Scarab: A Prototyping Tool for SAT-based Constraint Programming Systems

Takehide Soh¹, Naoyuki Tamura¹, Mutsunori Banbara¹, Daniel Le Berre² and Stéphanie Roussel²

1) Kobe University
2) CRIL-CNRS, UMR 8188

Joint Seminar on Theory, Implementation, and Applications of Logic and Inference
(July 25th, 2013 at Hokkaido University)
Modern fast SAT solvers have promoted the development of **SAT-based systems** for various problems.

For an intended problem, we usually need to develop a dedicated program that encodes it into SAT.

It sometimes bothers focusing on **problem modeling** which plays an important role in the system development process.

**In this talk**

- We introduce the **Scarab** system, which is a prototyping tool for developing SAT-based systems.
- Its features are also introduced through examples of **Graph Coloring** and **Pandiagonal Latin Square**.
SAT technology ... it can solve CNF problems of immense size.

But solving CNF problems ignores one important fact: there are NO problems that are originally CNF.

Modeling is important

Problem $\xrightarrow{\text{Modeling}}$ Conceptual Model $\xrightarrow{\text{Encoding}}$ Design Model

- Conceptual Model: A formal mathematical statement
- Design Model: In the form that can be handled by a solver
Contents of Talk

1. Getting Started: Overview of Scarab
2. Designing Constraint Models in Scarab
3. Advanced Solving Techniques using Sat4j
Contents of Talk

1. Getting Started: Overview of Scarab
2. Designing Constraint Models in Scarab
3. Advanced Solving Techniques using Sat4j
Scarab is a prototyping tool for developing SAT-based Constraint Programming (CP) systems.
Scarab is a prototyping tool for developing SAT-based Constraint Programming (CP) systems. It consists of 1) CP Domain-Specific Language, 2) API of CSP solver, 3) SAT encoding module, and 4) API of SAT solvers. It uses Order Encoding and Sat4j in default. Implemented by 500 lines of Scala.
**Scarab** is a prototyping tool for developing SAT-based Constraint Programming (CP) systems.

- It consists of 1) CP Domain-Specific Language, 2) API of CSP solver, 3) SAT encoding module, and 4) API of SAT solvers.
- It uses **Order Encoding** and **Sat4j** in default.

Scarab is developed to be an expressive, efficient, customizable, and portable workbench. The tight integration to Sat4j enables advanced CSP solving such as incremental solving and the use of assumptions.
**Scarab** is a prototyping tool for developing SAT-based Constraint Programming (CP) systems. It consists of 1) CP Domain-Specific Language, 2) API of CSP solver, 3) SAT encoding module, and 4) API of SAT solvers. It uses **Order Encoding** and **Sat4j** in default.
**Scarab** is a prototyping tool for developing SAT-based Constraint Programming (CP) systems.

- It consists of 1) CP Domain-Specific Language, 2) API of CSP solver, 3) SAT encoding module, and 4) API of SAT solvers.
- It uses **Order Encoding** and **Sat4j** in default.
Scarab is a prototyping tool for developing SAT-based Constraint Programming (CP) systems. It consists of 1) CP Domain-Specific Language, 2) API of CSP solver, 3) SAT encoding module, and 4) API of SAT solvers. It uses Order Encoding and Sat4j in default.
Scarab is a prototyping tool for developing SAT-based Constraint Programming (CP) systems. It consists of 1) CP Domain-Specific Language, 2) API of CSP solver, 3) SAT encoding module, and 4) API of SAT solvers. It uses Order Encoding and Sat4j in default. It is developed to be an expressive, efficient, customizable, and portable workbench.
Scarab is a prototyping tool for developing SAT-based Constraint Programming (CP) systems. It consists of 1) CP Domain-Specific Language, 2) API of CSP solver, 3) SAT encoding module, and 4) API of SAT solvers. It uses Order Encoding and Sat4j in default. It is developed to be an expressive, efficient, customizable, and portable workbench. The tight integration to Sat4j enables advanced CSP solving such as incremental solving and the use of assumptions.
Graph coloring problem (GCP) is a problem of finding a coloring of the nodes such that colors of adjacent nodes are different.

```
1: import jp.kobe_u.scarab.csp._
2: import jp.kobe_u.scarab.solver._
3: import jp.kobe_u.scarab.sapp._
4:
5: val nodes = Seq(1,2,3,4,5)
6: val edges = Seq((1,2),(1,5),(2,3),(2,4),(3,4),(4,5))
7: val colors = 3
8: for (i <- nodes) int('n(i),1,colors)
9: for ((i,j) <- edges) add('n(i) !== 'n(j))
10:
11: if (find) println(solution)
```
Imports

import jp.kobe_u.scarab.csp._
import jp.kobe_u.scarab.solver._
import jp.kobe_u.scarab.sapp._

- First 2 lines import classes of CSP and CSP solver.
- Third line imports the default CSP, Encoder, SAT Solver, and CSP Solver objects.
- It also imports DSL methods provided by Scarab.
  - `int(x, lb, ub)` method defines an integer variable.
  - `add(c)` method defines a constraint.
  - `find` method searches a solution.
  - `solution` method returns the solution.
  - etc.
Contents of Talk

1. Getting Started: Overview of Scarab
2. Designing Constraint Models in Scarab
3. Advanced Solving Techniques using Sat4j
Contents of Talk

1. Getting Started: Overview of Scarab
2. Designing Constraint Models in Scarab
3. Advanced Solving Techniques using Sat4j
Pandigonal Latin Square: \( PLS(n) \)

Place different \( n \) numbers into \( n \times n \) matrix such that each number appears exactly once for each row, column, diagonally down right, and diagonally up right.

\[
\begin{array}{cccc}
2 & 3 & 5 & 1 & 4 \\
5 & 1 & 4 & 2 & 3 \\
4 & 2 & 3 & 5 & 1 \\
3 & 5 & 1 & 4 & 2 \\
1 & 4 & 2 & 3 & 5 \\
\end{array}
\]
Pandigonal Latin Square: $PLS(n)$

Place different $n$ numbers into $n \times n$ matrix such that each number appears exactly once for each row, column, diagonally down right, and diagonally up right.

We can write five SAT-based PLS Solvers within 35 lines.

<table>
<thead>
<tr>
<th>Name</th>
<th>Modeling</th>
<th>Encoding</th>
<th>Lines</th>
</tr>
</thead>
<tbody>
<tr>
<td>AD1</td>
<td>alldiff</td>
<td>naive</td>
<td>17</td>
</tr>
<tr>
<td>AD2</td>
<td>with Perm. &amp; P. H. Const.</td>
<td>31</td>
<td></td>
</tr>
<tr>
<td>BC1</td>
<td>Boolean</td>
<td>Pairwise</td>
<td>22</td>
</tr>
<tr>
<td>BC2</td>
<td>Cardinality</td>
<td>Totalizer [Bailleux ‘03]</td>
<td>35</td>
</tr>
<tr>
<td>BC3</td>
<td>Seq. Counter [Sinz ‘05]</td>
<td>27</td>
<td></td>
</tr>
</tbody>
</table>

Let’s have a look their performance. Note that, in CSP Solver Comp. 2009, NO CSP solver (except Sugar) could solve $n > 8$. 

T. Soh, N, Tamura, M. Banbara, D. Le Berre, and S. Roussel
Scarab: a Prototyping Tool for SAT-based CP Systems
## Results (CPU Time in Seconds)

<table>
<thead>
<tr>
<th>n</th>
<th>SAT/UNSAT</th>
<th>AD1</th>
<th>AD2</th>
<th>BC1</th>
<th>BC2</th>
<th>BC3</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>SAT</td>
<td>0.2</td>
<td>0.2</td>
<td>0.2</td>
<td>0.3</td>
<td>0.3</td>
</tr>
<tr>
<td>8</td>
<td>UNSAT</td>
<td>T.O.</td>
<td>0.5</td>
<td>0.3</td>
<td>0.3</td>
<td>0.3</td>
</tr>
<tr>
<td>9</td>
<td>UNSAT</td>
<td>T.O.</td>
<td>0.3</td>
<td>0.5</td>
<td>0.3</td>
<td>0.2</td>
</tr>
<tr>
<td>10</td>
<td>UNSAT</td>
<td>T.O.</td>
<td>0.4</td>
<td>1.0</td>
<td>0.3</td>
<td>0.3</td>
</tr>
<tr>
<td>11</td>
<td>SAT</td>
<td>0.3</td>
<td>0.3</td>
<td>2.3</td>
<td>0.5</td>
<td>0.4</td>
</tr>
<tr>
<td>12</td>
<td>UNSAT</td>
<td>T.O.</td>
<td>1.0</td>
<td>5.3</td>
<td>0.8</td>
<td>0.8</td>
</tr>
<tr>
<td>13</td>
<td>SAT</td>
<td>T.O.</td>
<td>0.5</td>
<td>T.O.</td>
<td>T.O.</td>
<td>T.O.</td>
</tr>
<tr>
<td>14</td>
<td>UNSAT</td>
<td>T.O.</td>
<td>9.7</td>
<td>32.4</td>
<td>8.2</td>
<td>6.8</td>
</tr>
<tr>
<td>15</td>
<td>UNSAT</td>
<td>T.O.</td>
<td>388.9</td>
<td>322.7</td>
<td>194.6</td>
<td>155.8</td>
</tr>
<tr>
<td>16</td>
<td>UNSAT</td>
<td>T.O.</td>
<td>457.1</td>
<td>546.6</td>
<td>300.7</td>
<td>414.8</td>
</tr>
</tbody>
</table>

- Optimized version of alldiff model (AD2) solved all instances.
- **Modeling** and **encoding** have an important role in developing SAT-based systems and **Scarab** helps us to focus on them.
Contents of Talk

1. Getting Started: Overview of Scarab
2. Designing Constraint Models in Scarab
3. Advanced Solving Techniques using Sat4j
Contents of Talk

1. Getting Started: Overview of Scarab
2. Designing Constraint Models in Scarab
3. Advanced Solving Techniques using Sat4j
Advanced Solving

- Incremental SAT Solving
- CSP Solving under Assumption
- Commit/Rollback
Conclusion

- Introducing Architecture and Features of Scarab
- Using Scarab, we can write various constraint models without developing dedicated encoders, which allows us to focus on problem modeling and encoding.

Future Work
- Introducing more features from Sat4j
- Introducing more kinds of back-end solvers
Supplemental Slides
## Table: Truth table of $p(x \leq a)$

<table>
<thead>
<tr>
<th>$x$</th>
<th>$p(x \leq 0)$</th>
<th>$p(x \leq 1)$</th>
<th>$p(x \leq 2)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
Scala

- Scala is a relatively new programming language receiving an increasing interest for developing real-world applications.
- Scala is an integration of both functional and object-oriented programming paradigms.
- The main features of Scala are:
  - type inferences,
  - higher order functions,
  - immutable collections, and
  - concurrent computation.
- It is also suitable for implementing Domain-Specific Language (DSL) embedded in Scala.
- The Scala compiler generates Java Virtual Machine (JVM) bytecode, and Java class libraries can be used in Scala.
Contents of Talk

1 Getting Started: Overview of Scarab
   - Architecture and Features
   - Example: Graph Coloring Problem

2 Designing Constraint Models in Scarab
   - Pandiagonal Latin Square
   - alldiff Model
   - Boolean Cardinality Model

3 Advanced Solving Techniques using Sat4j
   - Incremental SAT Solving
   - CSP Solving under Assumption
Pandigonal Latin Square: $PLS(n)$

Place different $n$ numbers into $n \times n$ matrix such that each number appears exactly once for each row, column, diagonally down right, and diagonally up right.

\[
\begin{array}{cccc}
2 & 3 & 5 & 1 \\
5 & 1 & 4 & 2 \\
4 & 2 & 3 & 5 \\
3 & 5 & 1 & 4 \\
1 & 4 & 2 & 3 \\
\end{array}
\]
Pandigonal Latin Square: $PLS(n)$

Place different $n$ numbers into $n \times n$ matrix such that each number appears exactly once for each row, column, diagonally down right, and diagonally up right.

We can write five SAT-based PLS Solvers within 35 lines.

<table>
<thead>
<tr>
<th>Name</th>
<th>Modeling</th>
<th>Encoding</th>
<th>Lines</th>
</tr>
</thead>
<tbody>
<tr>
<td>AD1</td>
<td>alldiff</td>
<td>naive</td>
<td>17</td>
</tr>
<tr>
<td>AD2</td>
<td></td>
<td>with Perm. &amp; P. H. Const.</td>
<td>31</td>
</tr>
<tr>
<td>BC1</td>
<td>Boolean</td>
<td>Pairwise</td>
<td>22</td>
</tr>
<tr>
<td>BC2</td>
<td>Cardinality</td>
<td>Totalizer [Bailleux ‘03]</td>
<td>35</td>
</tr>
<tr>
<td>BC3</td>
<td></td>
<td>Seq. Counter [Sinz ‘05]</td>
<td>27</td>
</tr>
</tbody>
</table>

Let’s have a look their performance. Note that, in CSP Solver Comp. 2009, NO CSP solver (except Sugar) could solve $n > 8$. 
**Pandiagonal Latin Square** $PLS(n)$ is a problem of placing different $n$ numbers into $n \times n$ matrix such that each number is occurring exactly once for each row, column, diagonally down right, and diagonally up right.

- **alldiff Model**
  - One uses alldiff constraint, which is one of the best known and most studied global constraints in constraint programming.
  - The constraint $\text{alldiff}(a_1, \ldots, a_n)$ ensures that the values assigned to the variable $a_1, \ldots, a_n$ must be pairwise distinct.

- **Boolean Cardinality Model**
  - One uses Boolean cardinality constraint.
**Pandiagonal Latin Square** $PLS(5)$

<table>
<thead>
<tr>
<th></th>
<th>$x_{11}$</th>
<th>$x_{12}$</th>
<th>$x_{13}$</th>
<th>$x_{14}$</th>
<th>$x_{15}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$x_{21}$</td>
<td>$x_{22}$</td>
<td>$x_{23}$</td>
<td>$x_{24}$</td>
<td>$x_{25}$</td>
<td></td>
</tr>
<tr>
<td>$x_{31}$</td>
<td>$x_{32}$</td>
<td>$x_{33}$</td>
<td>$x_{34}$</td>
<td>$x_{35}$</td>
<td></td>
</tr>
<tr>
<td>$x_{41}$</td>
<td>$x_{42}$</td>
<td>$x_{43}$</td>
<td>$x_{44}$</td>
<td>$x_{45}$</td>
<td></td>
</tr>
<tr>
<td>$x_{51}$</td>
<td>$x_{52}$</td>
<td>$x_{53}$</td>
<td>$x_{54}$</td>
<td>$x_{55}$</td>
<td></td>
</tr>
</tbody>
</table>

- $x_{ij} \in \{1, 2, 3, 4, 5\}$

The Pandiagonal Latin Square $PLS(5)$ is satisfiable.
alldiff Model

**Pandiagonal Latin Square** $\text{PLS}(5)$

<table>
<thead>
<tr>
<th>$x_{11}$</th>
<th>$x_{12}$</th>
<th>$x_{13}$</th>
<th>$x_{14}$</th>
<th>$x_{15}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$x_{21}$</td>
<td>$x_{22}$</td>
<td>$x_{23}$</td>
<td>$x_{24}$</td>
<td>$x_{25}$</td>
</tr>
<tr>
<td>$x_{31}$</td>
<td>$x_{32}$</td>
<td>$x_{33}$</td>
<td>$x_{34}$</td>
<td>$x_{35}$</td>
</tr>
<tr>
<td>$x_{41}$</td>
<td>$x_{42}$</td>
<td>$x_{43}$</td>
<td>$x_{44}$</td>
<td>$x_{45}$</td>
</tr>
<tr>
<td>$x_{51}$</td>
<td>$x_{52}$</td>
<td>$x_{53}$</td>
<td>$x_{54}$</td>
<td>$x_{55}$</td>
</tr>
</tbody>
</table>

- $x_{ij} \in \{1, 2, 3, 4, 5\}$
- alldiff in each row (5 rows)

$\text{PLS}(5)$ is satisfiable.

T. Soh, N. Tamura, M. Banbara, D. Le Berre, and S. Roussel

Scarab: a Prototyping Tool for SAT-based CP Systems
alldiff Model

Pandigonal Latin Square \( PLS(5) \)

<table>
<thead>
<tr>
<th>( x_{11} )</th>
<th>( x_{12} )</th>
<th>( x_{13} )</th>
<th>( x_{14} )</th>
<th>( x_{15} )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( x_{21} )</td>
<td>( x_{22} )</td>
<td>( x_{23} )</td>
<td>( x_{24} )</td>
<td>( x_{25} )</td>
</tr>
<tr>
<td>( x_{31} )</td>
<td>( x_{32} )</td>
<td>( x_{33} )</td>
<td>( x_{34} )</td>
<td>( x_{35} )</td>
</tr>
<tr>
<td>( x_{41} )</td>
<td>( x_{42} )</td>
<td>( x_{43} )</td>
<td>( x_{44} )</td>
<td>( x_{45} )</td>
</tr>
<tr>
<td>( x_{51} )</td>
<td>( x_{52} )</td>
<td>( x_{53} )</td>
<td>( x_{54} )</td>
<td>( x_{55} )</td>
</tr>
</tbody>
</table>

- \( x_{ij} \in \{1, 2, 3, 4, 5\} \)
- alldiff in each row (5 rows)
alldiff Model

### Pandiagonal Latin Square \( PLS(5) \)

<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>4</td>
<td>5</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>4</td>
<td>5</td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>5</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
</tr>
</tbody>
</table>

- \( x_{ij} \in \{1, 2, 3, 4, 5\} \)
- alldiff in each row (5 rows)
- alldiff in each column (5 columns)
alldiff Model

Pandiagonal Latin Square $PLS(5)$

<table>
<thead>
<tr>
<th></th>
<th>$x_{11}$</th>
<th>$x_{12}$</th>
<th>$x_{13}$</th>
<th>$x_{14}$</th>
<th>$x_{15}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$x_{21}$</td>
<td></td>
<td>$x_{22}$</td>
<td>$x_{23}$</td>
<td>$x_{24}$</td>
<td>$x_{25}$</td>
</tr>
<tr>
<td>$x_{31}$</td>
<td>$x_{32}$</td>
<td></td>
<td>$x_{33}$</td>
<td>$x_{34}$</td>
<td>$x_{35}$</td>
</tr>
<tr>
<td>$x_{41}$</td>
<td>$x_{42}$</td>
<td>$x_{43}$</td>
<td></td>
<td>$x_{44}$</td>
<td>$x_{45}$</td>
</tr>
<tr>
<td>$x_{51}$</td>
<td>$x_{52}$</td>
<td>$x_{53}$</td>
<td>$x_{54}$</td>
<td></td>
<td>$x_{55}$</td>
</tr>
</tbody>
</table>

- $x_{ij} \in \{1, 2, 3, 4, 5\}$
- alldiff in each row (5 rows)
- alldiff in each column (5 columns)
**alldiff Model**

### Pandiagonal Latin Square $PLS(5)$

<table>
<thead>
<tr>
<th></th>
<th>$x_{11}$</th>
<th>$x_{12}$</th>
<th>$x_{13}$</th>
<th>$x_{14}$</th>
<th>$x_{15}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$x_{21}$</td>
<td>$x_{22}$</td>
<td>$x_{23}$</td>
<td>$x_{24}$</td>
<td>$x_{25}$</td>
<td></td>
</tr>
<tr>
<td>$x_{31}$</td>
<td>$x_{32}$</td>
<td>$x_{33}$</td>
<td>$x_{34}$</td>
<td>$x_{35}$</td>
<td></td>
</tr>
<tr>
<td>$x_{41}$</td>
<td>$x_{42}$</td>
<td>$x_{43}$</td>
<td>$x_{44}$</td>
<td>$x_{45}$</td>
<td></td>
</tr>
<tr>
<td>$x_{51}$</td>
<td>$x_{52}$</td>
<td>$x_{53}$</td>
<td>$x_{54}$</td>
<td>$x_{55}$</td>
<td></td>
</tr>
</tbody>
</table>

- $x_{ij} \in \{1, 2, 3, 4, 5\}$
- alldiff in each row (5 rows)
- alldiff in each column (5 columns)
- alldiff in each pandiagonal (10 pandiagonals)
alldiff Model

Pandiagonal Latin Square $PLS(5)$

- $x_{ij} \in \{1, 2, 3, 4, 5\}$
- alldiff in each row (5 rows)
- alldiff in each column (5 columns)
- alldiff in each pandiagonal (10 pandiagonals)
alldiff Model

Pandiagonal Latin Square \( PLS(5) \)

- \( x_{ij} \in \{1, 2, 3, 4, 5\} \)
- alldiff in each row (5 rows)
- alldiff in each column (5 columns)
- alldiff in each pandiagonal (10 pandiagonals)
### Pandiagonal Latin Square $PLS(5)$

<table>
<thead>
<tr>
<th>$x_{11}$</th>
<th>$x_{12}$</th>
<th>$x_{13}$</th>
<th>$x_{14}$</th>
<th>$x_{15}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$x_{21}$</td>
<td>$x_{22}$</td>
<td>$x_{23}$</td>
<td>$x_{24}$</td>
<td>$x_{25}$</td>
</tr>
<tr>
<td>$x_{31}$</td>
<td>$x_{32}$</td>
<td>$x_{33}$</td>
<td>$x_{34}$</td>
<td>$x_{35}$</td>
</tr>
<tr>
<td>$x_{41}$</td>
<td>$x_{42}$</td>
<td>$x_{43}$</td>
<td>$x_{44}$</td>
<td>$x_{45}$</td>
</tr>
<tr>
<td>$x_{51}$</td>
<td>$x_{52}$</td>
<td>$x_{53}$</td>
<td>$x_{54}$</td>
<td>$x_{55}$</td>
</tr>
</tbody>
</table>

- $x_{ij} \in \{1, 2, 3, 4, 5\}$
- alldiff in each row (5 rows)
- alldiff in each column (5 columns)
- alldiff in each pandiagonal (10 pandiagonals)
### Pandiagonal Latin Square $PLS(5)$

<table>
<thead>
<tr>
<th></th>
<th>$x_{11}$</th>
<th>$x_{12}$</th>
<th>$x_{13}$</th>
<th>$x_{14}$</th>
<th>$x_{15}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$x_{21}$</td>
<td>$x_{22}$</td>
<td>$x_{23}$</td>
<td>$x_{24}$</td>
<td>$x_{25}$</td>
<td></td>
</tr>
<tr>
<td>$x_{31}$</td>
<td>$x_{32}$</td>
<td>$x_{33}$</td>
<td>$x_{34}$</td>
<td>$x_{35}$</td>
<td></td>
</tr>
<tr>
<td>$x_{41}$</td>
<td>$x_{42}$</td>
<td>$x_{43}$</td>
<td>$x_{44}$</td>
<td>$x_{45}$</td>
<td></td>
</tr>
<tr>
<td>$x_{51}$</td>
<td>$x_{52}$</td>
<td>$x_{53}$</td>
<td>$x_{54}$</td>
<td>$x_{55}$</td>
<td></td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>5</td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
</tbody>
</table>

- $x_{ij} \in \{1, 2, 3, 4, 5\}$
- alldiff in each row (5 rows)
- alldiff in each column (5 columns)
- alldiff in each pandiagonal (10 pandiagonals)
- $PLS(5)$ is satisfiable.
Scarab Program for alldiff Model

```scala
1: import jp.kobe_u.scarab.csp._
2: import jp.kobe_u.scarab.solver._
3: import jp.kobe_u.scarab.sapp._
4:
5: val n = args(0).toInt
6:
7: for (i <- 1 to n; j <- 1 to n) int('x(i,j),1,n)
8: for (i <- 1 to n) {
9:   add(alldiff((1 to n).map(j => 'x(i,j))))
10:  add(alldiff((1 to n).map(j => 'x(j,i))))
11:  add(alldiff((1 to n).map(j => 'x(j,(i+(j-1)*(n-1))%n+1))))
12:  add(alldiff((1 to n).map(j => 'x(j,(i+j-1)%n+1))))
13: }
14:
15: if (find) println(solution)
```
In Scarab, all we have to do for implementing global constraints is just decomposing them into simple arithmetic constraints [Bessiere et al. ‘09].

In the case of \text{alldiff}(a_1, \ldots, a_n),

It is decomposed into pairwise not-equal constraints

\[
\bigwedge_{1 \leq i < j \leq n} (a_i \neq a_j)
\]

. 

This (naive) \text{alldiff} is enough to just have a feasible constraint model for \text{PLS}(n).

But, one probably want to improve this :)
Extra Constraints for \text{alldiff}(a_1, \ldots, a_n)

- In Pandiagonal Latin Square \(PLS(n)\), all integer variables \(a_1, \ldots, a_n\) have the same domain \(\{1, \ldots, n\}\).
- Then, we can add the following extra constraints.

**Permutation constraints:**

\[
\bigwedge_{i=1}^{n} \bigvee_{j=1}^{n} (a_j = i)
\]

- It represents that one of \(a_1, \ldots, a_n\) must be assigned to \(i\).

**Pigeon hole constraint:**

\[
\neg \bigwedge_{i=1}^{n} (a_i < n) \land \neg \bigwedge_{i=1}^{n} (a_i > 1)
\]

- It represents that mutually different \(n\) variables cannot be assigned within the interval of the size \(n - 1\).
alldiff (naive)

def alldiff(xs: Seq[Var]) =
    And(for (Seq(x, y) <- xs.combinations(2))
        yield x !== y)
def alldiff(xs: Seq[Var]) = {
    val lb = for (x <- xs) yield csp.dom(x).lb
    val ub = for (x <- xs) yield csp.dom(x).ub
    // pigeon hole
    val ph =
        And(Or(for (x <- xs) yield !(x < lb.min+xs.size-1)),
            Or(for (x <- xs) yield !(x > ub.max-xs.size+1)))
    // permutation
    def perm =
        And(for (num <- lb.min to ub.max)
            yield Or(for (x <- xs) yield x === num))
    val extra = if (ub.max-lb.min+1 == xs.size) And(ph,perm)
                 else ph
    And(And(for (Seq(x, y) <- xs.combinations(2))
             yield x !== y),extra)
}
**Boolean Cardinality Model**

\[
\begin{array}{ccccc}
    y_{11k} & y_{12k} & y_{13k} & y_{14k} & y_{15k} \\
    y_{21k} & y_{22k} & y_{23k} & y_{24k} & y_{25k} \\
    y_{31k} & y_{32k} & y_{33k} & y_{34k} & y_{35k} \\
    y_{41k} & y_{42k} & y_{43k} & y_{44k} & y_{45k} \\
    y_{51k} & y_{52k} & y_{53k} & y_{54k} & y_{55k} \\
\end{array}
\]

- \( y_{ijk} \in \{0, 1\} \)
- \( y_{ijk} = 1 \iff k \text{ is placed at } (i, j) \)
### Boolean Cardinality Model

<table>
<thead>
<tr>
<th>$y_{11k}$</th>
<th>$y_{12k}$</th>
<th>$y_{13k}$</th>
<th>$y_{14k}$</th>
<th>$y_{15k}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$y_{21k}$</td>
<td>$y_{22k}$</td>
<td>$y_{23k}$</td>
<td>$y_{24k}$</td>
<td>$y_{25k}$</td>
</tr>
<tr>
<td>$y_{31k}$</td>
<td>$y_{32k}$</td>
<td>$y_{33k}$</td>
<td>$y_{34k}$</td>
<td>$y_{35k}$</td>
</tr>
<tr>
<td>$y_{41k}$</td>
<td>$y_{42k}$</td>
<td>$y_{43k}$</td>
<td>$y_{44k}$</td>
<td>$y_{45k}$</td>
</tr>
<tr>
<td>$y_{51k}$</td>
<td>$y_{52k}$</td>
<td>$y_{53k}$</td>
<td>$y_{54k}$</td>
<td>$y_{55k}$</td>
</tr>
</tbody>
</table>

- $y_{ijk} \in \{0, 1\}$
- $y_{ijk} = 1 \iff k$ is placed at $(i, j)$
- for each value (5 values)
  - for each row (5 rows)
    - $y_{i1k} + y_{i2k} + y_{i3k} + y_{i4k} + y_{i5k} = 1$
**Boolean Cardinality Model**

\[
\begin{array}{cccccc}
 y_{11k} & y_{12k} & y_{13k} & y_{14k} & y_{15k} \\
 y_{21k} & y_{22k} & y_{23k} & y_{24k} & y_{25k} \\
 y_{31k} & y_{32k} & y_{33k} & y_{34k} & y_{35k} \\
 y_{41k} & y_{42k} & y_{43k} & y_{44k} & y_{45k} \\
 y_{51k} & y_{52k} & y_{53k} & y_{54k} & y_{55k} \\
\end{array}
\]

- \( y_{ijk} \in \{0, 1\} \)  
- \( y_{ijk} = 1 \iff k \text{ is placed at } (i, j) \)
- for each value (5 values)
  - for each row (5 rows)  
    - \( y_{i1k} + y_{i2k} + y_{i3k} + y_{i4k} + y_{i5k} = 1 \)
### Boolean Cardinality Model

| \( y_{11k} \) | \( y_{12k} \) | \( y_{13k} \) | \( y_{14k} \) | \( y_{15k} \) |
| \( y_{21k} \) | \( y_{22k} \) | \( y_{23k} \) | \( y_{24k} \) | \( y_{25k} \) |
| \( y_{31k} \) | \( y_{32k} \) | \( y_{33k} \) | \( y_{34k} \) | \( y_{35k} \) |
| \( y_{41k} \) | \( y_{42k} \) | \( y_{43k} \) | \( y_{44k} \) | \( y_{45k} \) |
| \( y_{51k} \) | \( y_{52k} \) | \( y_{53k} \) | \( y_{54k} \) | \( y_{55k} \) |

- \( y_{ijk} \in \{0, 1\} \)
- \( y_{ijk} = 1 \iff k \) is placed at \((i, j)\)
- for each value (5 values)
  - for each row (5 rows)
  - for each column (5 columns)

\[
\begin{align*}
\sum_{i=1}^{5} y_{1ik} &= 1 \\
\sum_{i=1}^{5} y_{ijk} &= 1 \\
\sum_{i=1}^{5} y_{ijk} &= 1 \\
\sum_{i=1}^{5} y_{ijk} &= 1 \\
\sum_{i=1}^{5} y_{ijk} &= 1
\end{align*}
\]
## Boolean Cardinality Model

<table>
<thead>
<tr>
<th>$y_{11k}$</th>
<th>$y_{12k}$</th>
<th>$y_{13k}$</th>
<th>$y_{14k}$</th>
<th>$y_{15k}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$y_{21k}$</td>
<td>$y_{22k}$</td>
<td>$y_{23k}$</td>
<td>$y_{24k}$</td>
<td>$y_{25k}$</td>
</tr>
<tr>
<td>$y_{31k}$</td>
<td>$y_{32k}$</td>
<td>$y_{33k}$</td>
<td>$y_{34k}$</td>
<td>$y_{35k}$</td>
</tr>
<tr>
<td>$y_{41k}$</td>
<td>$y_{42k}$</td>
<td>$y_{43k}$</td>
<td>$y_{44k}$</td>
<td>$y_{45k}$</td>
</tr>
<tr>
<td>$y_{51k}$</td>
<td>$y_{52k}$</td>
<td>$y_{53k}$</td>
<td>$y_{54k}$</td>
<td>$y_{55k}$</td>
</tr>
</tbody>
</table>

- $y_{ijk} \in \{0, 1\}$
- $y_{ijk} = 1 \iff k$ is placed at $(i, j)$

- for each value (5 values)
  - for each row (5 rows) $y_{i1k} + y_{i2k} + y_{i3k} + y_{i4k} + y_{i5k} = 1$
  - for each column (5 columns) $y_{1jk} + y_{2jk} + y_{3jk} + y_{4jk} + y_{5jk} = 1$
### Boolean Cardinality Model

<table>
<thead>
<tr>
<th>$y_{11k}$</th>
<th>$y_{12k}$</th>
<th>$y_{13k}$</th>
<th>$y_{14k}$</th>
<th>$y_{15k}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$y_{21k}$</td>
<td>$y_{22k}$</td>
<td>$y_{23k}$</td>
<td>$y_{24k}$</td>
<td>$y_{25k}$</td>
</tr>
<tr>
<td>$y_{31k}$</td>
<td>$y_{32k}$</td>
<td>$y_{33k}$</td>
<td>$y_{34k}$</td>
<td>$y_{35k}$</td>
</tr>
<tr>
<td>$y_{41k}$</td>
<td>$y_{42k}$</td>
<td>$y_{43k}$</td>
<td>$y_{44k}$</td>
<td>$y_{45k}$</td>
</tr>
<tr>
<td>$y_{51k}$</td>
<td>$y_{52k}$</td>
<td>$y_{53k}$</td>
<td>$y_{54k}$</td>
<td>$y_{55k}$</td>
</tr>
</tbody>
</table>

- $y_{ijk} \in \{0, 1\}$
- $y_{ijk} = 1 \iff k$ is placed at $(i, j)$

For each value (5 values):
- For each row (5 rows)
  - $y_{i1k} + y_{i2k} + y_{i3k} + y_{i4k} + y_{i5k} = 1$
- For each column (5 columns)
  - $y_{1jk} + y_{2jk} + y_{3jk} + y_{4jk} + y_{5jk} = 1$
- For each pandiagonal (10 pandiagonals)
  - $y_{11k} + y_{22k} + y_{33k} + y_{44k} + y_{55k} = 1$
### Boolean Cardinality Model

<table>
<thead>
<tr>
<th>$y_{11k}$</th>
<th>$y_{12k}$</th>
<th>$y_{13k}$</th>
<th>$y_{14k}$</th>
<th>$y_{15k}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$y_{21k}$</td>
<td>$y_{22k}$</td>
<td>$y_{23k}$</td>
<td>$y_{24k}$</td>
<td>$y_{25k}$</td>
</tr>
<tr>
<td>$y_{31k}$</td>
<td>$y_{32k}$</td>
<td>$y_{33k}$</td>
<td>$y_{34k}$</td>
<td>$y_{35k}$</td>
</tr>
<tr>
<td>$y_{41k}$</td>
<td>$y_{42k}$</td>
<td>$y_{43k}$</td>
<td>$y_{44k}$</td>
<td>$y_{45k}$</td>
</tr>
<tr>
<td>$y_{51k}$</td>
<td>$y_{52k}$</td>
<td>$y_{53k}$</td>
<td>$y_{54k}$</td>
<td>$y_{55k}$</td>
</tr>
</tbody>
</table>

- $y_{ijk} \in \{0, 1\}$
- $y_{ijk} = 1 \iff k$ is placed at $(i, j)$

- For each value (5 values)
  - For each row (5 rows) $y_{i1k} + y_{i2k} + y_{i3k} + y_{i4k} + y_{i5k} = 1$
  - For each column (5 columns) $y_{1jk} + y_{2jk} + y_{3jk} + y_{4jk} + y_{5jk} = 1$
  - For each pandiagonal (10 pandiagonals) $y_{11k} + y_{22k} + y_{33k} + y_{44k} + y_{55k} = 1$
**Boolean Cardinality Model**

- \( y_{ijk} \in \{0, 1\} \quad y_{ijk} = 1 \iff k \) is placed at \((i,j)\)
- for each value (5 values)
  - for each row (5 rows) \( y_{i1k} + y_{i2k} + y_{i3k} + y_{i4k} + y_{i5k} = 1 \)
  - for each column (5 columns) \( y_{1jk} + y_{2jk} + y_{3jk} + y_{4jk} + y_{5jk} = 1 \)
  - for each pandiagonal (10 pandiagonals) \( y_{11k} + y_{22k} + y_{33k} + y_{44k} + y_{55k} = 1 \)
Boolean Cardinality Model

\[
\begin{array}{cccccc}
y_{11k} & y_{12k} & y_{13k} & y_{14k} & y_{15k} \\
y_{21k} & y_{22k} & y_{23k} & y_{24k} & y_{25k} \\
y_{31k} & y_{32k} & y_{33k} & y_{34k} & y_{35k} \\
y_{41k} & y_{42k} & y_{43k} & y_{44k} & y_{45k} \\
y_{51k} & y_{52k} & y_{53k} & y_{54k} & y_{55k}
\end{array}
\]

- \( y_{ijk} \in \{0, 1\} \)
- \( y_{ijk} = 1 \iff k \text{ is placed at } (i, j) \)

For each value (5 values):
- For each row (5 rows): \( y_{i1k} + y_{i2k} + y_{i3k} + y_{i4k} + y_{i5k} = 1 \)
- For each column (5 columns): \( y_{1jk} + y_{2jk} + y_{3jk} + y_{4jk} + y_{5jk} = 1 \)
- For each pandiagonal (10 pandiagonals):
  \[
  y_{11k} + y_{22k} + y_{33k} + y_{44k} + y_{55k} = 1
  \]
Boolean Cardinality Model

\[ y_{ijk} \in \{0, 1\} \quad y_{ijk} = 1 \iff k \text{ is placed at } (i, j) \]

- for each value (5 values)
  - for each row (5 rows)
  - for each column (5 columns)
  - for each pandiagonal (10 pandiagonals)
  - for each \((i, j)\) position (25 positions)
Scarab Program for Boolean Cardinality Model

1: import jp.kobe_u.scarab.csp._
2: import jp.kobe_u.scarab.solver._
3: import jp.kobe_u.scarab.sapp._
4:
5: for (i <- 1 to n; j <- 1 to n; num <- 1 to n)
6: int('y(i,j,num),0,1)
7:
8: for (num <- 1 to n) {
9:   for (i <- 1 to n) {
10:      add(BC((1 to n).map(j => 'y(i,j,num)))===1)
11:      add(BC((1 to n).map(j => 'y(j,i,num)))===1)
12:      add(BC((1 to n).map(j => 'y(j,(i+j-1)%n+1,num))) === 1)
13:      add(BC((1 to n).map(j => 'y(j,(i+(j-1)*(n-1))%n+1,num))) === 1)
14:   }
15: }
16:
17: for (i <- 1 to n; j <- 1 to n)
18:   add(BC((1 to n).map(k => 'y(i,j,k)) === 1)
19:
20: if (find) println(solution)
There are several ways for encoding Boolean cardinality.

In Scarab, we can easily write the following encoding methods by defining your own BC methods.

- Pairwise
- Totalizer [Bailleux ‘03]
- Sequential Counter [Sinz ‘05]

In total, 3 variants of Boolean cardinality model are obtained.

- BC1: Pairwise (implemented by 2 lines)
- BC2: Totalizer [Bailleux ‘03] (implemented by 15 lines)
- BC3: Sequential Counter [Sinz ‘05] (implemented by 7 lines)

Good point to use Scarab is that we can test those models without writing dedicated programs.
Experiments

### Comparison on Solving Pandiagonal Latin Square

To show the differences in performance, we compared the following 5 models.

1. AD1: naive alldiff
2. AD2: optimized alldiff
3. BC1: Pairwise
4. BC2: [Bailleux ‘03]
5. BC3: [Sinz ‘05]

### Benchmark and Experimental Environment

- Benchmark: Pandiagonal Latin Square ($n = 7$ to $n = 16$)
- CPU: 2.93GHz, Mem: 2GB, Time Limit: 3600 seconds
### Results (CPU Time in Seconds)

<table>
<thead>
<tr>
<th>n</th>
<th>SAT/UNSAT</th>
<th>AD1</th>
<th>AD2</th>
<th>BC1</th>
<th>BC2</th>
<th>BC3</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>SAT</td>
<td>0.2</td>
<td>0.2</td>
<td>0.2</td>
<td>0.3</td>
<td>0.3</td>
</tr>
<tr>
<td>8</td>
<td>UNSAT</td>
<td>T.O.</td>
<td>0.5</td>
<td>0.3</td>
<td>0.3</td>
<td>0.3</td>
</tr>
<tr>
<td>9</td>
<td>UNSAT</td>
<td>T.O.</td>
<td>0.3</td>
<td>0.5</td>
<td>0.3</td>
<td>0.2</td>
</tr>
<tr>
<td>10</td>
<td>UNSAT</td>
<td>T.O.</td>
<td>0.4</td>
<td>1.0</td>
<td>0.3</td>
<td>0.3</td>
</tr>
<tr>
<td>11</td>
<td>SAT</td>
<td>0.3</td>
<td>0.3</td>
<td>2.3</td>
<td>0.5</td>
<td>0.4</td>
</tr>
<tr>
<td>12</td>
<td>UNSAT</td>
<td>T.O.</td>
<td>1.0</td>
<td>5.3</td>
<td>0.8</td>
<td>0.8</td>
</tr>
<tr>
<td>13</td>
<td>SAT</td>
<td>T.O.</td>
<td>0.5</td>
<td>T.O.</td>
<td>T.O.</td>
<td>T.O.</td>
</tr>
<tr>
<td>14</td>
<td>UNSAT</td>
<td>T.O.</td>
<td>9.7</td>
<td>32.4</td>
<td>8.2</td>
<td>6.8</td>
</tr>
<tr>
<td>15</td>
<td>UNSAT</td>
<td>T.O.</td>
<td>388.9</td>
<td>322.7</td>
<td>194.6</td>
<td>155.8</td>
</tr>
<tr>
<td>16</td>
<td>UNSAT</td>
<td>T.O.</td>
<td>457.1</td>
<td>546.6</td>
<td>300.7</td>
<td>414.8</td>
</tr>
</tbody>
</table>

- Only optimized version of alldiff model (AD2) solved all instances.
- Modeling and encoding have an important role in developing SAT-based systems.
- Scarab helps users to focus on them :)
BC1: Pairwise

**Definition of BC1**

```scala
def BC1(xs: Seq[Var]): Term = Sum(xs)
```
**BC1: Pairwise (cont.)**

**Scarab Program for** \( x + y + z = 1 \)

```plaintext
int('x,0,1)
int('y,0,1)
int('z,0,1)
add(BC1(Seq('x, 'y, 'z)) === 1)
```

**CNF Generated by Scarab**

\[
\begin{align*}
  p(x \leq 0) & \lor p(y \leq 0) & \lor & \lor & p(x \leq 0) & \lor p(z \leq 0) & \lor & \lor & p(y \leq 0) & \lor p(z \leq 0) & \lor & \lor & \neg p(x \leq 0) & \lor \neg p(y \leq 0) & \lor \neg p(z \leq 0) \\
  \{ x + y + z \leq 1 & \lor \lor & x + y + z \geq 1 \}
\end{align*}
\]
Definition of BC2

```scala
def BC2(xs: Seq[Var]): Term = {
  if (xs.size == 2) xs(0) + xs(1)
  else if (xs.size == 3) {
    val v = int(Var(), 0, 1)
    add(v === BC2(xs.drop(1)))
    xs(0) + v
  } else {
    val (xs1, xs2) = xs.splitAt(xs.size / 2)
    val v1 = int(Var(), 0, 1)
    val v2 = int(Var(), 0, 1)
    add(v1 === BC2(xs1))
    add(v2 === BC2(xs2))
    v1 + v2
  }
}
```
BC2: [Bailleux ‘03] (cont.)

Scarab Program for \( x + y + z = 1 \)

\[
\begin{align*}
\text{int}('x,0,1) \\
\text{int}('y,0,1) \\
\text{int}('z,0,1) \\
\text{add}(\text{BC2}(\text{Seq('x, 'y, 'z))} \iff 1)
\end{align*}
\]

CNF Generated by Scarab (q is auxiliary variable)

\[
\begin{align*}
q \lor \neg p(y \leq 0) \lor \neg p(z \leq 0) \\
\neg q \lor p(z \leq 0) \\
\neg q \lor p(y \leq 0) \\
p(y \leq 0) \lor p(z \leq 0) \\
q \lor p(x \leq 0) \\
\neg q \lor \neg p(x \leq 0)
\end{align*}
\]

\[
\begin{align*}
y + z = S \\
x + S = 1
\end{align*}
\]
Definition of BC3

```scala
def BC3(xs: Seq[Var]): Term = {
    val ss =
        for (i <- 1 until xs.size) yield int(Var(), 0, 1)
    add(ss(0) === xs(1) + xs(0))
    for (i <- 2 until xs.size)
        add(ss(i-1) === (xs(i) + ss(i-2)))
    ss(xs.size-2)
}
```
BC3: [Sinz ‘05] (cont.)

Program for $x + y + z = 1$

```
int('x,0,1)
int('y,0,1)
int('z,0,1)
add(BC3(Seq('x, 'y, 'z))===1)
```

CNF Generated by Scarab ($q_1$ and $q_2$ are auxiliary variables)

\[
\begin{align*}
q_1 &\lor \neg p(y \leq 0) \lor \neg p(x \leq 0) \\
\neg q_1 &\lor p(x \leq 0) \\
\neg q_1 &\lor p(y \leq 0) \\
\quad &\quad p(x \leq 0) \lor p(y \leq 0) \\
q_2 &\lor \neg q_1 \lor \neg p(z \leq 0) \\
\neg q_2 &\lor q_1 \\
\neg q_2 &\lor p(z \leq 0) \\
q_1 &\lor p(z \leq 0) \\
\neg q_2
\end{align*}
\]

\[
\begin{align*}
&\begin{cases}
\quad x + y = S_1 \\
\quad S_1 + z = S_2 \\
\quad S_2 = 1
\end{cases}
\end{align*}
\]
BC Native Encoder (work in progress)

- We have tested Boolean Cardinality Encoder (BC Native Encoder), which natively encodes Boolean cardinality constraints by using `addAtMost` or `addAtLeast` methods of Sat4j.
- Preliminary Results (CPU time in seconds)

<table>
<thead>
<tr>
<th>n</th>
<th>SAT/UNSAT</th>
<th>#Clauses (BC1)</th>
<th>#Constraints (BC Enc.)</th>
<th>time (sec) (BC1)</th>
<th>time (sec) (BC Enc.)</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>SAT</td>
<td>5341</td>
<td>441</td>
<td>0.1</td>
<td>0.1</td>
</tr>
<tr>
<td>8</td>
<td>UNSAT</td>
<td>9216</td>
<td>576</td>
<td>0.3</td>
<td>0.1</td>
</tr>
<tr>
<td>9</td>
<td>UNSAT</td>
<td>14904</td>
<td>729</td>
<td>0.5</td>
<td>0.1</td>
</tr>
<tr>
<td>10</td>
<td>UNSAT</td>
<td>22900</td>
<td>900</td>
<td>1.0</td>
<td>0.1</td>
</tr>
<tr>
<td>11</td>
<td>SAT</td>
<td>33759</td>
<td>1089</td>
<td>2.2</td>
<td>0.1</td>
</tr>
<tr>
<td>12</td>
<td>UNSAT</td>
<td>48096</td>
<td>1296</td>
<td>5.3</td>
<td>0.3</td>
</tr>
<tr>
<td>13</td>
<td>-</td>
<td>66586</td>
<td>1521</td>
<td>T.O.</td>
<td>T.O.</td>
</tr>
<tr>
<td>14</td>
<td>UNSAT</td>
<td>89964</td>
<td>1764</td>
<td>32.3</td>
<td>6.7</td>
</tr>
<tr>
<td>15</td>
<td>UNSAT</td>
<td>119025</td>
<td>2025</td>
<td>322.6</td>
<td>672.5</td>
</tr>
<tr>
<td>16</td>
<td>UNSAT</td>
<td>154624</td>
<td>2304</td>
<td>546.5</td>
<td>1321.4</td>
</tr>
</tbody>
</table>
**Example: Square Packing**

- **Square Packing** $SP(n, s)$ is a problem of packing a set of squares of sizes $1 \times 1$ to $n \times n$ into an enclosing square of size $s \times s$ without overlapping.

**Example of $SP(15, 36)$**

- **Optimum solution of $SP(n, s)$** is the smallest size of the enclosing square having a feasible packing.
Non-overlapping Constraint Model for $SP(n, s)$

Integer variables

- $x_i \in \{0, \ldots, s - i\}$ and $y_i \in \{0, \ldots, s - i\}$
- Each pair $(x_i, y_i)$ represents the lower left coordinates of the square $i$.

Non-overlapping Constraint ($1 \leq i < j \leq n$)

$$(x_i + i \leq x_j) \lor (x_j + j \leq x_i) \lor (y_i + i \leq y_j) \lor (y_j + j \leq y_i)$$
Decremental Search

Scarab Program for $SP(n,s)$

```plaintext
for (i <- 1 to n) { int('x(i),0,s-i) ; int('y(i),0,s-i) }
for (i <- 1 to n; j <- i+1 to n)
    add(('x(i) + i <= 'x(j)) || ('x(j) + j <= 'x(i)) || ('y(i) + i <= 'y(j)) || ('y(j) + j <= 'y(i)))
```

Searching an Optimum Solution

```plaintext
val lb = n; var ub = s; int('m, lb, ub)
for (i <- 1 to n)
    add(('x(i)+i <= 'm) && ('y(i)+i <= 'm))

// Incremental solving
while (lb <= ub && find('m <= ub)) { // using an assumption.
    add('m <= ub)
    ub = solution.intMap('m) - 1
}
```
Bisection Search

```plaintext
var lb = n; var ub = s; commit

while (lb < ub) {
    var size = (lb + ub) / 2
    for (i <- 1 to n)
        add(('x(i)+i<=size)&&('y(i)+i<=size))
    if (find) {
        ub = size
        commit // commit current constraints
    } else {
        lb = size + 1
        rollback // rollback to the last commit point
    }
}
```

T. Soh, N. Tamura, M. Banbara, D. Le Berre, and S. Roussel
Scarab: a Prototyping Tool for SAT-based CP Systems