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

Takehide Soh[1], Naoyuki Tamura[1], Mutsunori Banbara[1], Daniel Le Berre[2] and Stéphanie Roussel[2]

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)

Introduction

- 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**.

Invited Talk by Prof. Stuckey, SAT 2013

- 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

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

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

Contents of Talk

Contents of Talk

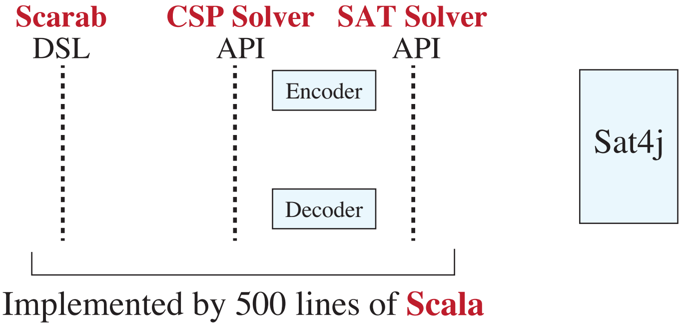
1. Getting Started: Overview of Scarab $\Longleftarrow$
2. Designing Constraint Models in Scarab
3. Advanced Solving Techniques using Sat4j

Architecture and Features

- **Scarab** is a prototyping tool for developing SAT-based Constraint Programming (CP) systems.

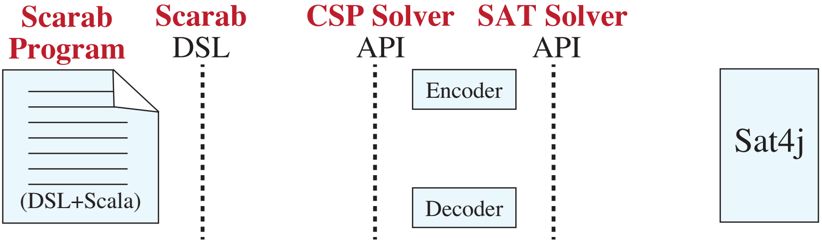
Architecture and Features

- **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.



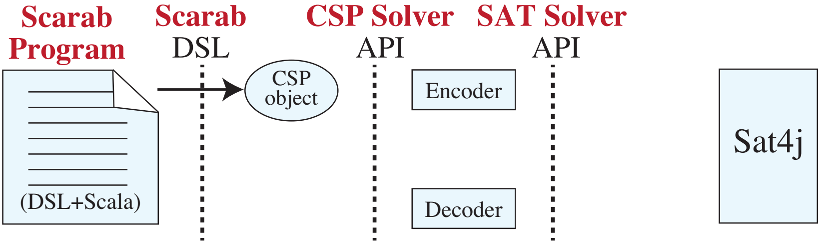
Architecture and Features

- **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.



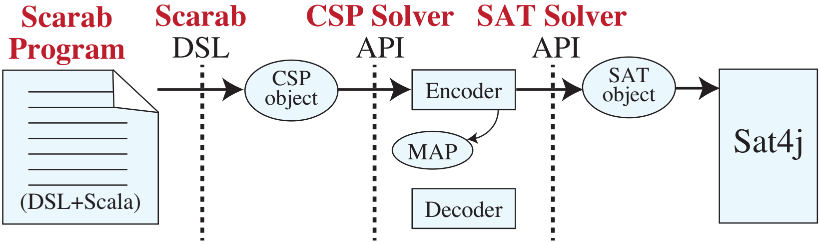
Architecture and Features

- **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.



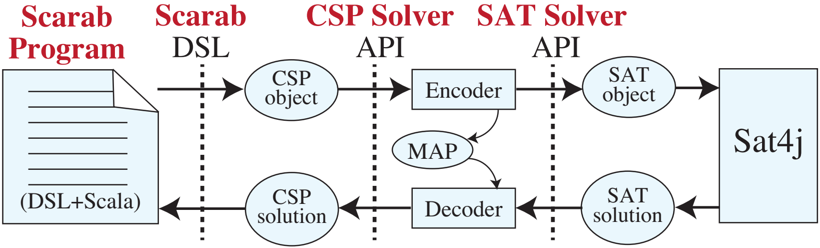
Architecture and Features

- **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.



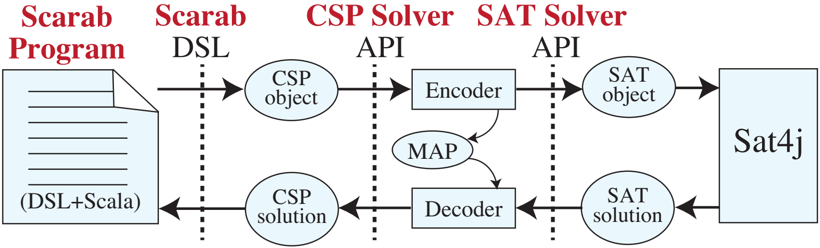
Architecture and Features

- **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.



Architecture and Features

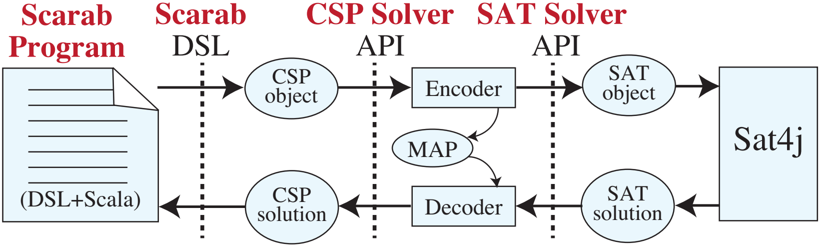
- **Scarab** is a prototyping tool for developing SAT-based Constraint Programming (CP) systems.
- It consists of 1) CP Domain-Specific Language, 2) API of CSP solver, 3) SAT encoding module, and 4) API of SAT solvers.
- It uses **Order Encoding** and **Sat4j** in default.



- It is developed to be an expressive, efficient, customizable, and portable workbench.

Architecture and Features

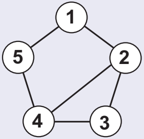
- **Scarab** is a prototyping tool for developing SAT-based Constraint Programming (CP) systems.
- It consists of 1) CP Domain-Specific Language, 2) API of CSP solver, 3) SAT encoding module, and 4) API of SAT solvers.
- It uses **Order Encoding** and **Sat4j** in default.



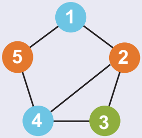
- It is developed to be an expressive, efficient, customizable, and portable workbench.
- The tight integration to Sat4j enables advanced CSP solving such as **incremental solving** and the use of **assumptions**.

Example of Scarab Program: GCP.scala

Graph coloring problem (GCP) is a problem of finding a coloring of the nodes such that colors of adjacent nodes are different.



Input



Solution

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

val nodes = Seq(1,2,3,4,5)
val edges = Seq((1,2),(1,5),(2,3),(2,4),(3,4),(4,5))
val colors = 3
for (i <- nodes) int('n(i),1,colors)
for ((i,j) <- edges)  add('n(i) !== 'n(j))

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

Pandiagonal 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.

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

Pandiagonal 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.

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

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

Name	Modeling	Encoding	Lines
AD1	alldiff	naive	17
AD2		with Perm. & P. H. Const.	31
BC1	Boolean	Pairwise	22
BC2	Cardinality	Totalizer [Bailleux '03]	35
BC3		Seq. Counter [Sinz '05]	27

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

Results (CPU Time in Seconds)

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

- 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

Contents of Talk

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

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

Encoded Variables in Order Encoding

Table: Truth table of $p(x \le a)$

x	$p(x \le 0)$	$p(x \le 1)$	$p(x \le 2)$
0	1	1	1
1	0	1	1
2	0	0	1
3	0	0	0

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** $\Longleftarrow$
   - Pandiagonal Latin Square
   - alldiff Model
   - Boolean Cardinality Model
3. **Advanced Solving Techniques using Sat4j**
   - Incremental SAT Solving
   - CSP Solving under Assumption

Pandiagonal 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.

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

Pandiagonal 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.

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

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

Name	Modeling	Encoding	Lines
AD1	alldiff	naive	17
AD2		with Perm. & P. H. Const.	31
BC1	Boolean	Pairwise	22
BC2	Cardinality	Totalizer [Bailleux '03]	35
BC3		Seq. Counter [Sinz '05]	27

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

Designing Constraint Models in Scarab

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 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.

alldiff Model

Pandiagonal Latin Square $PLS(5)$

x_{11}	x_{12}	x_{13}	x_{14}	x_{15}
x_{21}	x_{22}	x_{23}	x_{24}	x_{25}
x_{31}	x_{32}	x_{33}	x_{34}	x_{35}
x_{41}	x_{42}	x_{43}	x_{44}	x_{45}
x_{51}	x_{52}	x_{53}	x_{54}	x_{55}

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

alldiff Model

Pandiagonal Latin Square $PLS(5)$

x_{11}	x_{12}	x_{13}	x_{14}	x_{15}
x_{21}	x_{22}	x_{23}	x_{24}	x_{25}
x_{31}	x_{32}	x_{33}	x_{34}	x_{35}
x_{41}	x_{42}	x_{43}	x_{44}	x_{45}
x_{51}	x_{52}	x_{53}	x_{54}	x_{55}

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

alldiff Model

Pandiagonal Latin Square $PLS(5)$

x_{11}	x_{12}	x_{13}	x_{14}	x_{15}
x_{21}	x_{22}	x_{23}	x_{24}	x_{25}
x_{31}	x_{32}	x_{33}	x_{34}	x_{35}
x_{41}	x_{42}	x_{43}	x_{44}	x_{45}
x_{51}	x_{52}	x_{53}	x_{54}	x_{55}

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

alldiff Model

Pandiagonal Latin Square $PLS(5)$

x_{11}	x_{12}	x_{13}	x_{14}	x_{15}
x_{21}	x_{22}	x_{23}	x_{24}	x_{25}
x_{31}	x_{32}	x_{33}	x_{34}	x_{35}
x_{41}	x_{42}	x_{43}	x_{44}	x_{45}
x_{51}	x_{52}	x_{53}	x_{54}	x_{55}

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

alldiff Model

Pandiagonal Latin Square $PLS(5)$

x_{11}	x_{12}	x_{13}	x_{14}	x_{15}
x_{21}	x_{22}	x_{23}	x_{24}	x_{25}
x_{31}	x_{32}	x_{33}	x_{34}	x_{35}
x_{41}	x_{42}	x_{43}	x_{44}	x_{45}
x_{51}	x_{52}	x_{53}	x_{54}	x_{55}

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

alldiff Model

Pandiagonal Latin Square $PLS(5)$

x_{11}	x_{12}	x_{13}	x_{14}	x_{15}
x_{21}	x_{22}	x_{23}	x_{24}	x_{25}
x_{31}	x_{32}	x_{33}	x_{34}	x_{35}
x_{41}	x_{42}	x_{43}	x_{44}	x_{45}
x_{51}	x_{52}	x_{53}	x_{54}	x_{55}

- $x_{ij} \in \{1, 2, 3, 4, 5\}$
- 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_{11}	x_{12}	x_{13}	x_{14}	x_{15}
x_{21}	x_{22}	x_{23}	x_{24}	x_{25}
x_{31}	x_{32}	x_{33}	x_{34}	x_{35}
x_{41}	x_{42}	x_{43}	x_{44}	x_{45}
x_{51}	x_{52}	x_{53}	x_{54}	x_{55}

- $x_{ij} \in \{1, 2, 3, 4, 5\}$
- 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_{11}	x_{12}	x_{13}	x_{14}	x_{15}
x_{21}	x_{22}	x_{23}	x_{24}	x_{25}
x_{31}	x_{32}	x_{33}	x_{34}	x_{35}
x_{41}	x_{42}	x_{43}	x_{44}	x_{45}
x_{51}	x_{52}	x_{53}	x_{54}	x_{55}

- $x_{ij} \in \{1, 2, 3, 4, 5\}$
- 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_{11}	x_{12}	x_{13}	x_{14}	x_{15}
x_{21}	x_{22}	x_{23}	x_{24}	x_{25}
x_{31}	x_{32}	x_{33}	x_{34}	x_{35}
x_{41}	x_{42}	x_{43}	x_{44}	x_{45}
x_{51}	x_{52}	x_{53}	x_{54}	x_{55}

- $x_{ij} \in \{1, 2, 3, 4, 5\}$
- 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_{11}	x_{12}	x_{13}	x_{14}	x_{15}
x_{21}	x_{22}	x_{23}	x_{24}	x_{25}
x_{31}	x_{32}	x_{33}	x_{34}	x_{35}
x_{41}	x_{42}	x_{43}	x_{44}	x_{45}
x_{51}	x_{52}	x_{53}	x_{54}	x_{55}

1	2	3	4	5
3	4	5	1	2
5	1	2	3	4
2	3	4	5	1
4	5	1	2	3

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

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

val n = args(0).toInt

for (i <- 1 to n; j <- 1 to n)  int('x(i,j),1,n)
for (i <- 1 to n) {
 add(alldiff((1 to n).map(j => 'x(i,j))))
 add(alldiff((1 to n).map(j => 'x(j,i))))
 add(alldiff((1 to n).map(j => 'x(j,(i+j-1)%n+1))))
 add(alldiff((1 to n).map(j => 'x(j,(i+(j-1)*(n-1))%n+1))))
}

if (find)  println(solution)
```

Encoding alldiff

- 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 alldiff($a_1, \ldots, a_n$),

It is decomposed into pairwise not-equal constraints

$$\bigwedge_{1 \le i < j \le n} (a_i \neq a_j)$$

.

- This (naive) alldiff is enough to just have a feasible constraint model for $PLS(n)$.
- But, one probably want to improve this :)

Extra Constraints for 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**:

$$\neg \bigwedge_{i=1}^{n} (a_i < n) \wedge \neg \bigwedge_{i=1}^{n} (a_i > 1)$$

  - 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)
```

alldiff (optimized)

```
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

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}

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

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}

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

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}

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

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}

- $y_{ijk} \in \{0, 1\}$ $\qquad$ $y_{ijk} = 1 \Leftrightarrow k$ is placed at (i, j)
- for each value (5 values)
  - for each row (5 rows) $\qquad y_{i1k} + y_{i2k} + y_{i3k} + y_{i4k} + y_{i5k} = 1$
  - for each column (5 columns) $\qquad y_{1jk} + y_{2jk} + y_{3jk} + y_{4jk} + y_{5jk} = 1$

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}

- $y_{ijk} \in \{0, 1\}$ $\quad$ $y_{ijk} = 1 \Leftrightarrow k$ is placed at (i, j)
- for each value (5 values)
  - for each row (5 rows) $\quad y_{i1k} + y_{i2k} + y_{i3k} + y_{i4k} + y_{i5k} = 1$
  - for each column (5 columns) $\quad y_{1jk} + y_{2jk} + y_{3jk} + y_{4jk} + y_{5jk} = 1$

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}

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

Boolean Cardinality Model

y_{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}

- $y_{ijk} \in \{0, 1\}$ $y_{ijk} = 1 \Leftrightarrow k$ 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_{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}

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

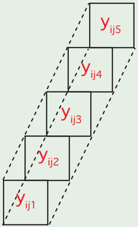
Boolean Cardinality Model

y_{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}

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

Boolean Cardinality Model

y_{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}



- $y_{ijk} \in \{0, 1\}$ $\quad$ $y_{ijk} = 1 \Leftrightarrow k$ is placed at (i, j)
- for each value (5 values)
  - for each row (5 rows) $\quad y_{i1k} + y_{i2k} + y_{i3k} + y_{i4k} + y_{i5k} = 1$
  - for each column (5 columns) $\quad y_{1jk} + y_{2jk} + y_{3jk} + y_{4jk} + y_{5jk} = 1$
  - for each pandiagonal (10 pandiagonals)
    $y_{11k} + y_{22k} + y_{33k} + y_{44k} + y_{55k} = 1$
- for each (i, j) position (25 positions) $\quad y_{ij1} + y_{ij2} + y_{ij3} + y_{ij4} + y_{ij5} = 1$

Scarab Program for Boolean Cardinality Model

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

for (i <- 1 to n; j <- 1 to n; num <- 1 to n)
  int('y(i,j,num),0,1)

for (num <- 1 to n) {
  for (i <- 1 to n) {
   add(BC((1 to n).map(j => 'y(i,j,num)))===1)
   add(BC((1 to n).map(j => 'y(j,i,num)))===1)
   add(BC((1 to n).map(j => 'y(j,(i+j-1)%n+1,num))) === 1)
   add(BC((1 to n).map(j => 'y(j,(i+(j-1)*(n-1))%n+1,num))) === 1)
  }
}

for (i <- 1 to n; j <- 1 to n)
  add(BC((1 to n).map(k => 'y(i,j,k))) === 1)

if (find)  println(solution)
```

SAT Encoding of Boolean Cardinality in Scarab

- 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)

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

- 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

```
def BC1(xs: Seq[Var]): Term = Sum(xs)
```

BC1: Pairwise (cont.)

Scarab Program for $x + y + z = 1$

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

CNF Generated by Scarab

$$\left.\begin{array}{lcl} p(x \le 0) & \vee & p(y \le 0) \\ p(x \le 0) & \vee & p(z \le 0) \\ p(y \le 0) & \vee & p(z \le 0) \end{array}\right\} x + y + z \le 1$$

$$\neg p(x \le 0) \vee \neg p(y \le 0) \vee \neg p(z \le 0) \quad \} \, x + y + z \ge 1$$

BC2: [Bailleux '03]

Definition of BC2

```
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 $x + y + z = 1$

```
int('x,0,1)
int('y,0,1)
int('z,0,1)
add(BC2(Seq('x, 'y, 'z)) === 1)
```

CNF Generated by Scarab (q is auxiliary variable)

$$\left.\begin{array}{ccccc} q & \vee & \neg p(y \leq 0) & \vee & \neg p(z \leq 0) \\ \neg q & \vee & p(z \leq 0) & & \\ \neg q & \vee & p(y \leq 0) & & \\ p(y \leq 0) & \vee & p(z \leq 0) & & \end{array}\right\} y + z = S$$

$$\left.\begin{array}{ccc} q & \vee & p(x \leq 0) \\ \neg q & \vee & \neg p(x \leq 0) \end{array}\right\} x + S = 1$$

BC3: [Sinz '05]

Definition of BC3

```
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)

$$\left.\begin{array}{lclcl} q_1 & \vee & \neg p(y \le 0) & \vee & \neg p(x \le 0) \\ \neg q_1 & \vee & p(x \le 0) & & \\ \neg q_1 & \vee & p(y \le 0) & & \\ & & p(x \le 0) & \vee & p(y \le 0) \end{array}\right\} x + y = S_1$$

$$\left.\begin{array}{lclcl} q_2 & \vee & \neg q_1 & \vee & \neg p(z \le 0) \\ \neg q_2 & \vee & q_1 & & \\ \neg q_2 & \vee & p(z \le 0) & & \\ q_1 & \vee & p(z \le 0) & & \end{array}\right\} S_1 + z = S_2$$

$$\neg q_2 \quad \} \; S_2 = 1$$

BC Native Encoder (work in progress)

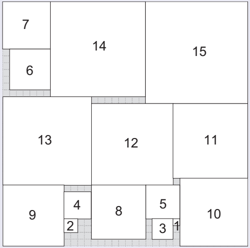
- We have tested Boolean Cardinality Encoder (BC Native Encoder), which natively encodes Boolean cardinality constraints by using `addAtMost` or `addAtLeast` methods of Sat4j
- Preliminary Results (CPU time in seconds)

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

Example: Square Packing

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

Example of $SP(15, 36)$



- **Optimum soluiton 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 $(1 \leq i < j \leq n)$

$$(x_i + i \leq x_j) \vee (x_j + j \leq x_i) \vee (y_i + i \leq y_j) \vee (y_j + j \leq y_i)$$

Decremental Seach

Scarab Program for $SP(n, s)$

```
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

```
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

Bisection Search

```
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
  }
}
```