What is SAT?

ph: SAT based CP System

SAT (Boolean satisfiability testing) Problems

SAT (Boolean satisfiability testing) Problems

SAT Solvers

There are 2^n combinations for assignments.

We cannot solve any SAT instances even for small n (e.g. $n = 100$)?

SAT Solvers

There are 2^n combinations for assignments.

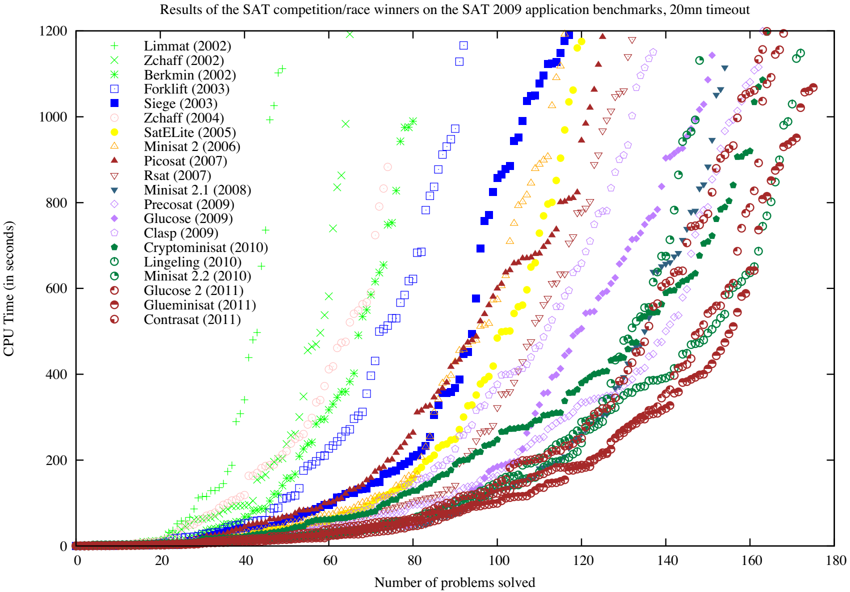
We cannot solve any SAT instances even for small n (e.g. $n = 100$)?

SAT Solvers

There are 2^n combinations for assignments.

We cannot solve any SAT instances even for small n (e.g. $n = 100$)?

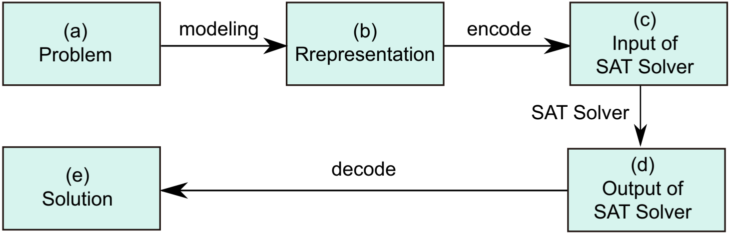
Progress of SAT Solvers (shown by [Simon 2011])



Cactus Plot shown by [Simon 2011]

Problem Solving using SAT Solvers

Thanks to the remarkable progress of SAT solvers, **SAT-based Problem Solving** have been actively studied.



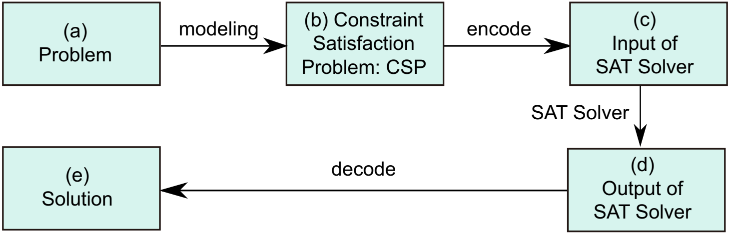
SAT-based Systems are implementations of SAT-based problem solving.

Many research topics in this field. Among them, the **importance of modeling and encoding** are re-recognized.

Good modeling/encodings are developed considering the size of solver input and propagations in SAT solvers (and many many trial/errors are necessary!).

Problem Solving using SAT Solvers

Thanks to the remarkable progress of SAT solvers, **SAT-based Problem Solving** have been actively studied.



- **SAT-based Systems** are implementations of SAT-based problem solving.
- Many research topics in this field. Among them, the **importance of modeling and encoding** are re-recognized.
- Good modeling/encodings are developed considering the size of solver input and propagations in SAT solvers (and many many trial/errors are necessary!).

SAT encodings

There have been several methods proposed to encode CSP into SAT.

Direct encoding is the most widely used one [de Kleer, 1989].

Other encodings:

Multivalued encoding [Selman et al., 1992]
Support encoding [Kasif, 1990]
Log encoding [Iwama and Miyazaki, 1994]
Log-support encoding [Gavanelli, 2007]

Applications of SAT Technology

Planning (SATPLAN, Blackbox) [Kautz and Selman, 1992]

Job-shop Scheduling [Crawford and Baker, 1994]

Bounded Model Checking [Biere et al., 1999]

Term Rewriting (AProVE) [Giesl et al. 2004]

Constraint Satisfaction Problem[Tamura et al., 2006]

- **Sugar**, SAT-based CSP Solvr, which is the Winner of 2008 and 2009 CSP Solver Competitions in GLOBAL categories.
  It adopts Order Encoding.

Others

- Test Case Generation,
  Systems Biology,
  Timetabling,
  Packing,
  Puzzle, and more!

Other News around SAT

A **SAT solver Sat4j** implemented on Java has been integrated into **Eclipse** for managing plugins dependencies in their update manager.

Donald E. Knuth gave an invited talk about SAT at the International Conference on Theory and Applications of Satisfiability Testing 2012.

  SAT will be appeared in Volume 4b of **The Art Of Computer Programming**.

Contents of Talk

What is SAT?

Sugar: SAT-based CP System

Motivation

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.

Scarab

Scarab

Scarab

Scarab

Scarab

Scarab

Scarab

Example of Scarab Program: GCP.scala

Example of Scarab Program: GCP.scala

Imports

Instance Structure

Defining CSP

Solving CSP

We can do more in GCP?

We can do more in GCP?

We can do more in GCP?

What is SAT?

ph: SAT-based CP System

Pandiagonal Latin Square: $PLS(n)$

Pandiagonal Latin Square: $PLS(n)$

alldiff Model

alldiff Model

alldiff Model

alldiff Model

alldiff Model

alldiff Model

alldiff Model

alldiff Model

alldiff Model

alldiff Model

Boolean Cardinality Model

Boolean Cardinality Model

Boolean Cardinality Model

Boolean Cardinality Model

Boolean Cardinality Model

Boolean Cardinality Model

Boolean Cardinality Model

Boolean Cardinality Model

Boolean Cardinality Model

Boolean Cardinality Model

Experiments

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. Just using SAT solvers is not enough!

Scarab helps users to focus on them ;)

Experiments on Hamiltonian Cycle Problem

We evaluate the effectiveness of (1) CEGAR-HCP, (2) Native BC, (3) Implementation on Tightly Integrated System.
We also have (4) a comparison with other specialized methods.

Experiments on Hamiltonian Cycle Problem

We evaluate the effectiveness of (1) CEGAR-HCP, (2) Native BC,
(3) Implementation on Tightly Integrated System.
We also have (4) a comparison with other specialized methods.

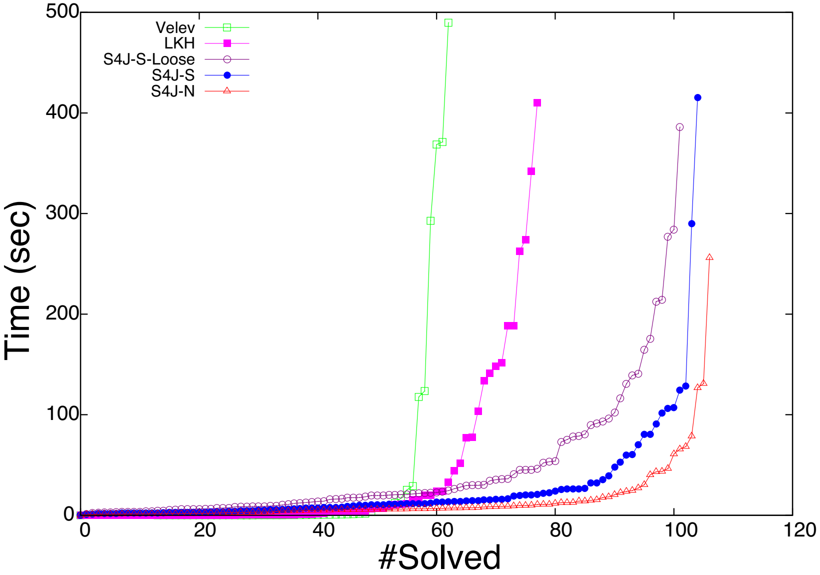
Experiments on Hamiltonian Cycle Problem

We evaluate the effectiveness of (1) CEGAR-HCP, (2) Native BC,
(3) Implementation on Tightly Integrated System.
We also have (4) a comparison with other specialized methods.

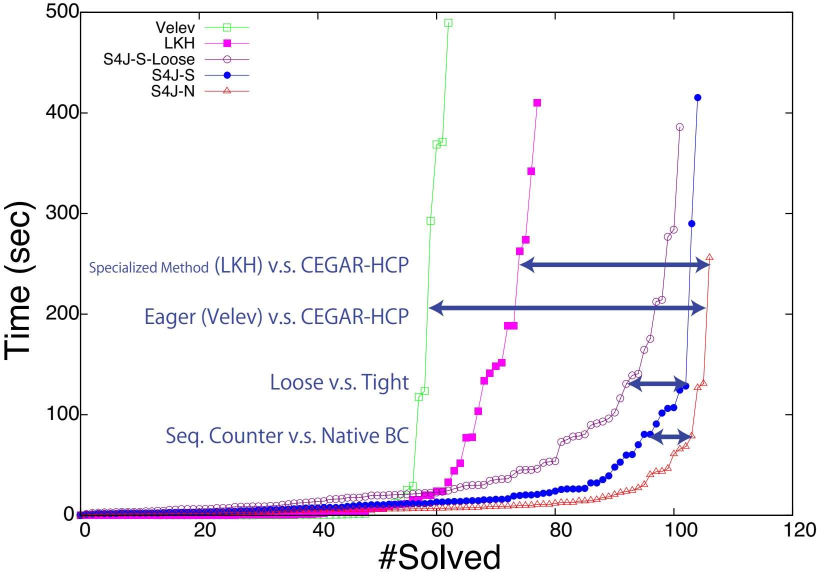
Experiments on Hamiltonian Cycle Problem

We evaluate the effectiveness of (1) CEGAR-HCP, (2) Native BC, (3) Implementation on Tightly Integrated System.
We also have (4) a comparison with other specialized methods.

Cactus Plot (#Solved–CPU Time)



Cactus Plot (#Solved–CPU Time)



Features of Scarab

Efficiency

Scarab is efficient in the sense that it uses an optimized version of the order encoding for encoding CSP into SAT.

Portability

The combination of Scarab and Sat4j enables the development of portable applications on JVM (Java Virtual Machine).

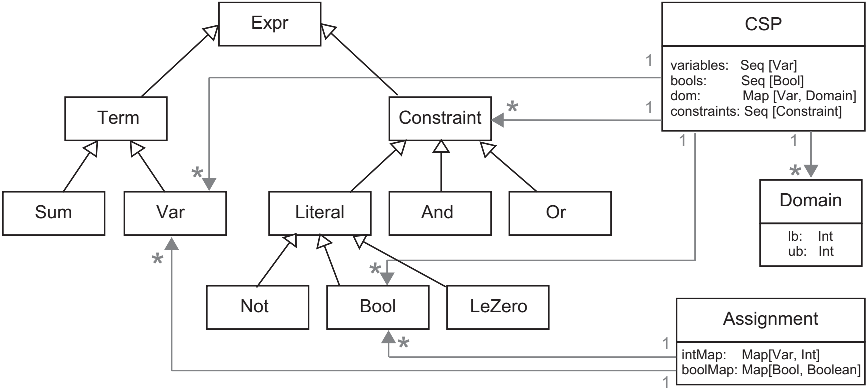
Customizability

Scarab is 800 lines long without comments.
Core of order encoding module is only 25 lines long.
It allows programmers to freely customize Scarab itself.

Availability of Advanced SAT Techniques

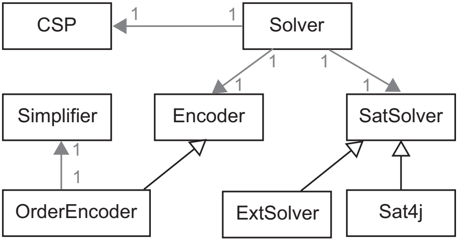
Thanks to the tight integration to Sat4j, it is available to use several SAT techniques, *e.g.*, incremental SAT solving and native handling constraints.

Class Diagrams



Class Diagrams for CSPs

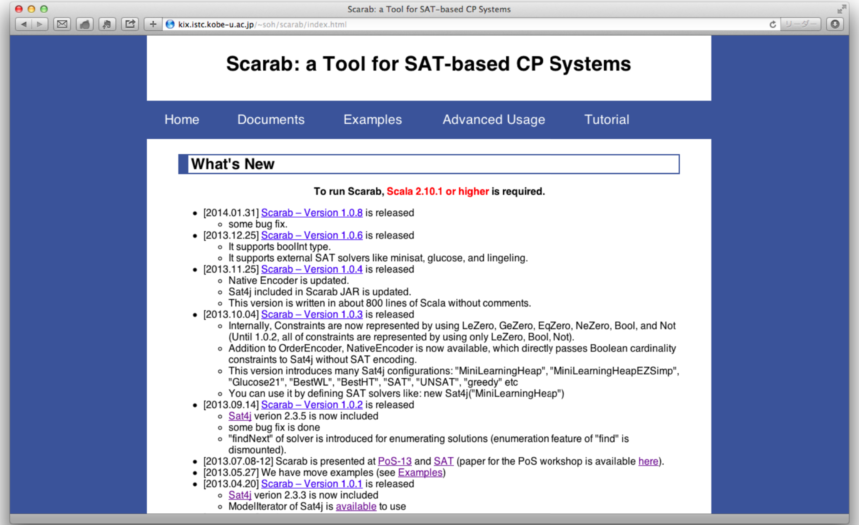
Class Diagrams



Class Diagrams for Solvers

Web Page for Scarab

http://kix.istc.kobe-u.ac.jp/~soh/scarab/



Scarab: a Tool for SAT-based CP Systems

kix.istc.kobe-u.ac.jp/~soh/scarab/index.html

Scarab: a Tool for SAT-based CP Systems

Home Documents Examples Advanced Usage Tutorial

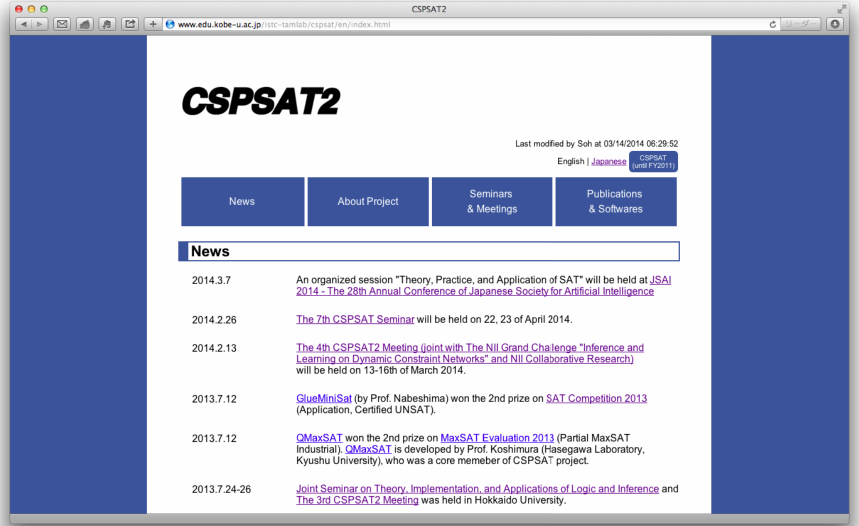
What's New

To run Scarab, Scala 2.10.1 or higher is required.

- [2014.01.31] Scarab – Version 1.0.8 is released
  - some bug fix.
- [2013.12.25] Scarab – Version 1.0.6 is released
  - It supports boolInt type.
  - It supports external SAT solvers like minisat, glucose, and lingeling.
- [2013.11.25] Scarab – Version 1.0.4 is released
  - Native Encoder is updated.
  - Sat4j included in Scarab JAR is updated.
  - This version is written in about 800 lines of Scala without comments.
- [2013.10.04] Scarab – Version 1.0.3 is released
  - Internally, Constraints are now represented by using LeZero, GeZero, EqZero, NeZero, Bool, and Not (Until 1.0.2, all of constraints are represented by using only LeZero, Bool, Not).
  - Addition to OrderEncoder, NativeEncoder is now available, which directly passes Boolean cardinality constraints to Sat4j without SAT encoding.
  - This version introduces many Sat4j configurations: "MiniLearningHeap", "MiniLearningHeapEZSimp", "Glucose21", "BestWL", "BestHT", "SAT", "UNSAT", "greedy" etc
  - You can use it by defining SAT solvers like: new Sat4j("MiniLearningHeap")
- [2013.09.14] Scarab – Version 1.0.2 is released
  - Sat4j verion 2.3.5 is now included
  - some bug fix is done
  - "findNext" of solver is introduced for enumerating solutions (enumeration feature of "find" is dismounted).
- [2013.07.08-12] Scarab is presented at PoS-13 and SAT (paper for the PoS workshop is available here).
- [2013.05.27] We have move examples (see Examples)
- [2013.04.20] Scarab – Version 1.0.1 is released
  - Sat4j verion 2.3.3 is now included
  - ModelIterator of Sat4j is available to use

Web Page for CSPSAT2

http://www.edu.kobe-u.ac.jp/istc-tamlab/cspsat/en/



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
  - Sat4j has various functions of finding MUS, optimization, solution enumeration, handling natively cardinality and pseudo-Boolean constraints.
- URL of Scarab http://kix.istc.kobe-u.ac.jp/~soh/scarab/

References I

Biere, A., Cimatti, A., Clarke, E. M., and Zhu, Y. (1999).
Symbolic model checking without BDDs.
In *Proceedings of the 5th International Conference on Tools and Algorithms for Construction and Analysis of Systems (TACAS 1999), LNCS 1579*, pages 193–207.

Cook, S. A. (1971).
The complexity of theorem-proving procedures.
In *Proceedings of the 3rd Annual ACM Symposium on Theory of Computing (STOC 1971)*, pages 151–158.

Crawford, J. M. and Baker, A. B. (1994).
Experimental results on the application of satisfiability algorithms to scheduling problems.
In *Proceedings of the 12th National Conference on Artificial Intelligence (AAAI 1994)*, pages 1092–1097.

References II

Davis, M., Logemann, G., and Loveland, D. W. (1962).
A machine program for theorem-proving.
Communications of the ACM, 5(7):394–397.

de Kleer, J. (1989).
A comparison of ATMS and CSP techniques.
In *Proceedings of the 11th International Joint Conference on Artificial Intelligence (IJCAI 1989)*, pages 290–296.

Gavanelli, M. (2007).
The log-support encoding of CSP into SAT.
In *Proceedings of the 13th International Conference on Principles and Practice of Constraint Programming (CP 2007), LNCS 4741*, pages 815–822.

References III

Iwama, K. and Miyazaki, S. (1994).
SAT-variable complexity of hard combinatorial problems.
In *Proceedings of the IFIP 13th World Computer Congress*, pages 253–258.

Kasif, S. (1990).
On the parallel complexity of discrete relaxation in constraint satisfaction networks.
Artificial Intelligence, 45(3):275–286.

Kautz, H. A. and Selman, B. (1992).
Planning as satisfiability.
In *Proceedings of the 10th European Conference on Artificial Intelligence (ECAI 1992)*, pages 359–363.

References IV

Petke, J. and Jeavons, P. (2011).
The order encoding: From tractable csp to tractable sat.
In *Proceedings of the 14th International Conference on Theory and Applications of Satisfiability Testing (SAT 2011), LNCS 6695*, pages 371–372.

Selman, B., Levesque, H. J., and Mitchell, D. G. (1992).
A new method for solving hard satisfiability problems.
In *Proceedings of the 10th National Conference on Artificial Intelligence (AAAI 1992)*, pages 440–446.

Tamura, N., Taga, A., Kitagawa, S., and Banbara, M. (2006).
Compiling finite linear CSP into SAT.
In *Proceedings of the 12th International Conference on Principles and Practice of Constraint Programming (CP 2006), LNCS 4204*, pages 590–603.

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.

Non-overlapping Constraint Model for $SP(n, s)$

Decremental Seach

Bisection Search

Advanced Solving Techniques using Sat4j

Thanks to the **tight integration to Sat4j**, Scarab provides the functions: Incremental solving and CSP solving with assumptions.

We explain it using the following program.

Incremental SAT Solving

CSP Solving under Assumption

Scarab Program for alldiff Model

```
import jp.kobe_u.scarab._ ; import dsl._

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].

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:

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

It represents that mutually different n variables cannot be assigned within the interval of the size $n - 1$.

alldiff (naive)

alldiff (optimized)

Scarab Program for Boolean Cardinality Model

```
import jp.kobe_u.scarab._ ; import dsl._

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**.