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

Takehide Soh, Naoyuki Tamura, and Mutsunori Banbara,

Kobe University

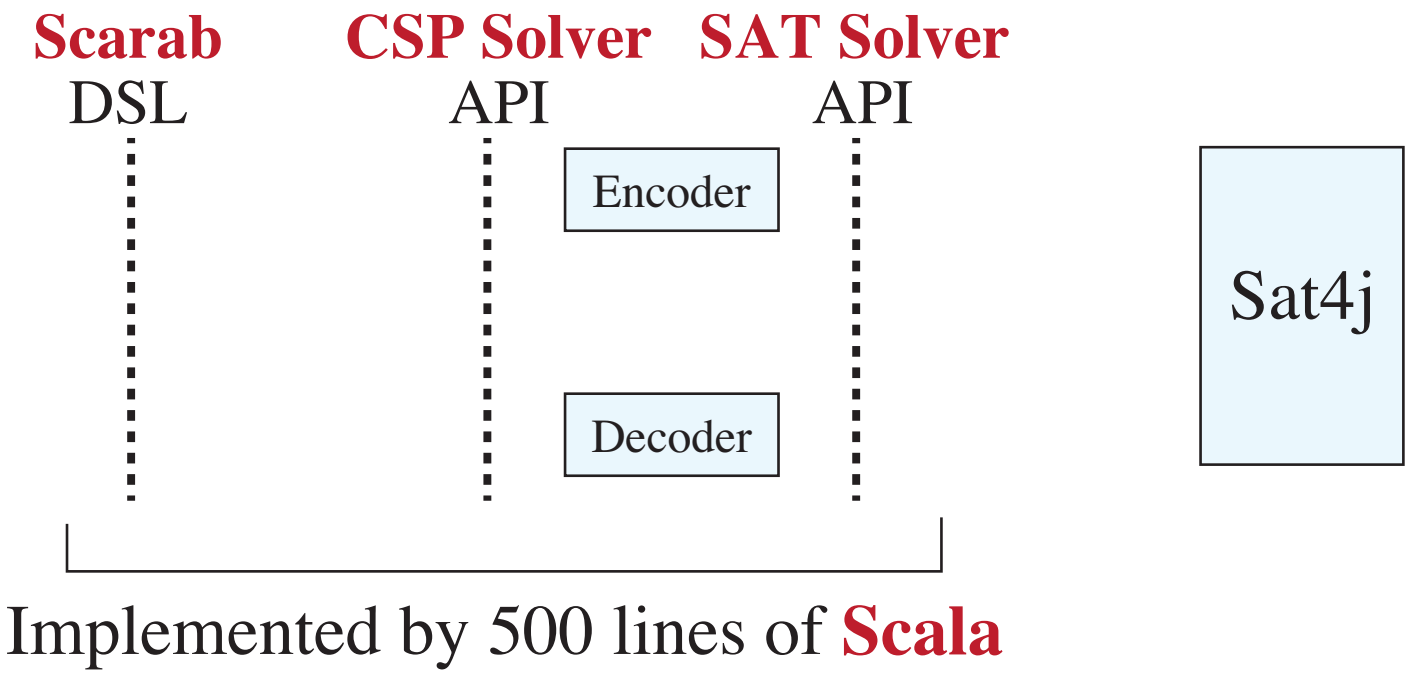
SAT 2013
(July 11th, 2013 at University of Helsinki)

Overview

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

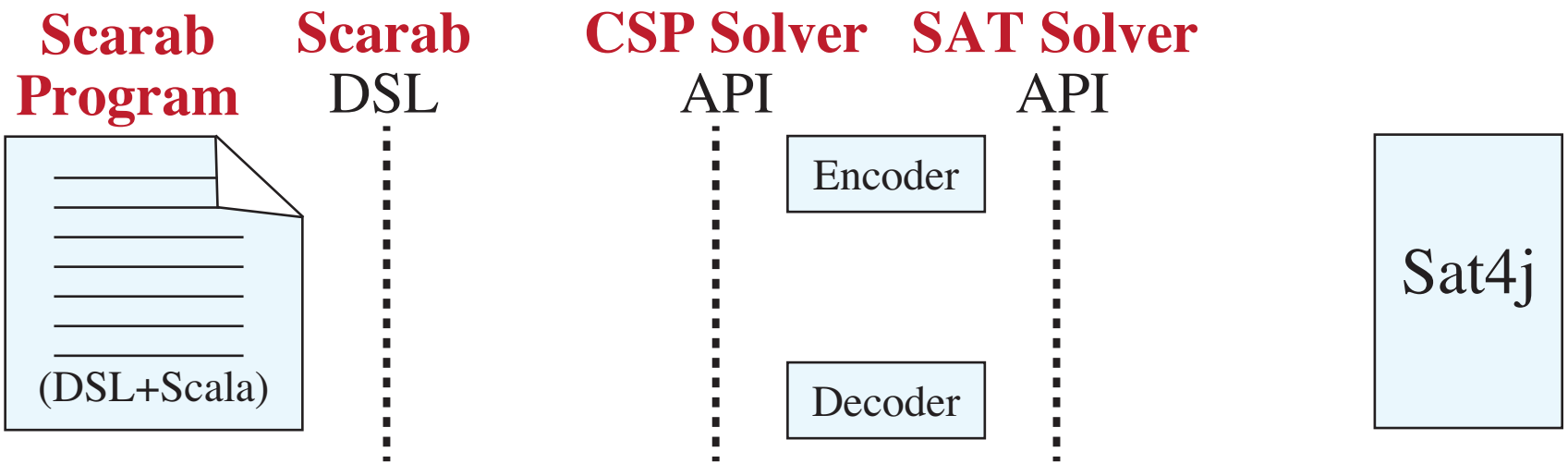
Overview

- **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.



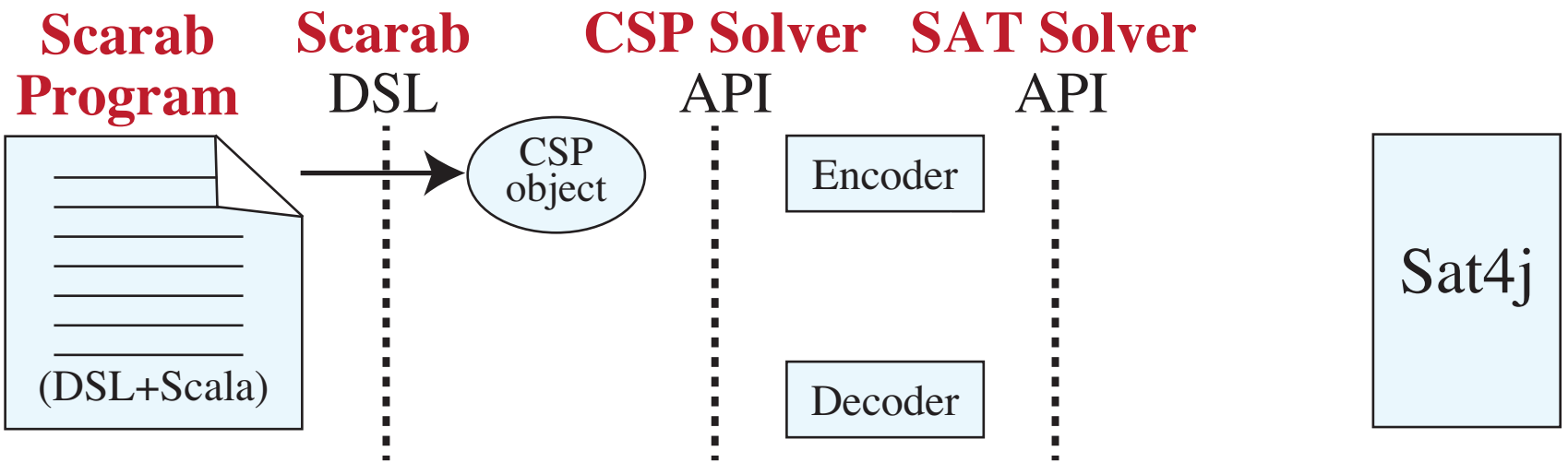
Overview

- **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.



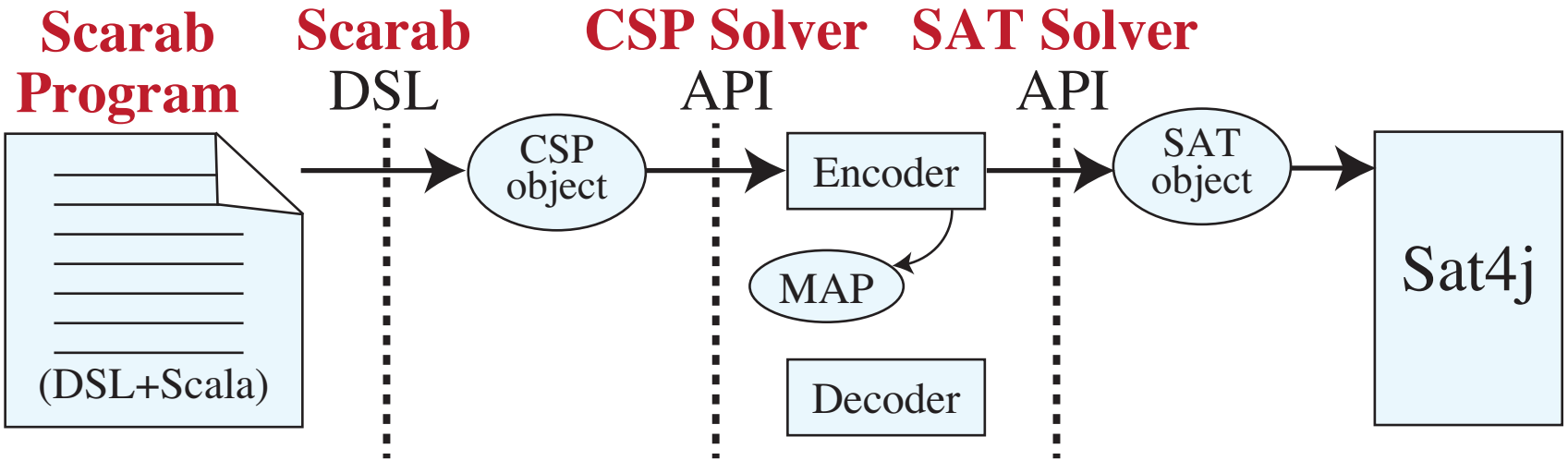
Overview

- **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.



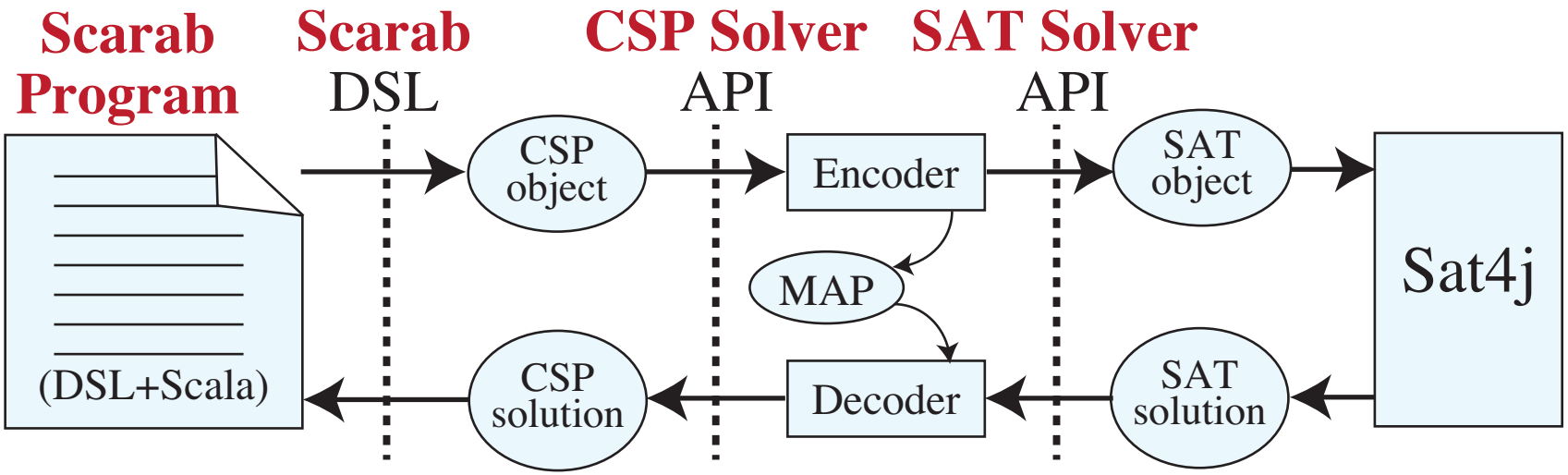
Overview

- **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.



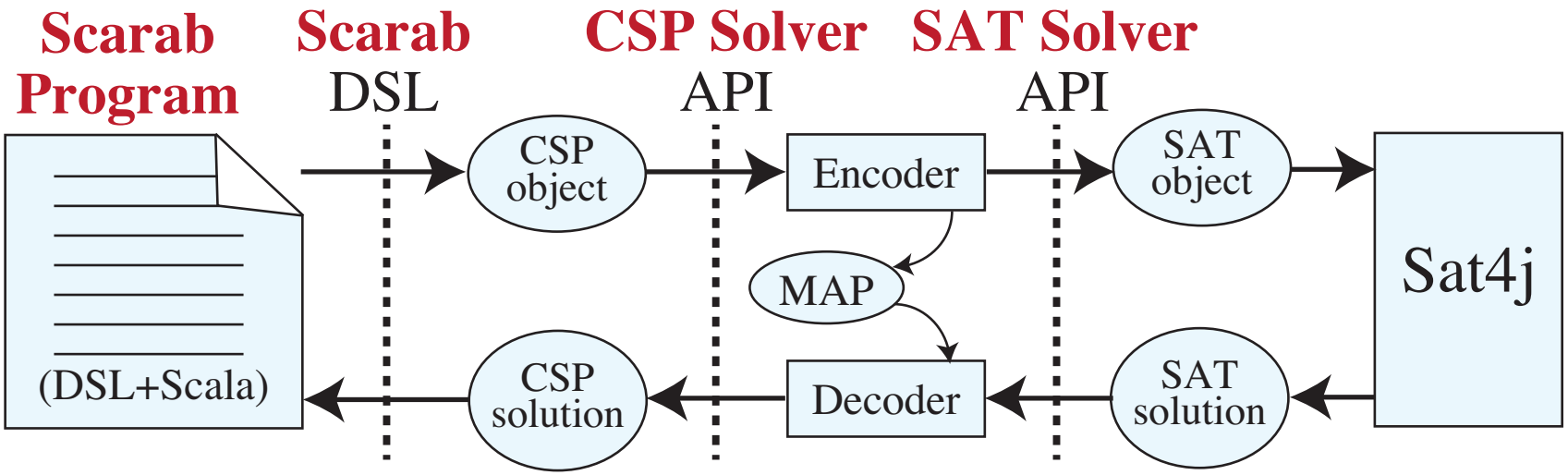
Overview

- **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.



Overview

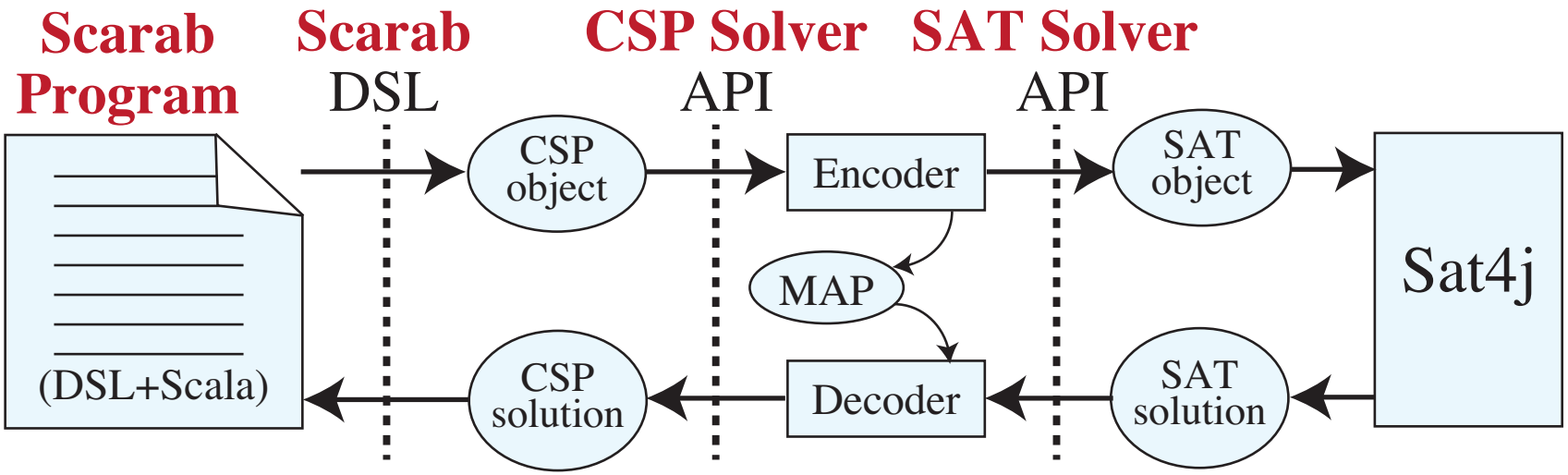
- **Scarab** is a prototyping tool for developing SAT-based Constraint Programming (CP) systems.
- It consists of 1) CP Domain-Specific Language, 2) API of CSP solver, 3) SAT encoding module, and 4) API of SAT solvers.
- It uses **Order Encoding** and **Sat4j** in default.



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

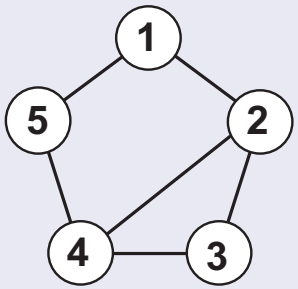
Overview

- **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.

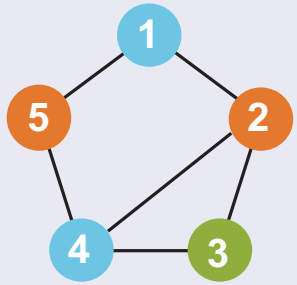


- It is developed to be an expressive, efficient, customizable, and portable workbench.
- The tight integration to Sat4j enables advanced CSP solving such as **incremental solving** and the use of **assumptions**.

Graph Coloring Problem (GCP)

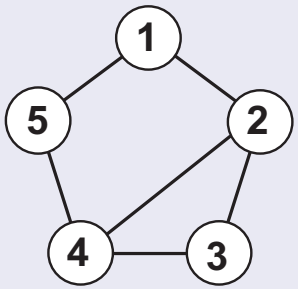


Input

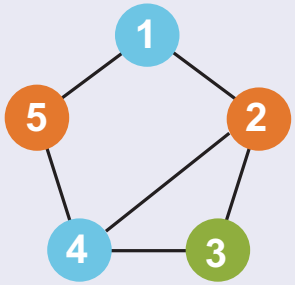


Solution

Graph Coloring Problem (GCP)



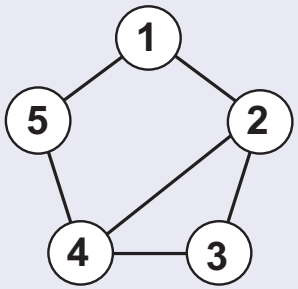
Input



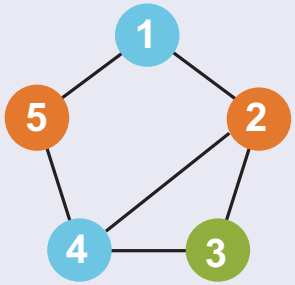
Solution

```
 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)
```

Graph Coloring Problem (GCP)



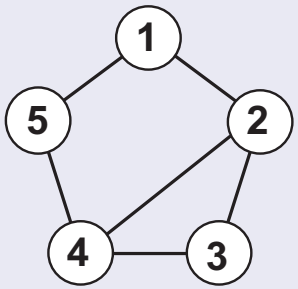
Input



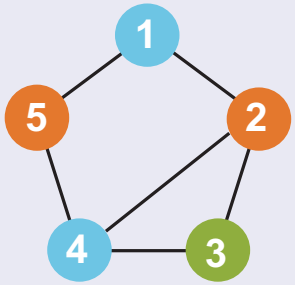
Solution

```
 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)
```

Graph Coloring Problem (GCP)



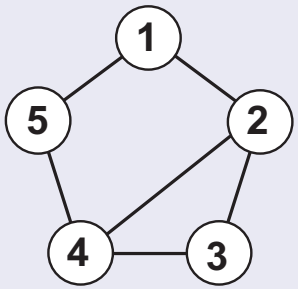
Input



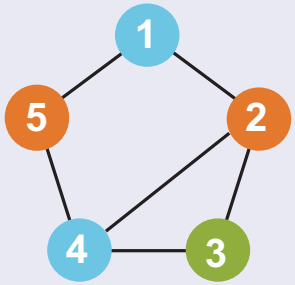
Solution

```
 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)
```

Graph Coloring Problem (GCP)



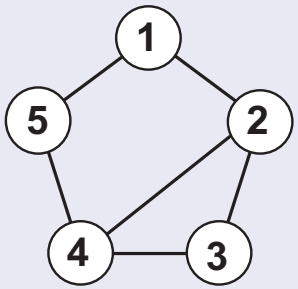
Input



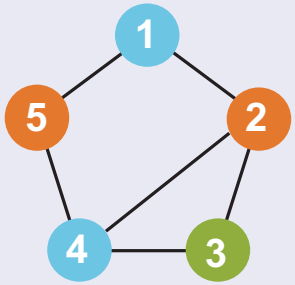
Solution

```
 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)
```

Graph Coloring Problem (GCP)



Input



Solution

```
 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)
```

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)

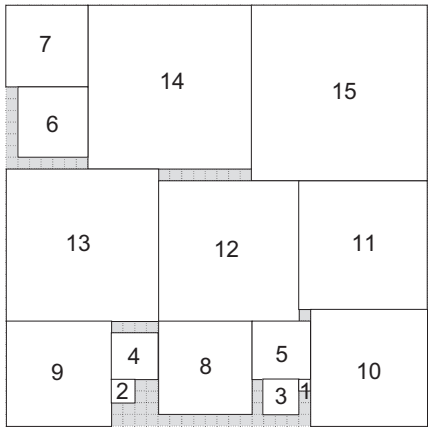
n	SAT/UNSAT	AD1	AD2	BC1	BC2	BC3
7	SAT	0.2	0.2	0.2	0.3	0.3
8	UNSAT	T.O.	0.5	0.3	0.3	0.3
9	UNSAT	T.O.	0.3	0.5	0.3	0.2
10	UNSAT	T.O.	0.4	1.0	0.3	0.3
11	SAT	0.3	0.3	2.3	0.5	0.4
12	UNSAT	T.O.	1.0	5.3	0.8	0.8
13	SAT	T.O.	0.5	T.O.	T.O.	T.O.
14	UNSAT	T.O.	9.7	32.4	8.2	6.8
15	UNSAT	T.O.	388.9	322.7	194.6	155.8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

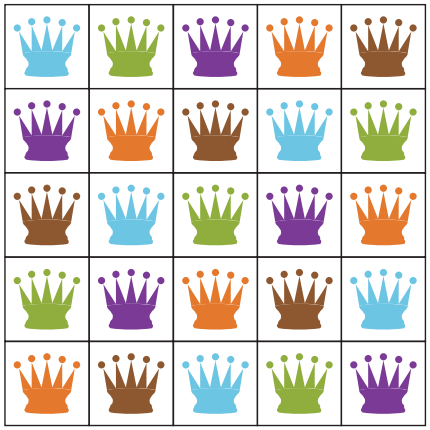
Demonstrations are available for . . .

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

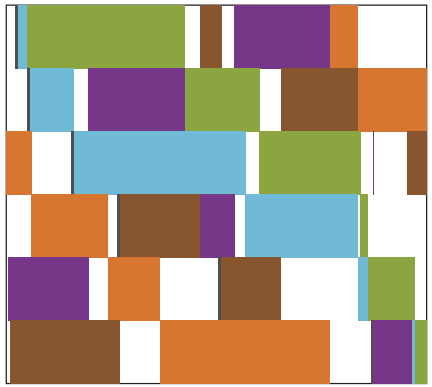
Pandiagonal Latin Square



Square Packing



Colored N Queen



Open-shop Scheduling

Langford Pairs, Alphametics, Magic Square, Optimization, Enumerating Solutions, etc.