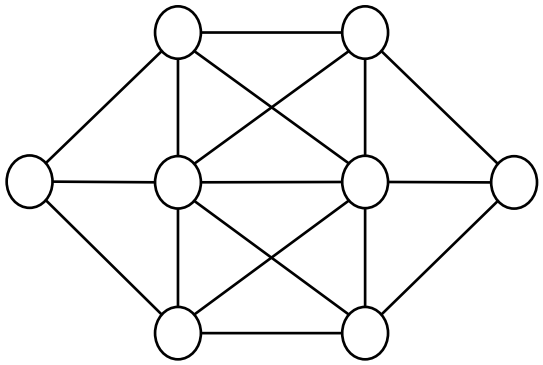
Introduction to Choco

The Constraint Satisfaction Problem

Given a set of *variables* $V = \{X_1, X_2, ..., X_n\}$, a set of *domains* $D = \{D_1, D_2, ..., D_n\}$ of allowable values for each variable, and a set of *constraints* $C = \{C_1, C_2, ..., C_m\}$ restricting the values that groups of variables can take simultaneously,

find an assignment for each variable X_i of a value v_i from its domain D_i, (i.e. an n-tuple $(v_1, v_2, ..., v_n)$, where $v_i \in D_i$), so that all constraints are satisfied.



Using only the numbers 1 to 8, put a different number in each circle so that adjacent circles do not have consecutive numbers

REMINDER

Constraint programming

Modelling:	what are the variables, values and constraints?
Searching:	guessing values, backtracking on failure
Heuristics:	which variable or value to try next
Inference:	ruling out options by reasoning
Symmetry:	spotting repeat patterns in the problem space
Complexity:	understanding the inherent difficulty
Optimisation:	are some solutions better than others?
Applications:	what real problems can we solve?
Programming:	how to implement all of this in programming languages, so that it is correct and allows us to find solutions efficiently

choco-solver.org | A Free

choco-solver.org

choco-solver.org

A Free and Open-Source Java Library for Constraint Programming

Home | Download | Support | Related projects | Log in

What is Choco ?

Choco is a *Free and Open-Source Software*[1] dedicated to **Constraint Programming**[2]. It is a Java library written under BSD license. It aims at describing hard combinatorial problems in the form of Constraint Satisfaction Problems and solving them with Constraint Programming techniques. The user models its problem in a declarative way by stating the set of constraints that need to be satisfied in every solution. Then, Choco solves the problem by alternating constraint filtering algorithms with a search mechanism. Choco is used for:

- teaching : easy to use
- research : easy to extend
- real-life applications : easy to integrate

Choco is among the fastest CP solvers on the market. In 2013 and 2014, Choco has been awarded many medals at the MiniZinc challenge that is the world-wide competition of constraint-programming solvers.

MiniZinc Challenge 2014 

MiniZinc Challenge 2013 

In addition to these performance results, Choco benefits from academic contributors, who provide long term improvements, and the consulting company COSLING, which provides services ranging from training, support to the development of interactive decision support web-services.

[1]: Choco is hosted on GitHub and distributed under BSD license (Copyright(c) 1999-2015, Ecole des Mines de Nantes).
[2]: Constraint programming is a technology at the crossroad between Artificial Intelligence and Operational Research, enabling to solve a wide range of complex problems arising in planning, scheduling, logistics, financial analysis, bioinformatics, etc. (read more on wikipedia).

A very simple problem

Three people have been selected for a job interview: Alice, Bob and Carol. Each interview lasts for 1 hour. Alice is only available at 2pm or 3pm. Bob is only available at 1pm or 2pm. Carol is only available at 1pm or 2pm. Only one person can be interviewed at a time. Find a schedule for the interviews.

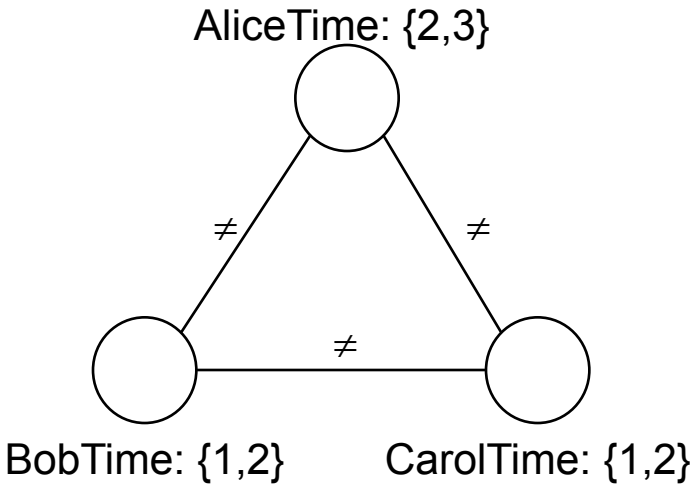
Analysis:	3 decisions to be made – a time for each person -- so three variables?
Variables?	AliceTime, BobTime, CarolTime
Domains?	the hours the people are available
Constraints?	for any pair of variables, they must be different

The very simple CSP

V = { AliceTime,
BobTime,
CarolTime
}

D = { {2,3} , //Alice
{1,2} , //Bob
{1,2} //Carol
}

C = {AliceTime $\neq$ BobTime,
AliceTime $\neq$ CarolTime,
BobTime $\neq$ CarolTime
}



AliceTime $\neq$BobTime
$\subseteq \{2,3\}\times\{1,2\}$

$\{(2,1),(3,1),(3,2)\}$

The Choco process

1. Create a solver object
2. Create the variables and their domains
3. Create the constraints and post to the solver
4. Specify a search strategy (or take the default)
5. Start the solving process
6. Print the results

The Choco program

```
import org.chocosolver.solver.Solver;
import org.chocosolver.solver.constraints.IntConstraintFactory;
import org.chocosolver.solver.trace.Chatterbox;
import org.chocosolver.solver.variables.IntVar;
import org.chocosolver.solver.variables.VariableFactory;

public class SimpleCSP {

    public static void main(String[] args) {
        //create a solver object that will solve the problem for us
        Solver solver = new Solver();

        //create the variables and domains for the problem, and add to the solver
        IntVar AliceTime = VariableFactory.enumerated("Alice time", 2, 3, solver);
        IntVar BobTime = VariableFactory.enumerated("Bob time", 1, 2, solver);
        IntVar CarolTime = VariableFactory.enumerated("Carol time", 1, 2, solver);

        //create the constraints
        solver.post(IntConstraintFactory.arithm(AliceTime, "!=", BobTime));
        solver.post(IntConstraintFactory.arithm(AliceTime, "!=", CarolTime));
        solver.post(IntConstraintFactory.arithm(BobTime, "!=", CarolTime));

         //use a pretty print object to display the results
         Chatterbox.showSolutions(solver); //just show the final result

         //ask the solver to find a solution
         solver.findSolution();

         //print out the search statistics
         Chatterbox.printStatistics(solver);
    }
}
```

```
import org.chocosolver.solver.Solver;
import org.chocosolver.solver.constraints.IntConstraintFactory;
import org.chocosolver.solver.trace.Chatterbox;
import org.chocosolver.solver.variables.IntVar;
import org.chocosolver.solver.variables.VariableFactory;

public class SimpleCSP {

    public static void main(String[] args) {
        //create a solver object that will solve the problem for us
        Solver solver = new Solver();

        //create the variables and domains for the problem, and add to the solver
        IntVar AliceTime = VariableFactory.enumerated("Alice time", 2, 3, solver);
        IntVar BobTime = VariableFactory.enumerated("Bob time", 1, 2, solver);
        IntVar CarolTime = VariableFactory.enumerated("Carol time", 1, 2, solver);

        //create the constraints
        solver.post(IntConstraintFactory.arithm(AliceTime, "!=", BobTime));
        solver.post(IntConstraintFactory.arithm(AliceTime, "!=", CarolTime));
        solver.post(IntConstraintFactory.arithm(BobTime, "!=", CarolTime));

         //use a pretty print object to display the results
         Chatterbox.showSolutions(solver); //just show the final result

         //ask the solver to find a solution
         solver.findSolution();

         //print out the search statistics
         Chatterbox.printStatistics(solver);
    }
}
```

packages required for this program

```
import org.chocosolver.solver.Solver;
import org.chocosolver.solver.constraints.IntConstraintFactory;
import org.chocosolver.solver.trace.Chatterbox;
import org.chocosolver.solver.variables.IntVar;
import org.chocosolver.solver.variables.VariableFactory;

public class SimpleCSP {

    public static void main(String[] args) {
        //create a solver object that will solve the problem for us
        Solver solver = new Solver();

        //create the variables and domains for the problem, and add to the solver
        IntVar AliceTime = VariableFactory.enumerated("Alice time", 2, 3, solver);
        IntVar BobTime = VariableFactory.enumerated("Bob time", 1, 2, solver);
        IntVar CarolTime = VariableFactory.enumerated("Carol time", 1, 2, solver);

        //create the constraints
        solver.post(IntConstraintFactory.arithm(AliceTime, "!=", BobTime));
        solver.post(IntConstraintFactory.arithm(AliceTime, "!=", CarolTime));
        solver.post(IntConstraintFactory.arithm(BobTime, "!=", CarolTime));

         //use a pretty print object to display the results
         Chatterbox.showSolutions(solver); //just show the final result

         //ask the solver to find a solution
         solver.findSolution();

         //print out the search statistics
         Chatterbox.printStatistics(solver);
    }
}
```

create the solver

```
import org.chocosolver.solver.Solver;
import org.chocosolver.solver.constraints.IntConstraintFactory;
import org.chocosolver.solver.trace.Chatterbox;
import org.chocosolver.solver.variables.IntVar;
import org.chocosolver.solver.variables.VariableFactory;

public class SimpleCSP {

    public static void main(String[] args) {
        //create a solver object that will solve the problem for us
        Solver solver = new Solver();

        //create the variables and domains for the problem, and add to the solver
        IntVar AliceTime = VariableFactory.enumerated("Alice time", 2, 3, solver);
        IntVar BobTime = VariableFactory.enumerated("Bob time", 1, 2, solver);
        IntVar CarolTime = VariableFactory.enumerated("Carol time", 1, 2, solver);

        //create the constraints
        solver.post(IntConstraintFactory.arithm(AliceTime, "!=", BobTime));
        solver.post(IntConstraintFactory.arithm(AliceTime, "!=", CarolTime));
        solver.post(IntConstraintFactory.arithm(BobTime, "!=", CarolTime));

         //use a pretty print object to display the results
         Chatterbox.showSolutions(solver); //just show the final result

         //ask the solver to find a solution
         solver.findSolution();

         //print out the search statistics
         Chatterbox.printStatistics(solver);
    }
}
```

VariableFactory creates the variables for us.
They have integer values, so we use "IntVar".
Add them to the solver.

```
import org.chocosolver.solver.Solver;
import org.chocosolver.solver.constraints.IntConst
import org.chocosolver.solver.trace.Chatterbox;
import org.chocosolver.solver.variables.IntVar;
import org.chocosolver.solver.variables.VariableFa

public class SimpleCSP {

    public static void main(String[] args) {
        //create a solver object that will solve the problem for us
        Solver solver = new Solver();

        //create the variables and domains for the problem, and add to the solver
        IntVar AliceTime = VariableFactory.enumerated("Alice time", 2, 3, solver);
        IntVar BobTime = VariableFactory.enumerated("Bob time", 1, 2, solver);
        IntVar CarolTime = VariableFactory.enumerated("Carol time", 1, 2, solver);

        //create the constraints
        solver.post(IntConstraintFactory.arithm(AliceTime, "!=", BobTime));
        solver.post(IntConstraintFactory.arithm(AliceTime, "!=", CarolTime));
        solver.post(IntConstraintFactory.arithm(BobTime, "!=", CarolTime));

         //use a pretty print object to display the results
         Chatterbox.showSolutions(solver); //just show the final result

         //ask the solver to find a solution
         solver.findSolution();

         //print out the search statistics
         Chatterbox.printStatistics(solver);
    }
}
```

Place all numbers between a lower and upper bound into the domain

```
import org.chocosolver.solver.Solver;
import org.chocosolver.solver.constraints.IntConst
import org.chocosolver.solver.trace.Chatterbox;
import org.chocosolver.solver.variables.IntVar;
import org.chocosolver.solver.variables.VariableFa

public class SimpleCSP {

    public static void main(String[] args) {
        //create a solver object that will solve the problem for us
        Solver solver = new Solver();

        //create the variables and domains for the problem, and add to the solver
        IntVar AliceTime = VariableFactory.enumerated("Alice time", 2, 3, solver);
        IntVar BobTime = VariableFactory.enumerated("Bob time", 1, 2, solver);
        IntVar CarolTime = VariableFactory.enumerated("Carol time", 1, 2, solver);

        //create the constraints
        solver.post(IntConstraintFactory.arithm(AliceTime, "!=", BobTime));
        solver.post(IntConstraintFactory.arithm(AliceTime, "!=", CarolTime));
        solver.post(IntConstraintFactory.arithm(BobTime, "!=", CarolTime));

         //use a pretty print object to display the results
         Chatterbox.showSolutions(solver); //just show the final result

         //ask the solver to find a solution
         solver.findSolution();

         //print out the search statistics
         Chatterbox.printStatistics(solver);
    }
}
```

IntConstraintFactory creates the constraints, and we post them to the solver.

```
import org.chocosolver.solver.Solver;
import org.chocosolver.solver.constraints.IntConstraintFactory;
import org.chocosolver.solver.trace.Chatterbox;
import org.chocosolver.solver.variables.IntVar;
import org.chocosolver.solver.variables.VariableFa

public class SimpleCSP {

    public static void main(String[] args) {
        //create a solver object that will solve the problem for us
        Solver solver = new Solver();

        //create the variables and domains for the problem, and add to the solver
        IntVar AliceTime = VariableFactory.enumerated("Alice time", 2, 3, solver);
        IntVar BobTime = VariableFactory.enumerated("Bob time", 1, 2, solver);
        IntVar CarolTime = VariableFactory.enumerated("Carol time", 1, 2, solver);

        //create the constraints
        solver.post(IntConstraintFactory.arithm(AliceTime, "!=", BobTime));
        solver.post(IntConstraintFactory.arithm(AliceTime, "!=", CarolTime));
        solver.post(IntConstraintFactory.arithm(BobTime, "!=", CarolTime));

         //use a pretty print object to display the results
         Chatterbox.showSolutions(solver); //just show the final result

         //ask the solver to find a solution
         solver.findSolution();

         //print out the search statistics
         Chatterbox.printStatistics(solver);
    }
}
```

The constraint is of type *arithmetic* (because we are dealing with numbers)

```
import org.chocosolver.solver.Solver;
import org.chocosolver.solver.constraints.IntConst
import org.chocosolver.solver.trace.Chatterbox;
import org.chocosolver.solver.variables.IntVar;
import org.chocosolver.solver.variables.VariableFa

public class SimpleCSP {

    public static void main(String[] args) {
        //create a solver object that will solve the problem for us
        Solver solver = new Solver();

        //create the variables and domains for the problem, and add to the solver
        IntVar AliceTime = VariableFactory.enumerated("Alice time", 2, 3, solver);
        IntVar BobTime = VariableFactory.enumerated("Bob time", 1, 2, solver);
        IntVar CarolTime = VariableFactory.enumerated("Carol time", 1, 2, solver);

        //create the constraints
        solver.post(IntConstraintFactory.arithm(AliceTime, "!=", BobTime));
        solver.post(IntConstraintFactory.arithm(AliceTime, "!=", CarolTime));
        solver.post(IntConstraintFactory.arithm(BobTime, "!=", CarolTime));

         //use a pretty print object to display the results
         Chatterbox.showSolutions(solver); //just show the final result

         //ask the solver to find a solution
         solver.findSolution();

         //print out the search statistics
         Chatterbox.printStatistics(solver);
    }
}
```

We will use the default search method, so don't need to do anything for that.

```
import org.chocosolver.solver.Solver;
import org.chocosolver.solver.constraints.IntConst
import org.chocosolver.solver.trace.Chatterbox;
import org.chocosolver.solver.variables.IntVar;
import org.chocosolver.solver.variables.VariableFactory;

public class SimpleCSP {

    public static void main(String[] args) {
        //create a solver object that will solve the problem for us
        Solver solver = new Solver();

        //create the variables and domains for the problem, and add to the solver
        IntVar AliceTime = VariableFactory.enumerated("Alice time", 2, 3, solver);
        IntVar BobTime = VariableFactory.enumerated("Bob time", 1, 2, solver);
        IntVar CarolTime = VariableFactory.enumerated("Carol time", 1, 2, solver);

        //create the constraints
        solver.post(IntConstraintFactory.arithm(AliceTime, "!=", BobTime));
        solver.post(IntConstraintFactory.arithm(AliceTime, "!=", CarolTime));
        solver.post(IntConstraintFactory.arithm(BobTime, "!=", CarolTime));

         //use a pretty print object to display the results
         Chatterbox.showSolutions(solver); //just show the final result

         //ask the solver to find a solution
         solver.findSolution();

         //print out the search statistics
         Chatterbox.printStatistics(solver);
    }
}
```

Start preparing the output – we want the final result

```
import org.chocosolver.solver.Solver;
import org.chocosolver.solver.constraints.IntConst
import org.chocosolver.solver.trace.Chatterbox;
import org.chocosolver.solver.variables.IntVar;
import org.chocosolver.solver.variables.VariableFactory;

public class SimpleCSP {

    public static void main(String[] args) {
        //create a solver object that will solve the problem for us
        Solver solver = new Solver();

        //create the variables and domains for the problem, and add to the solver
        IntVar AliceTime = VariableFactory.enumerated("Alice time", 2, 3, solver);
        IntVar BobTime = VariableFactory.enumerated("Bob time", 1, 2, solver);
        IntVar CarolTime = VariableFactory.enumerated("Carol time", 1, 2, solver);

        //create the constraints
        solver.post(IntConstraintFactory.arithm(AliceTime, "!=", BobTime));
        solver.post(IntConstraintFactory.arithm(AliceTime, "!=", CarolTime));
        solver.post(IntConstraintFactory.arithm(BobTime, "!=", CarolTime));

         //use a pretty print object to display the results
         Chatterbox.showSolutions(solver); //just show the final result

         //ask the solver to find a solution
         solver.findSolution();

         //print out the search statistics
         Chatterbox.printStatistics(solver);
    }
}
```

Go find the solution

```
import org.chocosolver.solver.Solver;
import org.chocosolver.solver.constraints.IntConst
import org.chocosolver.solver.trace.Chatterbox;
import org.chocosolver.solver.variables.IntVar;
import org.chocosolver.solver.variables.VariableFactory;

public class SimpleCSP {

    public static void main(String[] args) {
        //create a solver object that will solve the problem for us
        Solver solver = new Solver();

        //create the variables and domains for the problem, and add to the solver
        IntVar AliceTime = VariableFactory.enumerated("Alice time", 2, 3, solver);
        IntVar BobTime = VariableFactory.enumerated("Bob time", 1, 2, solver);
        IntVar CarolTime = VariableFactory.enumerated("Carol time", 1, 2, solver);

        //create the constraints
        solver.post(IntConstraintFactory.arithm(AliceTime, "!=", BobTime));
        solver.post(IntConstraintFactory.arithm(AliceTime, "!=", CarolTime));
        solver.post(IntConstraintFactory.arithm(BobTime, "!=", CarolTime));

         //use a pretty print object to display the results
         Chatterbox.showSolutions(solver); //just show the final result

         //ask the solver to find a solution
         solver.findSolution();

         //print out the search statistics
         Chatterbox.printStatistics(solver);
    }
}
```

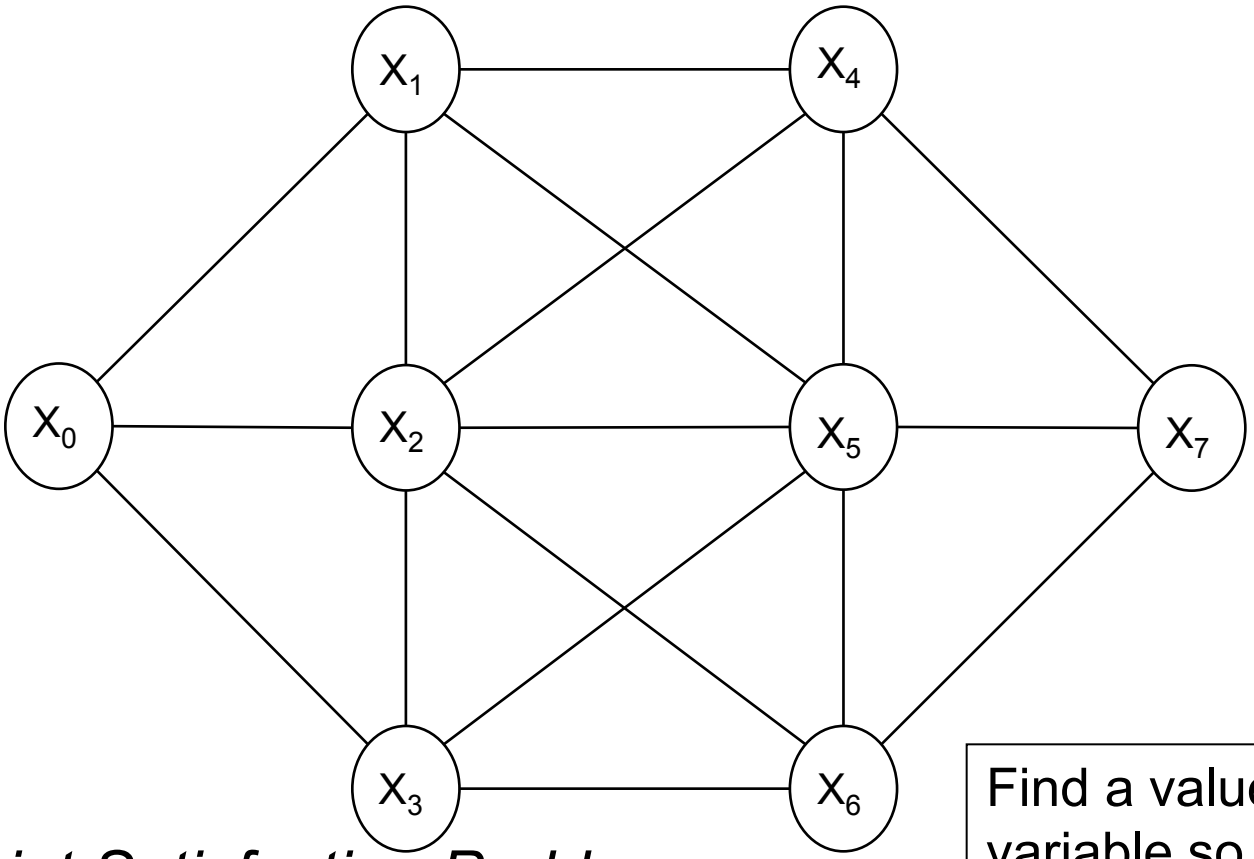
Tell us how much work it required

```
- Solution #1 found. Solver[Solver-0], 1 Solutions, Resolution time 0.022s, 3 Nodes
(139.4 n/s), 1 Backtracks, 1 Fails, 0 Restarts
        Alice time = 3 Bob time = 2 Carol time = 1 .
** Choco 3.3.3 (2015-12) : Constraint Programming Solver, Copyleft (c) 2010-2015
- Solver[Solver-0] features:
        Variables : 3
        Constraints : 3
        Default search strategy : yes
        Completed search strategy : no
- Complete search - 1 solution found.
        Solver[Solver-0]
        Solutions: 1
        Building time : 0.165s
        Resolution time : 0.100s
        Nodes: 3 (29.9 n/s)
        Backtracks: 1
        Fails: 1
        Restarts: 0
        Variables: 3
        Constraints: 3
```

```
- Solution #1 found. Solver[Solver-0], 1 Solutions, Resolution time 0.022s, 3 Nodes
(139.4 n/s), 1 Backtracks, 1 Fails, 0 Restarts
        Alice time = 3 Bob time = 2 Carol time = 1 .
** Choco 3.3.3 (2015-12) : Constraint Programming Solver, Copyleft (c) 2010-2015
- Solver[Solver-0] features:
        Variables : 3
        Constraints : 3
        Default search strategy : yes
        Completed search strategy : no
- Complete search - 1 solution found.
        Solver[Solver-0]
        Solutions: 1
        Building time : 0.165s
        Resolution time : 0.100s
        Nodes: 3 (29.9 n/s)
        Backtracks: 1
        Fails: 1
        Restarts: 0
        Variables: 3
        Constraints: 3
```

The solution

The search statistics



Find a value for each variable so that all constraints are satisfied simultaneously

A *Constraint Satisfaction Problem*:

Variables: $\{X_0, X_1, X_2, X_3, X_4, X_5, X_6, X_7\}$

Values: $\{1,2,3,4,5,6,7,8\}$

Constraints: $\{\text{alldifferent}(\{X_0, X_1, X_2, X_3, X_4, X_5, X_6, X_7\}), \ldots,$
$|X_0-X_1|>1, |X_0-X_2|>1, \ldots\}$

```
public static void main(String[] args) {

   //create a solver object that will solve the problem for us
    Solver solver = new Solver();

   //create the variables and domains for the problem, and add to the solver
   IntVar[] places = VariableFactory.enumeratedArray("places", 8, 1, 8, solver);

   //now create and post the constraints

   //use the built-in global constraint to say that they are all different
   solver.post(IntConstraintFactory.alldifferent(places));


   //then add the edge constraints
   solver.post(IntConstraintFactory.distance(places[0], places[1], ">", 1));
   solver.post(IntConstraintFactory.distance(places[0], places[2], ">", 1));
   solver.post(IntConstraintFactory.distance(places[0], places[3], ">", 1));
   solver.post(IntConstraintFactory.distance(places[1], places[2], ">", 1));
   solver.post(IntConstraintFactory.distance(places[1], places[4], ">", 1));
   solver.post(IntConstraintFactory.distance(places[1], places[5], ">", 1));
   solver.post(IntConstraintFactory.distance(places[2], places[3], ">", 1));
   solver.post(IntConstraintFactory.distance(places[2], places[4], ">", 1));
   solver.post(IntConstraintFactory.distance(places[2], places[5], ">", 1));
   solver.post(IntConstraintFactory.distance(places[2], places[6], ">", 1));
   solver.post(IntConstraintFactory.distance(places[3], places[5], ">", 1));
   solver.post(IntConstraintFactory.distance(places[3], places[6], ">", 1));
   solver.post(IntConstraintFactory.distance(places[4], places[5], ">", 1));
   solver.post(IntConstraintFactory.distance(places[4], places[7], ">", 1));
   solver.post(IntConstraintFactory.distance(places[5], places[6], ">", 1));
   solver.post(IntConstraintFactory.distance(places[5], places[7], ">", 1));
   solver.post(IntConstraintFactory.distance(places[6], places[7], ">", 1));
```

```
public static void main(String[] args) {

    //create a solver object that will solve the problem for us
     Solver solver = new Solver();

    //create the variables and domains for the problem, and add to the solver
    IntVar[] places = VariableFactory.enumeratedArray("places", 8, 1, 8, solver);

    //now create and post the constraints

    //use the built-in global constrain
    solver.post(IntConstraintFactory.al


    //then add the edge constraints
    solver.post(IntConstraintFactory.distance(places[0], places[1], ">", 1));
    solver.post(IntConstraintFactory.distance(places[0], places[2], ">", 1));
    solver.post(IntConstraintFactory.distance(places[0], places[3], ">", 1));
    solver.post(IntConstraintFactory.distance(places[1], places[2], ">", 1));
    solver.post(IntConstraintFactory.distance(places[1], places[4], ">", 1));
    solver.post(IntConstraintFactory.distance(places[1], places[5], ">", 1));
    solver.post(IntConstraintFactory.distance(places[2], places[3], ">", 1));
    solver.post(IntConstraintFactory.distance(places[2], places[4], ">", 1));
    solver.post(IntConstraintFactory.distance(places[2], places[5], ">", 1));
    solver.post(IntConstraintFactory.distance(places[2], places[6], ">", 1));
    solver.post(IntConstraintFactory.distance(places[3], places[5], ">", 1));
    solver.post(IntConstraintFactory.distance(places[3], places[6], ">", 1));
    solver.post(IntConstraintFactory.distance(places[4], places[5], ">", 1));
    solver.post(IntConstraintFactory.distance(places[4], places[7], ">", 1));
    solver.post(IntConstraintFactory.distance(places[5], places[6], ">", 1));
    solver.post(IntConstraintFactory.distance(places[5], places[7], ">", 1));
    solver.post(IntConstraintFactory.distance(places[6], places[7], ">", 1));
```

create an array of 8 enumerated IntVars with domain {1,2,3,...,8}

```
public static void main(String[] args) {

    //create a solver object that will solv
     Solver solver = new Solver();

    //create the variables and domains for
    IntVar[] places = VariableFactory.enume

    //now create and post the constraints

    //use the built-in global constraint to say that they are all different
    solver.post(IntConstraintFactory.alldifferent(places));


    //then add the edge constraints
    solver.post(IntConstraintFactory.distance(places[0], places[1], ">", 1));
    solver.post(IntConstraintFactory.distance(places[0], places[2], ">", 1));
    solver.post(IntConstraintFactory.distance(places[0], places[3], ">", 1));
    solver.post(IntConstraintFactory.distance(places[1], places[2], ">", 1));
    solver.post(IntConstraintFactory.distance(places[1], places[4], ">", 1));
    solver.post(IntConstraintFactory.distance(places[1], places[5], ">", 1));
    solver.post(IntConstraintFactory.distance(places[2], places[3], ">", 1));
    solver.post(IntConstraintFactory.distance(places[2], places[4], ">", 1));
    solver.post(IntConstraintFactory.distance(places[2], places[5], ">", 1));
    solver.post(IntConstraintFactory.distance(places[2], places[6], ">", 1));
    solver.post(IntConstraintFactory.distance(places[3], places[5], ">", 1));
    solver.post(IntConstraintFactory.distance(places[3], places[6], ">", 1));
    solver.post(IntConstraintFactory.distance(places[4], places[5], ">", 1));
    solver.post(IntConstraintFactory.distance(places[4], places[7], ">", 1));
    solver.post(IntConstraintFactory.distance(places[5], places[6], ">", 1));
    solver.post(IntConstraintFactory.distance(places[5], places[7], ">", 1));
    solver.post(IntConstraintFactory.distance(places[6], places[7], ">", 1));
```

alldifferent(vars) is easier to express than nested loops of $x_i \neq x_j$.
More importantly, the solver can do deeper reasoning.

```
public static void main(String[] args) {

    //create a solver object that will solv
     Solver solver = new Solver();

    //create the variables and domains for
    IntVar[] places = VariableFactory.enume

    //now create and post the constraints

    //use the built-in global constraint to
    solver.post(IntConstraintFactory.alldifferent(places));


    //then add the edge constraints
    solver.post(IntConstraintFactory.distance(places[0], places[1], ">", 1));
    solver.post(IntConstraintFactory.distance(places[0], places[2], ">", 1));
    solver.post(IntConstraintFactory.distance(places[0], places[3], ">", 1));
    solver.post(IntConstraintFactory.distance(places[1], places[2], ">", 1));
    solver.post(IntConstraintFactory.distance(places[1], places[4], ">", 1));
    solver.post(IntConstraintFactory.distance(places[1], places[5], ">", 1));
    solver.post(IntConstraintFactory.distance(places[2], places[3], ">", 1));
    solver.post(IntConstraintFactory.distance(places[2], places[4], ">", 1));
    solver.post(IntConstraintFactory.distance(places[2], places[5], ">", 1));
    solver.post(IntConstraintFactory.distance(places[2], places[6], ">", 1));
    solver.post(IntConstraintFactory.distance(places[3], places[5], ">", 1));
    solver.post(IntConstraintFactory.distance(places[3], places[6], ">", 1));
    solver.post(IntConstraintFactory.distance(places[4], places[5], ">", 1));
    solver.post(IntConstraintFactory.distance(places[4], places[7], ">", 1));
    solver.post(IntConstraintFactory.distance(places[5], places[6], ">", 1));
    solver.post(IntConstraintFactory.distance(places[5], places[7], ">", 1));
    solver.post(IntConstraintFactory.distance(places[6], places[7], ">", 1));
```

distance(X,Y,Rel,Z) is |X-Y| Rel Z
so this says

| places[0]-places[1] | > 1

(i.e. cannot be consecutive)

```
    //set a search strategy
    solver.set(IntStrategyFactory.domOverWDeg(places, 0));

    //Decide what to display in the output
     Chatterbox.showSolutions(solver);

     //solve the problem (and the Chatterbox will display some output)
     solver.findSolution();

     //display the search statistics
     Chatterbox.printStatistics(solver);
}
```

```
- Solution #1 found. 1 Solutions, Resolution 0.012s, 8 Nodes, 6 Backtracks, 5 Fails
    places[0] = 2 places[1] = 5 places[2] = 8 places[3] = 6 places[4] = 3 places[5]
= 1 places[6] = 4 places[7] = 7 .
- Complete search -
    Solutions: 1
    Building time : 0.101s
    Initialisation : 0.000s
    Initial propagation : 0.004s
    Resolution : 0.017s
    Nodes: 8
    Backtracks: 6
    Fails: 5
    Restarts: 0
    Max depth: 6
    Propagations: 247 + 0
    Memory: 0mb
    Variables: 10
    Constraints: 18
```

```
    //set a search strategy
    solver.set(IntStrategyFactory.domOverWDeg(places, 0));

    //Decide what to display in the output
     Chatterbox.showSolutions(solver);

     //solve the problem (and the Chatte
     solver.findSolution();

     //display the search statistics
     Chatterbox.printStatistics(solver);
}
```

tell the solver which search strategy to use

```
- Solution #1 found. Solver[Solver-0], 1 Solutions, Resolution time 0.013s, 8 Nodes
(637.4 n/s), 6 Backtracks, 5 Fails, 0 Restarts
    places[0] = 2 places[1] = 5 places[2] = 8 places[3] = 6 places[4] = 3 places[5]
= 1 places[6] = 4 places[7] = 7 .
...

- Complete search - 1 solution found.
    Solver[Solver-0]
    Solutions: 1
    Building time : 0.131s
    Resolution time : 0.044s
    Nodes: 8 (183.5 n/s)
    Backtracks: 6
    Fails: 5
    Restarts: 0
    Variables: 8
    Constraints: 18
```

choco-solver.org | A Free

Choco-3.3.3: an Open-Sou

file:///C:/Users/kbrown/choco-3.3.3/apidocs/index.html

All Classes

Packages

org.chocosolver.memory
org.chocosolver.memory.copy
org.chocosolver.memory.copy.stc
org.chocosolver.memory.generat
org.chocosolver.memory.structure
org.chocosolver.memory.trailing
org.chocosolver.memory.trailing.t
org.chocosolver.memory.trailing.t
org.chocosolver.memory.trailing.t
org.chocosolver.memory.trailing.t
org.chocosolver.samples

All Classes

AbsoluteEvaluation
AbstractBenchmarking
AbstractEnvironment
AbstractEnvironment.Type
AbstractLengauerTarjanDominat
AbstractNQueen
AbstractPenaltyFunction
AbstractProblem
AbstractProblem.Level
AbstractRestartStrategy
AbstractStoredObject
AbstractStrategy
AbstractVariable
ACounter
ActivityBased
AdaptiveNeighborhood
AirPlaneLanding
AlgoAllDiffAC
AlgoAllDiffBC
AllDifferent
AllIntervalSeries
AllSolutionsRecorder
Alpha
AlphaDominatorsFinder
AntiFirstFail

OVERVIEW PACKAGE CLASS USE TREE DEPRECATED INDEX HELP

PREV NEXT FRAMES NO FRAMES

Choco-3.3.3: an Open-Source Constraint Solver 3.3.3 API

Packages

Package	Description
org.chocosolver.memory	A package devoted to backtrackable data structures.
org.chocosolver.memory.copy	
org.chocosolver.memory.copy.store	
org.chocosolver.memory.generator	
org.chocosolver.memory.structure	
org.chocosolver.memory.trailing	
org.chocosolver.memory.trailing.trail	
org.chocosolver.memory.trailing.trail.chunck	
org.chocosolver.memory.trailing.trail.flatten	
org.chocosolver.memory.trailing.trail.unsafe	
org.chocosolver.samples	
org.chocosolver.samples.explanation	
org.chocosolver.samples.graph.input	
org.chocosolver.samples.integer	
org.chocosolver.samples.nqueen	
org.chocosolver.samples.nsp	
org.chocosolver.samples.pert	
org.chocosolver.samples.real	
org.chocosolver.samples.set	

org.chocosolver.solver.constraints

Class IntConstraintFactory

java.lang.Object
    org.chocosolver.solver.constraints.IntConstraintFactory

Direct Known Subclasses:

ICF

```
public class IntConstraintFactory
extends Object
```

A Factory to declare constraint based on integer variables (only). One can call directly the constructor of constraints, but it is recommended to use the Factory, because signatures and javadoc are ensured to be up-to-date.
As much as possible, the API names of global constraints must match those define in the Global Constraint Catalog.

Note that, for the sack of readability, the Java naming convention is not respected for methods arguments.

Constraints are ordered as the following: 1) Unary constraints 2) Binary constraints 3) Terary constraints 4) Global constraints

Since:

21/01/13

Author:

Charles Prud'homme

Method Summary

All Methods **Static Methods** **Concrete Methods**

choco-solver.org | A Free ... | Choco-3.3.3: an Open-Sou...

file:///C:/Users/kbrown/choco-3.3.3/apidocs/index.html

All Classes

Packages

org.chocosolver.memory
org.chocosolver.memory.copy
org.chocosolver.memory.copy.stc
org.chocosolver.memory.generat
org.chocosolver.memory.structure
org.chocosolver.memory.trailing
org.chocosolver.memory.trailing.t
org.chocosolver.memory.trailing.t
org.chocosolver.memory.trailing.t
org.chocosolver.memory.trailing.t
org.chocosolver.samples

ImpactBased
Indexable
IndexedBipartiteSet
IndexedObject
IndexFactory
INeighbor
InputOrder
IntCell
IntCircularQueue
IntComparator
IntConstraintFactory
IntDecision
IntDelta
IntDomainMax
IntDomainMedian
IntDomainMiddle
IntDomainMin
IntDomainRandom
IntDomainRandomBound
IntEqRealConstraint
IntervalDelta
IntervalDeltaMonitor
IntervalIntVarImpl
IntEventType
IntEvtScheduler
IntIterableBitSet
IntIterableSet

OP - an operator

VAR2 - second variable

arithm

```
public static Constraint arithm(IntVar VAR1,
                                String OP1,
                                IntVar VAR2,
                                String OP2,
                                int CSTE)
```

Ensures: VAR1 OP VAR2, where OP in {"=", "!=",">","<",">=","<="} or {"+", "-"}

Parameters:

VAR1 - first variable

OP1 - an operator

VAR2 - second variable

Arithmetic expressions as constraints:
`IntConstraintFactory.arithm(X1,"+",X2,"=",c)`
creates the constraint
X1 + X2 == c

```
                                IntVar VAR2,
                                String OP,
                                int CSTE)
```

Ensures:
|VAR1-VAR2| OP CSTE
where OP can take its value among {"=", ">", "<", "!="}

element

user_guide-3.3.3.pdf - Adobe Acrobat Standard

File Edit View Document Comments Forms Tools Advanced Window Help

Choco3 Documentation

Release 3.3.3

Charles Prud'homme, Jean-Guillaume Fages, Xavier Lorca

December 22, 2015

user_guide-3.3.3.pdf - Adobe Acrobat Standard

File Edit View Document Comments Forms Tools Advanced Window Help

Exercise:

(Make sure Java is on your system)
(Make sure Eclipse is on your system)
Download and install Choco as an Eclipse project

Model and solve the following problem:

$V = \{V1,V2,V3,V4\}$
$D = \{1,2,3,4,5\}$ for each var
$C = \{V1 \leq V4-1,$
$V1 < V2,$
$V2 + V3 > 6,$
$V2 + V4 = 5,$
$V4 < V3\}$

Setting up Eclipse with Choco on Windows

Choco: http://choco-solver.org/

Download choco 3.3.3, and unzip. Then unzip apidocs.zip
Make sure you have Java 1.8 JDK installed
Make sure you have Eclipse installed (I use Luna 4.4.2)

In Eclipse, create a new project (e.g. *CS4093*)
Select the project (CS4093),
right-click, Build Path / Configure Build Path / Libraries / Add External Jars ...
browse to your choco folder and select it, select choco-solver...dependencies.jar, then OK
In PackageExplorer/CS4093/Referenced Libraries,
right-click the jar file / Properties /Java Source Attachment / External Location,
browse to choco folder, select choco-solver...sources.jar, OK
In PackageExplorer/CS4093/Referenced Libraries,
right-click the jar file / Properties / Javadoc Location / Javadoc URL,
browse to Choco folder, apidocs subfolder, OK
In PackageExplorer/CS4093/src, create a new class, and start coding

Next lecture ...

modeling