U.PORTO
FEUP FACULDADE DE ENGENHARIA
UNIVERSIDADE DO PORTO

DEPARTMENT OF INFORMATICS ENGINEERING

Logic and Constraint Programming

Masters in Informatics and Computing Engineering
2022/2023 – 2nd Semester

Constraint Systems

Daniel Castro Silva
dcs@fe.up.pt

Agenda

- IBM ILOG CP Optimizer
- Docplex.CP
- Google OR Tools CP-SAT Solver
- Example Exercises

IBM ILOG CP Optimizer

Constraint Systems

IBM ILOG CP Optimizer (CPLEX)

- ILOG was one of the first companies (late 80s, early 90s) with a commercial tool using Constraint Programming technology
  - ILOG Solver was used in many industrial successful cases, mostly in Europe at first, but also around the world later on
- It was then acquired by IBM (late 2000s)
  - Developments continue, and IBM ILOG CP Optimizer continues to be one of the foremost constraint programming tools, used with special success in scheduling problems

See https://www.ibm.com/analytics/cplex-cp-optimizer

IBM ILOG CP Optimizer

- IBM ILOG CP Optimizer can be used directly from the IBM ILOG CPLEX Optimization Studio, using OPL (Optimization Programming Language), or using one of the available interfaces: C++, Java, C#/.Net, and Python (DOcplex)

IBM ILOG CP Optimizer

- CPLEX Optimization Studio and OPL provide
  - Separation of concerns between model and data
  - Support for external data sources (e.g., Excel files)
  - Good support for arrays, ranges, tuples and sets
  - Support for integer decision variables (*dvar*), but also floating-point decision expressions (*dexpr*) (e.g. for use as a cost function)
  - Many scheduling-related constraints
  - Some other global constraints
  - Many more features

IBM ILOG CP Optimizer

- IBM ILOG CP Optimizer provides elements to concisely represent complex scheduling problems:
  - Variables of type **interval**, with start, end, size and intensity attributes
  - Precedence constraints
  - Cumulative expressions to define resource constraints
  - Other elements to model sequencing, synchronization, and other constraints

  - Documentation center (v. 22.1.1)
  - Other documents
    - CP Optimizer User's Manual
    - OPL Language User's Manual
    - OPL Language Reference Manual
    - OPL Functions (Language Quick Reference)

Arithmetic Operations and Expressions

- Arithmetic operations
  - addition
  - subtraction
  - multiplication
  - scalar products
  - integer division
  - floating-point division
  - modular arithmetic
- Arithmetic expressions
  - standard deviation
  - minimum
  - maximum
  - counting
  - absolute value
  - element or index

See https://www.ibm.com/docs/en/icos/22.1.1?topic=expressions-arithmetic

Arithmetic and Logical Constraints

- Arithmetic constraints
  - equal to (==)
  - not equal to (!=)
  - strictly less than (<)
  - strictly greater than (>)
  - less than or equal to (<=)
  - greater than or equal to (>=)

- Logical constraints
  - Logical AND (&&)
  - Logical OR (||)
  - Logical NOT (!)
  - Logical XOR (!=)
  - Equivalence (==)
  - Imply (=>)

Scheduling Precedence Constraints

- endAtEnd
- endAtStart
- endBeforeEnd
- endBeforeStart
- startAtEnd
- startAtStart
- startBeforeEnd
- startBeforeStart

All can include a delay to further separate events

Other Scheduling Constraints

- alternative
- span
- synchronize
- isomorphism
- presenceOf
- first / last
- before / prev
- noOverlap
- <= / alwaysIn
- alwaysConstant / alwaysEqual / alwaysNoState

Other Specialized Constraints

- allDifferent
- allMinDistance
- pack
- distribute
- inverse
- lex

Variable and Value Selection

- Search phase can be guided in order to increase performance
  - Variable selection
    - What variable should we select next to attribute a value?
  - Value selection
    - What value should we try first for the selected variable?

- IBM ILOG CP Optimizer also supports different ‘search types’
  - Depth-first
  - Restart
  - Multi-point
  - Iterative-diving

Variable Selection

- selectSmallest(eval)
- selectLargest(eval)
- selectRandomVar()

Where eval may be:

- cp.factory.domainSize()
- cp.factory.domainMin()
- cp.factory.domainMax()
- cp.factory.regretOnMin()
- cp.factory.regretOnMax()
- cp.factory.successRate()
- cp.factory.impact()
- cp.factory.localImpact()
- cp.factory.impactOfLastBranch()
- cp.factory.varIndex(dvar int[])
- cp.factory.explicitVarEval(dvar int[],int[])

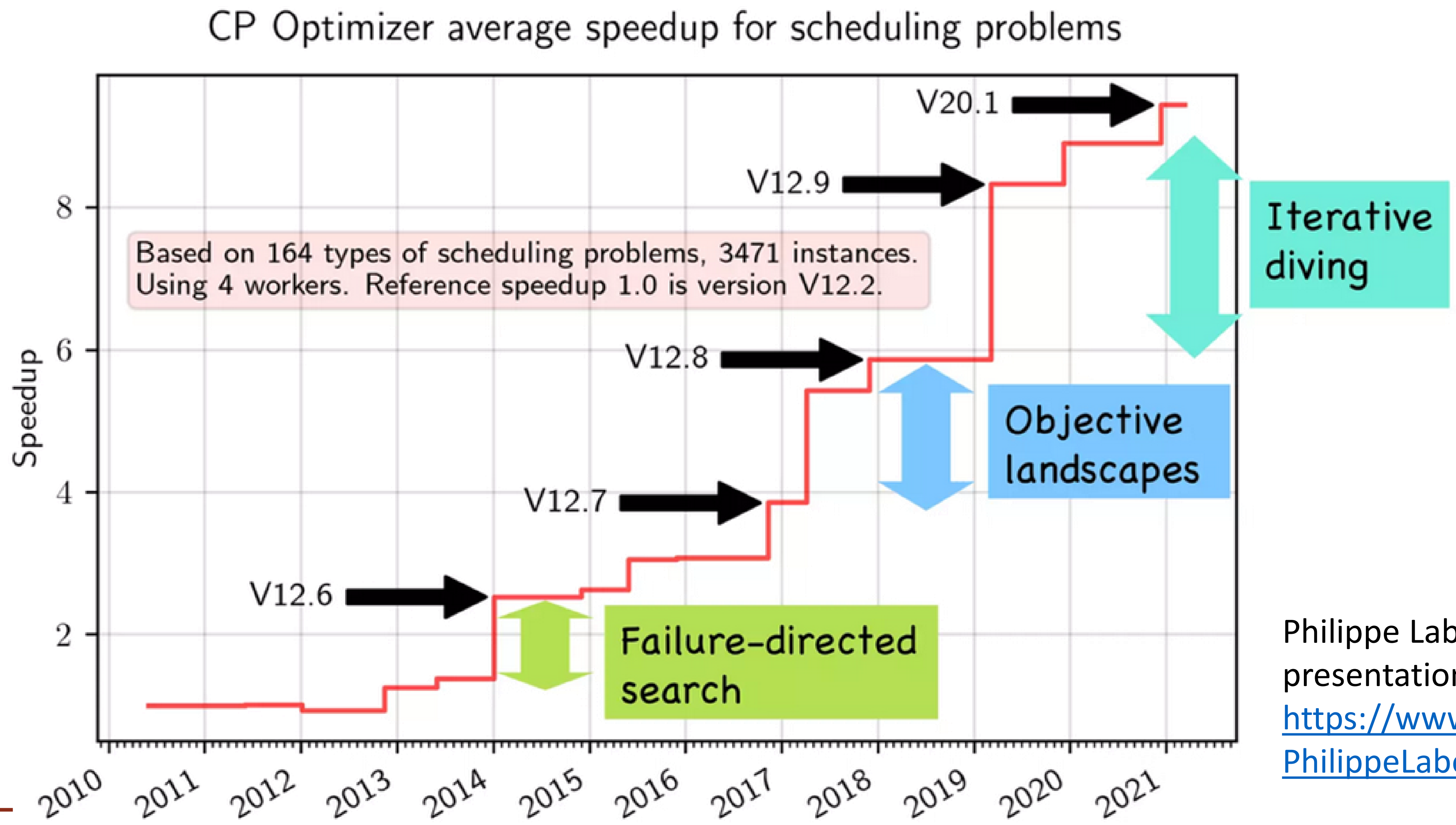
Value Selection

- selectSmallest(eval)
- selectLargest(eval)
- selectRandomValue()

Where eval may be:

- cp.factory.value()
- cp.factory.valueImpact()
- cp.factory.valueSuccessRate()
- cp.factory.valueIndex(int[])
- cp.factory.explicitValueEval(int[],int[])

IBM ILOG CP Optimizer



Philippe Laborie's presentations: https://www.slideshare.net/PhilippeLaborie

DOcplex.CP

Constraint Systems

DOcplex

- Python interface to IBM ILOG CP Optimizer
  - Provides access to both mathematical programming modeling and constraint programming modeling
  - Can work with local installation of CPLEX or connect to the cloud

See http://ibmdecisionoptimization.github.io/docplex-doc/

Typical Program Structure

- Import Solver module

  from docplex.cp.model import CpoModel

- Declare model

  model = CpoModel()

- Add Variables to the model
- Add Constraints to the model
- Solve the model

  solution = model.solve( )

  solution = model.solve(TimeLimit=120)

Adding Variables

- Creating a single integer variable

  varName = model.integer_var(MinValue, MaxValue, “VarNameInModel")

  - Integer variables can also be created with explicit domains

    varName = model.integer_var(name=“VarNameInModel“, domain=(1,3,5,7,9))

    varName = model.integer_var(name=“VarNameInModel“, domain=(1, (3,7), 9))

- Creating a list of integer variables at once

  varsName = model.integer_var_list(NVars, MinVal, MaxVal, “VarsNameInModel")

  - An explicit domain can also be used as with a single variable

    vars = model.integer_var_list(NVars, name=“VarsName“, domain=(1,3,5,7,9))

Adding Variables

- Creating a single interval variable

  varName = model.interval_var(Start, End, Length, "VarNameInModel")

  - Intervals can also be created as optional (*optional=True*)
  - There is a distinction between interval *Length* and *Size*
    - *Length* is the duration of the interval if the *intensity* is always 100%
    - *Size* is the actual duration of the interval (can be higher than *Length* if *intensity* is sometimes below 100%)
    - The *intensity* is a stepwise function that can describe efficiency over time
  - Lists of intervals can also be created at once

Adding Variables

- In addition to integers and intervals, DOcplex also has:
  - Binary variables (and binary variable lists) (these are equivalent to integer variables with domain [0, 1])

    varName = model.binary_var(“VarNameInModel")

    varsName = model.binary_var_list(Size, “VarNameInModel")

  - Sequence variables (they represent a sequence of intervals)

    varName = model.sequence_var(ListOfIntervals, “VarNameInModel")

Adding Variables

- You can also create dictionaries of (integer, interval or binary) variables in addition to lists

varsName = model.integer_var_dict(Keys, MinVal, MaxVal, “VarsNameInModel")

- Variable names are optional, but they are useful when visualizing (printing) the solution

See http://ibmdecisionoptimization.github.io/docplex-doc/cp/docplex.cp.expression.py.html for more information on creating variables using DOcplex

Adding Constraints

- The *add(Expr)* method allows adding expressions to the model, which can be constraints, objectives, search phases, ...
  - Arithmetic expressions
  - Logical expressions
  - Constraints
  - Objectives (minimize / maximize)
  - Search phases (variable / value selectors)

See http://ibmdecisionoptimization.github.io/docplex-doc/cp/docplex.cp.modeler.py.html for a list of expressions and constraints

Adding Constraints

- Examples:

```
model.add( model.all_diff(ListOfVars) )

model.add( sumVar == model.sum(ListOfVars) )

model.add( nValuesVar == model.count_different(ListOfVars) )

model.add( model.distribute(Occurrences, ListOfVars, Values) )

model.add ( model.minimize(VarName) )

model.add(
    model.search_phase(varchooser=model.select_smallest(model.domain_size()),
    valuechooser=model.select_smallest(model.value()) ) )
```

Adding Constraints

- Several properties and functions can be used to obtain information from variables, useful in specifying constraints
  - From integer variables we can determine:
    - The lower and upper bounds of the domain: *varName.lb*, *varName.ub*
    - The domain of the variable: *varName.get_domain()*
    - Whether a value is contained in the domain: *varName.domain_contains(Value)*
    - Whether the variable is binary: *varName.is_binary()*
    - And also set the domain of a variable
      - The domain is represented in the same manner as when declaring an integer variable given a domain

See http://ibmdecisionoptimization.github.io/docplex-doc/cp/docplex.cp.expression.py.html#docplex.cp.expression.CpoIntVar

Adding Constraints

- Interval variables also provide much information and functionality:
  - Obtaining interval start, end, length, size, intensity, etc.: *varName.get_start(), varName.get_end(), varName.get_length(), varName.get_size(), …*
  - Setting interval start, end, length, size, intensity, etc. using either intervals or specifying minimum and/or maximum values: *varName.set_start(Interval), varName.set_end(Interval), varName.set_length(Interval), varName.set_start_min(Value), varName.set_start_max(Value), …*
  - Determining whether the interval is optional, present or absent: *varName.is_optional(), varName.is_absent(), varName.is_present(), …*
  - Setting interval as optional, present or absent: *varName.set_optional(), varName.set_absent(), varName.set_present(), …*

See http://ibmdecisionoptimization.github.io/docplex-doc/cp/docplex.cp.expression.py.html#docplex.cp.expression.CpoIntervalVar

Google OR-Tools CP-SAT Solver

Constraint Systems

Google OR-Tools CP-SAT Solver

- OR-Tools provides many functionalities
  - Dedicated algorithms for specific problems (e.g. knapsack problem)
  - Includes a CP-SAT Solver
    - External interfaces (Python, C++, Java, .Net)
    - Several global constraints
    - Very good performance
      - eg, very good results in the MiniZinc Challenge

Google OR-Tools CP-SAT Solver

- Google OR-Tools provides elements to concisely represent several problems:
  - Boolean and Integer Variables
  - Intervals (and Optional Intervals)
  - Constants

  - For more information
    - OR-Tools – Constraint Programming
    - Python CP-SAT Reference

Typical Program Structure

- Import SAT-CP module

  from ortools.sat.python import cp_model

- Declare model

  model = cp_model.CpModel()

- Add Variables to the model
- Add Constraints to the model
- Declare the solver and solve the model

  solver = cp_model.CpSolver()

  status = solver.Solve(model)

Variable Types

- Constants
  - *model.NewConstant(Value)*
- Booleans
  - *model.NewBoolVar(Name)*
- Integers
  - *model.NewIntVar(Lower Bound, Upper Bound, Name)*
  - *model.NewIntVarFromDomain(Domain, Name)*
- Intervals
  - *model.NewIntervalVar(Start, Duration, End, Name)*
  - *model.NewOptionalIntervalVar(Start, Dur, End, Is Present, Name)*

Domains

- Domains can be constructed from lists of Values or Intervals

  dom = cp_model.Domain.FromValues([1,3,5,7,9])

  dom = cp_model.Domain.FromIntervals([[1-5], [7,7], [9,9]])

- There are some methods to obtain information about a domain

  dom.IsEmpty()

  dom.Size()

  dom.Min()

  dom.Max()

  dom.Contains(Value)

- There are also useful domain manipulation methods

  dom.UnionWith(anotherDomain)

  dom.IntersectionWith(anotherDomain)

Constraints

- Constraints over linear expressions
  - *Add(Linear Expression)*
  - *AddLinearConstraint(Linear Expression, Lower, Upper)* [Low<= Expr<= Up]
  - *AddLinearExpressionInDomain(Linear Expression, Domain)*
- Propositional Constraints
  - *AddBoolAnd(Literals)*
  - *AddBoolOr (Literals)*
  - *AddBoolXOr (Literals)*
  - *AddImplication(Antecedent, Consequent)*
  - Negation: *Var.Not()* [for Boolean variables]
- Absolute and Modulo
  - *AddAbsEquality(Value, Variable)* [Value = abs(Variable)]
  - *AddModuloEquality(Value, Variable, Modulo)* [Value = Variable % Modulo]

Constraints

- Division and Multiplication
  - *AddDivisionEquality(Value, Numerator, Denominator)*
  - *AddMultiplicationEquality(Value, List of Variables)*
- Sum, Term and Scalar Product
  - *Sum(Expressions)*
  - *Term(Expression, Coefficient)*
  - *ScalProd(Expressions, Coefficients)*
- Minimum and Maximum
  - *AddMaxEquality(Max Value, Variables)*
  - *AddMinEquality(Min Value, Variables)*
- Domain Mapping
  - *AddMapDomain(Variable, List of Bools, Offset)*

Constraints

- All Different
  - *AddAllDifferent(List of Variables)*
- Element
  - *AddElement(Index, List of Variables, Value)*
- Allowed and Forbidden Assignments
  - *AddAllowedAssignments(List of Variables, List of Tuples)*
  - *AddForbiddenAssignments(List of Variables, List of Tuples)*
- Circuit
  - *AddCircuit(List of Arcs)*
    - *Arcs are tuples (Source Node, Destination Node, Literal)*
- Cumulative
  - *AddCumulative(Intervals, Demands, Capacity)*

Constraints

- Reservoir
    - *AddReservoirConstraint(Times, Demands, Min, Max)*
    - *AddReservoirConstraintWithActive(Times, Demands, Actives, Min, Max)*
- Automaton
    - *AddAutomaton(Transitions Variables, Start State, Final States, Transitions)*
        - *Transitions is a list of tuples (From State, Variable Value, Destination State)*
- Inverse
    - *AddInverse(Variables, Inverse Variables)*
- No Overlapping Constraints
    - *AddNoOverlap(List of Intervals)*
    - *AddNoOverlap2D(X Intervals, Y Intervals)*

Other Features

- Reified Constraints
  - *Constraint.OnlyEnforceIf(Literal or List of Literals)*

- Optimization
  - *Minimize(Objective)*
  - *Maximize(Objective)*

- Search Strategy
  - *AddDecisionStrategy(Variables, Variable Strategy, Value Strategy)*

Solver Features

- Timeout
  - *solver.parameters.max_time_in_seconds = 10.0*
- Solve
  - *Solve(Model)*
  - *SearchForAllSolutions(Model, Callback)*
  - *SolveWithSolutionCallback(Model, Callback)*
- Statistics
  - *NumBooleans*
  - *NumBranches*
  - *NumConflicts*
  - *ObjectiveValue*
  - *UserTime*
  - *WallTime*
  - *Value(Variable or Expression)*

Example Exercises

Constraint Systems

Example Exercises

- 3x3 Magic Square
- Wedding Table
- Lazy Mailman
- Map Coloring
- Bus Company

3x3 Magic Square

- Solve the 3x3 Magic Square
  - Fill the square with values from 1 to 9
  - All cells must have a different value
  - The sum of each line, column or diagonal must be the same

A	B	C
D	E	F
G	H	I

3x3 Magic Square – (SICStus) Prolog Model

```
:-use_module(library(clpfd)).

magicSquare:-
        Vars = [C1, C2, C3, C4, C5, C6, C7, C8, C9],
        domain(Vars, 1, 9),
        all_distinct(Vars),
        C1 + C2 + C3 #= Soma,        % Rows
        C4 + C5 + C6 #= Soma,
        C7 + C8 + C9 #= Soma,
        C1 + C4 + C7 #= Soma,        % Cols
        C2 + C5 + C8 #= Soma,
        C3 + C6 + C9 #= Soma,
        C1 + C5 + C9 #= Soma,        % Diagonals
        C3 + C5 + C7 #= Soma,
        C1 #< C3, C3 #< C7,          % Symmetry-breaking constraints
        labeling([], Vars),
        write(Vars).
```

3x3 Magic Square – OPL Model

```
using CP;

dvar int numbers[1..9] in 1..9;
dvar int summ in 6..24;            //int summ = 15;    // Given by n (n^2 + 1)/2 (more efficient)

constraints
{
    allDifferent(numbers);
    numbers[1] + numbers[2] + numbers[3] == summ;    // Rows
    numbers[4] + numbers[5] + numbers[6] == summ;
    numbers[7] + numbers[8] + numbers[9] == summ;
    numbers[1] + numbers[4] + numbers[7] == summ;    // Cols
    numbers[2] + numbers[5] + numbers[8] == summ;
    numbers[3] + numbers[6] + numbers[9] == summ;
    numbers[1] + numbers[5] + numbers[9] == summ;    // Diagonals
    numbers[3] + numbers[5] + numbers[7] == summ;
    numbers[1]<numbers[3];                           // Symmetry-breaking constraints
    numbers[3]<numbers[7];
}
```

3x3 Magic Square – DOcplex.CP Model

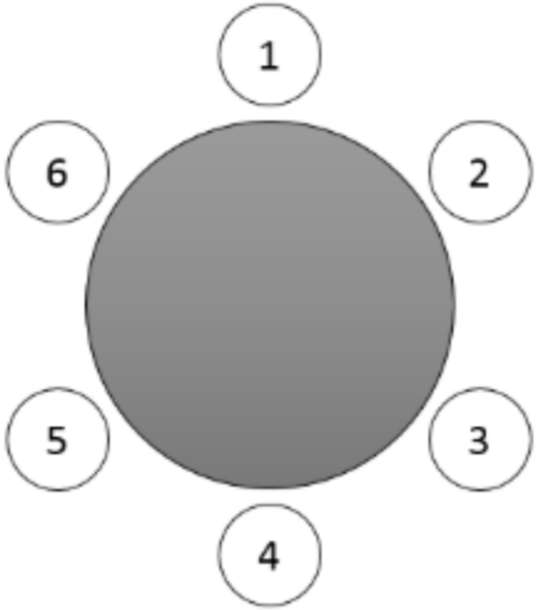
```
model = CpoModel()

Square = model.integer_var_list(9, 1, 9, "Squares")
Sum = model.integer_var(6, 24, "Sum")                                  # Sum = 15

for i in range(3):
    model.add( Square[i*3] + Square[i*3+1] + Square[i*3+2] == Sum )       # Row i
    model.add( Square[i] + Square[3+i] + Square[6+i] == Sum )             # Col i
model.add( Square[0] + Square[4] + Square[8] == Sum )                    # Diagonal \
model.add( Square[6] + Square[4] + Square[2] == Sum )                    # Diagonal /
model.add( model.all_diff(Square) )

model.add( Square[0] < Square[2] )                                     # Symmetry-breaking constraints
model.add( Square[2] < Square[6] )

solution = model.solve()
if solution:
    solution.print_solution()
```

3x3 Magic Square – OR-Tools Python Model

```
model = cp_model.CpModel()

List = [ model.NewIntVar(1, 9, 'v'+str(x+1)) for x in range(9) ]

Sum = model.NewIntVar(6, 24, "Sum")            # Sum = model.NewConstant(15)

for i in range(3):
    model.Add( List[i*3] + List[i*3+1] + List[i*3+2] == Sum)     # Rows
    model.Add( List[i] + List[3+i] + List[6+i] == Sum)           # Cols
model.Add( List[0] + List[4] + List[8] == Sum)           # Diagonal \
model.Add( List[6] + List[4] + List[2] == Sum)           # Diagonal /

model.AddAllDifferent(List)

# model.Add( List[0] < List[2] )            # Symmetry-breaking constraints
# model.Add( List[2] < List[6] )

solver = cp_model.CpSolver()
status = solver.Solve(model)
```

Wedding Table

- We want to sit six people in a round table such that some constraints are met:
  - Adam and Bernadette should sit together
  - Christina and Emmet should sit together
  - Emmet and Francis should NOT sit together
  - Adam and Emmet should NOT sit together



- Try to make the model flexible so as to adapt to different problem sizes
- Try to improve performance by avoiding symmetries

Wedding Table – (SICStus) Prolog Model

- List with the place each person is sitting at

```
wedTable(TableSize, Adj, Dist):-
        % Example input:
        %       TableSize = 6,  Adj = [1-2, 3-5],  Dist = [5-6, 1-5],

        length(PersonSeat, TableSize),
        domain(PersonSeat, 1, TableSize),
        all_distinct(PersonSeat),
        processAdj(TableSize, PersonSeat, Adj),
        processDist(TableSize, PersonSeat, Dist),

        element(1, PersonSeat, 1),              % Avoid Rotate Symmetry
        element(2, PersonSeat, S),              % Avoid Mirror Symmetry
        element(TableSize, PersonSeat, L),
        S #< L,

        labeling([], PersonSeat),
        write(PersonSeat).
```

Adam	1
Bernadette	2
Christina	3
Dina	4
Emmet	5
Francis	6

Wedding Table – (SICStus) Prolog Model

- Constraints

```
processAdj(_TableSize, _PersonSeat, []).
processAdj(TableSize, PersonSeat, [F-S|Adj]):-
        element(F, PersonSeat, FP),
        element(S, PersonSeat, SP),
        abs(FP-SP) #= 1 #\/ abs(FP-SP) #= TableSize-1,
        processAdj(TableSize, PersonSeat, Adj).

processDist(_TableSize, _PersonSeat, []).
processDist(TableSize, PersonSeat, [F-S|Dist]):-
        element(F, PersonSeat, FP),
        element(S, PersonSeat, SP),
        abs(FP-SP) #\= 1, abs(FP-SP) #\= TableSize-1,
        processDist(TableSize, PersonSeat, Dist).
```

Wedding Table – OPL Model

- List with the place each person is sitting at and problem input

```
using CP;

int TableSize = 6;

dvar int PersonSeat [1..TableSize] in 1..TableSize;

tuple Pair {
	int First;
	int Second;
};

int NAdjacencies = 2;
int NDistances = 2;
Pair adj [1..NAdjacencies] = [ <1, 2>, <3, 5> ];
Pair dist [1..NDistances] = [ <5, 6>, <1, 5> ];
```

Adam	1
Bernadette	2
Christina	3
Dina	4
Emmet	5
Francis	6

Wedding Table – OPL Model

- Constraints
  - The model can work with any problem size (table size, number of constraints), merely by adjusting input

```
constraints
{
      allDifferent(PersonSeat);                    //Everyone is sitting in a different place

      // people who should be sitting together are sitting together
      forall(i in 1..NAdjacencies)
            abs( PersonSeat[adj[i].First] - PersonSeat[adj[i].Second] ) == 1
            || abs( PersonSeat[adj [i].First] - PersonSeat[adj[i].Second] ) == TableSize - 1;

      // people who should not be sitting together are sitting apart
      forall(i in 1..NDistances)
            abs( PersonSeat[dist[i].First] - PersonSeat[dist[i].Second] ) != 1
            && abs( PersonSeat[dist[i].First] - PersonSeat[dist[i].Second] ) != TableSize - 1 ;
}
```

Wedding Table – OPL Model

- Removal of symmetries
  - We can add some constraints to limit the number of symmetrical solutions, thus improving the performance of the solver

```
constraints
{
        ...

        PersonSeat[1] == 1;          // Adam is sitting in place 1 (avoid rotations)

        PersonSeat [2] == 2;         // Bernadette is sitting next to Adam in place 2 (avoid mirrored solution)
                                        (relies on knowledge from
                                         specific problem instance)

        PersonSeat [2] < PersonSeat [TableSize];        // alternative constraint (does not depend on
                                                            problem instance)

}
```

Wedding Table – DOcplex.CP Model

- List with the place each person is sitting at

Adam	1
Bernadette	2
Christina	3
Dina	4
Emmet	5
Francis	6

```
TableSize = 6;
Adjacents = [ [1, 2], [3, 5] ]
Distants = [ [5, 6], [1, 5] ]

model = CpoModel()

PersonSeat = model.integer_var_list(TableSize, 1, TableSize, "PersonSeat")
model.add( model.all_diff(PersonSeat) )

distances = []
```

Wedding Table – DOcplex.CP Model

- Constraints and symmetry removal

```
for pair in Adjacents:
    distances.append(model.integer_var(domain=(-TableSize+1, -1, 1, TableSize-1),
                                       name="d"+str(pair[0])+str(pair[1])  ))
    model.add( PersonSeat[ pair[0]-1 ] - PersonSeat[ pair[1]-1 ] == distances[-1] )

for pair in Distants:
    model.add(PersonSeat[ pair[0]-1 ] - PersonSeat[ pair[1]-1 ] != 1)
    model.add(PersonSeat[ pair[0]-1 ] - PersonSeat[ pair[1]-1 ] != -1)
    model.add(PersonSeat[ pair[0]-1 ] - PersonSeat[ pair[1]-1 ] != TableSize-1)
    model.add(PersonSeat[ pair[0]-1 ] - PersonSeat[ pair[1]-1 ] != -TableSize+1)

model.add( PersonSeat[0] == 1)
model.add( PersonSeat[1] < PersonSeat[TableSize-1] )

solution = model.solve()

if solution:
    solution.print_solution()
```

Wedding Table – OR-Tools Python Model

- List with the place each person is sitting at

Adam	1
Bernadette	2
Christina	3
Dina	4
Emmet	5
Francis	6

```
TableSize = 6;
Adjacents = [ [1, 2], [3, 5] ]
Distants = [ [5, 6], [1, 5] ]

PersonSeat = [TableSize]        # To use 1-based indexes

model = cp_model.CpModel()

for i in range(TableSize):
    PersonSeat.append( model.NewIntVar(1, TableSize, "p"+str(i+1)) )

model.AddAllDifferent(PersonSeat[1:])
```

Wedding Table – OR-Tools Python Model

- Constraints and symmetry removal

```
distances = []

for pair in Adjacents:              # Aux var with possible adjacency distance values for each adjacent pair
    distances.append( model.NewIntVarFromDomain( cp_model.Domain.FromValues([-TableSize+1,
            -1, 1, TableSize-1]), "d"+str(pair[0])+str(pair[1])  ))
    model.Add( PersonSeat[ pair[0] ] - PersonSeat[ pair[1] ] == distances[-1] )

for pair in Distants:
    model.Add(PersonSeat[ pair[0] ] - PersonSeat[ pair[1] ] != 1)
    model.Add(PersonSeat[ pair[0] ] - PersonSeat[ pair[1] ] != -1)
    model.Add(PersonSeat[ pair[0] ] - PersonSeat[ pair[1] ] != TableSize-1)
    model.Add(PersonSeat[ pair[0] ] - PersonSeat[ pair[1] ] != -TableSize+1)

model.Add( PersonSeat[1] == 1)                        # Remove some symmetries
model.Add( PersonSeat[2] < PersonSeat[TableSize] )

solver = cp_model.CpSolver()
status = solver.Solve(model)
```

Lazy Mailman

- A lazy mailman with few letters to deliver has set a goal of taking as long as possible to deliver the mail in the last street of his round
  - It is a straight street, with ten houses, all ten meters apart from each other
  - He always walks at ten meters per minute, and wants to finish at house 6 (the person living there always offers him coffee and cake).
  - He has a (greedy) solution, starting in house 1, then going to house 10, then house 2, ..., and finally 6, which results in 45 minutes (9+8+7+6+5+4+3+2+1)
  - Model this problem using constraint programming to find a better solution (one that takes even longer than 45 minutes)
  - Note: consider that the mailman enters the street orthogonally and time starts counting from the first house he visits

Lazy Mailman – (SICStus) Prolog Model

- List of visited houses (order of visit)

```
lazy:-	Size = 10,
	length( HouseOrder, Size ),
	domain( HouseOrder, 1, Size ),
	all_distinct( HouseOrder ),
	element( Size, HouseOrder, 6 ),
	calcDistance( HouseOrder, Distance ),
	Distance #> 45,
	labeling( [maximize(Distance)], HouseOrder),
	write( HouseOrder-Distance ).

calcDistance( [Last], 0).
calcDistance( [F, N|R], Distance):-
	Step #= abs(F - N),
	calcDistance( [N|R], NDist),
	Distance #= NDist + Step.
```

Lazy Mailman – OPL Model

- List of visited houses (order of visit)

```
using CP;

dvar int houseOrder [1..10] in 1..10;

dexpr float distance = sum(i in 1..9) abs(houseOrder[i+1] - houseOrder[i]) ;

maximize distance;

subject to
{
	allDifferent(houseOrder);
	houseOrder[10] == 6;
	distance >= 45;
}
```

Lazy Mailman – OPL Model

- Possible improvement: *execute* block with instructions to change variable and value choice methods and/or search type

```
execute
{
    var f = cp.factory;

    //var phase1 = f.searchPhase(houseOrder,     f.selectSmallest(f.domainSize()),
    //                                           f.selectLargest(f.value()));

    var phase1 = f.searchPhase(houseOrder,       f.selectLargest(f.impact()),
                                                 f.selectLargest(f.value()));

    cp.setSearchPhases(phase1);

    cp.param.SearchType = "DepthFirst";
}
```

Lazy Mailman – DOcplex.CP Model

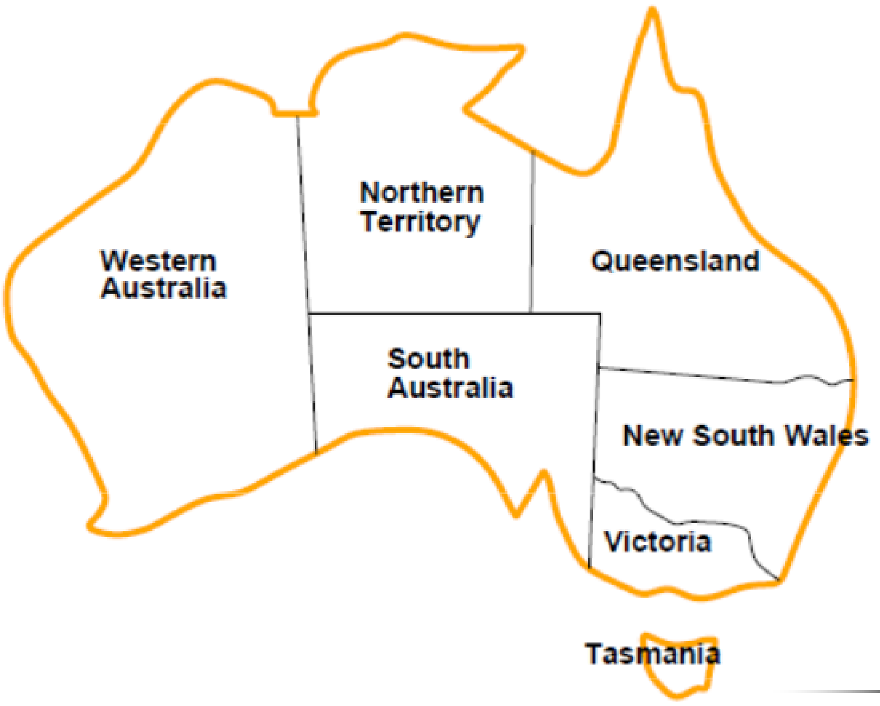
```
model = CpoModel()
NHouses = 10
houses = model.integer_var_list(NHouses, 1, NHouses, "Houses")

model.add( model.all_diff(houses) )
model.add( houses[NHouses-1] == 6 )

distances = model.integer_var_list(NHouses-1, 1, NHouses, "Distances")
for i in range(0, NHouses-1):
    model.add( distances[i] == model.abs( houses[i+1] - houses[i] ) )
dist = model.integer_var(0, NHouses * NHouses, "Dist")
model.add( dist == model.sum(distances) )
model.add( model.maximize(dist) )

solution = model.solve(TimeLimit=120)
if solution:
    solution.print_solution()
```

Lazy Mailman – OR-Tools Python Model

```
model = cp_model.CpModel()
NHOUSES = 10
houses = [ model.NewIntVar(1, NHOUSES, 'h'+str(i)) for i in range(1, NHOUSES+1) ]

model.AddAllDifferent(houses)
model.AddElement(NHOUSES-1, houses, 6)

travelTime = []
for i in range(NHOUSES-1):
    tempVar = model.NewIntVar( -NHOUSES, NHOUSES, 'o'+str(i) )
    model.Add( tempVar == houses[i+1] - houses[i] );
    travelTime.append( model.NewIntVar(1, NHOUSES, 'd'+str(i)) )
    model.AddAbsEquality( travelTime[-1], tempVar )
dist = model.NewIntVar(0, NHOUSES*NHOUSES, "Dist" )
model.Add( dist == sum(travelTime) )
model.Maximize(dist)
solver = cp_model.CpSolver()
status = solver.Solve(model)
```

Map Coloring

- Map Coloring is a classic problem, with the goal of coloring a map with N different colors such that no two adjacent areas have similar colors.
  - Solve the problem for Australia using the minimum amount of colors possible (and at most 5 colors)



Map Coloring – (SICStus) Prolog Model

```
mapColor:-
        length(StateColors, 7),
        domain(StateColors, 1, 5),
        % StateNames = ['WA', 'NT', 'SA', 'Q', 'NSW', 'V', 'T'],
        StateAdjacencies = [1-2, 1-3, 2-3, 2-4, 3-4, 3-5, 3-6, 4-5, 5-6],
        processAdj(StateColors, StateAdjacencies),
        maximum(MaxColor, StateColors),
        labeling([minimize(MaxColor)], StateColors),
        write(StateColors).

processAdj(_StateColors, []).
processAdj(StateColors, [F-S|Adj]):-
        element(F, StateColors, FC),
        element(S, StateColors, SC),
        FC #\= SC,
        processAdj(StateColors, Adj).
```

Map Coloring – OPL Model

```
using CP;

int NStates = 7;
//string StateNames[1..NStates] = ["WA", "NT", "SA", "Q", "NSW", "V", "T"];

tuple Pair{ int first; int second; }
{Pair} StateAdjacencies = {<1,2>, <1,3>, <2,3>, <2,4>, <3,4>, <3,5>, <3,6>, <4,5>, <5,6>};
int MaxColors = 5;

dvar int StateColors[1..NStates] in 1..MaxColors;

minimize max(i in 1..NStates) StateColors[i];

subject to
{
	forall(<a, b> in StateAdjacencies)
		StateColors[a] != StateColors[b];
}
```

Map Coloring – DOcplex.CP Model

```
model = CpoModel()

NStates = 7
StateNames = ["WA", "NT", "SA", "Q", "NSW", "V", "T"];
StateAdjacencies = [ (1,2), (1,3), (2,3), (2,4), (3,4), (3,5), (3,6), (4,5), (5,6) ]
MaxColors = 5

StateColors = model.integer_var_list(NStates, 1, MaxColors, "StateColors")
for a, b in StateAdjacencies:
    model.add(StateColors[a-1] != StateColors[b-1])

AllColors = list( range(1, MaxColors+1) )
ColorCounts = model.integer_var_list(MaxColors, 0, NStates, "ColorCounts")

model.add( model.distribute(ColorCounts, StateColors, AllColors) )
model.add( model.maximize( model.count(ColorCounts, 0) ) )

solution = model.solve(TimeLimit=120)
if solution:
    solution.print_solution()
```

Map Coloring – OR-Tools Python Model

```
MaxColors = 5
NStates = 7
StateNames = ["WA", "NT", "SA", "Q", "NSW", "V", "T"]
StateAdjacencies = [ (1,2), (1,3), (2,3), (2,4), (3,4), (3,5), (3,6), (4,5), (5,6) ]

StateColors = [ model.NewIntVar(1, MaxColors, 'State‘ + str(i)) for i in range(NStates) ]

for a, b in StateAdjacencies:
    model.Add(StateColors[a-1] != StateColors[b-1])

model.Minimize( max(StateColors) )

solver = cp_model.CpSolver()
status = solver.Solve(model)
```

Holiday Bus Company

- A bus company has several buses that can be used to ferry several groups of tourists to their vacation destination.
- Different buses have different capacities, and each group of tourists has a different size.
- The goal is to allocate groups of tourists to buses such that:
  - Each group is not separated (the entire group travels in the same bus)
  - The number of used buses is minimized (each bus may ferry several groups)
- Example problem input:
  - 4 buses with capacities of 11, 14, 10, 20
  - 5 groups of sizes 5, 5, 7, 4, 3

Holiday Bus Company – (SICStus) Prolog Model

```
busCompany:-
    Buses = [11, 14, 10, 20],
    length(Buses, NBuses),
    Groups = [5, 5, 7, 4, 3],
    length(Groups, NGroups),
    create_items(Groups, Items, AssignedBuses),
    domain(AssignedBuses, 1, NBuses),

    create_bins(Buses, 1, Bins),
    bin_packing(Items, Bins),
    nvalue(UsedBuses, AssignedBuses),

    labeling([minimize(UsedBuses)], AssignedBuses),
    write(UsedBuses-AssignedBuses).

create_items([], [], []).
create_items([Size | Gs], [item(Bin, Size) | Items], [Bin | IDs]):-
        create_items(Gs, Items, IDs).
```

```
create_bins([], _, []).
create_bins([Max | As], ID, [bin(ID, Cap) | Bs]):-
        Cap #=< Max,
        ID1 is ID + 1,
        create_bins(As, ID1, Bs).
```

Holiday Bus Company – OPL Model

```
using CP;

// Buses (Bin maxLoads)
int NBuses = 4;
int MaxLoads[1..NBuses] = [11, 14, 10, 20];
int MaxMaxLoad = max(i in 1..NBuses) MaxLoads[i];
dvar int Loads[1..NBuses] in 0..MaxMaxLoad;

// Groups (Item weights)
int NGroups = 5;
int Weights[1..NGroups] = [5, 5, 7, 4, 3];

// Attribution (Packing)
dvar int PackIDs [1..NGroups] in 1..NBuses;

// Used Buses (Non-zero)
dvar int NonZero in 1..NBuses;
```

```
minimize NonZero;

subject to
{
    forall(i in 1..NBuses)
        Loads[i] <= MaxLoads[i];
    pack(Loads, PackIDs, Weights, NonZero);
}
```

Holiday Bus Company – DOcplex.CP Model

```
model = CpoModel()

# Groups (Weights)
NGroups = 5
Weights = [5, 5, 7, 4, 3]

# Buses (MaxLoads)
NBuses = 4
MaxLoads = [11, 14, 10, 20]
MaxMaxLoad = max(MaxLoads)
Loads = model.integer_var_list(NBuses, 0, MaxMaxLoad, "Loads")

# Attribution (Packing)
PackIDs = model.integer_var_list(NGroups, 1, NBuses, "PackIDs")

# Used Buses (Non-zero)
NonZero = model.integer_var(1, NBuses, "NonZero")
```

Holiday Bus Company – DOcplex.CP Model

```
for i in range(NBuses):
    model.add( Loads[i] <= MaxLoads[i] )

model.add( model.pack(Loads, PackIDs, Weights, NonZero) )

model.add( model.minimize(NonZero) )

solution = model.solve( TimeLimit=120 )

if solution:
    solution.print_solution()
```

Holiday Bus Company – OR-Tools Python Model

- The new CP-SAT Solver lacks several global constraints that could be very useful:
  - *pack*, *nvalue*, *global_cardinality* (*distribute*), *count*, ...

- The original CP Solver has some of these constraints
  - Documentation on the original CP solver available online at https://developers.google.com/optimization/reference/python/constraint_solver/pywrapcp#solver_3

Holiday Bus Company – OR-Tools Python Model

```
model = cp_model.CpModel()

NGroups = 5
Weights = [5, 5, 7, 4, 3]

NBuses = 4
MaxLoads = [11, 14, 10, 20]

Loads = [model.NewIntVar(0, MaxLoads[i], "Loads"+str(i)) for i in range(NBuses)]

# Attribution (0-1 Matrix)
Attribution = [ [model.NewIntVar(0, 1, "Attr"+str(i)+str(j) ) for i in range(NBuses)] for j in range(NGroups)]

# Groups are exactly on one Bus
for i in range(NGroups):
    model.Add( 1 == sum(Attribution[i][j] for j in range(NBuses)) )
```

Holiday Bus Company – OR-Tools Python Model

```
# Max Loads on Buses
for i in range(NBuses):
    model.Add( Loads[i] == sum(Attribution[j][i]*Weights[j] for j in range(NGroups) ) )

Zeros = model.NewIntVar(1, NBuses, "Zeros")
add_count_eq(Loads, 0, Zeros, model)
model.Maximize( Zeros )

solver = cp_model.CpSolver()
status = solver.Solve(model)
```

```
def add_count_eq(vars, value, count, model):
    boolvars = [ ]
    for var in vars:
        boolvar = model.NewBoolVar('')
        model.Add(var == value).OnlyEnforceIf(boolvar)
        model.Add(var != value).OnlyEnforceIf(boolvar.Not())
        boolvars.append(boolvar)
    model.Add(count == sum(boolvars))
```

Q & A

?