A Brief Introduction to Constraint Programming with Minizinc

Tong Liu, 22/01/2018

Dip. Informatica, Mura Anteo Zamboni 7

BOLOGNA BUSINESS SCHOOL
Alma Mater Studiorum Università di Bologna

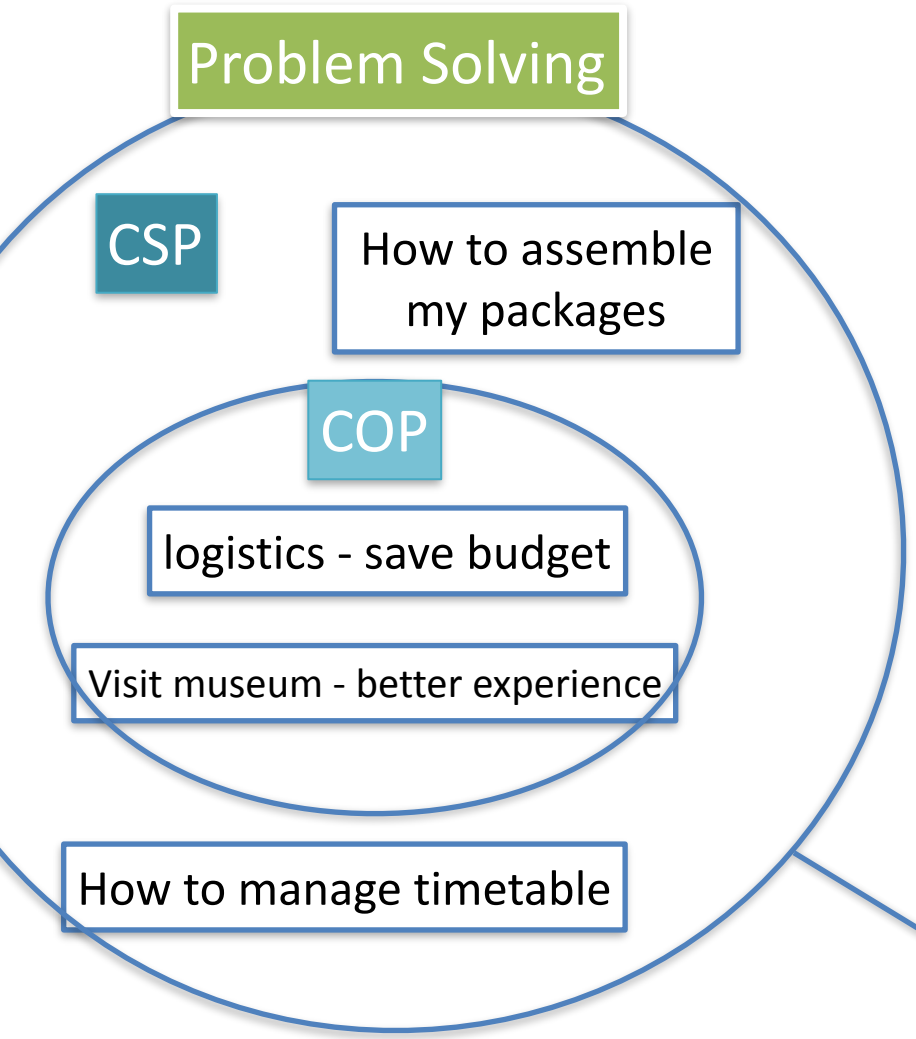
CP - Constraint Programming

"... the user states the problem, the computer solves it."
Eugene C. Freuder - UCC

Outline

- Constraint Programming in AI
- Procedural Language vs. Declarative Language
- CP Methodology
- CSP & COP
- Elements in Minizinc
- Examples with Minizinc
- Exercises

Constraint Programming in AI



Machine Learning

Robotics

Artificial Intelligence

Knowledge Reasoning

CP: Constraint Programming
ILP: Integer Linear Programming
Local Search
…

CP - Declarative Programming

Procedural:

1 Go to kitchen
2 Get Tea leaf and water
3 Mix them and heat over fire till it boils
4 Put that in a cup and bring it to me

e.g. Python, C, Java

Declarative:

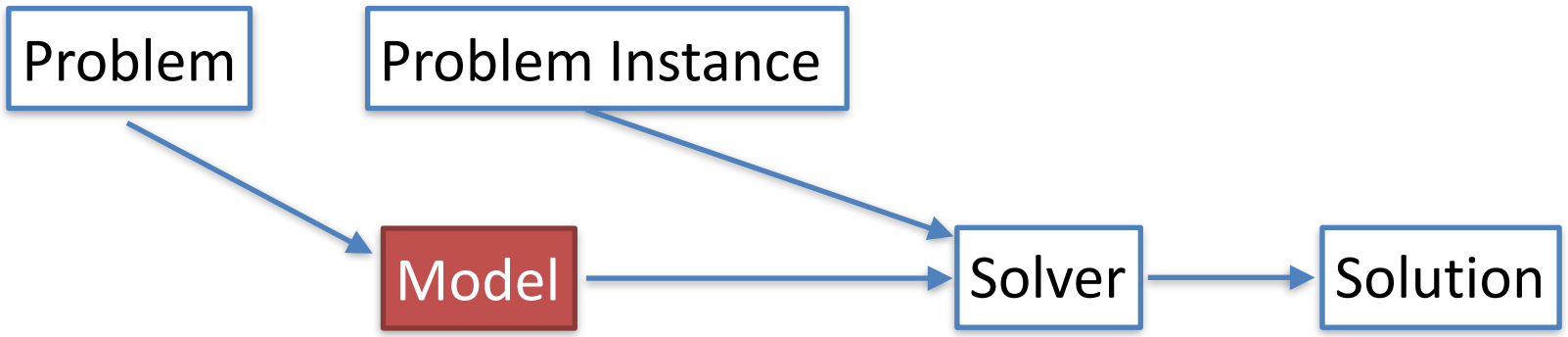
1 Tea is composed by tea leaf with hot water
2 Get me a cup of tea.

Minizinc

Declarative programming is where you describe what you want without having to say how to do it. With procedural programming, you have to specify exact steps to get the result.

CP - Methodology

Model Problem - Solve Model



state (describe) the problem

CP - How to describe a problem to Computer?

- Entities (Variables) in the problem and who are they?
  - number, string, set of (number/string)
  - the domain of Entities
- Relation (Constraints) between the Entities and what entity should look like
- My Goal (Objective)

- e.g. planing a tour
  - Entities: city, distance between cities …
  - Relation: I won't visit a city twice, I start from city x …
  - Goal: I want to save my time
  - => a path that satisfies the constraints

- => Formally, we describe it as a CSP (and COP)

CSP - Constraint Satisfaction Problem

- A CSP is defined by
  - a finite set of variables {*X1,…,Xn*}
  - a set of values(domain) for each variable *dom(X1)*,… *dom(Xn)*
  - A set of constraints {*C1*,…,*Cm*}
- A **solution** to a CSP is a complete assignment to all the variables that satisfies the constraints.

COP - Constraint Optimization Problem

A COP is a CSP defined on the variables x1, . . . , xn, together with an objective function f : D(x1) × · · · × D(xn) → Q that assigns a value to each variable. An optimal solution to a COP is a solution d that optimize the value of f (d).

Model CSP with Minizinc

Minizinc, a well-known Constraint Modeling Language which is becoming the standard

- Mainly at U of Melbourne and Monash U, Australia
- Introduced in 2007, v 2.1.7 in 2018
- Homepage: http://www.minizinc.org.
- Courses available also in Coursera

Some features of Minizinc

- Each expression terminates with ';'
- Variable domain, array index domain must all be specified
  - a domain could be 1..10 (Yes, '..', not '…') or int or an array
- Keyword 'constraint' denotes the rules that a solution should meet
- Index starts from '1' not '0'
- '%' for comments

```
1  include "globals.mzn";
2
3  int: n;
4  array[1..n,1..n] of int: dist;
5  int: start_city;
6  int: end_city;
7  array[1..n] of var 1..n: city;
8  array[1..n] of string: city_name;
9
10 constraint  city[1] = start_city;
11 constraint  city[n] = end_city;
12 constraint  all_different(city);
```

Import global constraint library in order to use 'all_different()'

Parameter (not variable), its value will be given by problem instance

Variable (a.k.a decision variable), its domain values will be checked in order to find solution

Constraints specify your requirements e.g. I start and end up my trip in specific cities

Minizinc Basics: Data Representation

- Parameters - values are passed by problem instance
- [domain] : [parameter name]
- e.g. int: n = 10;

- Variables - values depend on the solution
- var [domain] : [variable name]
- e.g. var int: total_distance; % traveling distance in a tour

- Arrays - can be a set of either parameters or variables
- array[index_domain] of [domain] %parameter array
- array[index_domain] of var [domain] %variable array
- e.g.
- array[1..n] of var 1..n: city; % the order of city I visit from 1 to n
- array[1..n,1..n] of int: distance; % distance between a pair of city

Minizinc Basics: Aggregation Functions

- sum, product, min, max
- e.g.
  - sum(array_x), % sum of all the elements in array_x
  - sum(i in 1..3)(array_x[i]), % sum of elements from 1 to 3 of array_x

- forall (*counter(s) in domain*) (*constraints*),
- forall (*counter(s) in domain* where *condition*) (*constraints*)
- e.g.
  - forall (i,j **in** 1..3 **where** i < j) (array_x[i] != array_x[j]) % first 3 elements in array_x are different

- exists (*counter(s) in domain*) (*constraints*)
- and others …

Minizinc Basics: Constraints

- Constraints are rules that a solution must respect
- constraints [expression]

- e.g.
- constraint city[1] = start_city
- constraint all_different(city);

- all_different is a global constraint, which deals with an arbitrary number of variables. Global constraints are notations easily recognized by CP solvers where efficient solving techniques will be applied.
- global constraints are also in fact composed by basic constraints:

```
all_different(array x) =
forall ( i, j in index(x) where i < j ) ( x[i] != x[j] )
```

Minizinc Basics: Constraints Language

- Arithmetic Operators
  - +, -, *, /,^, =, !=
  - e.g. x+y^2 = z

- Logical Operators
  - \/, or
  - /\, and,
    - constraint A /\ B; means also constraint A; constraint B;
  - ->, implies
  - ! negation
  - e.g. x=1 -> y != 5 % if x quals to 1 then y must not be 5

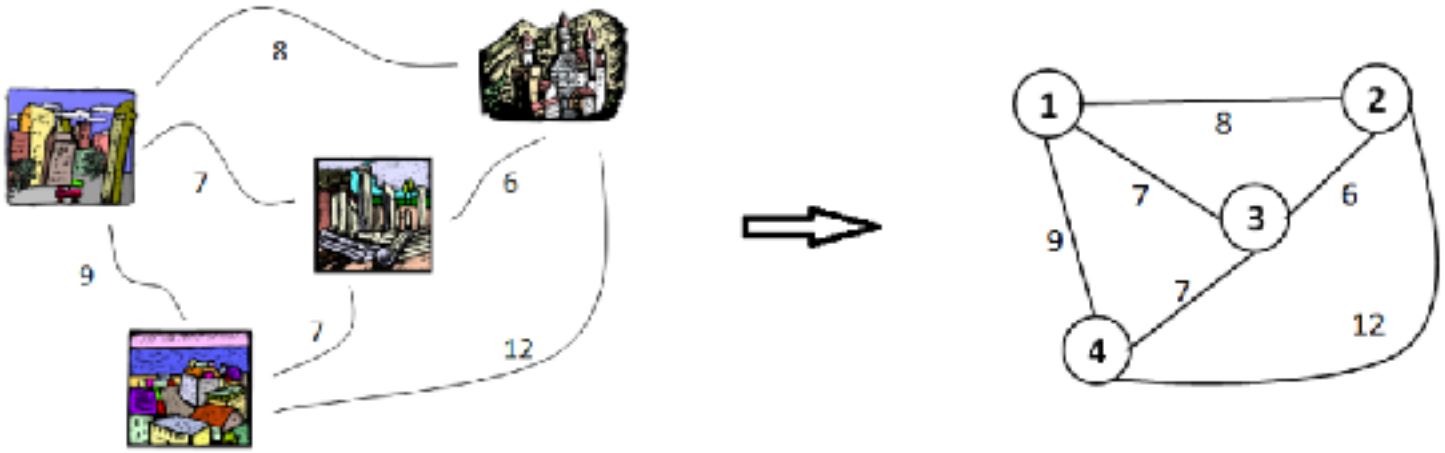
- Global Constraints
  - all_different()
  - all_equal() …

Minizinc Basics: Objective

- define the objective
- define the aim of solution

- var int: total_distance = sum(i in 2..n) (dist [city[i-1], city[i]]);
- solve minimize total_distance;

Understanding Minizinc with problems



Traveling Salesman Problem:

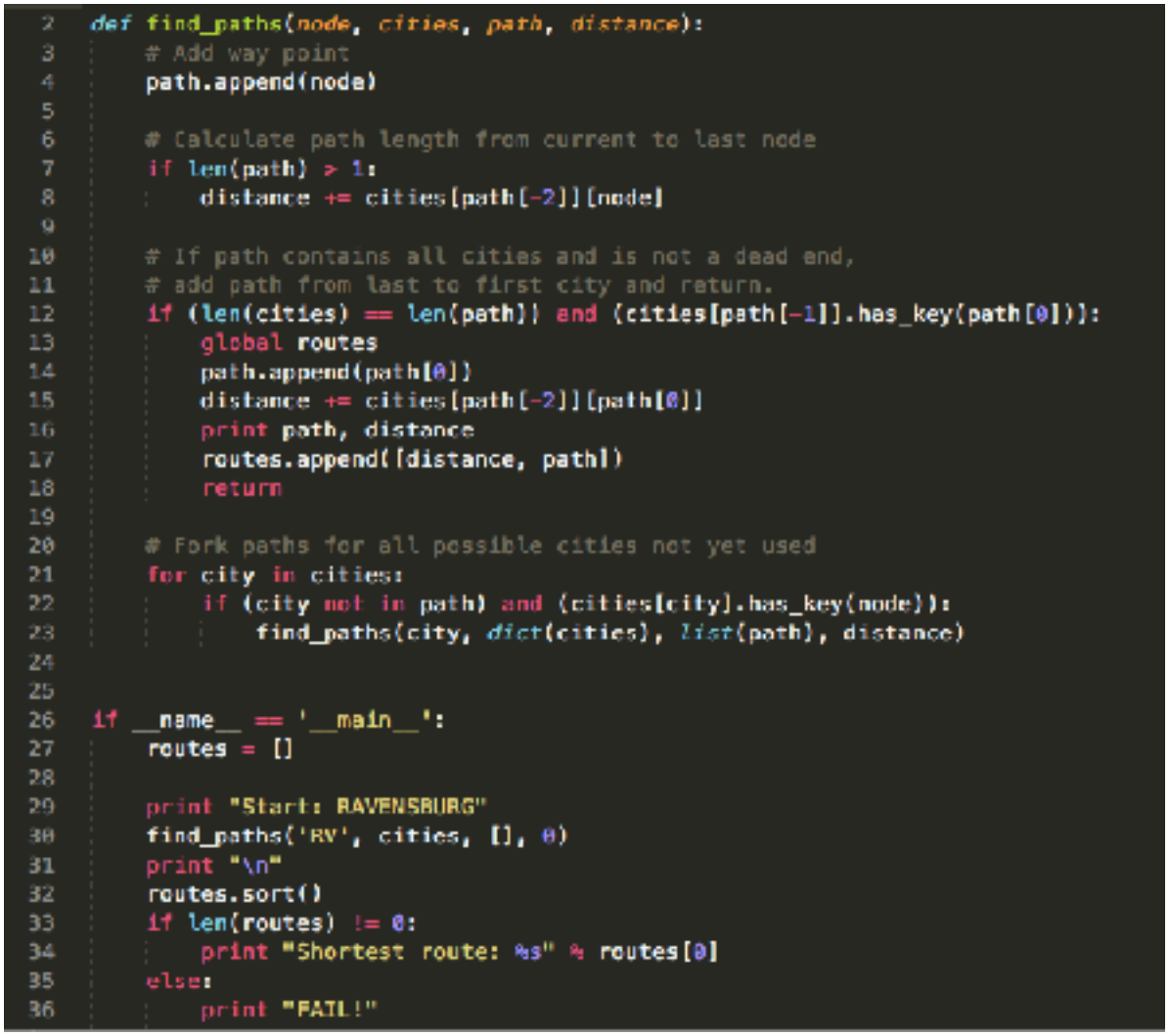
I visit each city once and I want to save my time

Complete Code - Traveling Salesman Problem

```
1  include "globals.mzn";
2
3  int: n;
4  array[1..n,1..n] of int: dist;
5  int: start_city;
6  int: end_city;
7
8  array[1..n] of var 1..n: city;
9  array[1..n] of string: city_name;
10
11 constraint  city[1] = start_city;
12 constraint  city[n] = end_city;
13 constraint  all_different(city);
14
15 var int: total_distance = sum(i in 2..n)(dist[city[i-1],city[i]]);
16 solve minimize total_distance;
17
18 output [city_name[fix(city[i])] ++ " -> " | i in 1..n ] ++
19         ["\nTotal hours travelled: ", show(total_distance) ];
20
```

Just state your problem

Traveling Salesman Problem in Python



```python
def find_paths(node, cities, path, distance):
    # Add way point
    path.append(node)

    # Calculate path length from current to last node
    if len(path) > 1:
        distance += cities[path[-2]][node]

    # If path contains all cities and is not a dead end,
    # add path from last to first city and return.
    if (len(cities) == len(path)) and (cities[path[-1]].has_key(path[0])):
        global routes
        path.append(path[0])
        distance += cities[path[-2]][path[0]]
        print path, distance
        routes.append([distance, path])
        return

    # Fork paths for all possible cities not yet used
    for city in cities:
        if (city not in path) and (cities[city].has_key(node)):
            find_paths(city, dict(cities), list(path), distance)


if __name__ == '__main__':
    routes = []

    print "Start: RAVENSBURG"
    find_paths('RV', cities, [], 0)
    print "\n"
    routes.sort()
    if len(routes) != 0:
        print "Shortest route: %s" % routes[0]
    else:
        print "FAIL!"
```

Problem and Solution are mixed

More Ideas

- Hopefully, you are becoming familiar with Minizinc …

- Can you personalize the model by considering …
  - 1) I must visit city X before Y.
  - 2) I do not visit Y right after X.

Model and Dataset:
http://cs.unibo.it/t.liu/mzn/

Suppose an investment problem (1)

Project name	Value	Budget (k)	Personnel
Ischia	6000	35	5
Speltra	4000	34	3
Hotel	1000	26	4
Restaurant	1500	12	2
ContoA	800	10	2
ContoB	1200	18	2
Scuola	2000	32	4
Dago	2200	11	1
Lago	900	10	1
small	3800	22	5
Iper	2900	27	3
Bivio	1300	28	2
Tela	800	16	2
Idro	2700	29	4
Batment	2800	22	3

- 225 k available
- 28 persons
- max 9 projects

Solution available at
http://cs.unibo.it/t.liu/mzn/

Suppose an investment problem (2)

Project name	Value	Budget	Personnel	Not With	Require
Ischia	6000	35	5	10	-
Speltra	4000	34	3	-	-
Hotel	1000	26	4	-	15
Restaurant	1500	12	2	-	15
ContoA	800	10	2	6	-
ContoB	1200	18	2	5	-
Scuola	2000	32	4	-	-
Dago	2200	11	1	-	7
Lago	900	10	1	-	-
small	3800	22	5	1	-
Iper	2900	27	3	15	-
Bivio	1300	28	2	-	-
Tela	800	16	2	-	2
Idro	2700	29	4	-	2
Batment	2800	22	3	11	-

- several projects cannot bet selected with others
- and some must be selected together

Problems solved with Minizinc

- factory scheduling (JSSP)
- vehicle routing (VRP)
- packing problems (NumPart and BinPack)
- timetabling (exams, lectures, trains)
- configuration and design (hardware)
- workforce management (call centres, etc)
- car sequencing (assembly line scheduling)
- supertree construction (bioinformatics)
- network design (telecoms problem)
- gate arrival (at airports)
- logistics (Desert Storm an example)
- aircraft maintenance schedules
- aircraft crew scheduling (commercial airlines)
- air cover for naval fleet

Summary

- AI => Problem Solving => CSP
- Constraint Programming Approach
  - problem and solution are separate
  - declarative programming
- Minizinc Paradigm
  - data types
  - aggregation functions
  - constraints
  - objective definition
- Minizinc exercises

How do you describe a problem to your friends?
describe it to a computer!

More resources

Minizinc Tutorial:
http://www.minizinc.org/downloads/doc-latest/minizinc-tute.pdf

Coursera:
https://www.coursera.org/learn/basic-modeling

More Minizinc Examples:
https://github.com/hakank/hakank/tree/master/minizinc
https://github.com/MiniZinc/minizinc-examples

Credits: Hakank, Chiarandini, Peter van BeeK