

# Constraint Programming

Alessandro Hill

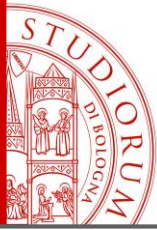
rev. 1.0(AH) – 2024



# Topics

---

- Concepts
- Solvers
- Applications



# Concepts

1. Decision variables are represented by domains
2. Constraints
3. Objective function

→ **Similarities with Mathematical Programming!**

*Special cases:*

- SAT Solving
- CP over finite domains

**Constraint Solving**

NorthAmerica-draft Home Tutorials Community Books Solvers Conferences Forum

FAO

**Constraint Solving**

Constraint programming is a programming paradigm where relations between variables can be stated in the form of constraints. Constraints differ from the common primitives of other programming languages in that they do not specify a step or sequence of steps to execute but rather the properties of a solution to be found.

**Why Constraint Solving?**

"Constraint Programming represents one of the closest approaches computer science has yet made to the Holy Grail of programming: the user states the problem, the computer solves it."

[E. Freuder]

**Applications of Constraint Solving**

Real-time applications that take advantage of constraint programming techniques have been increasing by leaps and bounds every year for more than a decade now. A lot of areas such as manufacturing, financial services, telecommunications, defense etc have been employing constraint and logic programming.

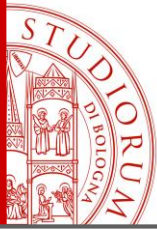
**CP and combinatorial optimization in circuit verification**

One of the three features of Constraint programming is modeling. A modeling language allows a software analyst to specify requirements of a software system on an architectural level. In designing and verifying VLSI circuits problem modeling is effectively achieved with constraints. Functional verification has become quite difficult these days what with

**About this site**

Site maintained by Martine Ceberio and her

<http://www.constraintsolving.com>

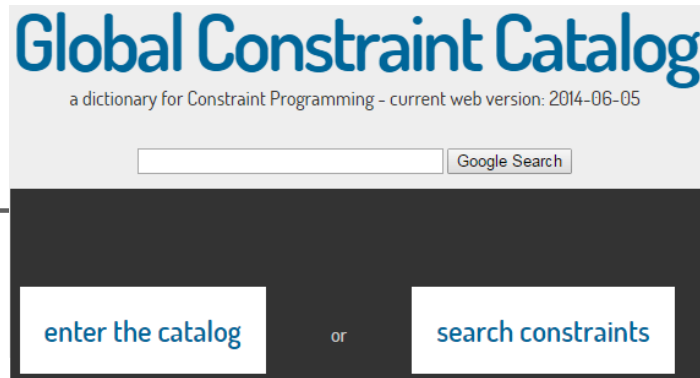


# Domains

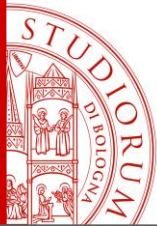
- **Domains [of variables] can be Boolean, integer, numerical.**  
(commonly finite domain)
- **Scheduling-oriented variables are provided**  
(e.g., Interval variable representing a job/task)
- **Finite domains are internally represented by intervals.**  
(e.g.,  $x=[1,5]=\{1,2,3,4,5\}$ ) **→ Leads to time-independent models!**
- **Explicit representation when “holes” appear.**  
(e.g.,  $x'=\{1,3,4,10\}$ )



# Constraints



- **Constraints link variables (domains) – not necessarily linearly.**
- **They can be defined individually...**  
(e.g.,  $x^2 > 2$ )
- **and there are many pre-defined.**  
(e.g., `ALL_DIFFERENT(x,x',x'')`)
- **A constraint needs a propagator, which identifies constraint implications on other domains.**  
(re-apply after domain change)

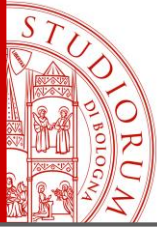


# Constraints (cont.)

1	1	1	1	1
2				2
3	3	3		
4		4	4	4
5				5
A	B	C	D	E
<del>1</del>	1	1	1	<del>1</del>
2				2
<del>3</del>	3	3		
<del>4</del>		4	4	<del>4</del>
5				5

**ALL\_DIFFERENT(A,B,C,D,E)**

Uses matching theory








# Objectives

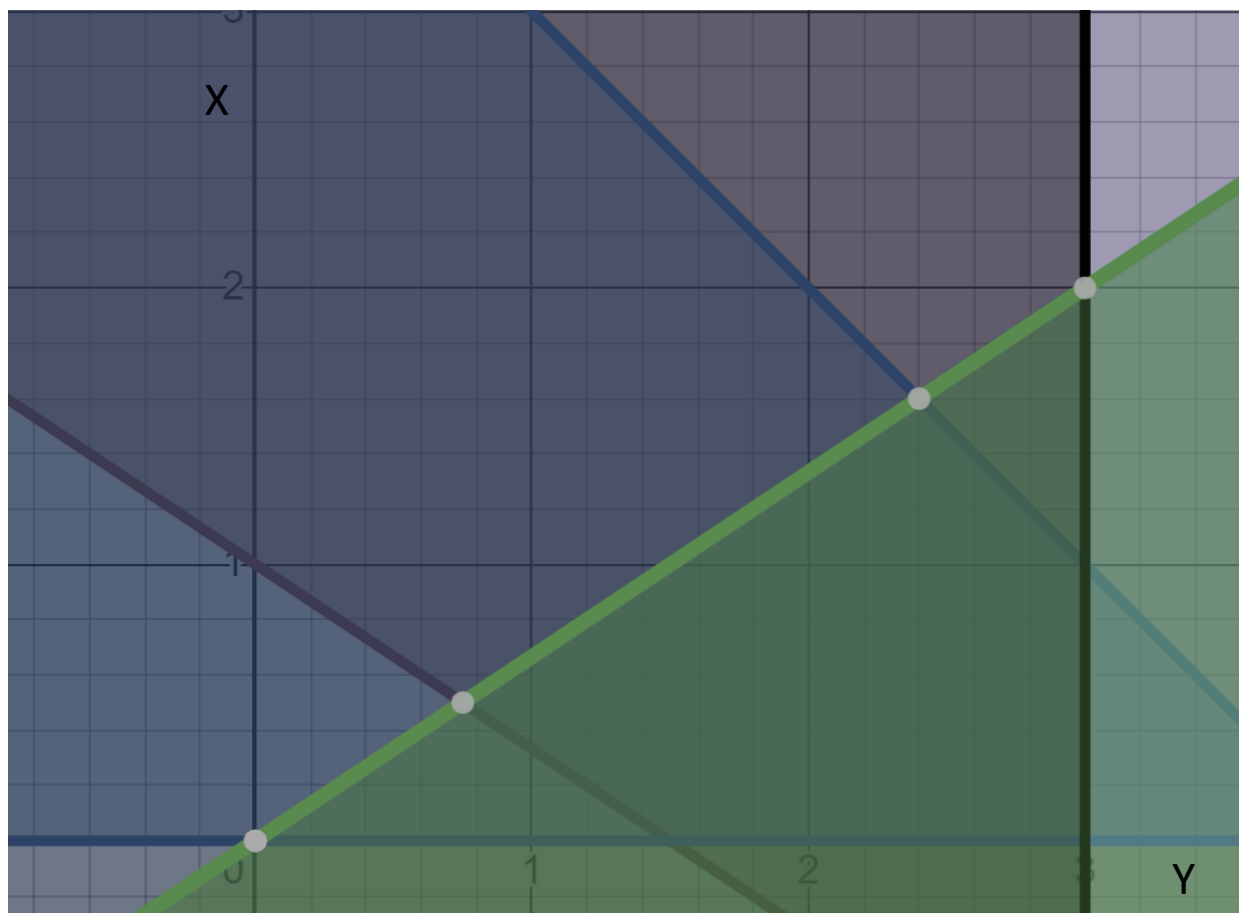
---

- **Build any expression**  
(e.g., use MAX/MIN, IF/THEN, POWER)
- **Access interval variable properties**
- **Some carefulness with expressions needed**

# Branch & Bound

$$\begin{array}{ll}
 \text{Min} & 2X + Y \\
 \text{s.t.} & 2X + 3Y \geq 3 \\
 & X + Y \leq 4 \\
 & 2X - 3Y \geq 0 \\
 & X \leq 3 \\
 & X, Y \text{ intere}
 \end{array}$$

- 1   $2x + 3y \geq 3$
- 2   $x + y \leq 4$
- 3   $2x - 3y \geq 0$
- 4   $y \geq 0$
- 5   $x \leq 3$





# Modelling languages

**FlatZinc:** Standard low-level modelling language

**MiniZinc:** High-level modelling language

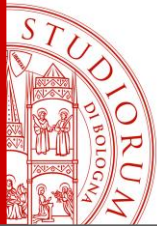
```
% Colouring Australia using nc colours
int: nc = 3;

var 1..nc: wa;   var 1..nc: nt;   var 1..nc: sa;   var 1..nc: q;
var 1..nc: nsw;  var 1..nc: v;    var 1..nc: t;

constraint wa != nt;
constraint wa != sa;
constraint nt != sa;
constraint nt != q;
constraint sa != q;
constraint sa != nsw;
constraint sa != v;
constraint q != nsw;
constraint nsw != v;
solve satisfy;

output ["wa=", show(wa), "\t nt=", show(nt),
        "\t sa=", show(sa), "\n", "q=", show(q),
        "\t nsw=", show(nsw), "\t v=", show(v), "\n",
        "t=", show(t), "\n"];
```





# Solvers

## Key ingredients:

- Branch and bound
- Domain filtering (Propagation)

## Also:

- Impact-based branching rules
- No-good learning
- Automated tuning
- Linear programming
- Neighborhood search
- Machine learning
- Probing

**Competition:** [MiniZinc Challenges](#)  
(yearly, 2008-today)

**Commercial:**  
**IBM ILOG CP Optimizer**  
(research: free)

**Free:**  
**GECODE** ([www.gecode.org](http://www.gecode.org))  
**CHUFFED** (SAT-hybrid)  
**CHOCO** (java, open-source,  
[www.choco-solver.org](http://www.choco-solver.org))

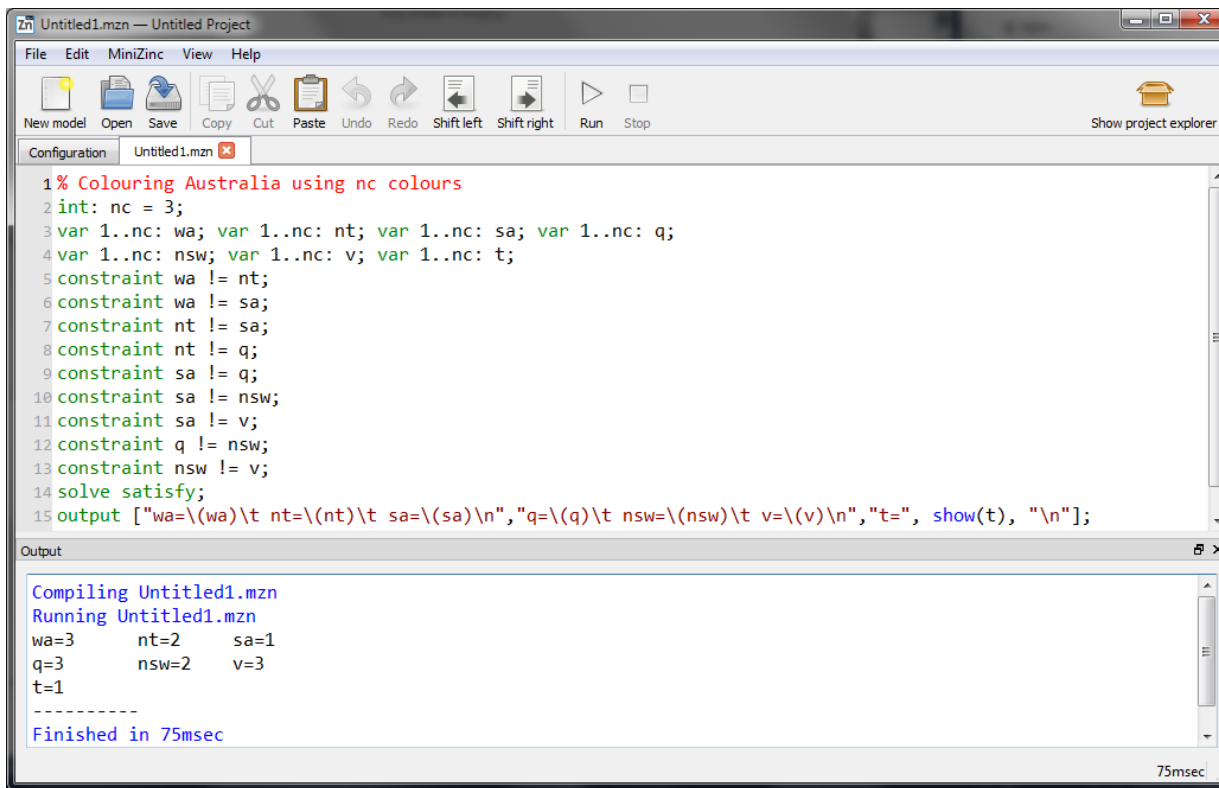
...

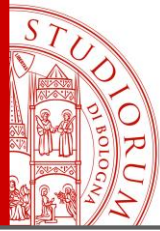


# MiniZinc Tool



- User interface
- Provides modelling language
- Translates model into FlatZinc
- Interfaces various solvers
- Free





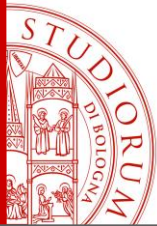
# IBM ILOG CP Optimizer

- **Interfaces:** C/C++, Java, Python, OP (decent APIs)
- **Focus on**
  - **Integer models** (allows floating point)
  - **Scheduling**
- **Rich constraint library** (e.g., packing, assignments, resources, precedences, set-up costs/times)
- **Automated tuning**
- **Conflict analysis**

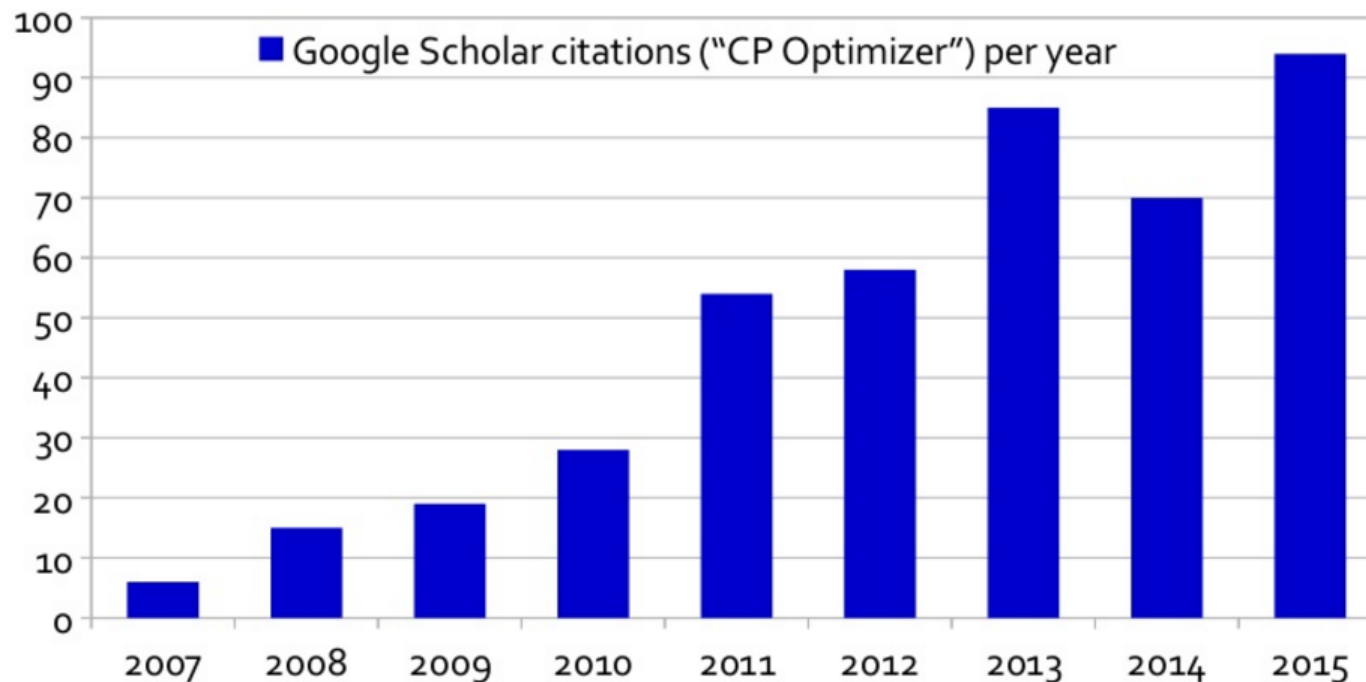
CP Optimizer model for RCPSP:

```
dvar interval a[i in Tasks] size i.pt;  
  
cumulFunction usage[r in Resources] =  
    sum (i in Tasks: i.qty[r]>0) pulse(a[i], i.qty[r]);  
  
minimize max(i in Tasks) endOf(a[i]);  
subject to {  
    forall (r in Resources)  
        usage[r] <= Capacity[r];  
    forall (i in Tasks, j in i.succs)  
        endBeforeStart(a[i], a[j]);  
}
```

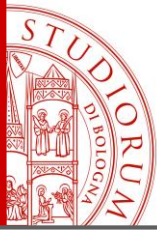
IBM



# IBM ILOG CP Optimizer (cont.)



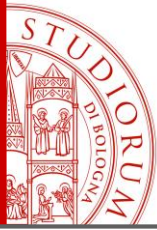
© 2016 IBM Corporation



# Successful applications

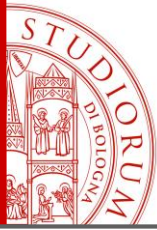
- **Scheduling**
- **Timetabling**
- **Product configuration**
- ...

Bench index	Problem type	MRD	# Imp. UBs / # Instances
1	Trolley	-10.2%	15/15
2	Hybrid flow-shop	-11.3%	19/20
3	Job-shop w/ E/T	-6.2%	41/48
4	Air traffic management	-7.0%	1/1
5	Max. quality RCPSP	-2.7%	NA/3600
6	Flow-shop w/ E/T	-1.1%	5/12
7	RCPSP w/ E/T	-2.1%	16/60
8	Cumulative job-shop	-0.1%	15/86
9	Semiconductor testing	-0.3%	7/18
10	Single proc. tardiness	0.3%	0/20
11	Open-shop	0.3%	0/28
12	MaScLib single machine	0.6%	0/60
13	Shop w/ setup times	0.4%	3/15
14	RCPSP	1.2%	2/600
15	Air land	0.0%	0/8
16	Parallel machine w/ E/T	1.6%	4/52
17	Job-shop	1.9%	0/33
18	Flow-shop	0.9%	4/22
19	Flow-shop w/ buffers	3.9%	11/30
20	Single machine w/ E/T	7.4%	0/40
21	Aircraft assembly	8.7%	0/1
22	Common due-date	6.8%	4/20



# CP: Pros

- Easy optimization **model setup** → **Model & Run**
- Modeling of **complex constraints**
- **Scheduling-oriented**
- Various strong **solvers** available
- **Efficient** for problems with complex combinatorial structure (up to a certain size)
- Finds **feasible solutions** fast
- Detects **infeasibility** fast

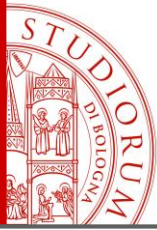


# CP: Cons

---

- Only **little structural information** for hard large-scale problems
- No direct solution **quality guarantee**
- **Black box behavior**
- **Efficient problem formulation is difficult**





# Some resources

---

- **IBM ILOG CP Optimizer for Detailed Scheduling Illustrated on Three Problems**, Philippe Laborie, 2009
- **CP Optimizer Walkthrough**, Paul Shaw, 2013
- **Modeling and Solving Scheduling Problems with CP Optimizer**, Philippe Laborie, 2014
- **An Introduction to CP Optimizer**, Philippe Laborie, 2016
- **Solver challenge:** [www.minizinc.org/challenge.html](http://www.minizinc.org/challenge.html)
- **Global Constraint Catalogue:** <http://sofdem.github.io/gccat>
- [www.constraintsolving.com](http://www.constraintsolving.com)
- **MiniZinc tool:** [www.minizinc.org](http://www.minizinc.org)