Session 1
Constraint Programming: Problems and Models

Daniel Diaz
Salvador Abreu

with contributions by Philippe Codognet

Course topics and structure

Problems and Models

Constraint Modeling and Programming

Global Constraints

Tools for Constraint Programming

Incomplete methods and Metaheuristics

Session pattern

Mixed: lectures, theory, exercises

Mixed: tutorials on tools and applications to problem-solving

Slides & other stuff (GDrive):

bit.ly/3GvB4sp

Some Context

Declarative programming

Express relations, not actions

Kowalski's equation:

Algorithms = Logic + Control

Argues that the logic describes **what** must be true and the control *may* be used to add information as to **how** to reach that.

Declarative programming

First-order logic, resolution, unification (Herbrand terms), some control hacks

Prolog

Issues with unification, free (unbound) variables and negation (as failure). Ways of dealing with it:

- make sure all variables are ground (not variable anymore)
- delay goals until variables are sufficiently specified

One step further would be to let variables “carry the disjunction”

CLP(*)

Declarative programming

By keeping just:

- The variables,
- Their domains and
- The relations that bind them (the constraints)

We get the topic of this course: **constraint programming**.

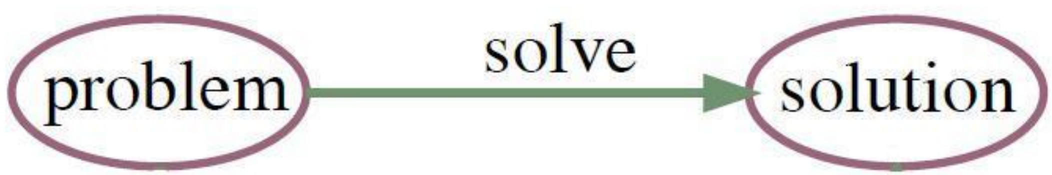
Problems and Models

Problems in the real world

A problem situation may be described by:

- A set of **general rules** (world axioms)
- Entities that represent the **state** of the system
- Specific **limits** on those entities
- Things that **must hold** true, involving the same entities and possibly some external ones

Ideally:

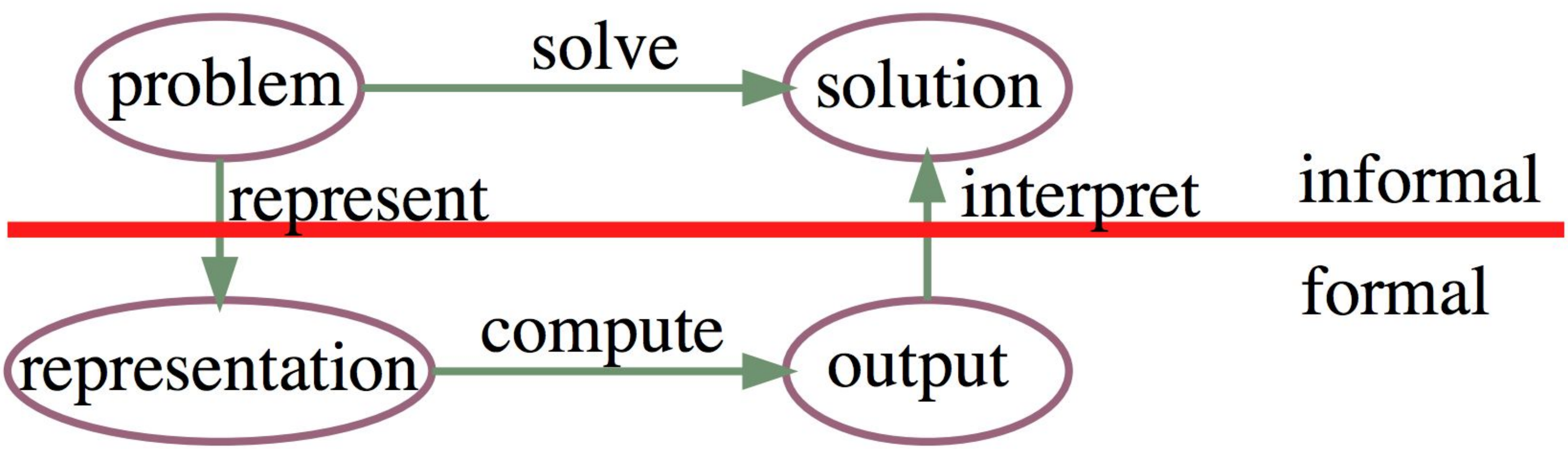


Models

Emerge after consideration of **multiple** concrete problem cases

Allow for solving problems in an **abstract** fashion

In practice



[from Poole & Mackworth 2010]

Take a real, concrete, problem situation

Let's pack a bunch of **boxes...** into a container, like this van:

Consider another -apparently similar- problem

A different set of boxes...

...into another van

After a few cases (instances)

We see **patterns** emerge (insights)

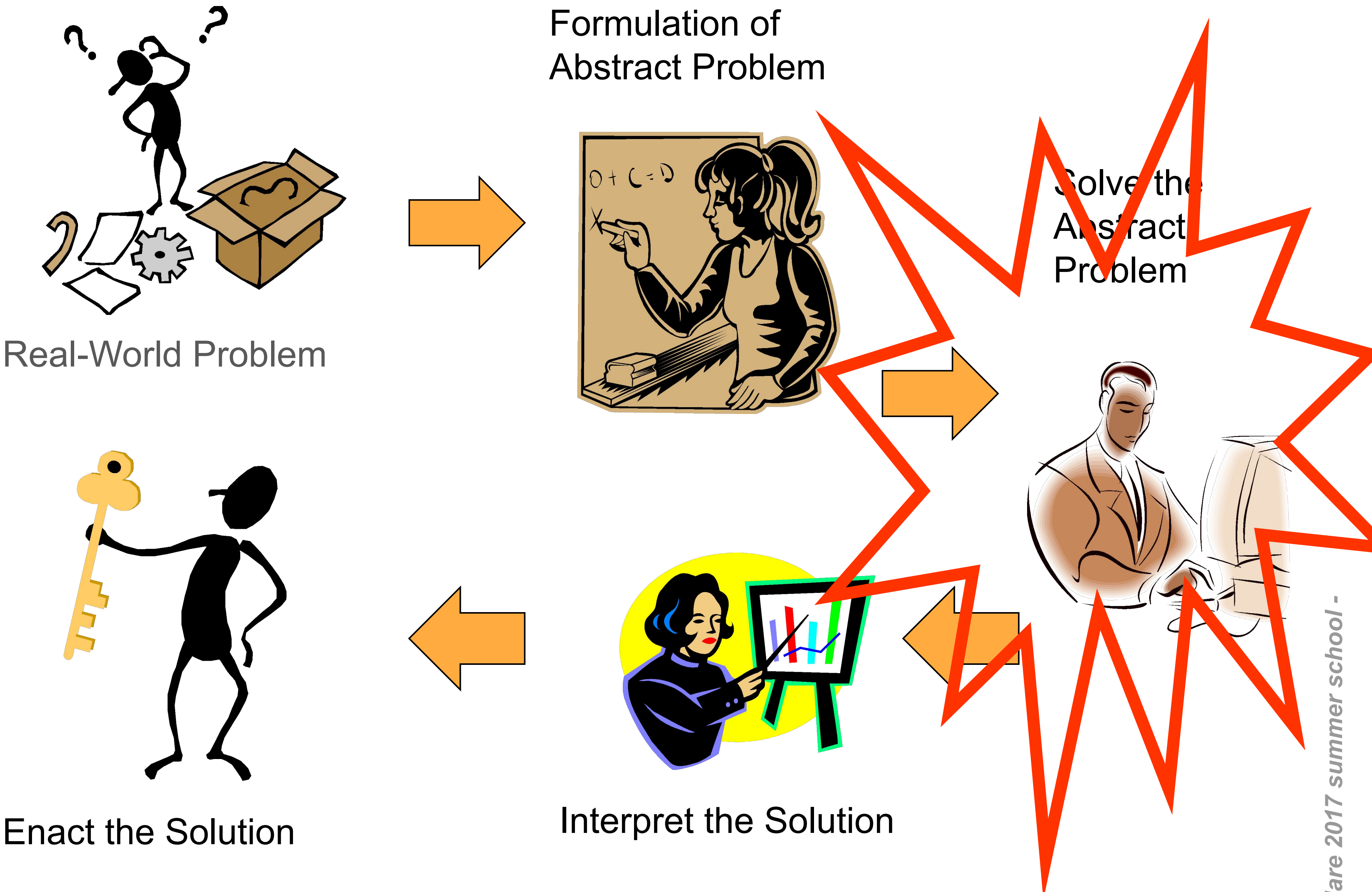
They will guide the abstract problem-solving process, i.e. the design of a **model**

For example,

- No two distinct boxes may occupy the same location
- All boxes must be placed inside the van
- ...

This leads us to a model for the problem, e.g. as a ***set of variables*** and a ***system of relations*** among the said variables.

Conceptually close to *systems of equations*, only it's more general.



[From A. Løkketangen]

Problem Representation

We want a representation to be:

- **Rich** enough to express the knowledge needed to solve the problem.
- As **close** as possible **to the problem**: compact, natural and maintainable.
- Amenable to **efficient** computation:
  - Able to express features of the problem that can be exploited for computational gain
  - Able to trade off accuracy and computation time or space
- Able to be acquired from people, data and past **experiences**.

[from Poole & Mackworth 2010]

Modeling

- We have to represent (model) the problem
  … in a modeling language
- and to have a notion of “solution”
  by reduction / simplification of the problem
- may use the whole mathematics toolbox:
  Logic, polynomial equations, differential equations, …
- key: model must be (efficiently) executable by a computer
- Modeling Language or Modeling Paradigm with associated computation algorithm(s)

Modeling

We may be aware that some forms are better than others.

- There is no unique formulation
- Some models may incorporate expert knowledge
- Some models may be more appropriate for describing (and solving) particular problems, but not others
- Sometimes it may be useful to combine different, otherwise sufficient, models (hybrid models)

What is a Solution?

- Formula to be satisfied or set of conditions to be achieved
- Issue: Unique solution? Several solutions?
- Issue: Some solutions are better than others?
    - Do we care?
    - Optimal solution?
- Issue: Sometimes too hard to find
    - Seek approximate solution
- Issue: Solution quality vs. time spent looking for it:
    - Anytime algorithms

Crypto-arithmetic:

$$\begin{array}{cccccc} & & S & E & N & D \\ + & & M & O & R & E \\ \hline = & M & O & N & E & Y \end{array}$$

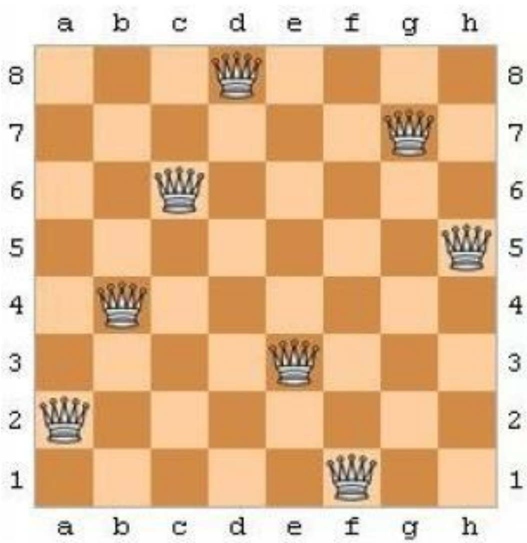
Mathematical and Logic Puzzles

boolean formulas (SAT)

Magic squares:

"Melencolia", Albrecht Dürer, 1514

N-Queens



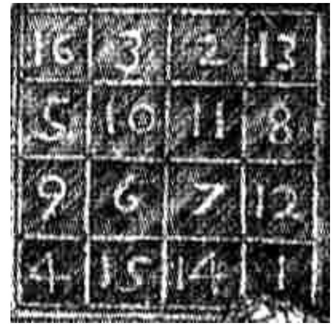
Sudoku

5	3			7				
6			1	9	5			
	9	8					6	
8				6				3
4			8		3			1
7				2				6
	6					2	8	
			4	1	9			5
				8			7	9

1μs/config: 10^{-6}s; 10^{13}s

Take Dürer's magic square:

16^{16} combinations $>10^{19}$…



Benjamin Franklin's Magic Square

52	61	4	13	20	29	36	45
14	3	62	51	46	35	30	19
53	60	5	12	21	28	37	44
11	6	59	54	43	38	27	22
55	58	7	10	23	26	39	42
9	8	57	56	41	40	25	24
50	63	2	15	18	31	34	47
16	1	64	49	48	33	32	17

52	73	7	64	21	15	35	98	99	44
91	58	25	6	66	19	41	79	84	43
31	60	62	11	5	26	29	68	36	74
100	40	2	3	20	61	65	86	24	88
4	38	14	76	87	71	16	80	53	97
34	22	85	89	82	18	77	69	47	56
8	9	57	67	50	78	42	10	96	70
90	1	13	39	46	33	81	49	27	59
83	30	48	12	51	45	55	92	28	23
95	93	63	32	72	17	94	75	37	54

Consider 10x10 magic square

- naïve search space = $100^{100} = 10^{200}$
- "much better" with permutations:

  100! ≈ *only* 10^{158}

For a 400x400 square:

- We'll have 400x400! = 160000! ≈ 10^{763175}

There are methods which can solve 400x400 in less than one hour CPU-time

Example: Simple Scheduling

What is the **minimal time** to build the house?
How to schedule the tasks to achieve the goal in minimal time (satisfying the precedence constraints)?

Task	Description	Duration	Predecessors
A	Masonry	7	none
B	Carpentry	3	A
C	Plumbing	8	A
D	Ceiling	3	A
E	Roofing	1	B
F	Painting	2	D
G	Windows	1	E
H	Facade	2	E, C
I	Garden	1	E, C
J	Moving in	1	G, H, I, F

Modeling

Variables: **start date of each task**

Constraints:

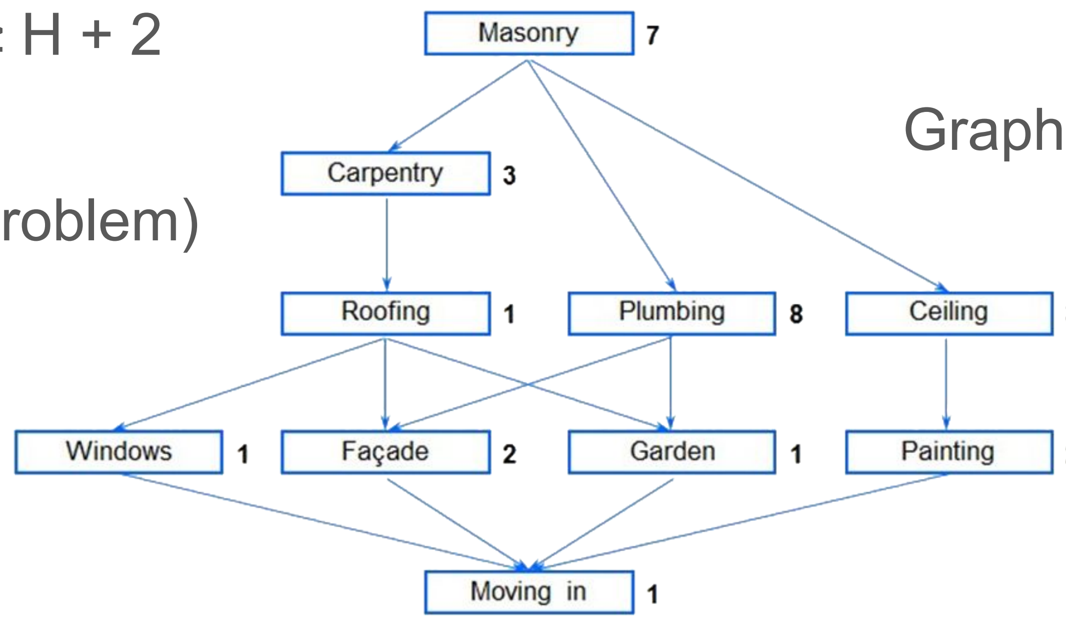
$B \geq A + 7$ $C \geq A + 7$ $D \geq A + 7$
$E \geq B + 3$ $F \geq D + 3$ $G \geq E + 1$
$H \geq E + 1$ $H \geq C + 8$ $I \geq E + 1$
$I \geq C + 8$ $J \geq G + 1$ $J \geq H + 2$
$J \geq I + 1$ $J \geq F + 2$

Minimise J (optimization problem)

Solution: 17 days:

A=0 B=C=D=7 E=F=10
G=11 H=I=15 J=**17**

Task	Description	Duration	Predecessors
A	Masonry	7	none
B	Carpentry	3	A
C	Plumbing	8	A
D	Ceiling	3	A
E	Roofing	1	B
F	Painting	2	D
G	Windows	1	E
H	Facade	2	E, C
I	Garden	1	E, C
J	Moving in	1	G, H, I, F



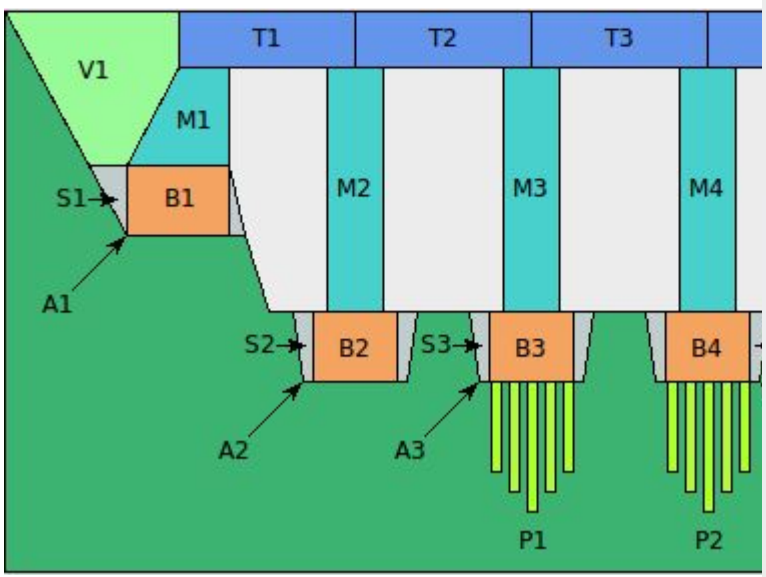
Graph

Example: Disjunctive S

Build a bridge in minimal tim

and resource requirements (

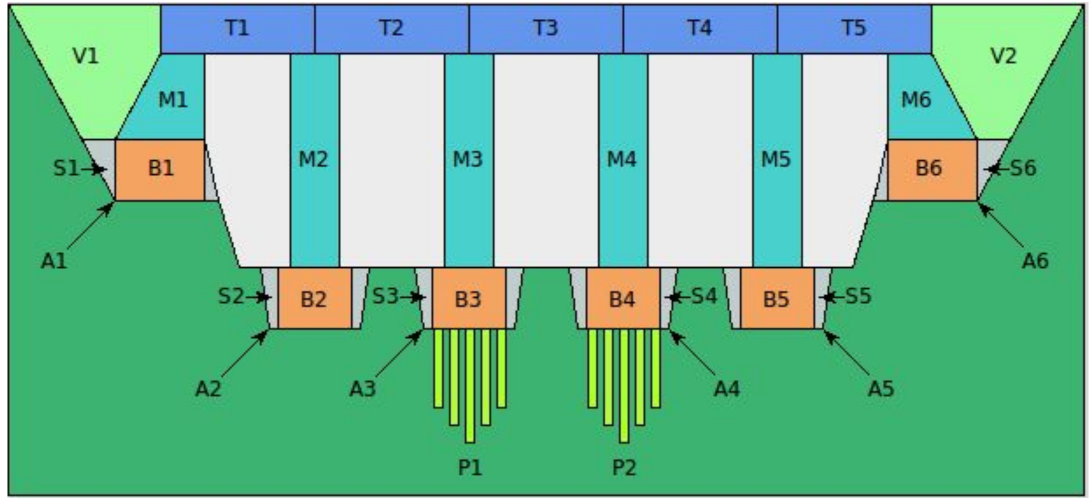
cannot overlap).

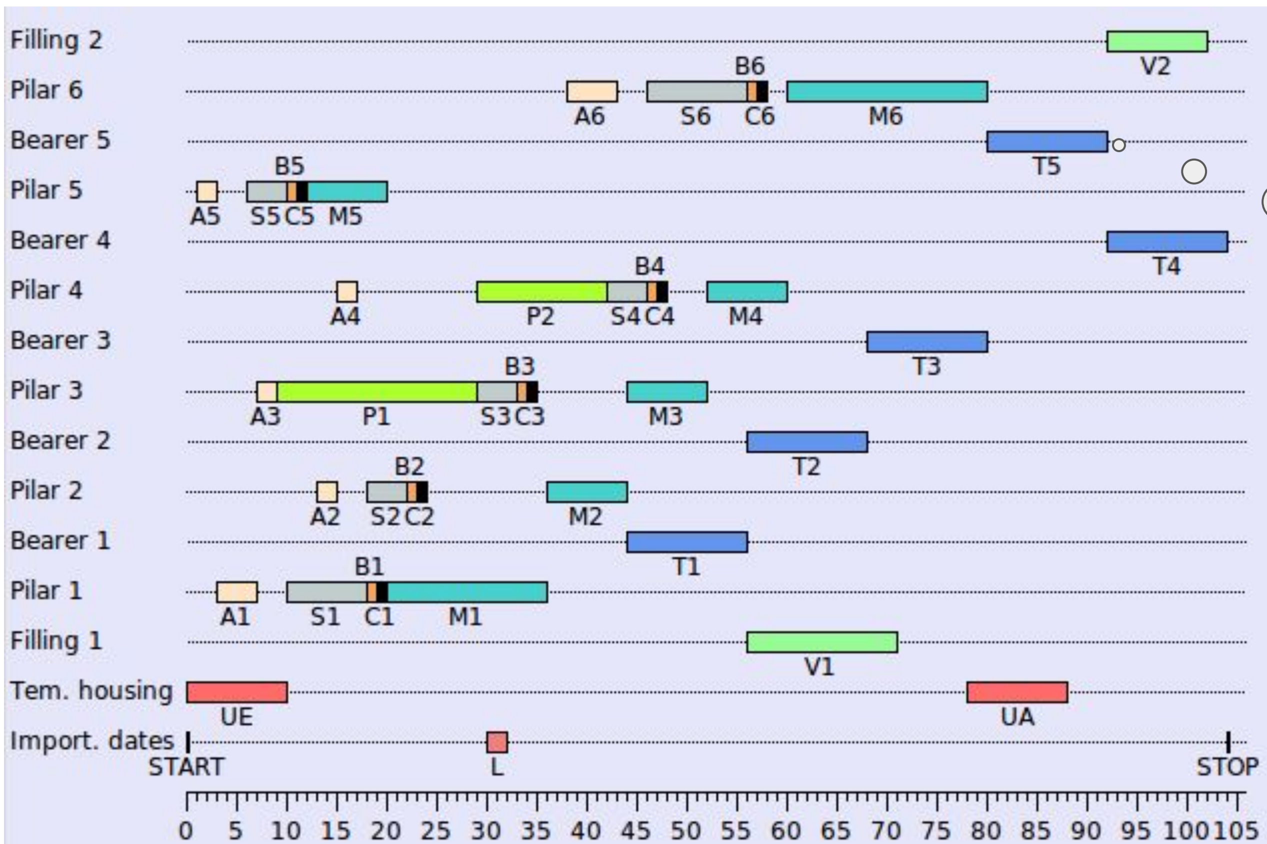


Task	Description	Dur.	Precedences	Resource
START	Beginning of the project	0		
A1	Excavation pillar 1 (abutment)	4	START	Excavator
A2	Excavation pillar 2	2	START	Excavator
A3	Excavation pillar 3	2	START	Excavator
A4	Excavation pillar 4	2	START	Excavator
A5	Excavation pillar 5	2	START	Excavator
A6	Excavation pillar 6 (abutment)	5	START	Excavator
P1	Extra foundation pillar 3	20	A3	Pile driver
P2	Extra foundation pillar 4	13	A4	Pile driver
UE	Erection of temporary housing	10	START	
S1	Formwork pillar 1	8	A1	Carpentry
S2	Formwork pillar 2	4	A2	Carpentry
S3	Formwork pillar 3	4	P1	Carpentry
S4	Formwork pillar 4	4	P2	Carpentry
S5	Formwork pillar 5	4	A5	Carpentry
S6	Formwork pillar 6	10	A6	Carpentry
B1	Concrete foundation pillar 1	1	S1	Concrete mixer
B2	Concrete foundation pillar 2	1	S2	Concrete mixer
B3	Concrete foundation pillar 3	1	S3	Concrete mixer
B4	Concrete foundation pillar 4	1	S4	Concrete mixer
B5	Concrete foundation pillar 5	1	S5	Concrete mixer
B6	Concrete foundation pillar 6	1	S6	Concrete mixer
C1	Concrete setting time pillar 1	1	B1	
C2	Concrete setting time pillar 1	1	B2	
C3	Concrete setting time pillar 2	1	B3	
C4	Concrete setting time pillar 3	1	B4	
C5	Concrete setting time pillar 4	1	B5	
C6	Concrete setting time pillar 6	1	B6	
M1	Masonry work pillar 1	16	C1	Bricklaying
M2	Masonry work pillar 2	8	C2	Bricklaying
M3	Masonry work pillar 3	8	C3	Bricklaying
M4	Masonry work pillar 4	8	C4	Bricklaying
M5	Masonry work pillar 5	8	C5	Bricklaying
M6	Masonry work pillar 6	20	C6	Bricklaying
L	Delivery of preformed bearers	2		Crane
T1	Positioning preformed bearer 1	12	M1 M2 L	Crane
T2	Positioning preformed bearer 2	12	M2 M3 L	Crane
T3	Positioning preformed bearer 3	12	M3 M4 L	Crane
T4	Positioning preformed bearer 4	12	M4 M5 L	Crane
T5	Positioning preformed bearer 5	12	M5 M6 L	Crane
UA	Removal of temporary housing	10		
V1	Filling 1	15	T1	Bulldozer
V2	Filling 2	10	T5	Bulldozer
STOP	End of the project	0	T2 T3 T4 V1 V2 UA	

A solution

Optimal end time = **103**





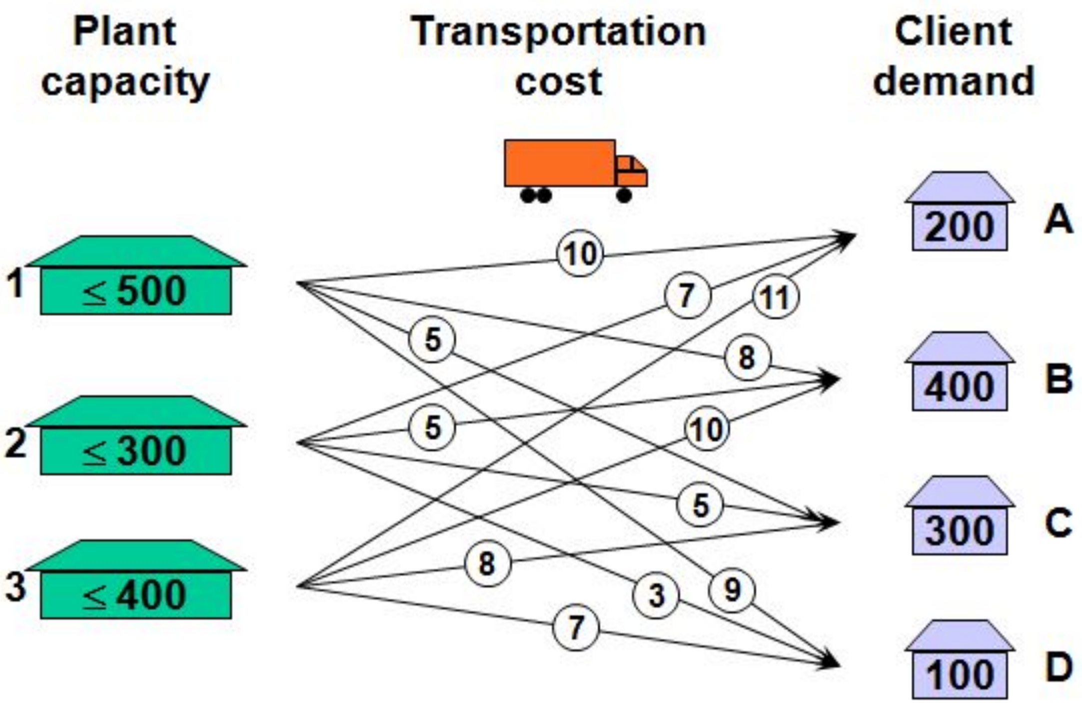
Color designates tasks (and also resources)

Example: Resource Allocation

Constraints:

A1 + A2 + A3 = 200
B1 + B2 + B3 = 400
C1 + C2 + C3 = 300
D1 + D2 + D3 = 100

A1 + B1 + C1 + D1 ≤ 500
A2 + B2 + C2 + D2 ≤ 300
A3 + B3 + C3 + D3 ≤ 400



Minimise the total cost:

10*A1 + 7*A2 + 11*A3
+ 8*B1 + 5*B2 + 10*B3
+ 5*C1 + 5*C2 + 8*C3
+ 9*D1 + 3*D2 + 7*D3

Solution: minimal cost = **6600**

A3=200, B1=200, B2=200, C1=300, D2=100
(all others vars = 0)

What do all these problems have in common?

- Variables take values over a **finite subset of integers**
- Clear constraints (easy to state in natural language)
- Large search space
- Well-identified “goal”
  Notion of solution is easy to define (declaratively)
- But we don’t know how to compute it!
- No algorithm to build a solution (incrementally or otherwise)
- Hence:
  - Need to explore (search) the possible solution space (all possibilities)
  - Either exhaustively or in an “intelligent”, “guided” manner

Methods covered in these lectures

- Constraint Programming
  - Represent problem by variables and constraints
  - Use specific solving algorithms to speed search up
  - Use “complete” solvers
- Modeling with constraints
  - Constraints over finite domains
  - Useful global constraints
- Constraint Optimisation vs. Constraint Satisfaction

Methods lightly approached in these lectures

- *Local Search and Metaheuristics*
    - *Evaluation function to check if state is “good” or not*
    - *Optimization of the evaluation function*
    - *Sacrifice completeness to (hopefully) quickly find a solution*
- *SAT (solvers for boolean-variable formulations)*
- *Modeling Languages*

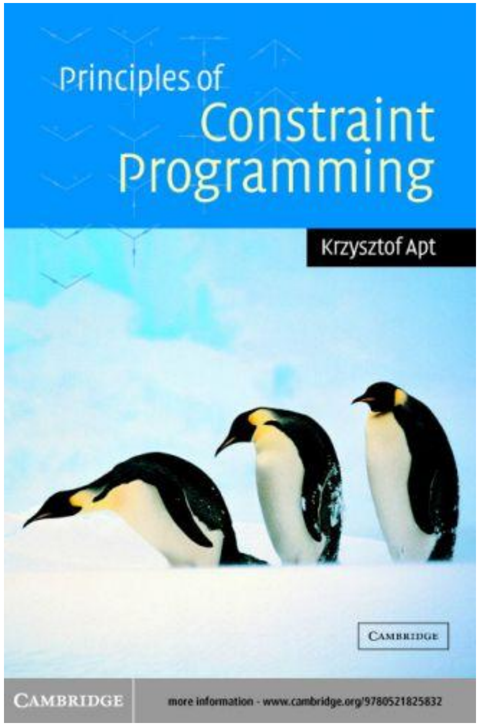
Methods **NOT** covered in this lecture series

- Graph Search
  - Representation of states and transitions/actions between states → graph
  - Explored explicitly or implicitly
- Numerical Optimization Methods
  - For continuous domains & twice differentiable functions
- Linear Optimization methods
  - For linear constraints & rational domains
  - Simplex algorithm, Interior Point Methods
  - (Mixed-)Integer Programming, cutting plane methods
- Dynamic Programming
  - Decomposable problem, recursive relation
- … :)

Useful Resources

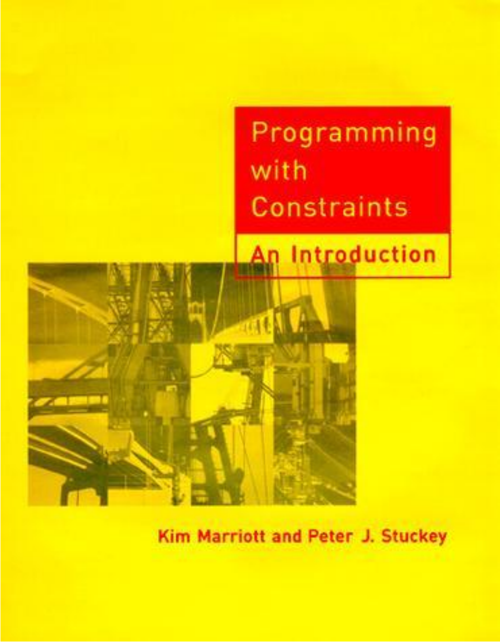
K. Apt

Principles of Constraint Programming,
Cambridge University Press 2003



K. Marriott and P. J. Stuckey

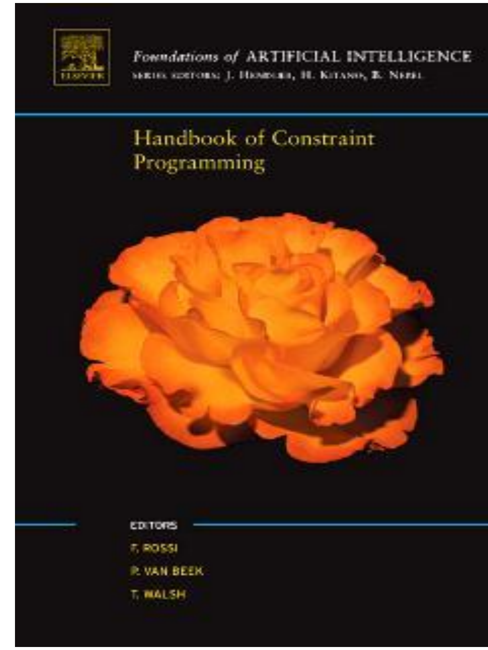
Programming with Constraints:
An Introduction
MIT Press, 1998



F. Rossi, P. Van Beek and T. Walsh

Handbook of Constraint Programming,

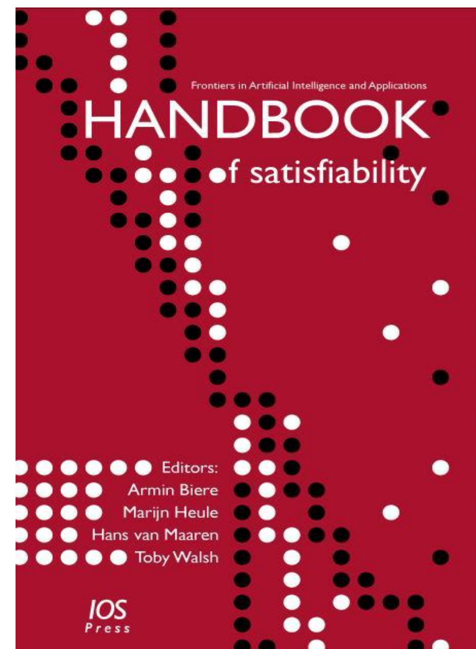
Elsevier 2006



A. Biere, M. Heule, H. van Maaren & T. Walsh

Handbook of Satisfiability,

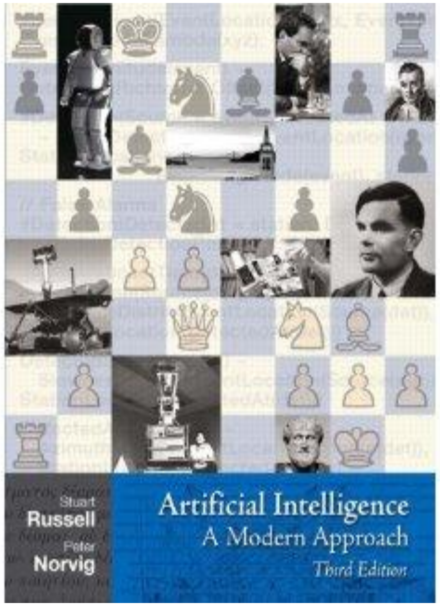
IOS Press 2009



S. Russell & P. Norvig

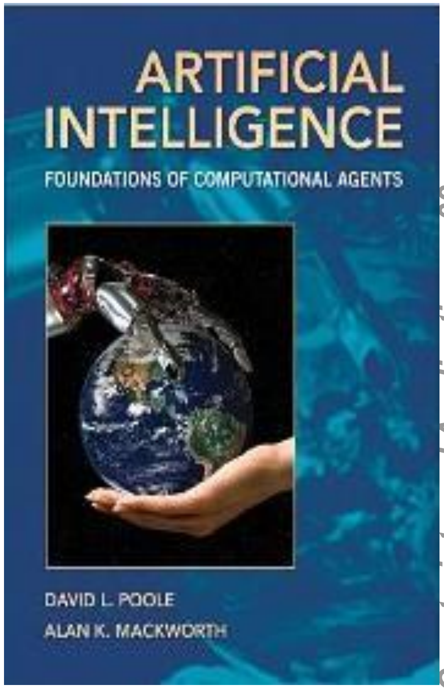
Artificial Intelligence: A Modern Approach, 3rd edition, Pearson 2010

http://aima.cs.berkeley.edu/



D. Poole & A. Mackworth, Artificial Intelligence: Foundation of Computational Agents, Cambridge University Press 2010

http://artint.info/



F. Rothlauf

Design of Modern Heuristics,

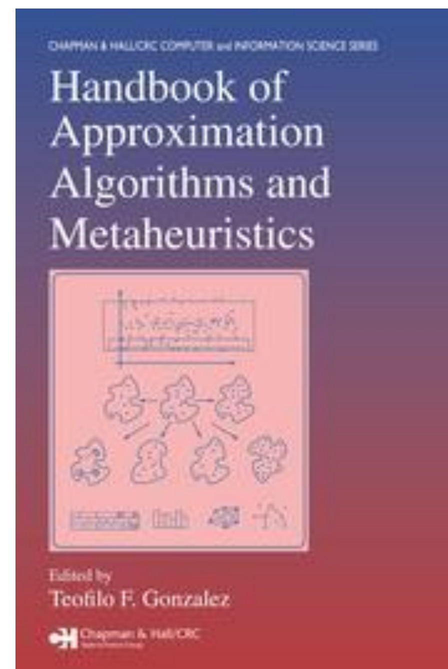
Springer Verlag 2011

T. Gonzalez

Handbook of Approximation

Algorithms and Metaheuristics,

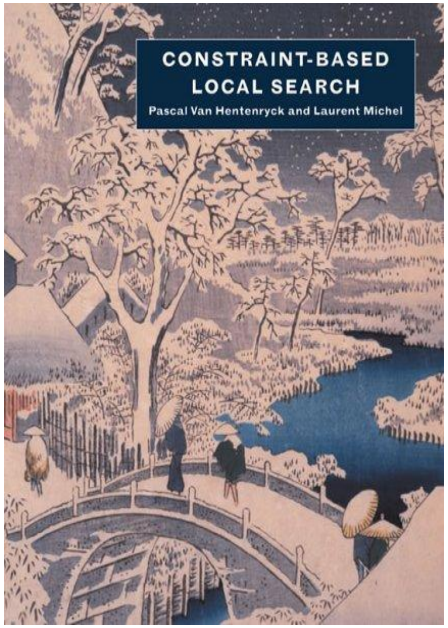
Chapman & Hall/CRC 2010



P. Van Hentenryck and L. Michel

Constraint-based Local Search

MIT Press 2005



Exercise: **ad-hoc** problem solving

Implement in **Java** (or Python) a solution for these problems:

1) TWO + TWO = FOUR (*)
2) N-queens (**), Try N=8, 10, 11, 12,...
3) Magic squares (**). Try 3x3, can you solve 4x4? Larger? Why?
4) ~~Square packing (***)~~
5) ~~Stable marriage (****)~~

Questions:

1) Is it ***easy*** to program?
2) Can it be ***generalized*** (for arbitrary size problems)?
3) Is it ***efficient***? Does it ***scale well*** (in terms of execution time)?
4) Measure execution time for different size problems