Problem Solving in Artificial Intelligence

4810-1208

Philippe Codognet

SHORT
INTRODUCTION
TO THE
COURSE TOPICS

Lecturer

- Philippe CODOGNET
  - Professor at University Pierre & Marie Curie (Paris)
  - Co-Director of the Japanese-French Laboratory for Informatics (JFLI), joint lab between CNRS, UPMC, University of Tokyo, Keio University, N.I.I.
- Office

  Dept. of Computer Science, Prefab A, room 204
- Email

  codognet@is.s.u-tokyo.ac.jp
- Slides

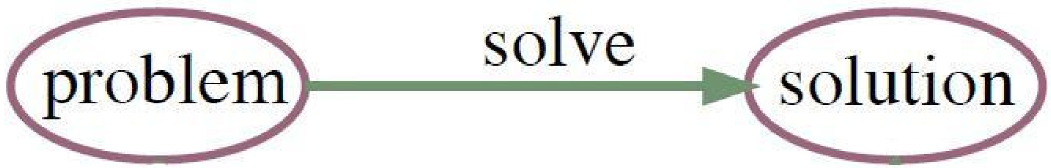
  webia.lip6.fr/~codognet/PSAI

Faculty of Engineering Bldg. 3
·Information &
Communication Engineering
Faculty of Science Bldg. 7
·Computer Science
Ikenohata Gate
0 50 100 200 300(m)
JFLI
Offices 204/205/206/213
Nezu Sta.
Kototoi St.
Faculty of Engineering Bldg. 2
·Mechano-Informatics
·Postgraduate Section II, Office of International Relations (Graduate School of IST)
·Office of International Relations
Faculty of Engineering Bldg. 6
·Mathematical Informatics
·Information Physics & Computing
Asano South Gate
Asano Main Gate
Yayoi Gate
Todai-konai Bus Stop
Kasuga St.
Tatsuoka Gate
Todaimae Sta.
Main Gate to Faculty of Agriculture
Main Gate
Red Gate
Hongo St.
Hongo-Sanchome Sta.
Faculty of Engineering Bldg. 8
·Mechano-Informatics
Faculty of Engineering Bldg. 14
·Information &
Communication Engineering
Reppin-kan
·Administration Office of
the School of Engineering/IST

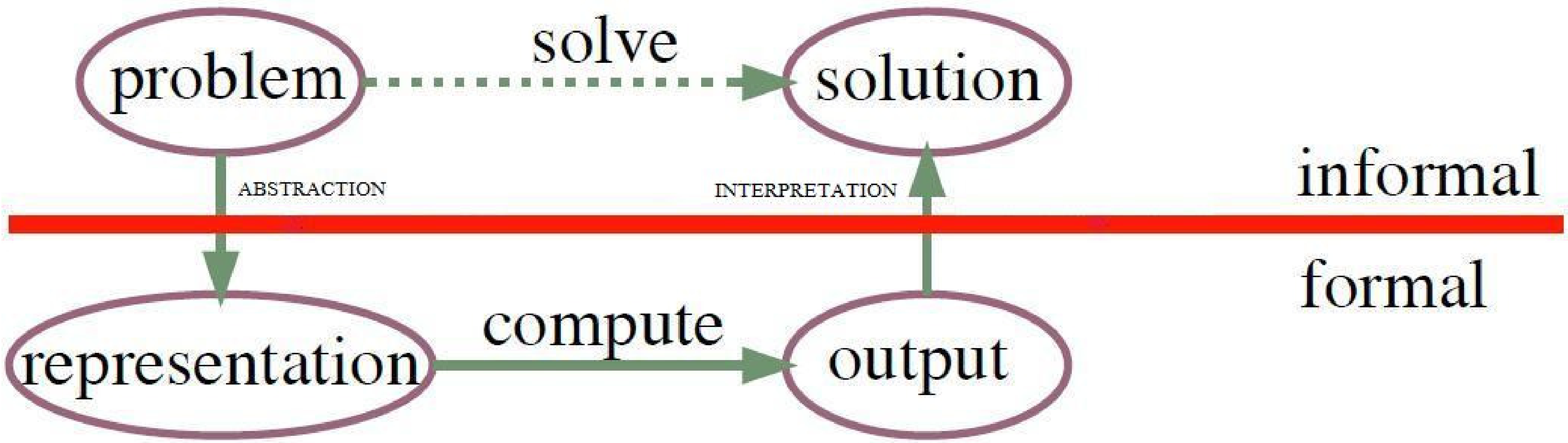
What is Problem solving ?

- We have a problem and want to find a solution !
- Different meanings in different contexts …
- From Wikipedia (!) :
  - In **psychology**, problem solving refers to a state of desire for reaching a definite *goal* from a present condition that either is not directly moving toward the goal, is far from it, or needs more complex logic for finding a missing description of conditions or steps toward the goal.
  - In **computer science** and in the part of artificial intelligence that deals with algorithms, problem solving encompasses a number of techniques known as algorithms, heuristics, root cause analysis, etc.

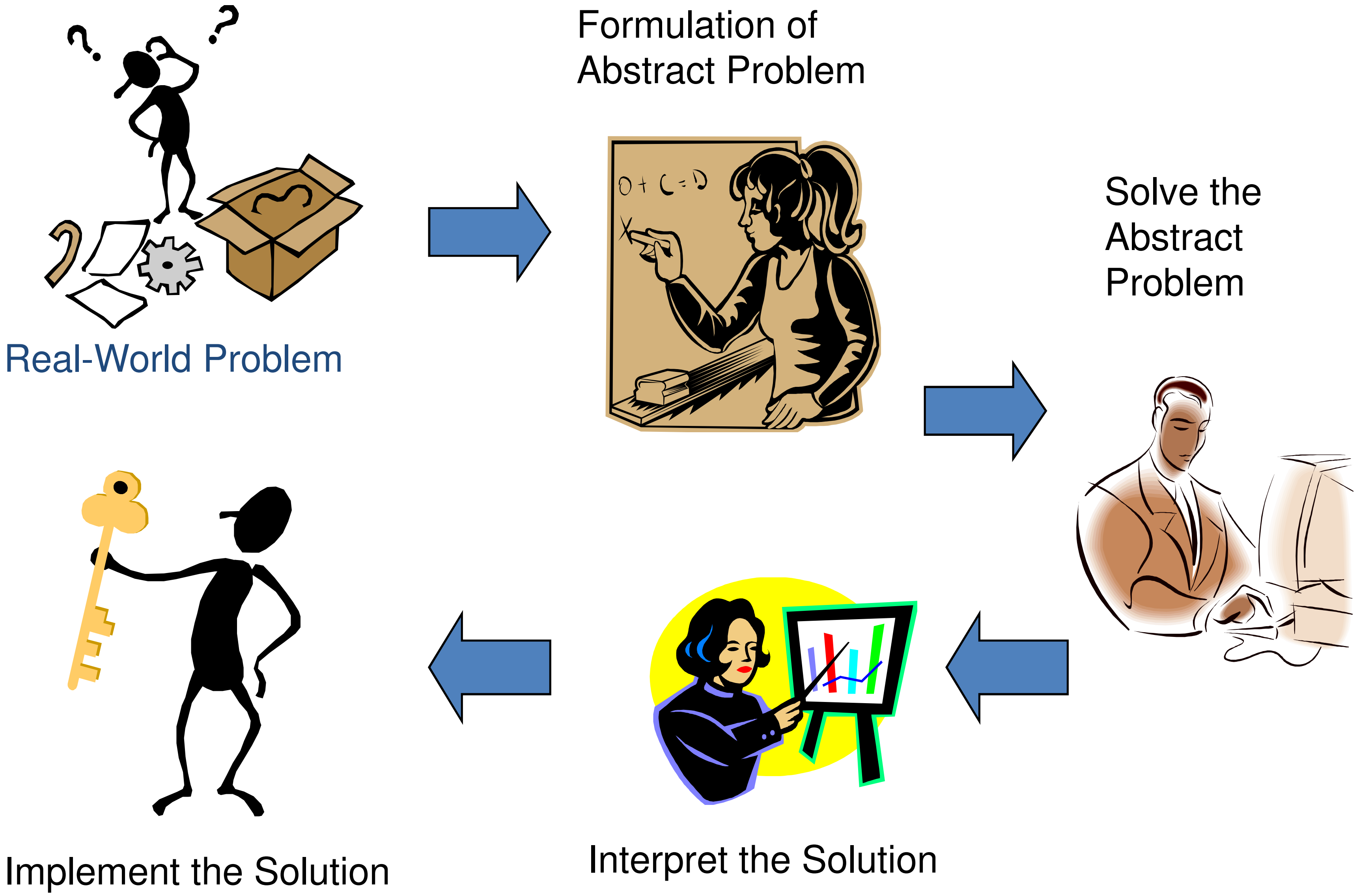
Ideally



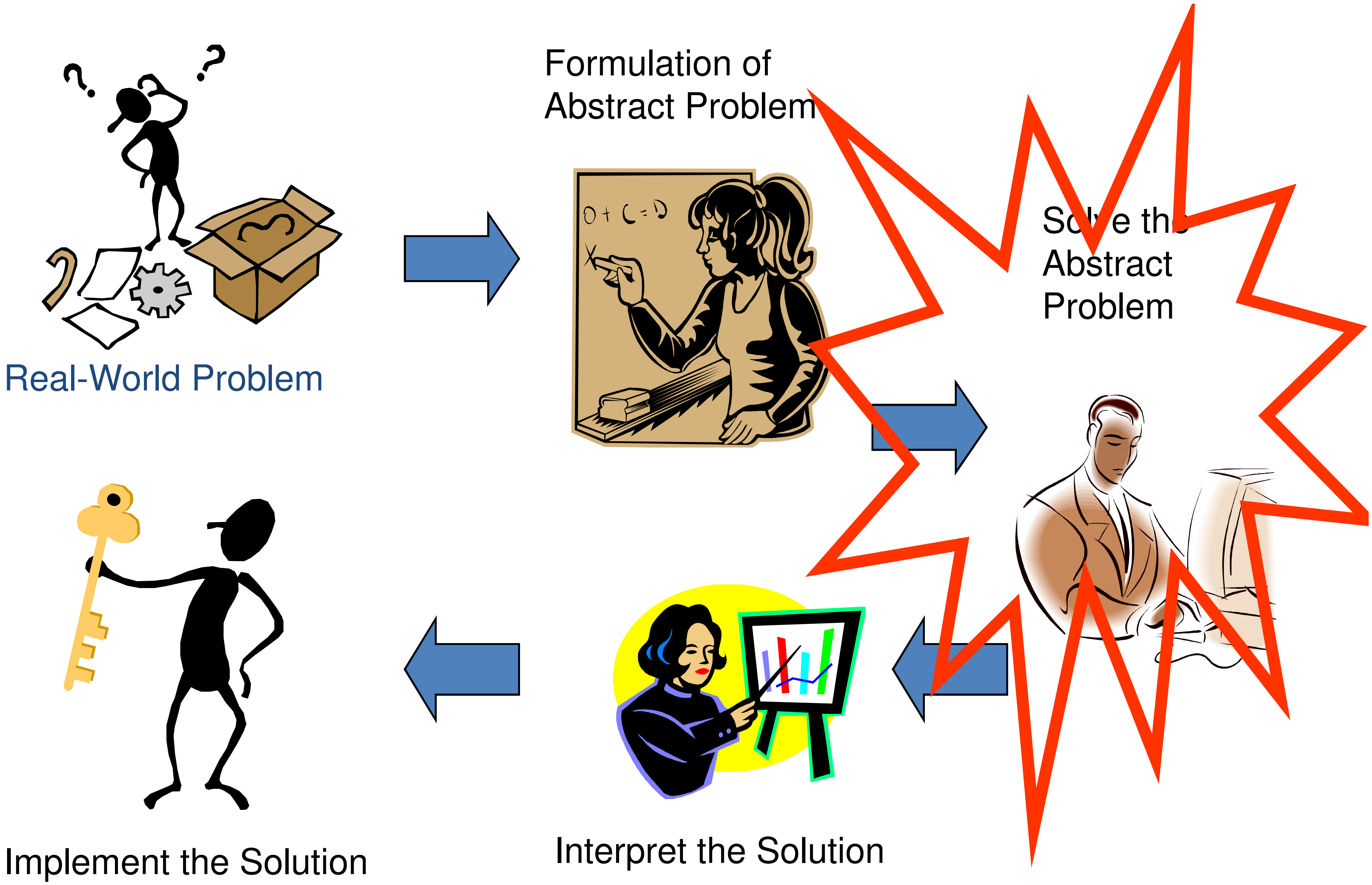
In practice



[from Poole & Mackworth 2010]



[From A. Løkketangen]



[From A. Løkketangen]

Problem Representation

- From [Poole & Mackworth 2010] :

We want a representation to be

- rich enough to express the knowledge needed to solve the problem;
- as close to the problem as possible: 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 and/or space
- able to be acquired from people, data and past experiences.

Modeling

- We have to model the problem
  … in a modeling language
- and to have a notion of "solution"
  by reduction / simplification of the problem
- Can use the mathematics toolbox:
  Logic, polynomial equations, differential equations,…
- Key: we want this model to be (efficiently) executable by a computer
- Modeling Language or Modeling Paradigm with associated computation algorithm(s)

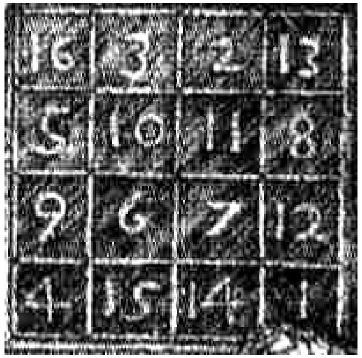
What is a solution ?

- Formula to be satisfied or set of conditions to be achieved
- unique solution ? Several solutions ?
- Some solution are better than others ?
  - Optimal solution
- Sometimes too hard to find …
  - Approximate solution
- Quality of solution improving with time:
  - Anytime algorithms

Simple examples

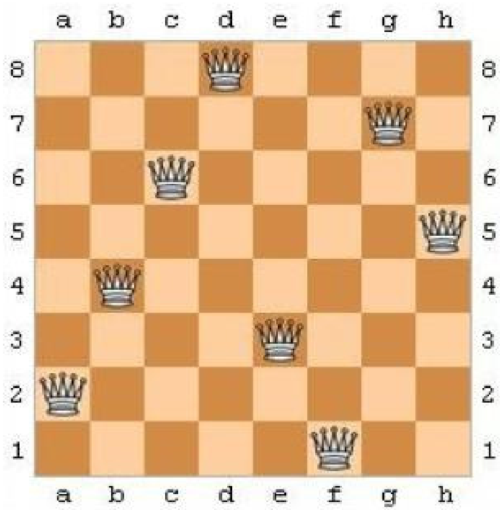
$$
\begin{array}{cccccc}
 & & S & E & N & D \\
+ & & M & O & R & E \\
\hline
= & M & O & N & E & Y
\end{array}
$$

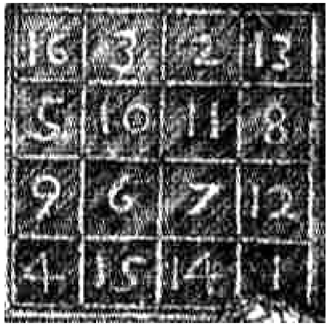
- Mathematical puzzles
  - Crypto-arithmetic, magic squares
- Logical puzzles
  - boolean formulas (SAT), N-Queens
- Sudoku



A. Dürer, *Melencolia I* (1514)

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





Simple ?

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

- Let's take magic square
- 10x10 magic square

  naïve search space $=100^{100} = 10^{200}$

  better with permutations:

  $100! \approx 10^{158}$

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

- 400x400 magic square

  search space $= 160000! \approx 10^{763175}$
- We will see methods which can solve 400x400 in less than one hour CPU-time

Simple Scheduling

Task	Description	Duration	Predecessor
a	Erecting Walls	7	none
b	Carpentry for Roof	3	a
c	Roof	1	b
d	Installations	8	a
e	Facade Painting	2	c, d
f	Windows	1	c, d
g	Garden	1	c, d
h	Ceilings	3	a
i	Painting	2	f, h
j	Moving in	1	i

what is the minimal time to build the house ?
How to schedule the tasks to achieve the goal in minimal time ?

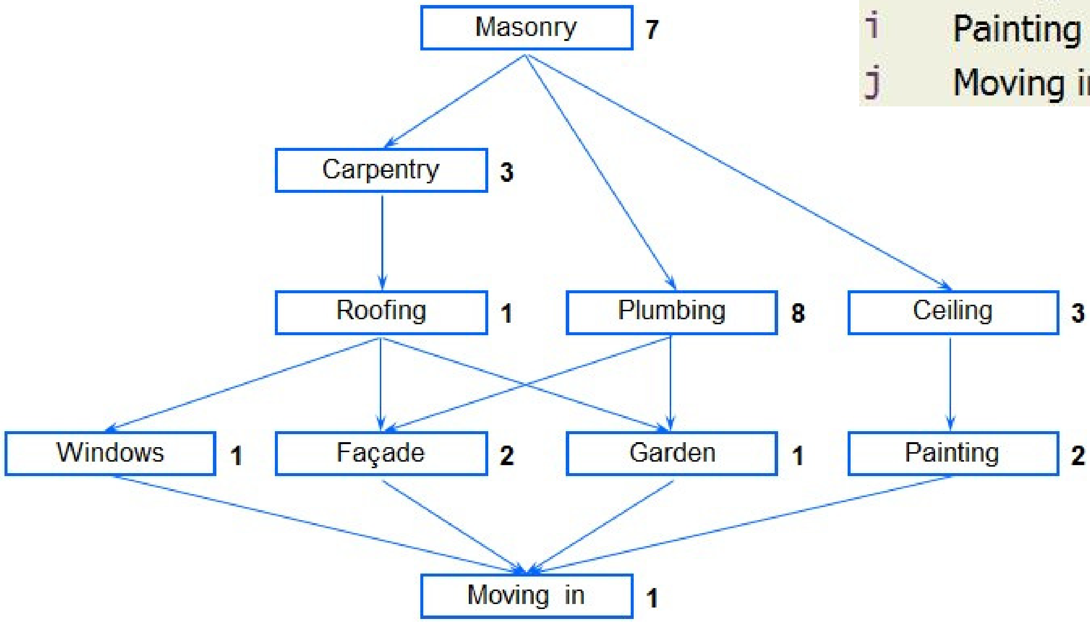
Representation(s)

- Constraints:

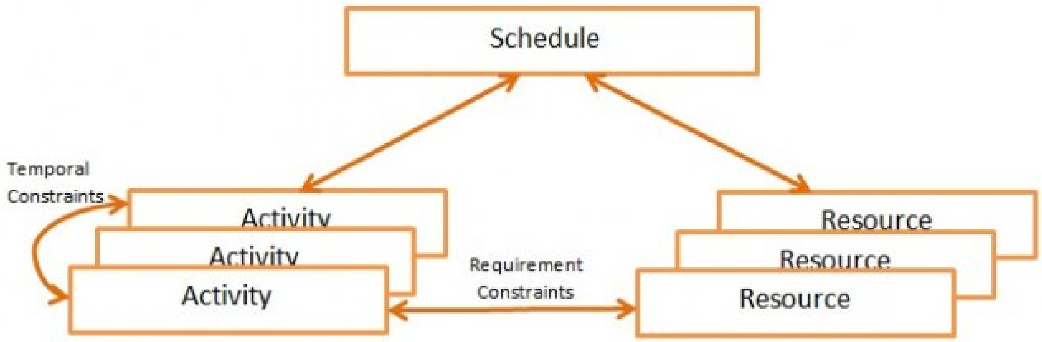
$$A+7\le B,\quad B+3\le C,\quad A+7\le D,\quad C+1\le E,$$
$$D+8\le E,\quad C+1\le F,\quad D+8\le F,\quad C+1\le G,$$
$$D+8\le G,\quad A+7\le H,\quad F+1\le I,\quad H+3\le I,$$
$$I+2\le J.$$

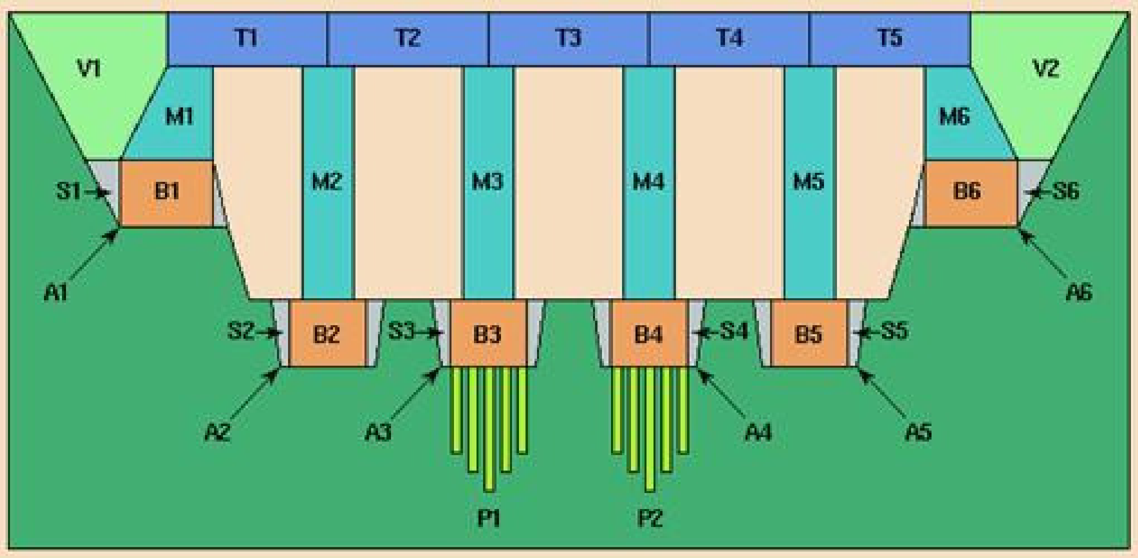
Task	Description	Duration	Predecessor
a	Erecting Walls	7	none
b	Carpentry for Roof	3	a
c	Roof	1	b
d	Installations	8	a
e	Facade Painting	2	c, d
f	Windows	1	c, d
g	Garden	1	c, d
h	Ceilings	3	a
i	Painting	2	f, h
j	Moving in	1	i

- Graph:



Disjunctive Scheduling

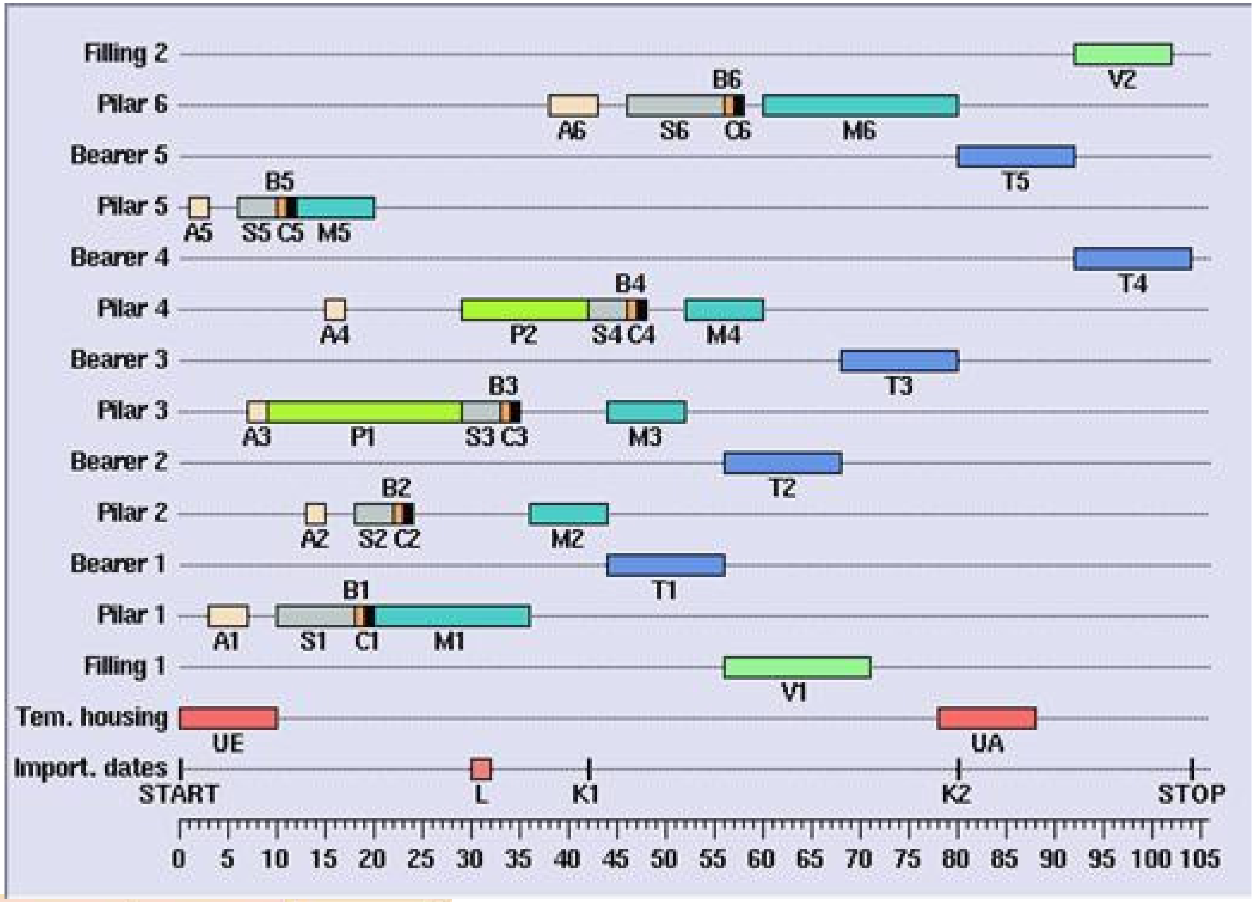


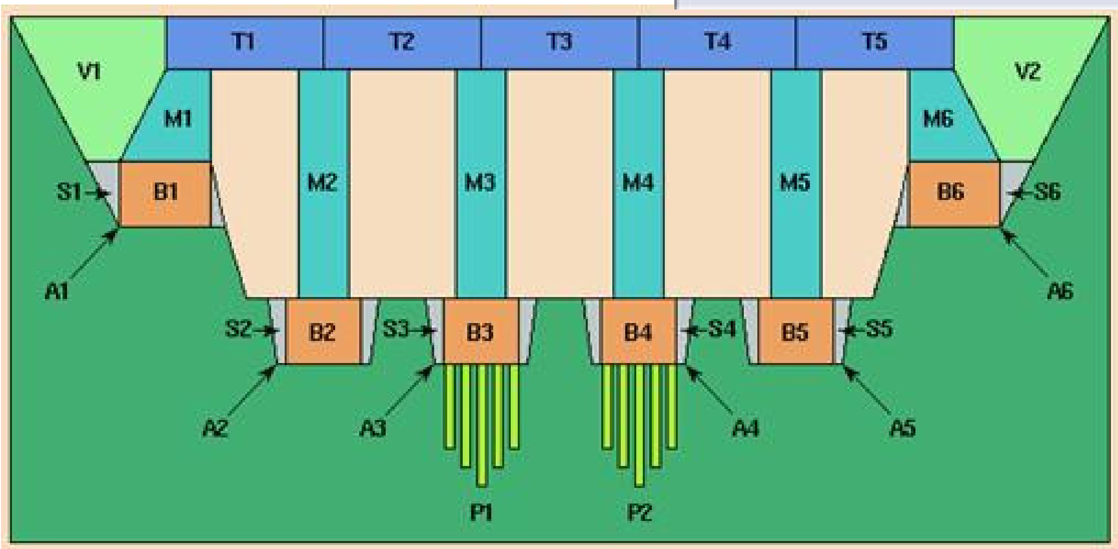


No	Na.	Description	Dur	Preds	Res
1	pa	beginning of project	0	-	noResource
2	a1	excavation (abutment 1)	4	pa	excavator
3	a2	excavation (pillar 1)	2	pa	excavator
4	a3	excavation (pillar 2)	2	pa	excavator
5	a4	excavation (pillar 3)	2	pa	excavator
6	a5	excavation (pillar 4)	2	pa	excavator
7	a6	excavation (abutment 2)	5	pa	excavator
8	p1	foundation piles 2	20	a3	pile driver
9	p2	foundation piles 3	13	a4	pile driver
10	ue	erection of temporary housing	10	pa	noResource
11	s1	formwork (abutment 1)	8	a1	carpentry
12	s2	formwork (pillar 1)	4	a2	carpentry
13	s3	formwork (pillar 2)	4	p1	carpentry
14	s4	formwork (pillar 3)	4	p2	carpentry
15	s5	formwork (pillar 4)	4	a5	carpentry
16	s6	formwork (abutment 2)	10	a6	carpentry
17	b1	concrete foundation (abutment 1)	1	s1	concrete mixer
18	b2	concrete foundation (pillar 1)	1	s2	concrete mixer
19	b3	concrete foundation (pillar 2)	1	s3	concrete mixer
20	b4	concrete foundation (pillar 3)	1	s4	concrete mixer
21	b5	concrete foundation (pillar 4)	1	s5	concrete mixer
22	b6	concrete foundation (abutment 2)	1	s6	concrete mixer
23	ab1	concrete setting time (abutment 1)	1	b1	noResource
24	ab2	concrete setting time (pillar 1)	1	b2	noResource
25	ab3	concrete setting time (pillar 2)	1	b3	noResource
26	ab4	concrete setting time (pillar 3)	1	b4	noResource
27	ab5	concrete setting time (pillar 4)	1	b5	noResource
28	ab6	concrete setting time (abutment 2)	1	b6	noResource
29	m1	masonry work (abutment 1)	16	ab1	bricklaying
30	m2	masonry work (pillar 1)	8	ab2	bricklaying
31	m3	masonry work (pillar 2)	8	ab3	bricklaying
32	m4	masonry work (pillar 3)	8	ab4	bricklaying
33	m5	masonry work (pillar 4)	8	ab5	bricklaying
34	m6	masonry work (abutment 2)	20	ab6	bricklaying
35	l	delivery of the preformed bearers	2	-	crane
36	t1	positioning (preformed bearer 1)	12	m1, m2, l	crane
37	t2	positioning (preformed bearer 2)	12	m2, m3, l	crane
38	t3	positioning (preformed bearer 3)	12	m3, m4, l	crane
39	t4	positioning (preformed bearer 4)	12	m4, m5, l	crane
40	t5	positioning (preformed bearer 5)	12	m5, m6, l	crane
41	ua	removal of the temporary housing	10	-	noResource
42	v1	filling 1	15	t1	caterpillar
43	v2	filling 2	10	t5	caterpillar
44	pe	end of project	0	t2, t3, t4, v1, v2, ua	noResource

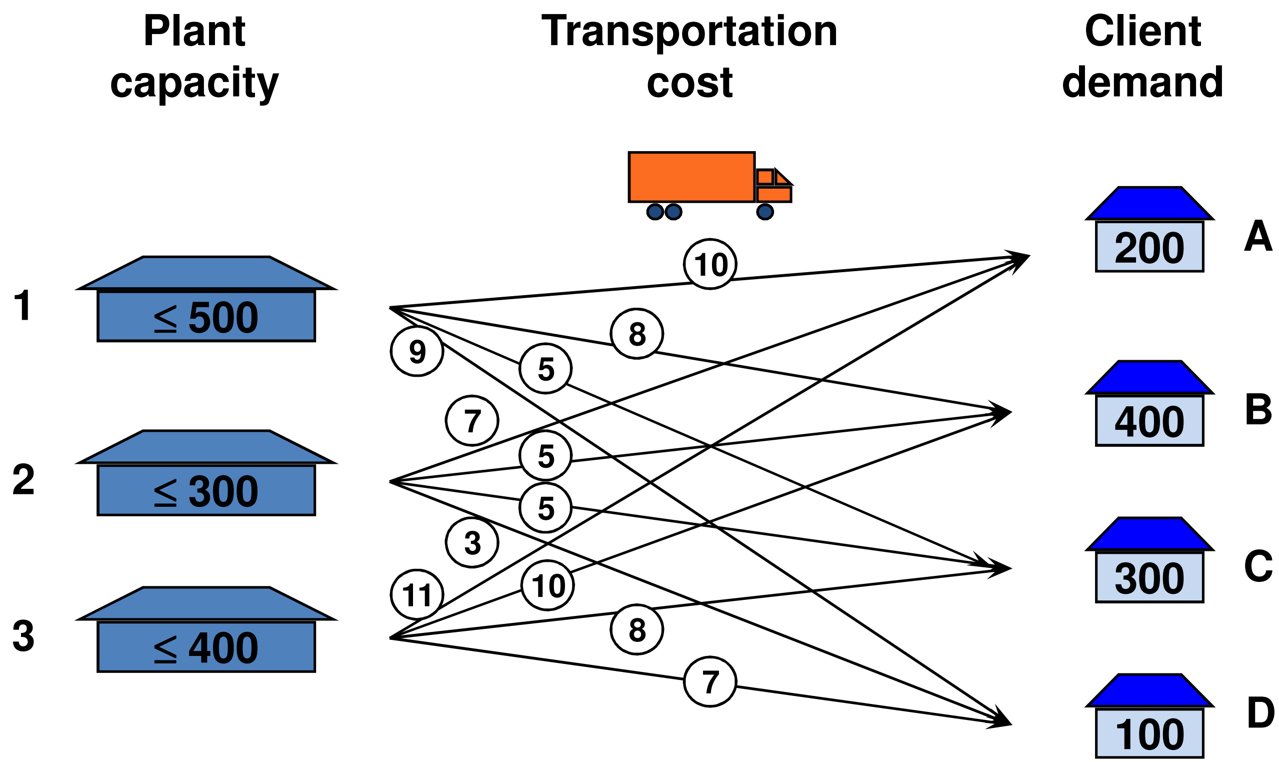
Solution:

(Gantt chart)





Ressource Allocation



Ressource Allocation

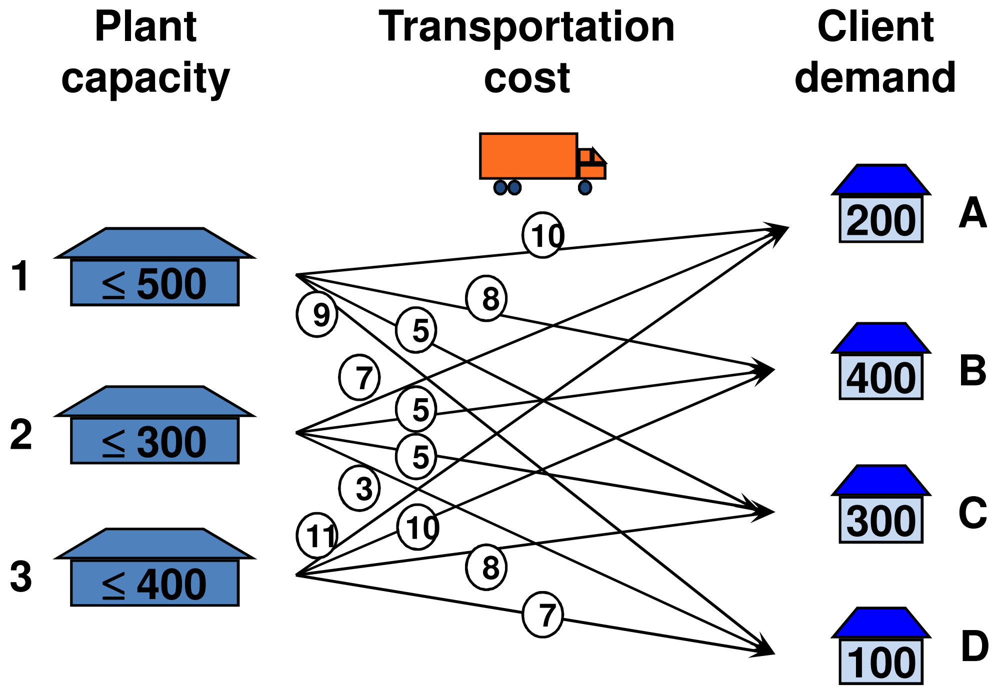
Constraints:

```
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
```

Goal: minimize 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
```



What have all this in common ?

- Large search space
- well-identified “goal”
  - Notion of solution is easy to define (declaratively)
- But we don’t know how to reach it
- No algorithm to build a solution incrementally
- Hence:
  - need to explore the search space
  - Either exhaustively or in an “intelligent”, “guided” manner

Methods detailed in this lecture series

- Graph Search
  - Representation of states and transitions/actions between states → graph
  - Explored explicitly or implicitly
- Constraint Solving
  - Represent problem by variables and constraints
  - Use specific solving algorithms to speedup search
- Local Search and Metaheuristics
  - Evaluation function to check if state is “good” or not
  - Optimization of the evaluation function

Methods NOT detailed in this lecture series

- Numerical Optimization Methods
  - For continuous domains & twice differentiable functions
- Linear Optimization methods
  - For Linear Constraints & rational domains
  - Simplex algorithm, Interior Point Methods
  - Integer Programming, cutting plane methods
- Dynamic Programming
  - Decomposable problem, recursive relation

Lectures

1. Introduction
   (now!)
2. classical A.I. : State-graphs and the A* algorithm
3. Constraint Satisfaction Problems (CSP)
4. Constraint Solving Techniques I
5. Constraint Solving Techniques II (indexicals)
6. Constraint Programming
7. Combinatorial Optimization Problems
8. Local Search techniques
9. Some Metaheuristics:
   Tabu search, simulated annealing
10. Population-based Methods
    Genetic algo., Beam search, //
11. Constraint-based local search
12. Parallel Local Search

LECTURE 1
INTRODUCTION

Graph Search

- A large variety of problems can be represented by a graph
- Solutions can be considered as defining specific nodes
- Solving the problem is reduced to searching the graph for those nodes
  - starting from an initial node
  - each transition in the graph corresponds to a possible action
  - ending when reaching a final node (solution)

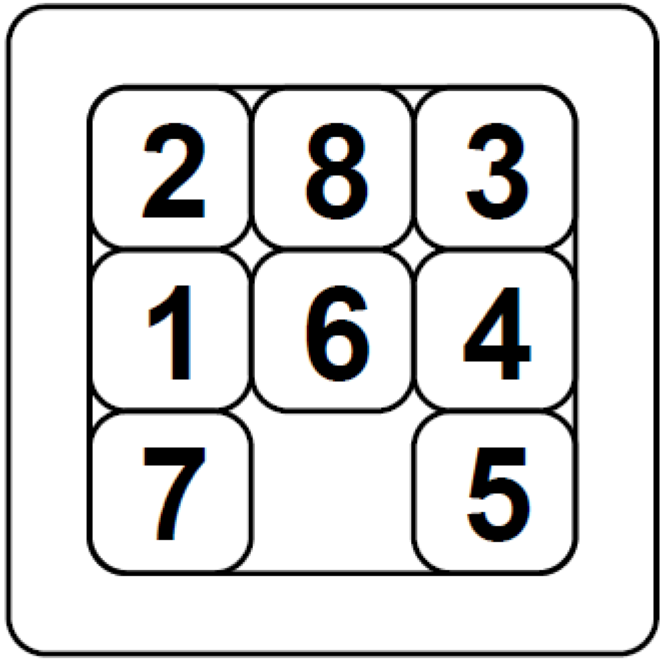
Single-state Graph Search

- A problem is defined by :

1. An initial state
2. A successor function S(X) = set of action-state pairs
3. A set of specific nodes: the goals
4. ? A path cost (additive)

A solution is the sequence of actions leading from the initial state to a goal

The 8-puzzle

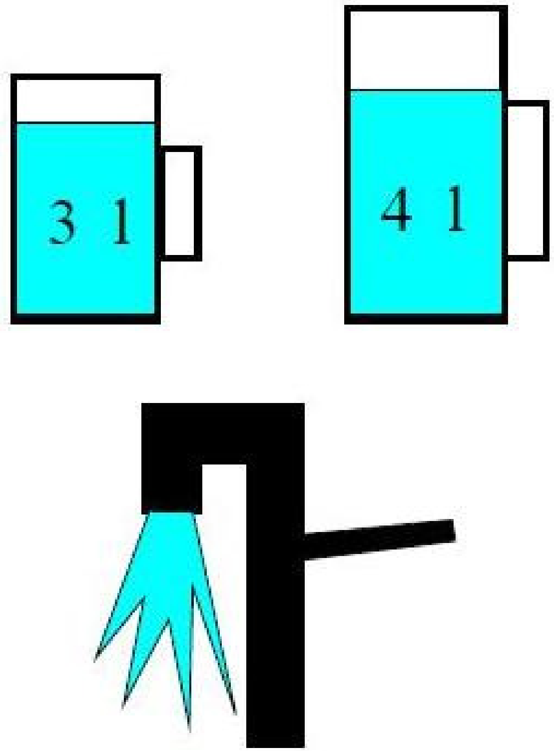


- can be generalized to 15-puzzle, 24-puzzle, etc.
- Any $(n^2 - 1)$-puzzle for $n \geq 3$
- state = permutation of (Ø, 1, 2, 3, 4, 5, 6, 7, 8)
- e.g. state above is: (2,8,3,1,6,4,7,Ø,5)
- 9! = 362,880 possible states
- Solution: (Ø,1,2,3,4,5,6,7,8)
- Actions: possible moves, e.g. : (2,8,3,1,6,4,7,Ø,5)→ (2,8,3,1,Ø,4,7,6,5)

Water Jug Problem

- Problem
  we have one jug of 3 liters, one jug of 4 liters
  we want to put exactly 2 liters of in the 4 l. jug

- Formulation of the problem:
  - state represents the content of jugs:
    thus 2 variables: J_3 and J_4
    Initial state: (0,0)
    Final state: (_,2)
  - Actions:
    - Fill jugs
    - Empty jugs
    - What else?



F4: fill jug4 from the pump.

precond: $J_4 < 4$ **effect:** $J_4' = 4$

E4: empty jug4 on the ground.

precond: $J_4 > 0$ **effect:** $J_4' = 0$

E4-3: pour water from jug4 into jug3 until jug3 is full.

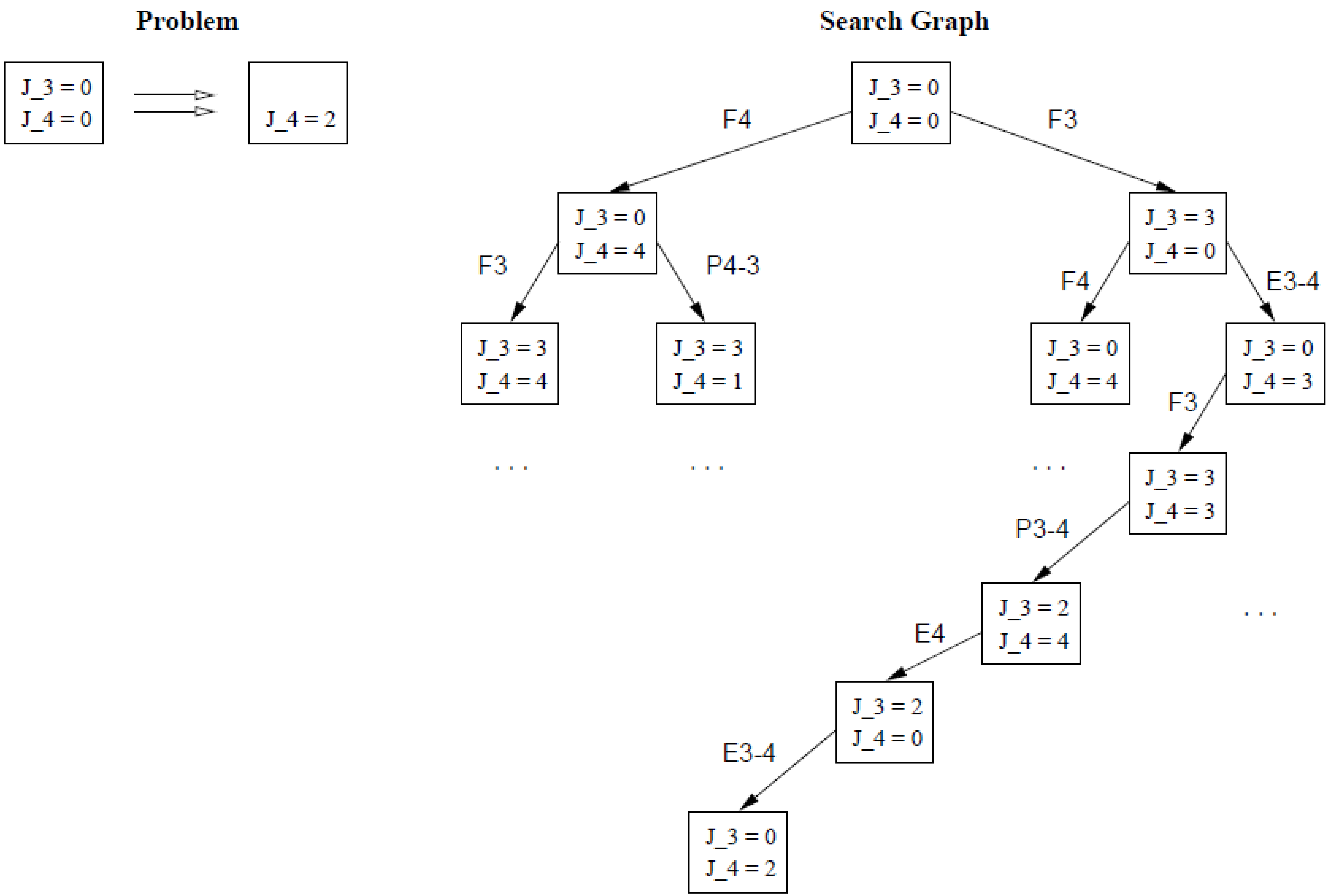
precond: $J_3 < 3,$ $J_4 \geq 3 - J_3$ **effect:** $J_3' = 3,$ $J_4' = J_4 - (3 - J_3)$

P3-4: pour water from jug3 into jug4 until jug4 is full.

precond: $J_4 < 4,$ $J_3 \geq 4 - J_4$ **effect:** $J_4' = 4,$ $J_3' = J_3 - (4 - J_4)$

E3-4: pour water from jug3 into jug4 until jug3 is empty.

precond: $J_3 + J_4 < 4,$ $J_3 > 0$ **effect:** $J_4' = J_3 + J_4,$ $J_3' = 0$

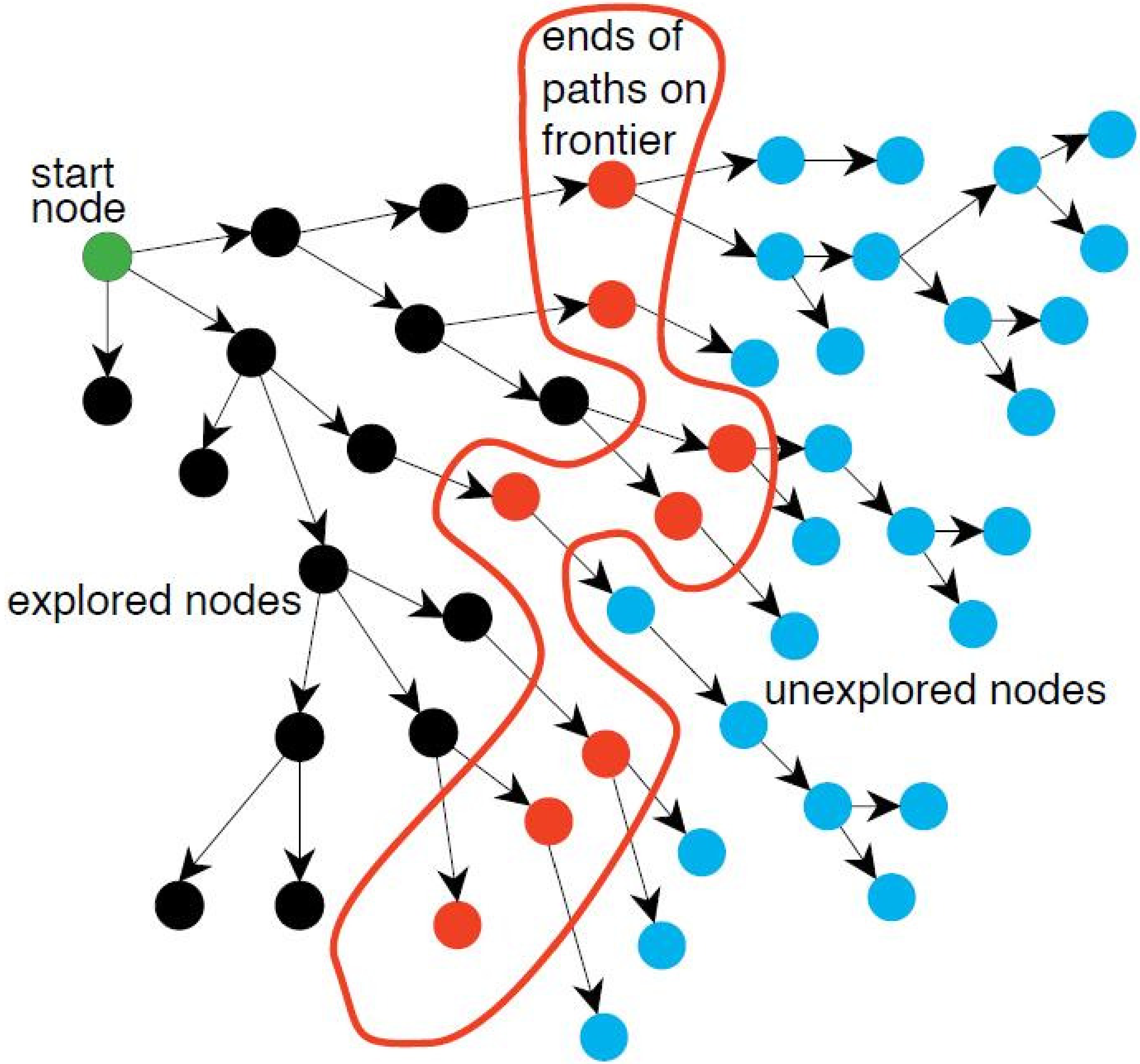


Problem
J_3 = 0
J_4 = 0
J_4 = 2
Search Graph
J_3 = 0
J_4 = 0
F4
F3
J_3 = 0
J_4 = 4
J_3 = 3
J_4 = 0
F3
P4-3
F4
E3-4
J_3 = 3
J_4 = 4
J_3 = 3
J_4 = 1
J_3 = 0
J_4 = 4
J_3 = 0
J_4 = 3
F3
J_3 = 3
J_4 = 3
P3-4
J_3 = 2
J_4 = 4
E4
J_3 = 2
J_4 = 0
E3-4
J_3 = 0
J_4 = 2

The set of all possible paths of a graph can be represented as a tree.

- A *tree* is a directed acyclic graph all of whose nodes have at most one parent.
- A *root* of a tree is a node with no parents.
- A *leaf* is a node with no children.
- The *branching factor* of a node is the number of its children.

Graphs can be turned into trees by duplicating nodes and breaking cyclic paths, if any.



ends of
paths on
frontier
start
node
explored nodes
unexplored nodes

Basic Graph Search Algorithm

Input: a graph,
a set of start nodes,
Boolean procedure $goal(n)$ that tests if n is a goal node.

$frontier := \{\langle s \rangle : s \text{ is a start node}\};$

while *frontier* is not empty:
  select and **remove** path $\langle n_0, \ldots, n_k \rangle$ from *frontier*;
  if $goal(n_k)$
    return $\langle n_0, \ldots, n_k \rangle$;
  for every neighbor n of n_k
    add $\langle n_0, \ldots, n_k, n \rangle$ to *frontier*;

end while

Basic Graph Search Algorithm

Input: a graph,
a set of start nodes,
Boolean procedure $goal(n)$ that tests if n is a goal node.
$frontier := \{\langle s \rangle : s \text{ is a start node}\}$;
while *frontier* is not empty:
select and **remove** path $\langle n_0, \ldots, n_k \rangle$ from *frontier*;
if $goal(n_k)$
return $\langle n_0, \ldots, n_k \rangle$;
for every neighbor n of n_k
add $\langle n_0, \ldots, n_k, n \rangle$ to *frontier*;
end while

… beware of cycles !

Different Search Algorithms

Criterion	Breadth-First	Uniform-Cost	Depth-First	Depth-Limited	Iterative Deepening
Complete?	Yes*	Yes*	No	Yes, if $l \geq d$	Yes
Time	b^{d+1}	$b^{\lceil C^*/\epsilon \rceil}$	b^m	b^l	b^d
Space	b^{d+1}	$b^{\lceil C^*/\epsilon \rceil}$	bm	bl	bd
Optimal?	Yes*	Yes*	No	No	Yes

Constraint Modeling

- **Declarative language** : modeling is easy
- **local** specification of the problem
- **global** consistency achieved (or approximated) by constraint solving techniques
- **compositionality** : constraints are combined implicitly through shared logical variables

Basic Objects

Variable: a place holder for values

$$X, Y, Z, L_3, U_{21}, List$$

Function Symbol: mapping of variables to values

$$+, -, \times, \div, \sin, \cos, ||$$

Relation Symbol: relation between variables

arithmetic relation: $=, \leq, \neq$

symbolic relation: *all_different*

Constraints

- Declarative relations between variables
- Constraints used to both model and solve the problem
- specific algorithms for efficient computation
- Constraints could be numeric or symbolic :

$X \leq 5$, $X + Y = Z$

all_different(X1,X2,…,Xn)

at_most(N,[X1,X2,X3],V)

- multi-directional relations

Constraint Satisfaction Problems

- Variables $X_1 \dots X_n$

  unknowns of the problem

- Domains $D_1 \dots D_n$

  search space

- Constraints $C_1 \dots C_p$

  partial information on the variables

Constraint Satisfaction Problem (CSP)

- a CSP is a triple < V , D, C > where :
  - $V=\{V_1,\dots,V_n\}$ is a (finite) set of *variables*
  - $D=\{D_1,\dots,D_n\}$ a set of *domains* D_i for each variable V_i (finite sets of possible values)
  - $C=\{C_1,\dots,C_p\}$ is a set of *constraints* on variables of V

- a constraint $c_i(V_{i1},\dots,V_{ik})$

  on variables $\{V_{i1},\dots,V_{ik}\}$ is defined as a subset of the cross-product $D_{i1} x \dots x D_{ik}$

Crypto-arithmetics as CSP

```
      S E N D
+   M O R E
_____________
  M O N E Y
```

each letter represents a (different) digit

and the addition should be correct !

... Solution ?

- Two different models with constraints

R_4 R_3 R_2 R_1

S E N D

\+ M O R E

____

M O N E Y

R_4 R_3 R_2 R_1

variables :

$\{S,E,N,D,M,O,R,Y,R_1, R_2, R_3, R_4\}$

domains :

{0,...,9} for the letters

{0,1} for the carries

constraints :

all_different(S,E,N,D,M,O,R,Y}

$S \neq 0$

$M \neq 0$

5 constraints for columns

$$D + E = Y + 10 * R1$$
$$R1 + N + R = E + 10 * R2$$
$$R2 + E + O = N + 10 * R3$$
$$R3 + S + M = O + 10 * R4$$
$$R4 = M$$

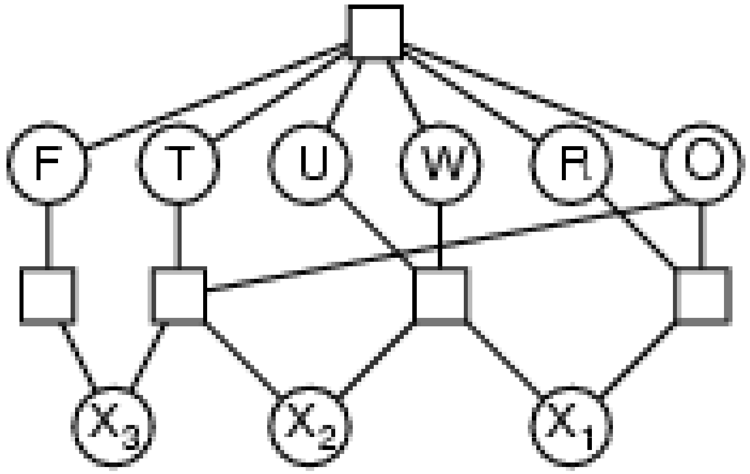
<u>or</u>

one single constraint

$$1000*S + 100*E + 10*N + D + 1000*M + 100*O + 10*R + E = 10000*M + 1000*O + 100*N + 10*E + Y$$

Constraint (hyper-)Graph

```
    T W O
+   T W O
---------
  F O U R
```



- Variables:
  $F\ T\ U\ W\ R\ O\ X_1\ X_2\ X_3$
- Domains:
  $\{0,1,2,3,4,5,6,7,8,9\}$
- Constraints:

all_different (F,T,U,W,R,O)

$T \neq 0 \quad F \neq 0 \qquad X3 = F$

$O + O = R + 10 * X_1$

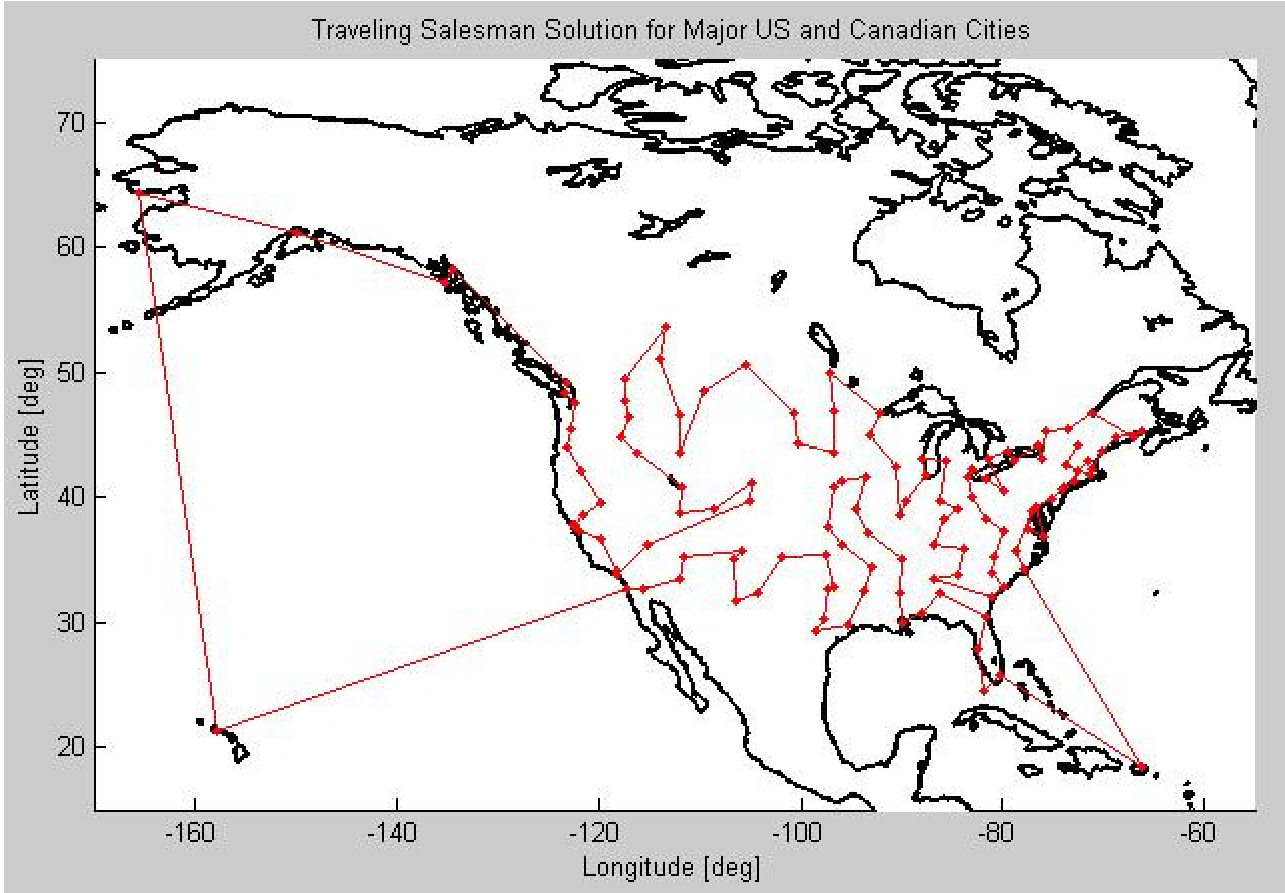
$X_1 + W + W = U + 10 * X_2$

$X_2 + T + T = O + 10 * X_3$

Local Search & Metaheuristics

- Heuristic methods (from Greek: "Εὑρίσκω“)
  - “guided”, but incomplete…
- To be used when search space is too big and cannot be searched exhaustively
- So-called « Metaheuristics » : general techniques to guide the search
- Experimented in various problems :
  - Traveling Salesman Problem (since 60 ’s)
  - scheduling, vehicle routing, cutting
  - SAT
- Simple but experimentally very efficient ...

Traveling Salesman Problem

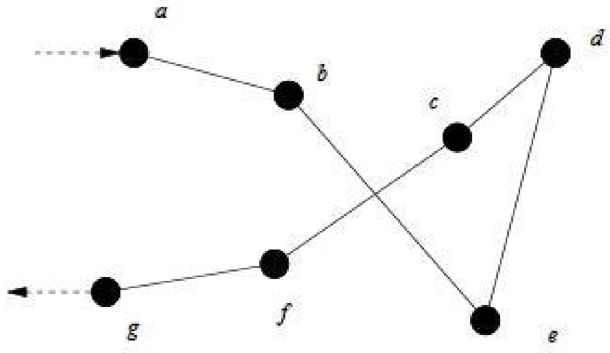


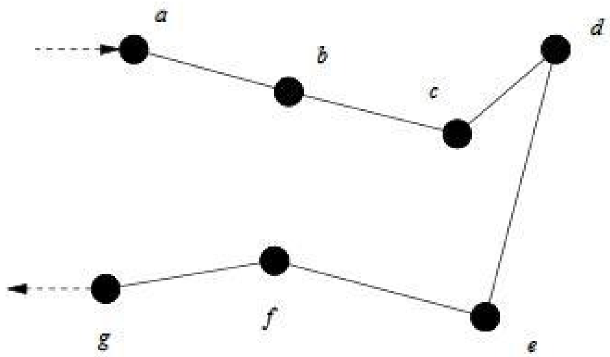
- Local search introduced by [Lin 1965]
- Idea: edge exchange (2-opt, 3-opt, k-opt)

TSP by Local Search

- A *tour* can be represented by a permutation of the list of city nodes
- 2-opt: Swap the visit of 2 nodes
- Cf. example:

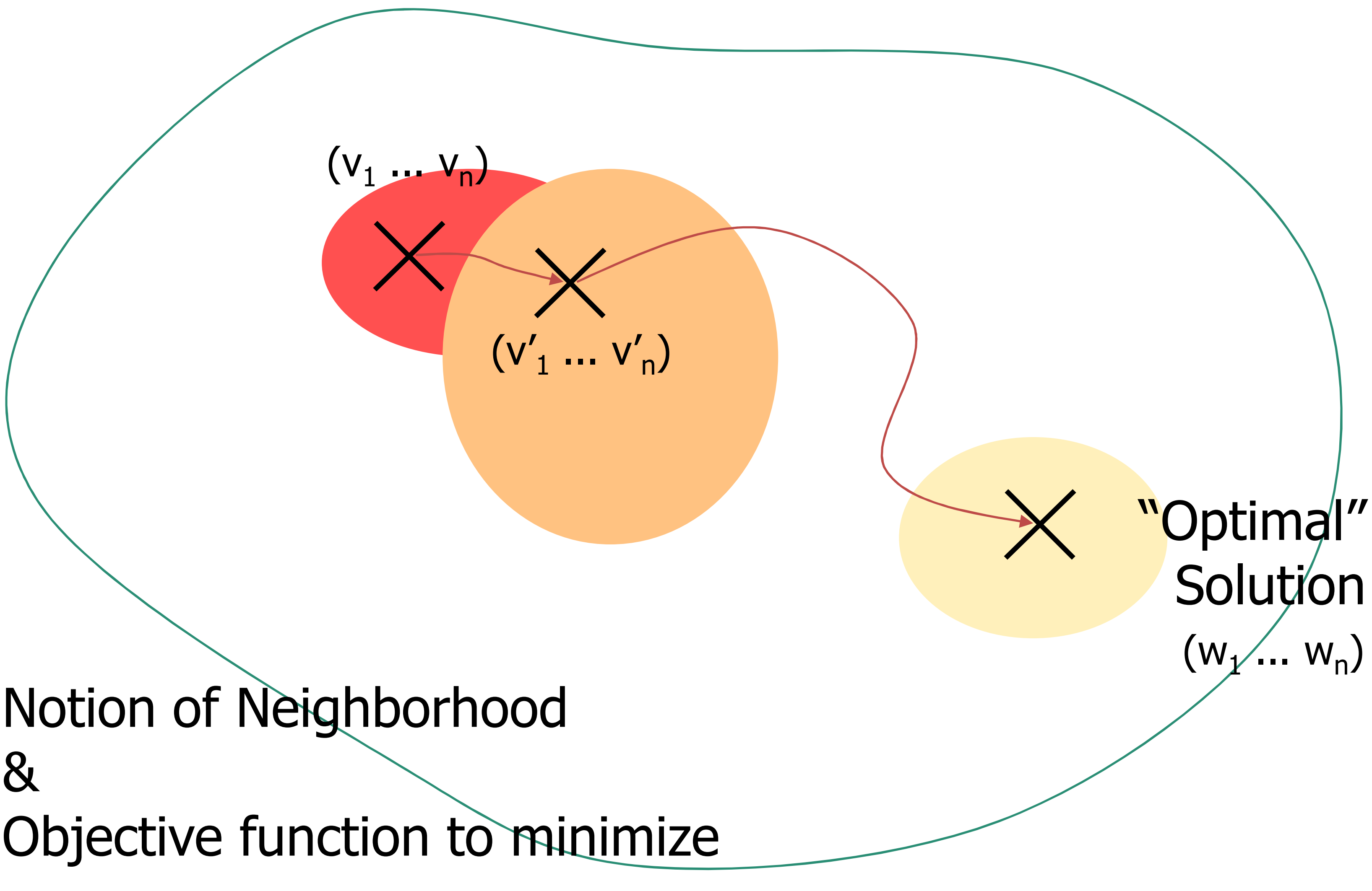
  *(a, b, **e,** d, **c,** f, g> → <a, b, **c,** d, **e,** f, g>*





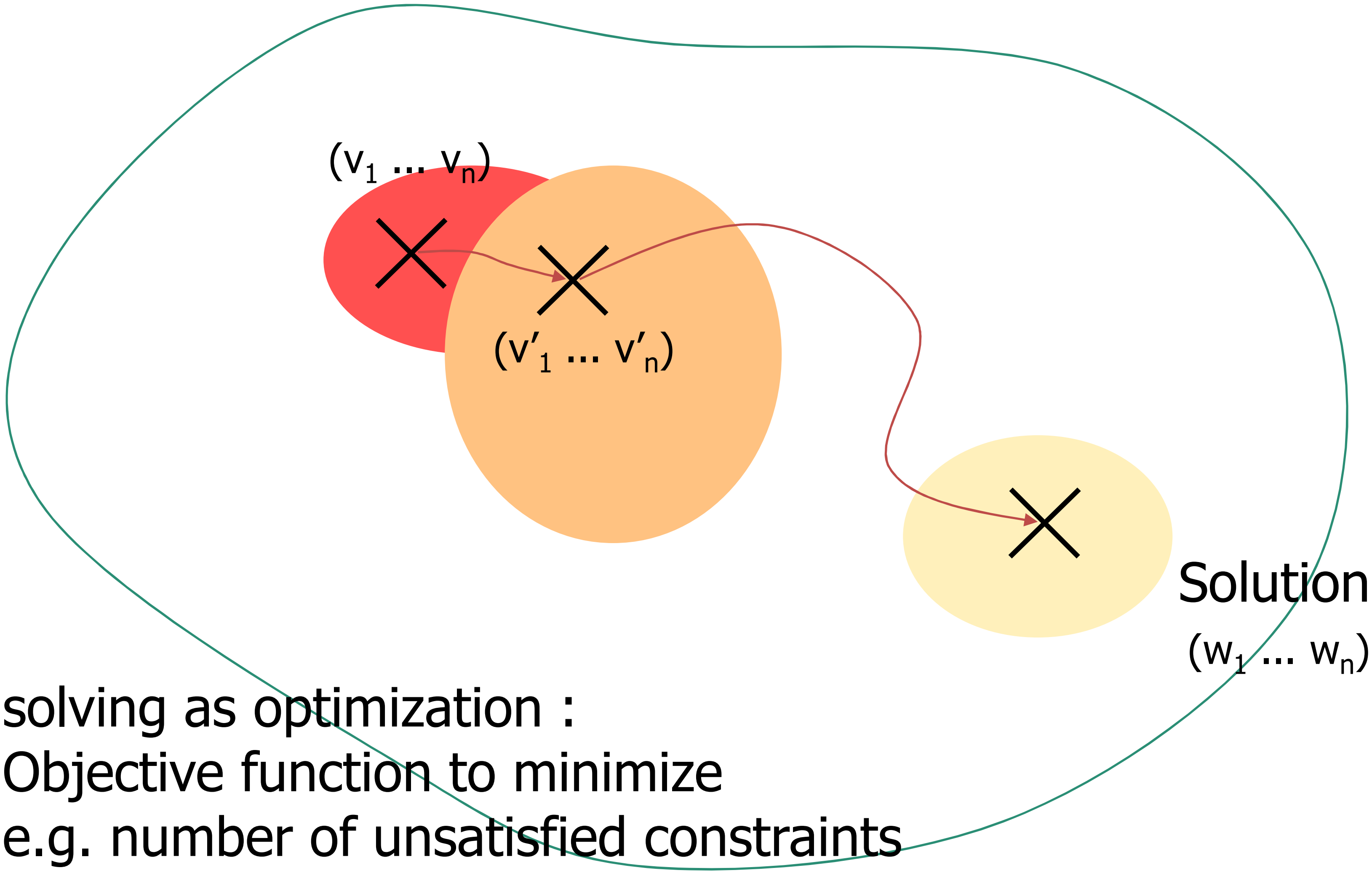
- Naïve Local Search algorithm
  - Start by a random tour
  - Consider all tours formed by executing swaps of 2 nodes
  - Take the one with best (lower) cost
  - Continue until optimum or time-limit reached

Local Search - Iterative Improvement



Notion of Neighborhood
&
Objective function to minimize

Local Search - Iterative Improvement

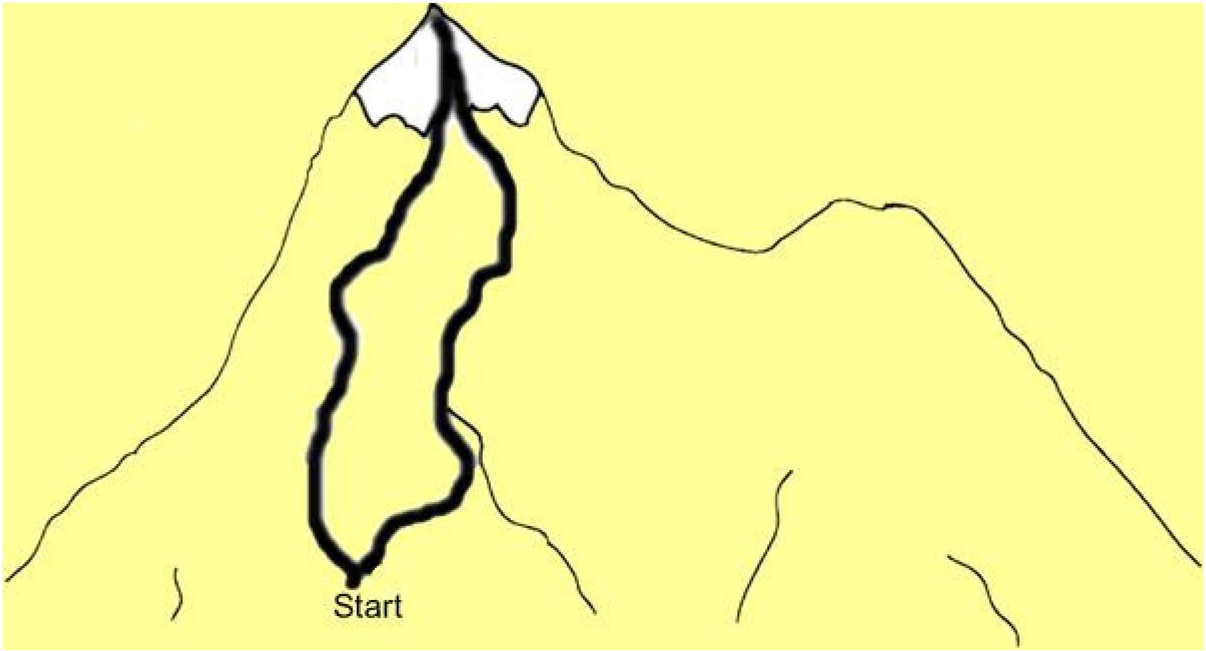


solving as optimization :
Objective function to minimize
e.g. number of unsatisfied constraints

Key Ideas

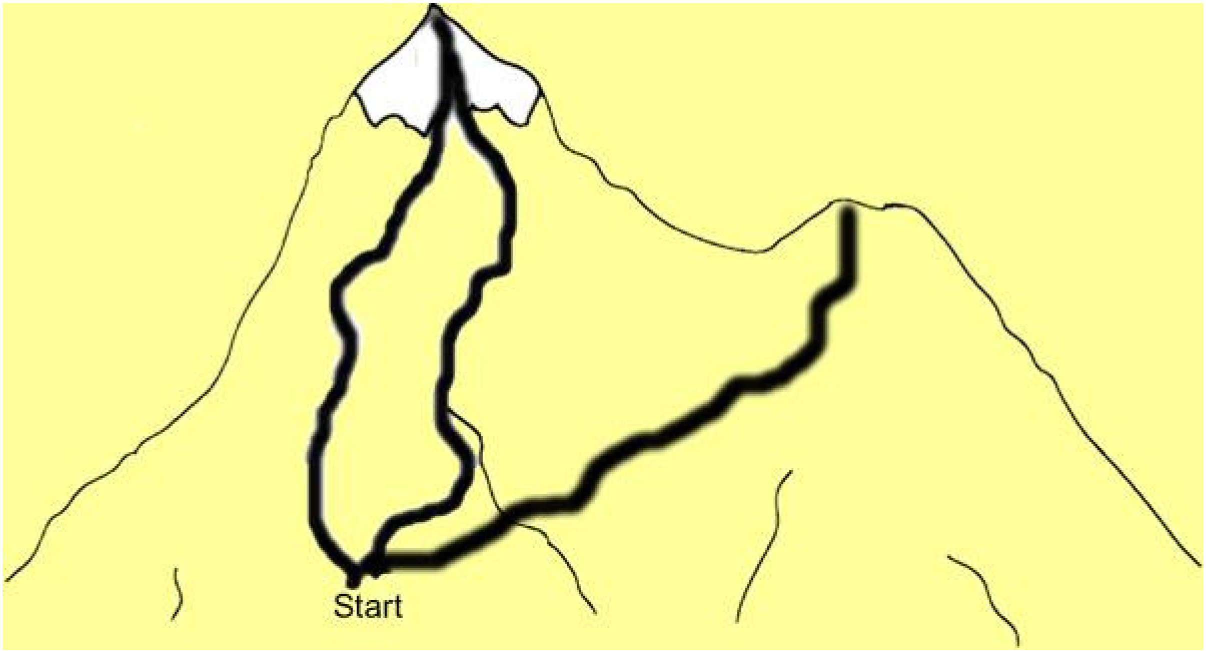
- Optimization problem with objective function
  - e.g. fitness function to maximize, or cost to minimize
- Basic algorithm :
  - start from a random assignment
  - Explore the « neighborhood »
  - move to a « better » candidate
  - continue until optimal solution is found
- iterative improvement
- *anytime* algorithm
  - outputs good if not optimal solution

Hill-Climbing / Gradient Descent



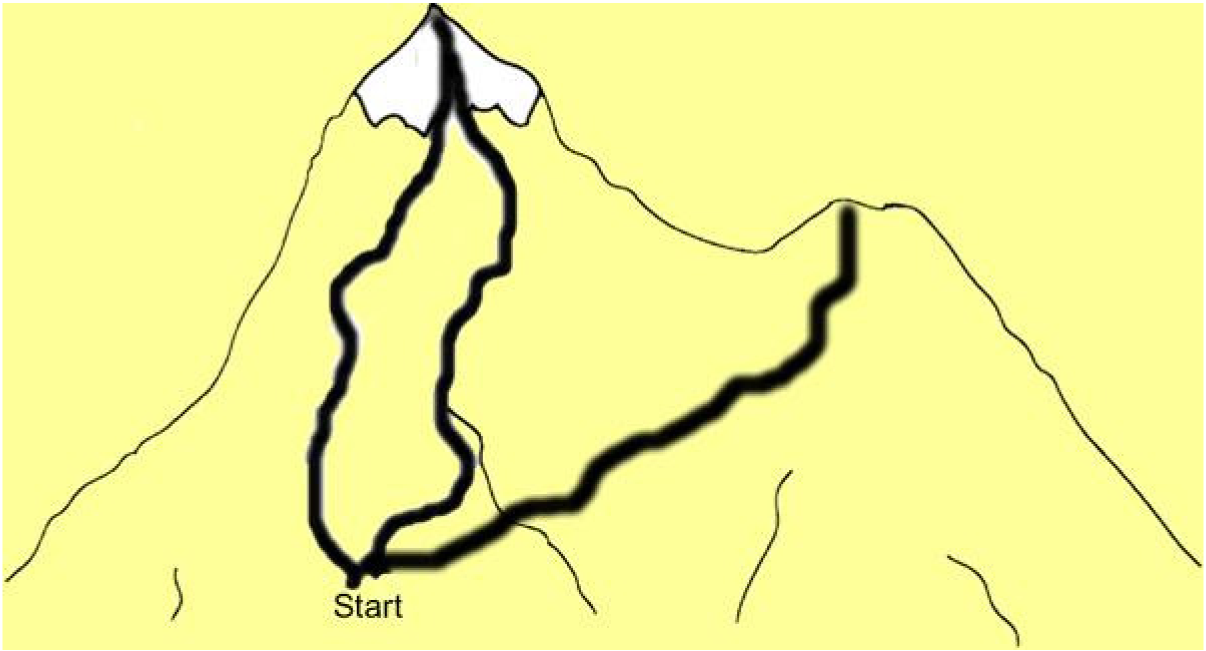
- Fitness/Cost/Objective function to optimize
  - Hill-Climbing = maximization
  - Gradient Descent = minimization

Hill-Climbing / Gradient Descent



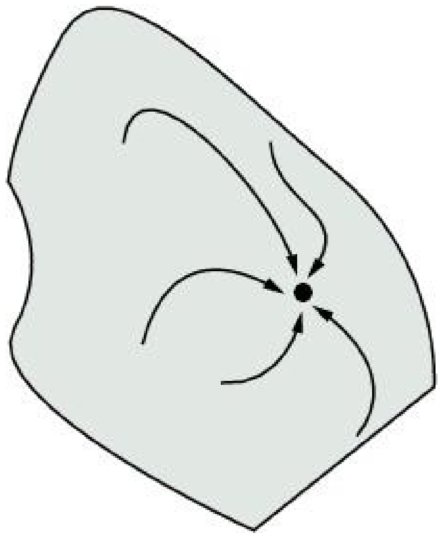
- Beware !

Hill-Climbing / Gradient Descent



- Beware !
- Many different methods to avoid this problem…

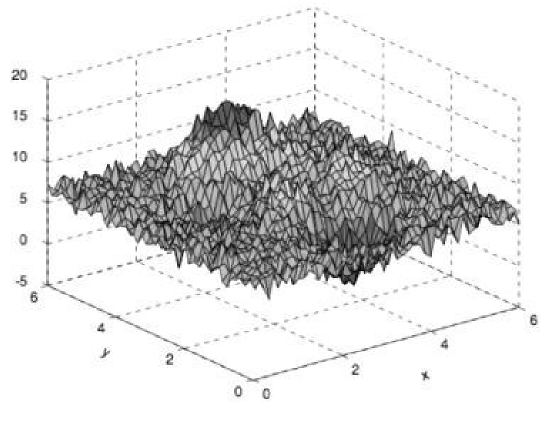
Caveats…



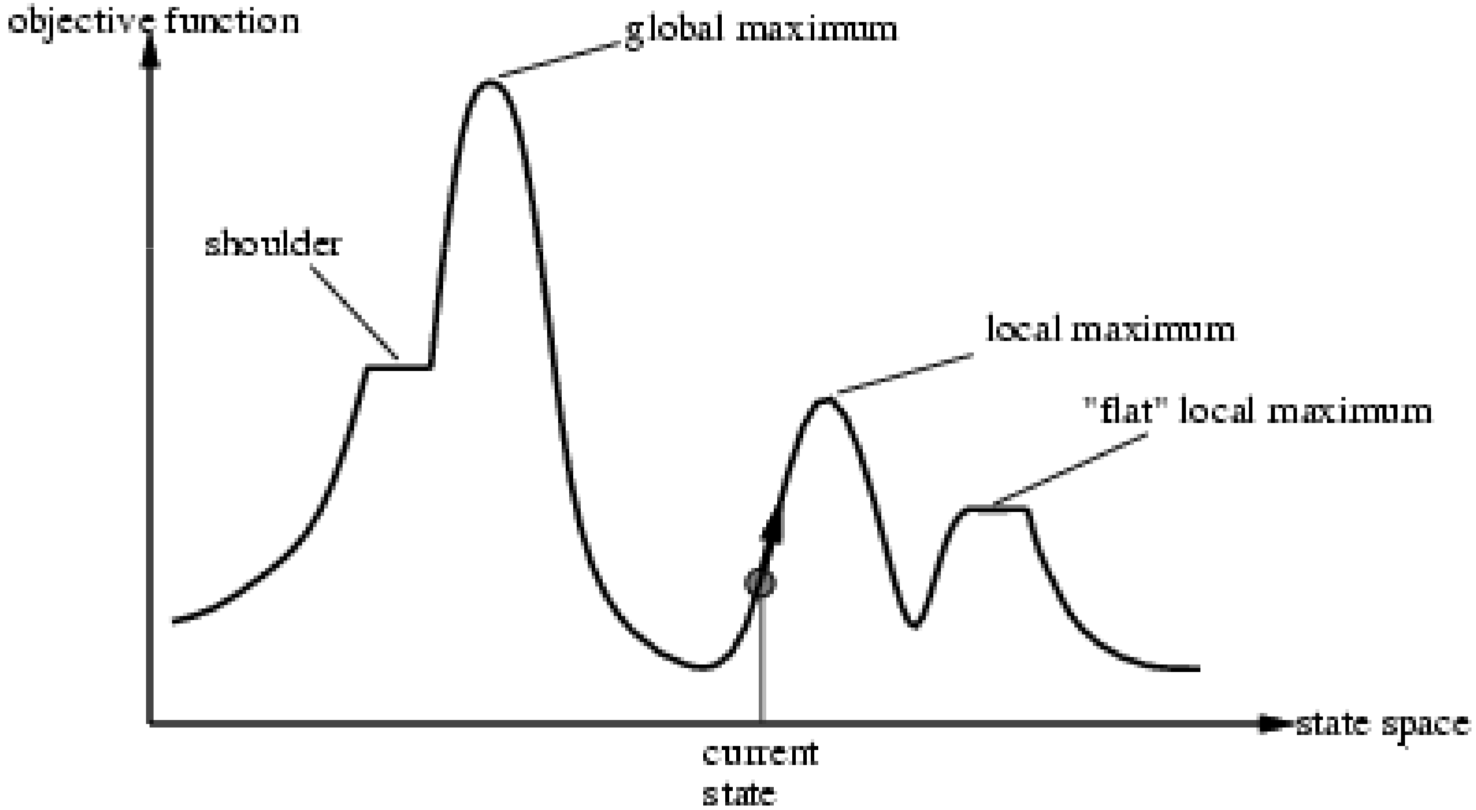
- Escape from local optima

  of evaluation/cost/objective function
- Need ways to re-start the search
  - Partial or global
- Intensification vs. diversification
  - faster toward optimum (but maybe local…)
  - diversify the search
- Shape / ruggedness of landscape

  Definition of a good objective function



Local versus Global



Summary: Which Method to use ?

- Graph-based search
  - basic method, e.g. when no structure is known
  - Useful if full path to solution is needed
- Constraint Satisfaction
  - Declarative model
  - Specialized algorithms, programming tools
- Local Search & Metaheuristics
  - When search space is huge
  - Different metaheuristics, different performances
  - Tuning is essential

Lectures

1. Introduction
2. classical A.I. : State-graphs and the A* algorithm
3. Constraint Satisfaction Problems (CSP)
4. Constraint Solving Techniques I
5. Constraint Solving Techniques II (indexicals)
6. Constraint Programming
7. Combinatorial Optimization Problems
8. Local Search techniques
9. Some Metaheuristics:
   Tabu search, simulated annealing
10. Population-based Methods
    Genetic algo., Beam search, //
11. Constraint-based local search
12. Parallel Local Search

No lecture on 12/11 & 12/18

Ressources (1)

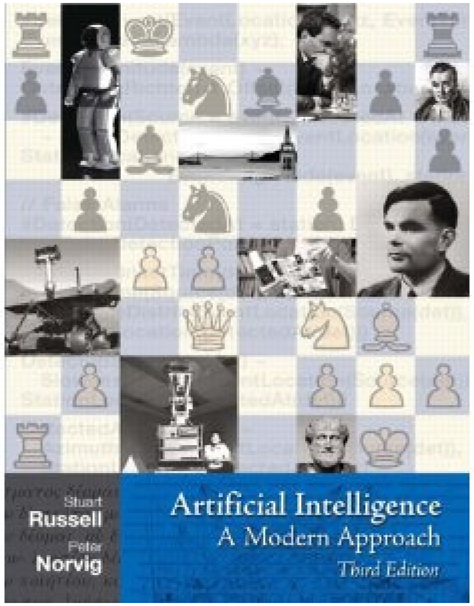
- S. Russell & P. Norvig

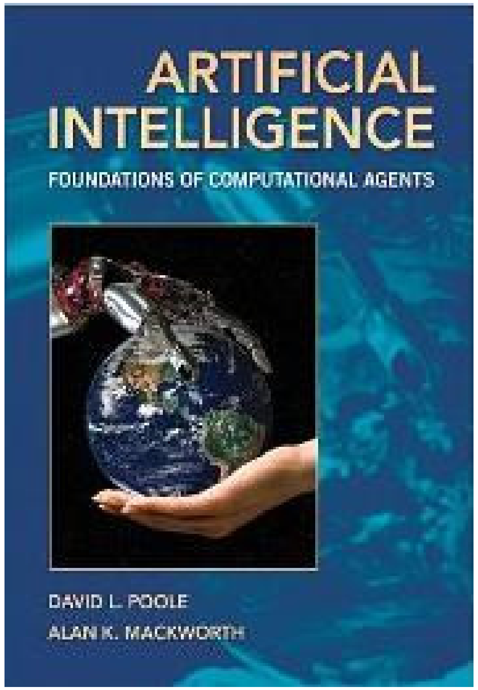
  Artificial Intelligence: A Modern Approach, 3rd edition, Pearson 2010

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

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

  http://artint.info/

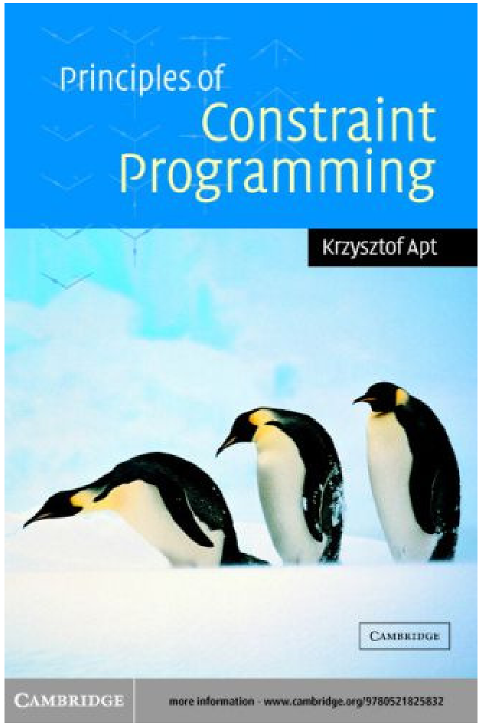




Ressources (2)

- K. Apt

  Principles of Constraint Programming, Cambridge University Press 2003

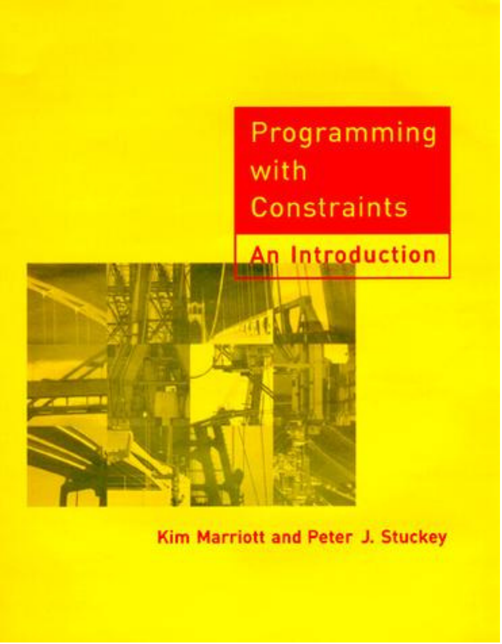


- K. Marriott and P. J. Stuckey

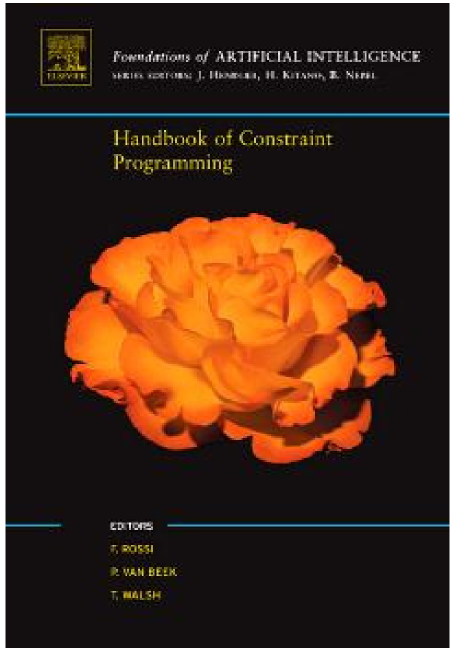
  Programming with Constraints:

  An Introduction

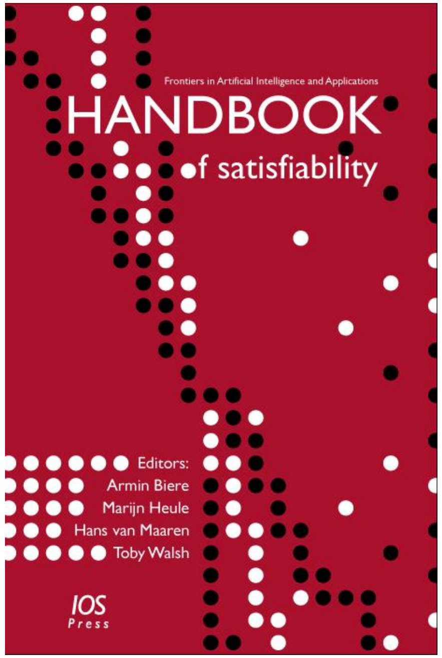
  MIT Press, 1998



Ressources (2')



- F. Rossi, P. Van Beek and T. Walsh
  Handbook of Constraint Programming, Elsevier 2006

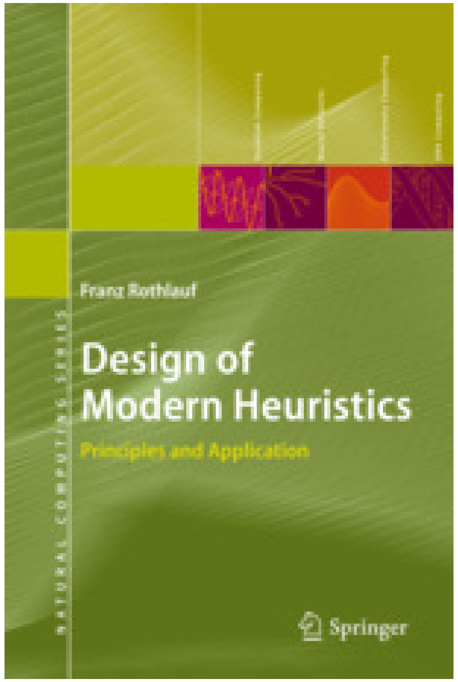


- A. Biere, M. Heule, H. van Maaren & T. Walsh
  Handbook of Satisfiability, IOS Press 2009

Ressources (3)

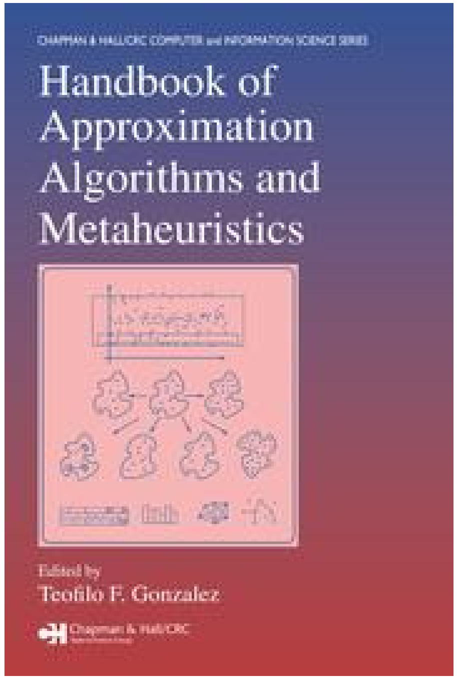
- F. Rothlauf

  Design of Modern Heuristics,
  Springer Verlag 2011



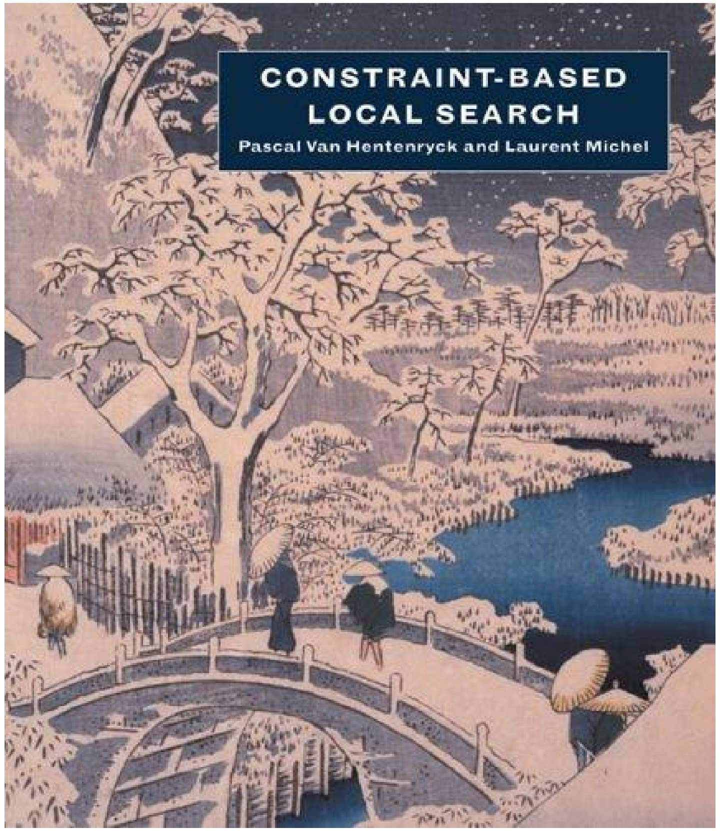
- T. Gonzalez

  Handbook of Approximation
  Algorithms and Metaheuristics,
  Chapman & Hall/CRC 2010



Ressources (4)

- P. Van Hentenryck and L. Michel
  Constraint-based Local Search
  MIT Press 2005



Programming Tools

- Comet 2.1 (CP & LS)
  http://dynadec.com/support/downloads/
- Gecode (CP library for C++)
  http://www.gecode.org/
- GNU Prolog (CLP language)
  http://www.gprolog.org/
- IBM ILOG CP CPLEX Optimizer (IP, MIP & CP)
  http://www-01.ibm.com/software/integration/optimization/cplex-optimization-studio/