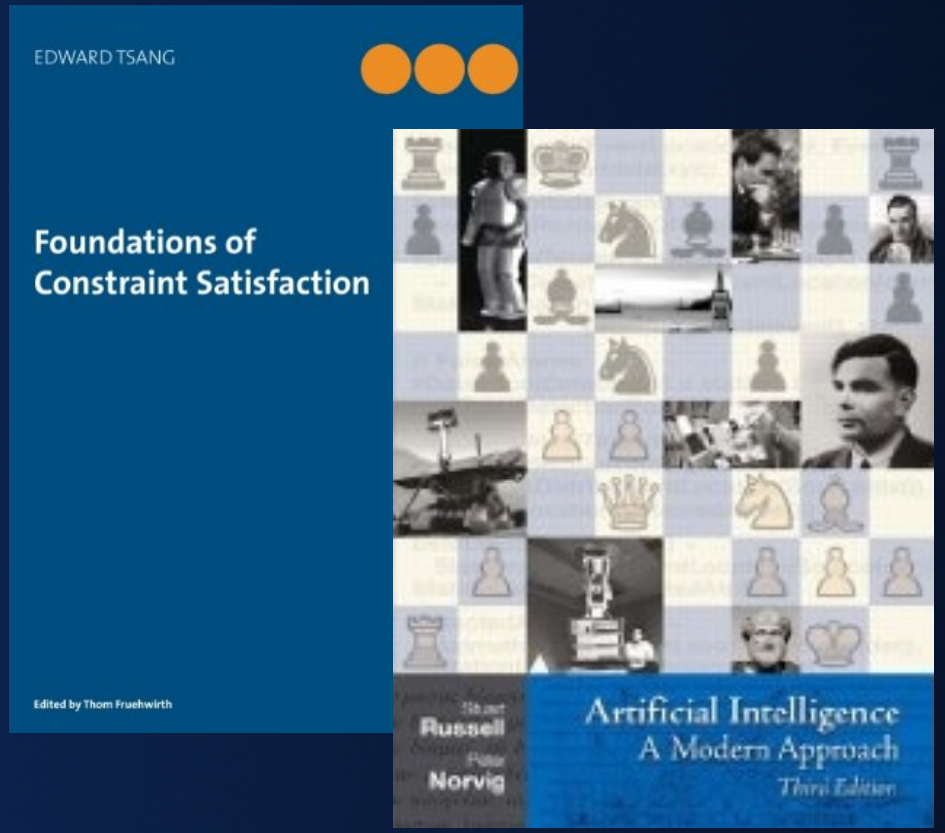


AI Fundamentals: Constraints Satisfaction

Maria Simi

Problem solving

LESSON 1 – PROBLEM SOLVING AS SEARCH

Problem solving as search

The dominant approach to solving problems in AI is formulating a problem as search in a **state space**.

The paradigm is quite general and follows these steps:

1. Define a goal (a set of states)
2. Formulate the problem as a search problem:
   - define a representation for states
   - define legal actions and transition functions
3. Find a solution (a sequence of actions) by means of a search process
4. Execute the plan

Problem structure

A problem can be defined formally by five components

1. Initial state
2. Possible actions in s: $Actions(s)$
3. Transition model: a function $Result$: $state \times action \rightarrow state$
   $Result(s, a) = s'$, a successor state
3. Goal test: a boolean function
   $Goal\text{-}Test\,(s) \rightarrow \{true, false\}$
4. Path cost: function that assigns a numeric cost to each path
   the sum of the cost of the actions on the path c(s, a, s')

Note: 1, 2, 3 implicitly define a graph

Searching algorithms

A **problem** is given as input to a **search algorithm**. A **solution** to a problem is a path (**action sequence**) that leads from the initial state to a goal state.

Solution quality is measured by the **path cost function**: an optimal solution has the lowest path cost among all solutions.

Different strategies (algorithms) for searching the state space and their *quality* (time and space complexity, completeness, optimality …)

- **Uninformed** search methods vs **informed/heuristic** search methods, which use an **heuristic evaluation function** of the nodes. A*.
- Direction of search
- Local search methods

Assumptions in problem solving

Simplifying assumptions (wrt to agent's design):

1. States are treated as **black boxes**, we only need to know their "heuristic value" and whether they are a goal, by applying the Boolean goal function. From the point of view of searching algorithms their internal structure does not matter.
2. The agent has **perfect knowledge** of the state (full accessibility): it knows in which state it is.
3. Actions are **deterministic**: the agent can plan in advance.

On the other hand:

The state space is **generated incrementally**, may not fit in memory, may be infinite.

Dynamic programming

Dynamic programming is a general method for optimization that involves **storing partial solutions to problems**, so that a solution that has already been found can be retrieved in a table rather than being recomputed.

Dynamic programming for graph searching can be seen as constructing the *perfect* heuristic function in advance so that, by keeping only one element of the frontier, it is guaranteed to find an optimal solution. It is based on a **pre-computed table** of the lowest *cost_to_goal(n)* value for each node.

The main limitations of dynamic programming are that:

- it only works when the **graph is finite** and the **table small enough** to fit into memory,
- an agent must re-compute a policy for each different goal.

Dynamic programming algorithms are not typical of AI but are used throughout AI.

Constraints satisfaction

LESSON 1 - AN INTRODUCTION AND PROBLEM FORMULATION

Searching vs constraint satisfaction

It is often better to describe states in terms of **features** and then to reason in terms of these features. We call this a **factored representation**.

For real world states it is usually **more natural and efficient** to describe the features that make up the state rather than explicitly enumerating the states. 10 binary features can describe 2^{10}=1024 states.

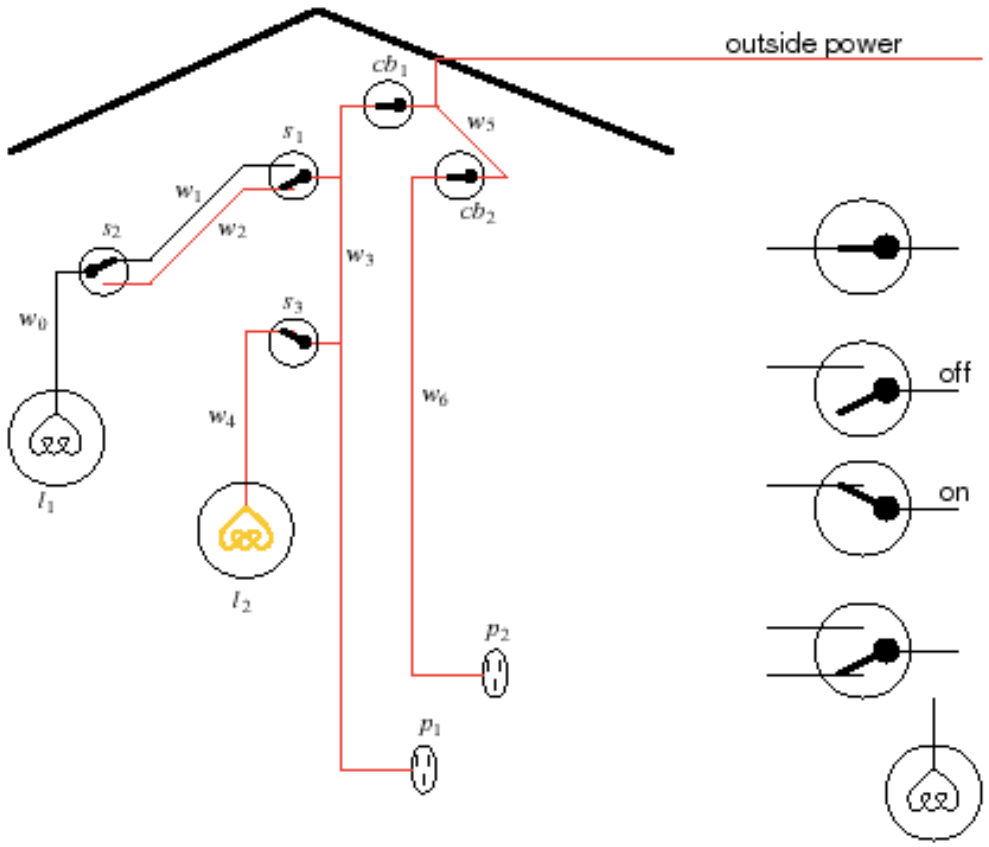
Often these features are not independent and there are **constraints** that specify legal combinations of assignments of values to them.

The mind discovers and exploits constraints to solve tasks. For many important problems this strategy can make problem solving more efficient.

Constraint satisfaction is about **generating assignments** that satisfy a set of hard constraints and how to optimize a collection of soft constraints (**preferences**).

A powerful mechanism spanning a large number of applications.

A feature representation for states



Electrical circuit in a house

- a feature for the position of each switch that specifies whether the switch is up or down.
- a feature for each light that specifies whether the light is lit or no
- a feature for each component specifying whether it is working properly or if it is broken.

A state consists of the position of every switch, the status of every device, and so on, i.e. an assignment of a value to each feature.

For example, a state may be described as switch 1 is up, switch 2 is down, fuse 1 is okay, wire 3 is broken, … the role of constraints.

What is a constraint satisfaction problem?

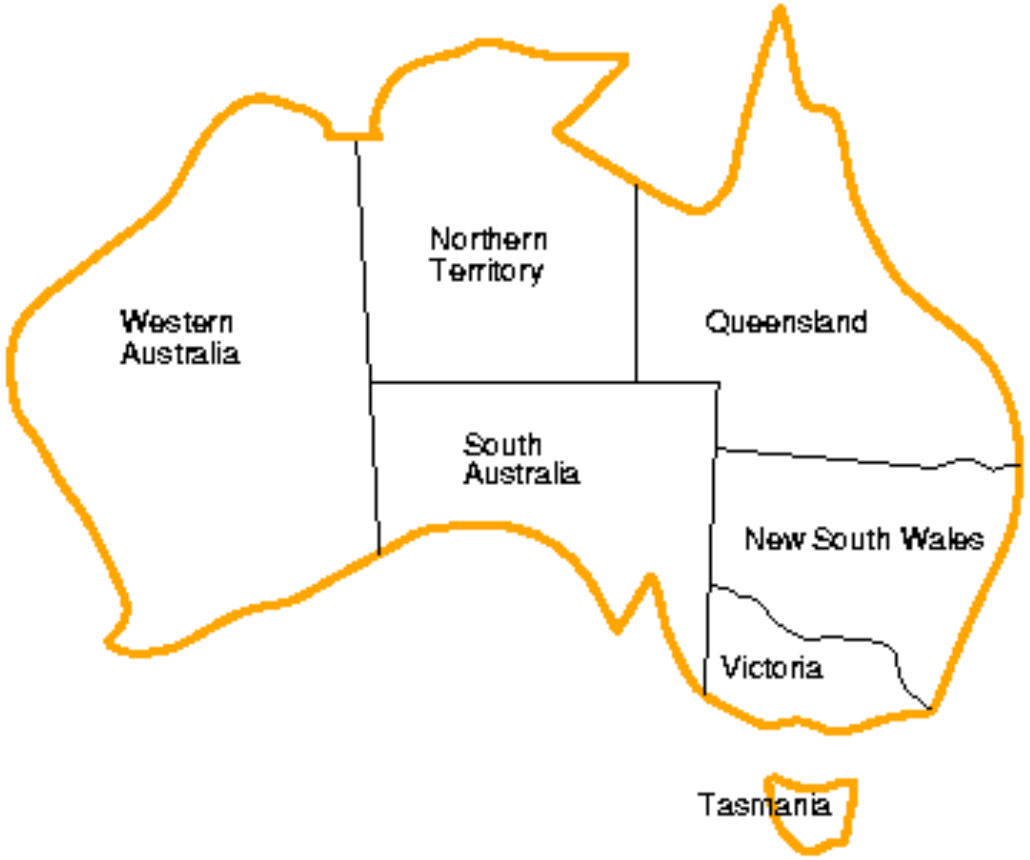
A CSP is a problem composed of

- a finite set of **variables**,
- each variable is associated with a **finite domain**
- and a **set of constraints** that restrict the values the variables can simultaneously take.

The task is to assign a value (from the associated domain) to each variable satisfying all the constraints

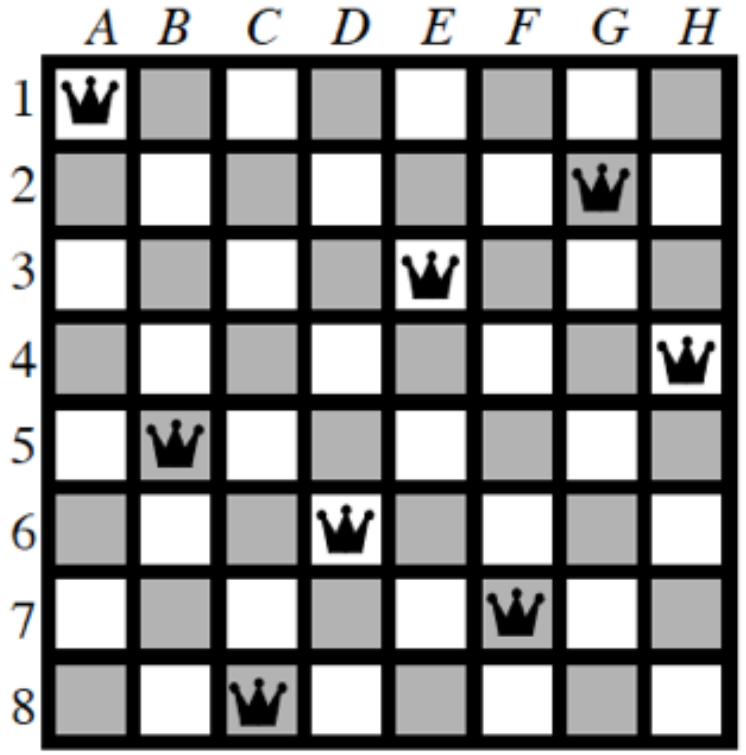
The problem in NP hard in the worst cases but general heuristic exist and structure can be exploited for efficiency.

Map coloring problem



The task is to color regions on the map with three colors in such a way that no two adjacent regions get the same color.

The *N*-queens problem



The problem is to place *N* queens on *N* different squares on a *N* x *N* chess board, satisfying the constraint that no two queens can threaten each other.

Remind: a queen can threaten any other pieces on the same row, column or diagonal.

The figure shows a solution to the 8-queens problem

Using constraints to solve problems

1. Informal problem description
2. Formalization of a constraint satisfaction problem by defining the set of variables, their domains and all the relevant constraints
3. Choose a model and an algorithm for solving the problem (or use constraint solving packages)
4. Implementation and actually solving the problem

There are many different ways to formulate the same problem; efficiency may vary

Different models and approaches: systematic search or repair/stochastic approach?

CSP: a formal *definition*

A Constraint Satisfaction Problem consists of three components, X, D, and C

$$CSP = \langle X, D, C \rangle$$

1. X is a set of variables, $\{X_1, \ldots, X_n\}$
2. D is a set of domains, $\{D_1, \ldots, D_n\}$, one for each variable.

The **domain** of a variable x is a set of all possible values $\{v_1, \ldots, v_k\}$ that can be assigned to the variable x

$Dom =$ a function which maps every variable in X to a set of objects of arbitrary type

$Dom(x) = D_x$ is the domain of x, the set of possible values for x

Domains can be numerical (integer or real numbers), boolean, symbolic ...

3. C is a set of constraints that specify allowable combinations of values

CSP: assignment

A **[partial] assignment** of values to a set of variables (also called **compound label**) is a set of pairs:

$$A = \{\langle x_i, v_i \rangle \langle x_j, v_j \rangle \dots\}$$

where values are taken from the variable domain: $v_i \in D_{xi}$

A **complete assignment** is an assignment to all the variables of the problem

A complete assignment can be **projected** to a smaller partial assignment by restricting the variables to a subset. We will use this notation for the projection:

$$\pi_{x1, \dots, xk} A$$

Where π is the projection operator of relational algebra

CSP: constraints representation

A **constraint on a set of variables** is a set of possible assignments for those variables

Each constraint C can be represented as a pair $\langle scope, rel \rangle$

- $scope$ is a tuple of variables that participate in the constraint $(x_1, x_2, \ldots x_k)$
- rel is a relation that defines the allowable combination of values for those variables, taken from their respective domains

The relation can be represented as:

- an explicit list of all tuples of values that satisfy the constraint, explicit relation.
- an implicit relation, an object that supports two operations: (1) testing if a tuple is a member of the relation and (2) enumerating the members of the relation.

We will also use $C_{x1, x2, \ldots xk} = rel$ to denote a constraint on the scope variables $x_1, x_2, \ldots x_k$: i.e. the constraint $C = \langle (x_1, x_2, \ldots x_k), rel \rangle$

Examples of constraint

Example 1

If x_1 and x_2 both have the domain $\{\mathrm{A}, \mathrm{B}\}$, the constraint that the two variables must have different values can be written as

$\langle (x_1, x_2), \{(\mathrm{A}, \mathrm{B}), (\mathrm{B}, \mathrm{A})\} \rangle$ by explicit enumeration $\mathrm{C}_{\mathrm{x1,\,x2}} = \{(\mathrm{A}, \mathrm{B}), (\mathrm{B}, \mathrm{A})\}$

$\langle (x_1, x_2), x_1 \neq x_2 \rangle$ by implicit definition

$C_{x1,\,x2} = \{(\mathrm{v}_1, \mathrm{v}_2) \mid \mathrm{v}_1 \in D_{x1}, \mathrm{v}_2 \in D_{x2}, \mathrm{v}_1 \neq \mathrm{v}_2\}$

Example 2 (ternary constraint)

$X = \{a, b, c\}$ $\mathrm{D}_a = \mathrm{D}_b = \mathrm{D}_c = \{1, 2, 3, 4, 5, 6\}$

$C_{a,b,c} = \{(1, 2, 3), (1, 4, 3), (4, 5, 6)\} =$

$\{\langle a, 1\rangle, \langle b, 2\rangle, \langle c, 3\rangle), (\langle a, 1\rangle \langle b, 4\rangle \langle c, 3\rangle), (\langle a, 4\rangle \langle b, 5\rangle \langle c, 6\rangle)\}$

Constraint satisfaction

Satisfies is a binary relationship between an assignment and a constraint:

$$Satisfies(\{\langle x_1,v_1\rangle\ \langle x_2,v_2\rangle\ \dots \langle x_k,v_k\rangle\}, C_{x1,\ x2\ \dots xk}) \equiv$$
$$\{\langle x_1,v_1\rangle\ \langle x_2,v_2\rangle\ \dots \langle x_k,v_k\rangle\} \in C_{x1,\ x2\ \dots xk}$$

Example:

$$Satisfies(\{\langle x_1, A\rangle, \langle x_2, B\rangle\}, \langle (x_1, x_2), \{(A, B), (B, A)\}\rangle)$$

$$Satisfies(\{\langle x_1, B\rangle, \langle x_2, A\rangle\}, \langle (x_1, x_2), x_1 \neq x_2\rangle)$$

$$NOT\ Satisfies(\{\langle x_1, B\rangle, \langle x_2, B\rangle\}, \langle (x_1, x_2), \{(A, B), (B, A)\}\rangle)$$

CSP: solution

To solve a CSP problem $\langle X, D, C \rangle$, we need to define a **state** space and the notion of a **solution**.

A **state** in a CSP is an **assignment** of values to some or all of the variables

Consistent assignment: one that satisfies all the constraints

- $Satisfies(\{\langle x_1, v_1\rangle \langle x_2, v_2\rangle \ldots \langle x_k, v_k\rangle\}, C_{x1,\, x2 \ldots xk})$ for any constraint in C

Partial/complete assignment:

- A **partial** assignment is one that assigns values to only some of the variables.
- A **complete** assignment is one in which every variable is assigned

A **solution** to a CSP is a **consistent, complete** assignment.

Types of constraints

The simplest kind of CSP involves variables that have **discrete**, **finite** domains

- Values can be numbers, strings, Boolens (True, False)

When variables are numbers, and the constraints are inequalities will can deal with variables with infinite domains or continuous domains with linear or integer programming (techniques used in Operations research).

According to the variables involved constraints can be:

- unary (ex. “x even”)
- binary (ex. “x > y”)
- higher-order constraints (ex. x+y = z)

Absolute/hard vs soft/preference constraints

- CSPs with preferences can be solved by optimization methods. These are called Constraint Optimization problem, or COP.

Problem formalization: examples

Map coloring

The *N*-queens problem

Scheduling

Car sequencing

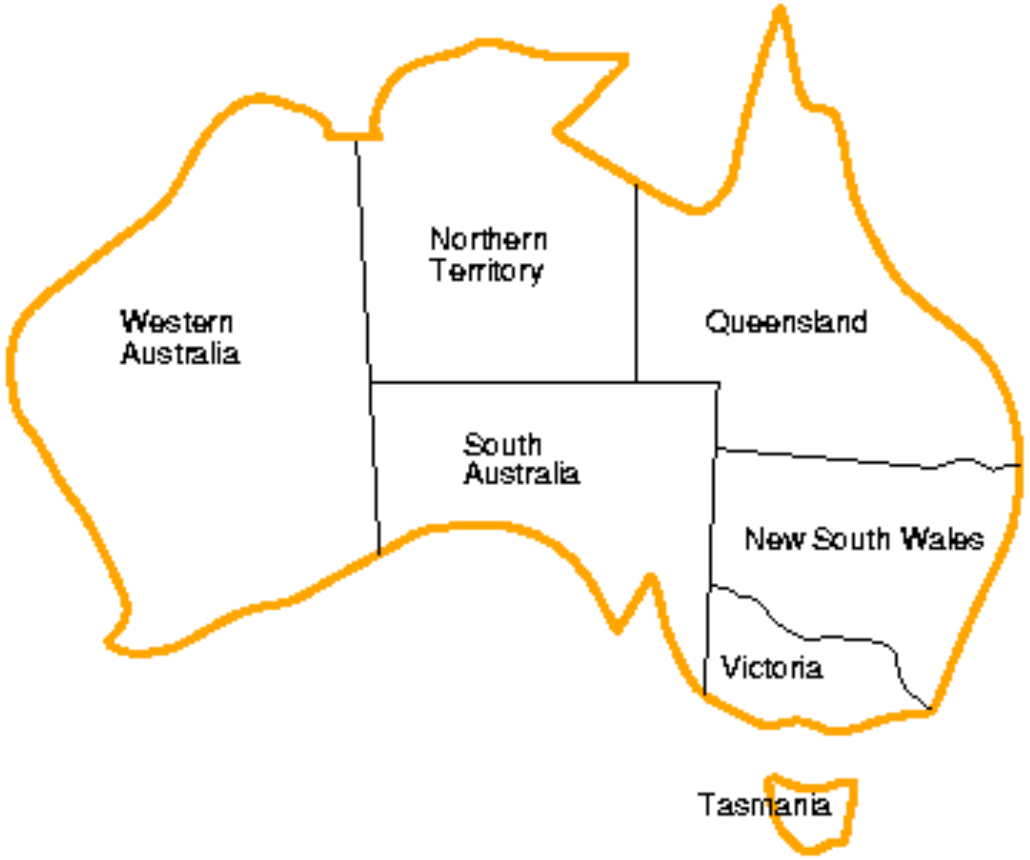
Scene labelling in vision

Temporal reasoning in planning

Subgraph matching in semantic networks

… other

Map coloring: constraint graph



Variables: *WA, NT, SA, Q, NSW, V, T*

Domains: {red, green, blue}

Constraints: *WA≠ NT, WA≠SA, NT≠ Q, SA≠ Q, SA≠ NSW SA≠ V NSW≠ V*

The 8-queens problem: explicit constraints

A queen for each column means 8 variables instead of 64 of naïve formulation.

$X = \{Q_1, Q_2, \ldots, Q_8\}$

$D_{Q1} = D_{Q2} = \ldots = D_{Q8} = \{1, 2, 3, 4, 5, 6, 7, 8\}$

$C = \{\langle (Q_1, Q_2),$ *scope*

$\{(1,3),(1,4),(1,5),(1,6),(1,7),(1,8),$ *rel*

$(2,4),(2,5),(2,6),(2,7),(2,8),(3,2),(3,5),(3,6),(3,7),(3,8),$

$(4,1),(4,2),(4,6),(4,7),(4,8),(5,1),(5,2),(5,3),(5,7),(5,8),$

$(6,1),(6,2),(6,3),(6,4),(6,8),(7,1),(7,2),(7,3),(7,4),(7,5),$

$(8,1),(8,2),(8,3),(8,4),(8,5),(8,6)\}\rangle,$

$\langle (Q_1, Q_3), \ldots\rangle, \langle (Q_1, Q_4), \ldots\rangle, \langle (Q_1, Q_5), \ldots\rangle \langle (Q_1, Q_6), \ldots\rangle\}$

The 8-queens problem: implicit constraints

A queen for each column.

$X = \{Q_1, Q_2, \ldots, Q_8\}$

$D_{Q1} = D_{Q2} = \ldots = D_{Q8} = \{1, 2, 3, 4, 5, 6, 7, 8\}$

(1) *not the same row*

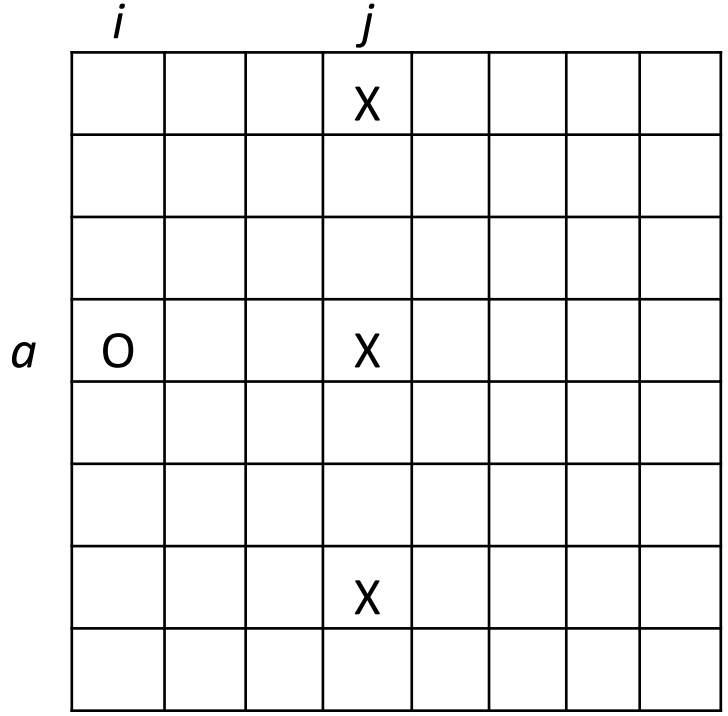
$\forall\, i, j \,.\, Q_i \neq Q_j$

(2) *not the same diagonal*

$\forall\, i, j \,.\, \text{If } Q_i = a \wedge Q_j = b,$

then $i - j \neq a - b$ and $i - j \neq b - a$ [Tsang]

Easy to write a function checking for constraint.



Job-Shop scheduling [AIMA]

Scheduling the assembling of a car, a job requiring several tasks; for example installing axles, installing wheels, tightening nuts, put on hubcap, inspect.

- X initial times of the tasks to be performed
- D finite number of times in an interval (minutes)
- C, temporal constraints among tasks

  - *Precedence constraints*: *i* must be completed before *j* begins

    $X_i + d_i < X_j$ *where* d_i *is the duration of task* i

  - *Disjunctive constraints between* i *and j, not overlapping in time* (two workers and one tool)

    $X_i + d_i < X_j$ or $X_j + d_j < X_i$

  - *Global duration* of the planned assembly jobs.

    Limitation on the interval of time considered.

Job-shop scheduling: example

$X = \{AxleF, AxleB, WheelRF, WheelLF, WheelRB, WheelLB, NutsRF, NutsLF, NutsRB, NutsLB, CapRF, CapLF, CapRB, CapLB, Inspect\}$

Precedence constraints:

$AxleF + 10 \leq WheelRF$ $\quad WheelRF + 1 \leq NutsRF$ $\quad NutsRB + 2 \leq CapRB$

$AxleF + 10 \leq WheelLF$ $\quad WheelLF + 1 \leq NutsLF$ $\quad NutsRF + 2 \leq CapRF$

$AxleB + 10 \leq WheelRB$ $\quad WheelRB + 1 \leq NutsRB$ $\quad NutsLF + 2 \leq CapLF$

$AxleB + 10 \leq WheelLB$ $\quad WheelLB + 1 \leq NutsLB$ $\quad NutsLB + 2 \leq CapLB$

Disjunctive constraint:

$(AxleF + 10 \leq AxleB)$ or $(AxleB + 10 \leq AxleF)$

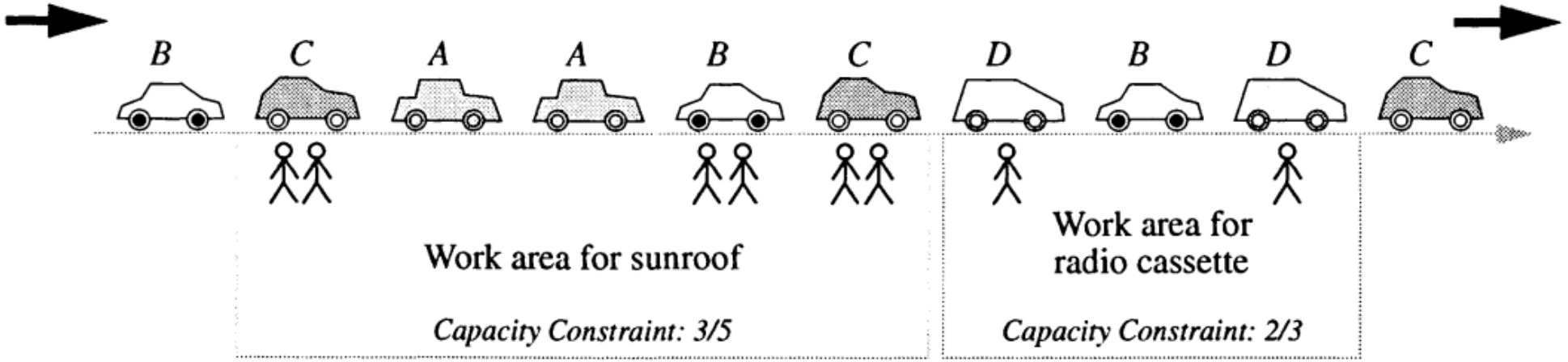
$X + d_X \leq Inspect$ $\qquad D_i = \{1, 2, 3, \ldots, 27\}$

Allowed time: 30 minutes

The car sequencing problem [Tsang]

Production Requirements:

Options (✓ = required, × = not):	Model A	Model B	Model C	Model D	
Sunroof	×	✓	✓	×	
Radio cassette	✓	×	✓	✓	
Air-conditioning	✓	✓	×	✓	
Anti-rust treatment	×	✓	✓	✓	
Power brakes	✓	×	✓	×	
Number of cars required:	30	30	20	40	Total: 120



The car sequencing problem: description

Cars are placed on conveyor belts which move through different work areas.

Work areas specialize to do a particular job: fitting sunroofs, car radios or air-conditioners, **depending on the model**.

When a car enters a work area, a team of engineers travels with the car while working on it. They must have enough time to finish the job while the car is in their work area. E.g. **the capacity of the work area** for fitting roofs is 5 cars: the job takes 20 min and the cars enter every 4 min.

Each work area has **capacity constraints**, for example if the number of teams fitting sunroofs is 3, they cannot deal with more than 3/5 cars.

A car-sequencing problem is specified by the production and option requirements and the capacity constraints. The output is a proper sequencing of the cars to be produced.

Car sequencing formulation: hint

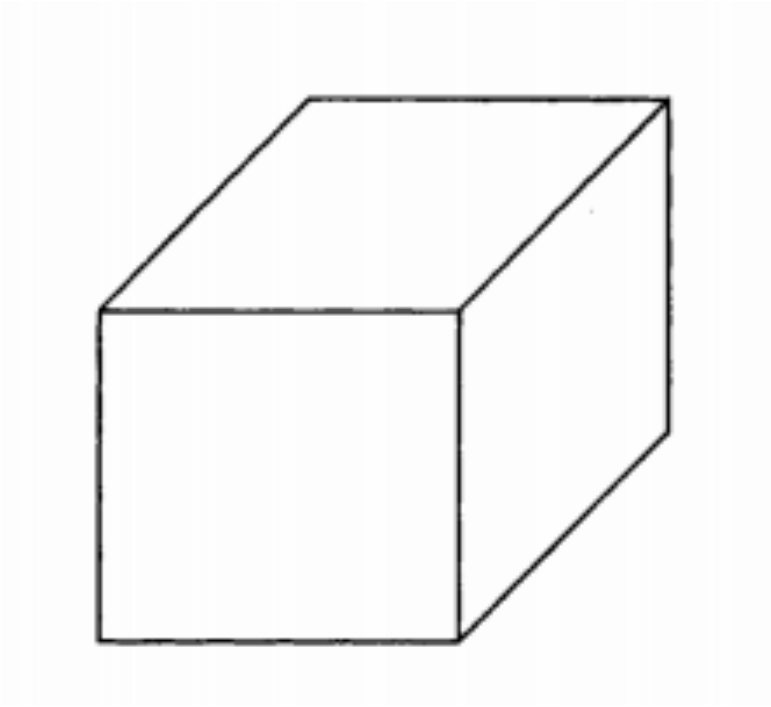
The car-sequencing problem can be formulated as a CSP in the following way.

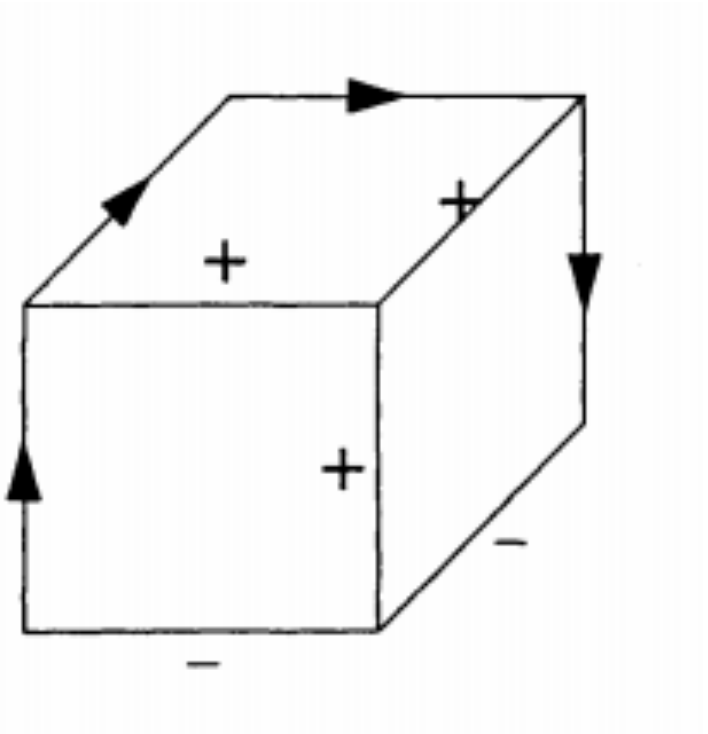
n variables: one for each car position on the conveyor belt

The domain of each variable is the set of car models: {A, B, C, D}

The task is to assign a value (a car model) to each variable (a position in the conveyor belt), satisfying both the production requirements and capacity constraints.

Scene labelling [Tsang]





The scene labelling problem [Tsang]

To recognize the objects in the scene, one must first interpret the lines in the drawings. One can categorize the lines in a scene into the following types:

(1) **convex edges** "+"

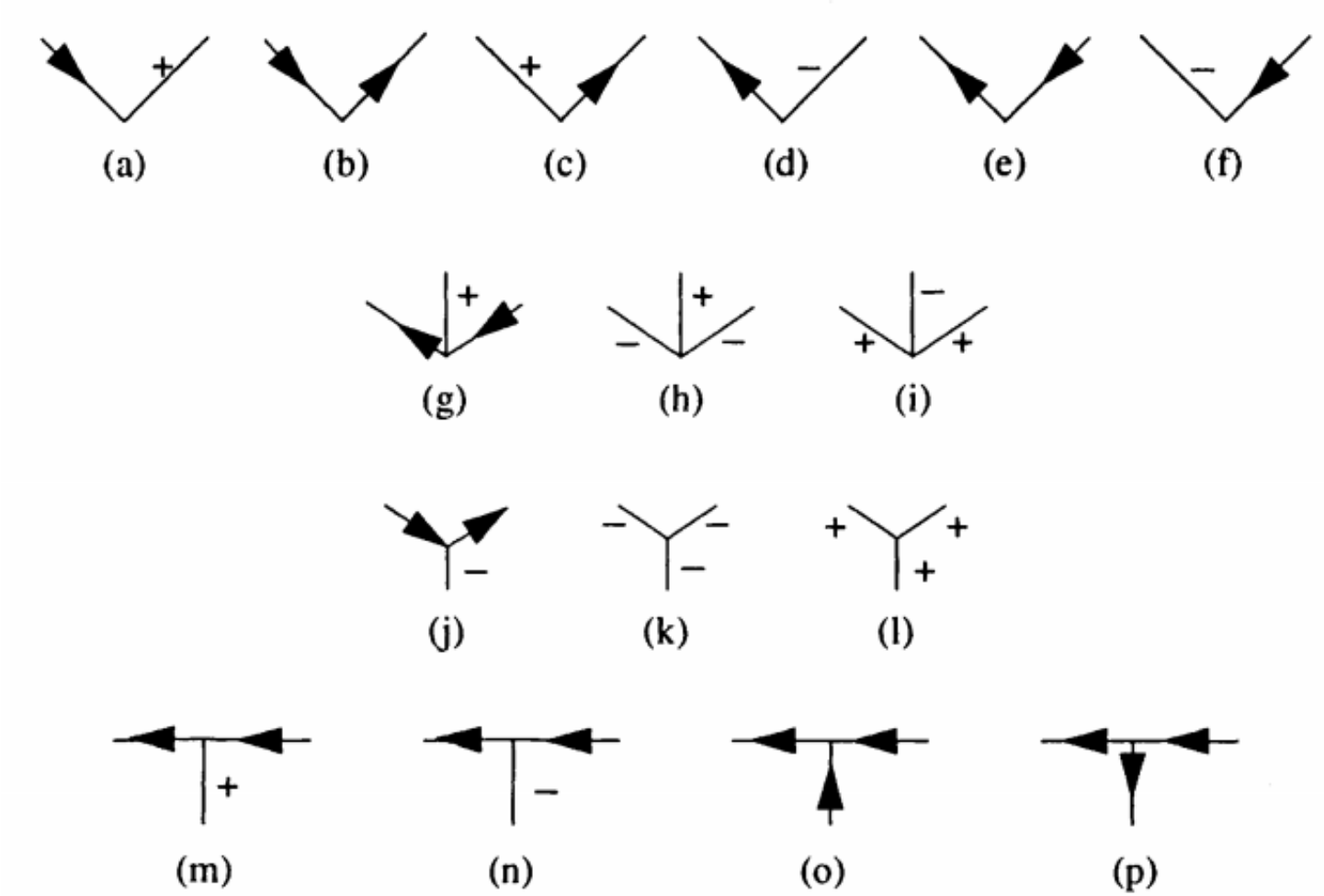
(2) **concave edges** "– "

(3) **occluding edges**

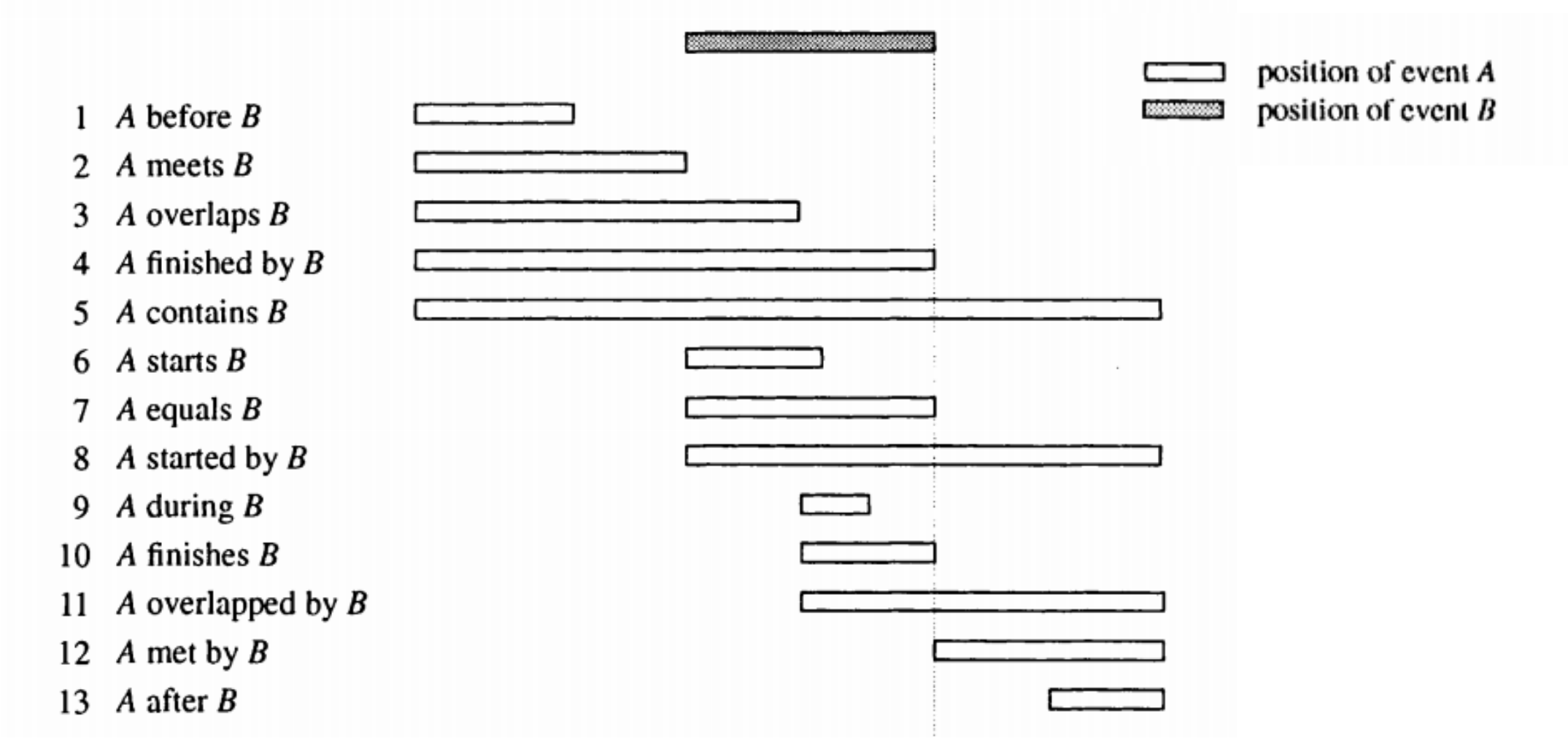
An occluding edge is a convex edge where one of the planes is not seen by the viewer. Occluding edges are marked with arrows according to whether moving in the direction of the arrow the visible plane is to right "→" or the left "←".

Constraints are dictated by physical constraints.

Legal labels for junctions



Temporal reasoning



Constraint graphs

Constraint graphs

A binary CSP, is a CSP with unary and binary constraints only.

A binary CSP may be represented as an undirected graph (V, E):

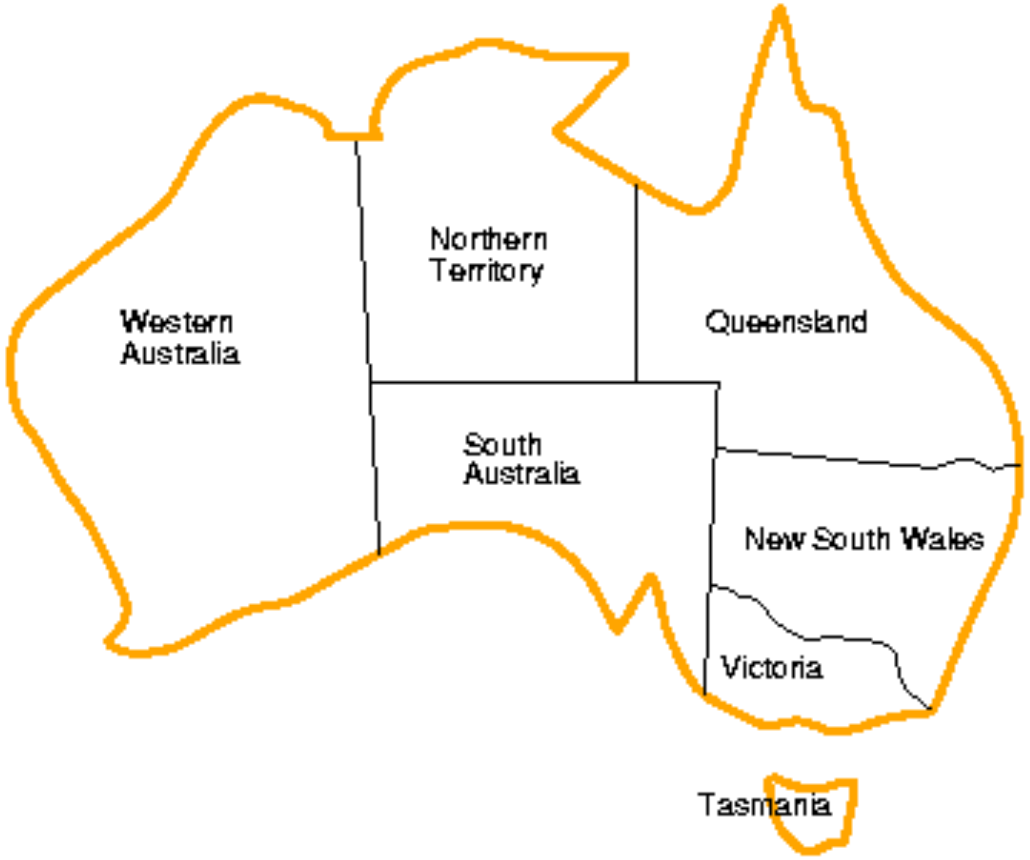
- Nodes correspond to variables (V)
- Edges correspond to binary constraints among variables $(E = V \times V)$

Note: **arcs** have a direction; **edges** are **undirected** arcs; an edge can be seen as a pair of arcs.

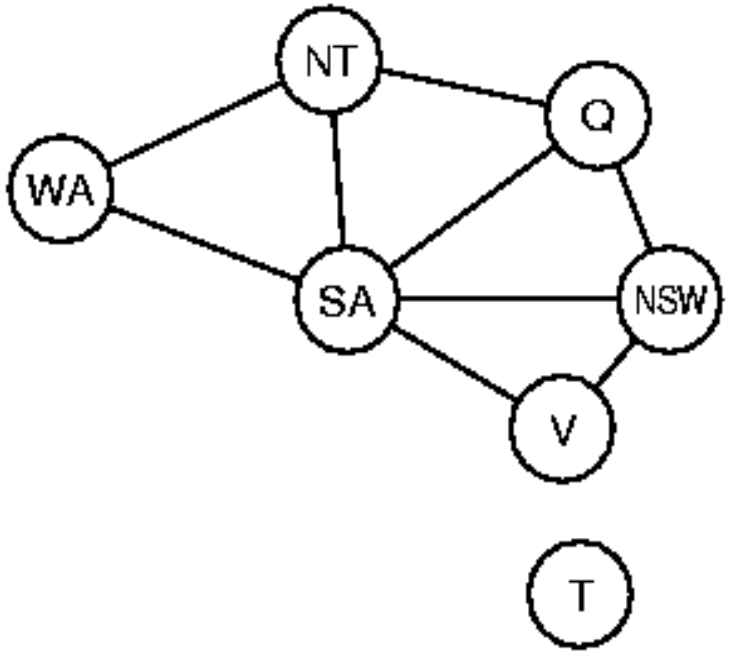
Node x is **adjacent** to node y if and only if (x, y) is in E

A graph is connected if there is a path among any two nodes

Map coloring: constraint graph



Binary constraint graph



Transformation into binary constraints

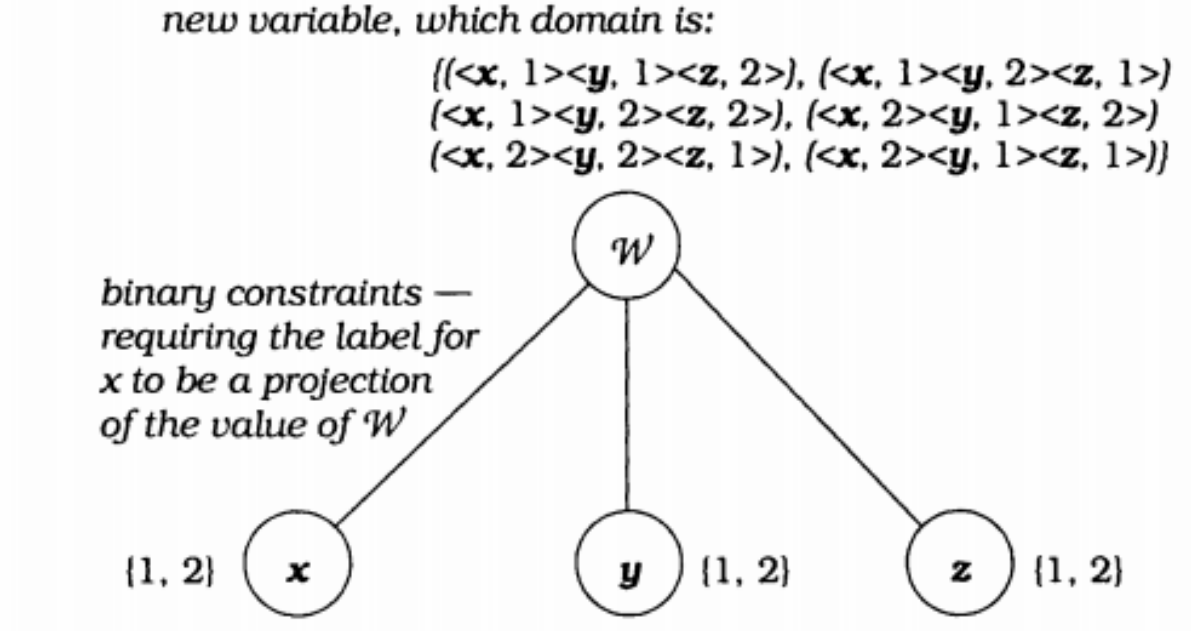
All problems can be transformed into binary constraint problems (not always worthwhile).

Example.

V = {x, y, z}

$D_x = D_y = D_z = \{1, 2\}$

C = { (<x,1>,<y,1>,<z,2>)
(<x,1>,<y,2>,<z,2>)
(<x,1>,<y,2>,<z,1>)
(<x,2>,<y,1>,<z,2>)
(<x,2>,<y,1>,<z,1>)
(<x,2>,<y,2>,<z,1>)}



Not all three variables have the same values

Constraints hypergraphs

In general, every CSP is associated with a constraint hypergraph.

Hypergraphs are a generalization of graphs. In a hypergraph, each hyperedge may connect more than two nodes.

The constraint hypergraph of a CSP (X, D, C) is a hypergraph in which each node represents a variable in X, and each hyperedge represents a higher order constraint in C.

Example: Cryptoarithmetic

Each letter stands for a distinct digit;
the aim is to find a substitution of digits
for letters such that the resulting sum is arithmetically correct

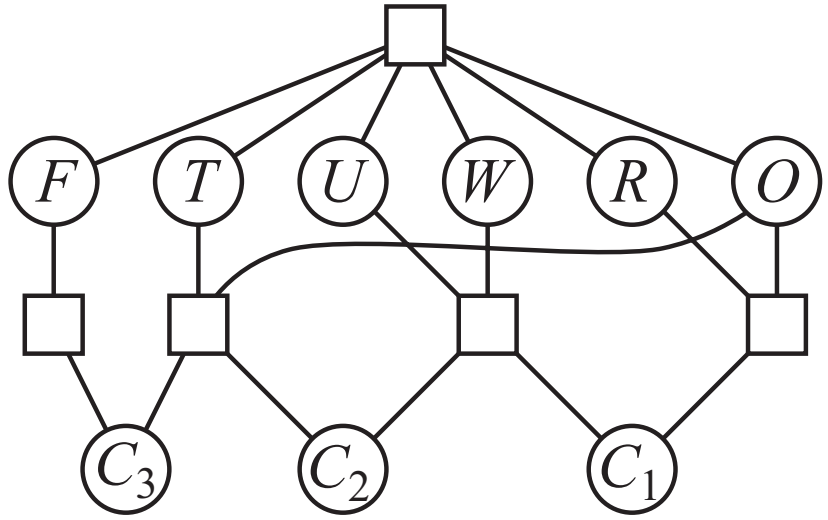
$$\begin{array}{cccc} C3 & C2 & C1 & \\ & T & W & O \\ + & T & W & O \\ \hline F & O & U & R \end{array}$$

(a)

Hypergraph: cryptoarithmetic example

$$\begin{array}{r} T\ W\ O \\ +\ T\ W\ O \\ \hline F\ O\ U\ R \end{array}$$

(a)



(b)

The constraint hypergraph for the cryptarithmetic problem, showing the Alldiff constraint (square box at the top) as well as the column addition constraints (four square boxes in the middle). The variables C1 , C2 , and C3 represent the carry digits for the three columns.

Square nodes are hypernodes representing n-ary constraints

CSP solving techniques: an overview

Problem reduction/Inference/Constraint propagation

- Techniques for transforming a CSP into an equivalent one which is easier to solve or recognizable as insoluble (removing values from domains and tightening constraints).

Searching

- Search in the space of labels: enumerate combinations of labels to find solutions.
- How to search efficiently: heuristics, intelligent backtracking … local search

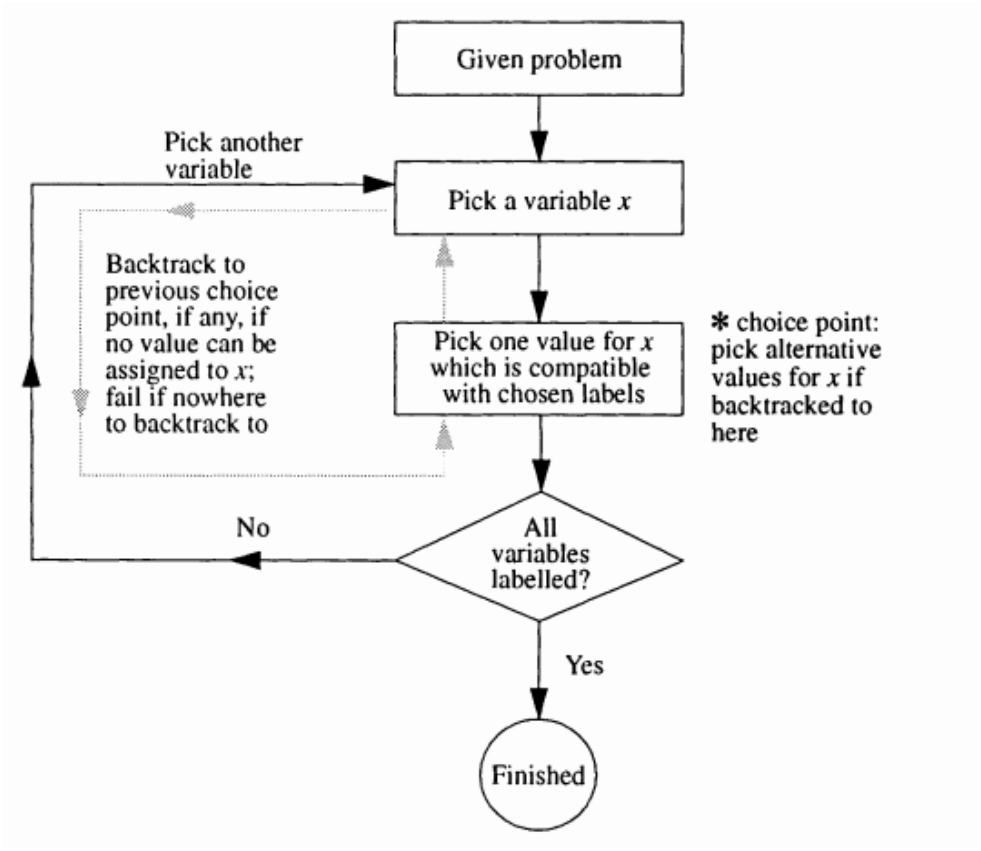
Exploiting the structure of the problem

- Independent sub-problems, tree-structured constraints, tree-decomposition, exploiting symmetry

[Solution synthesis

- Construct and extend partial solutions in order to generate the set of all solution tuples]

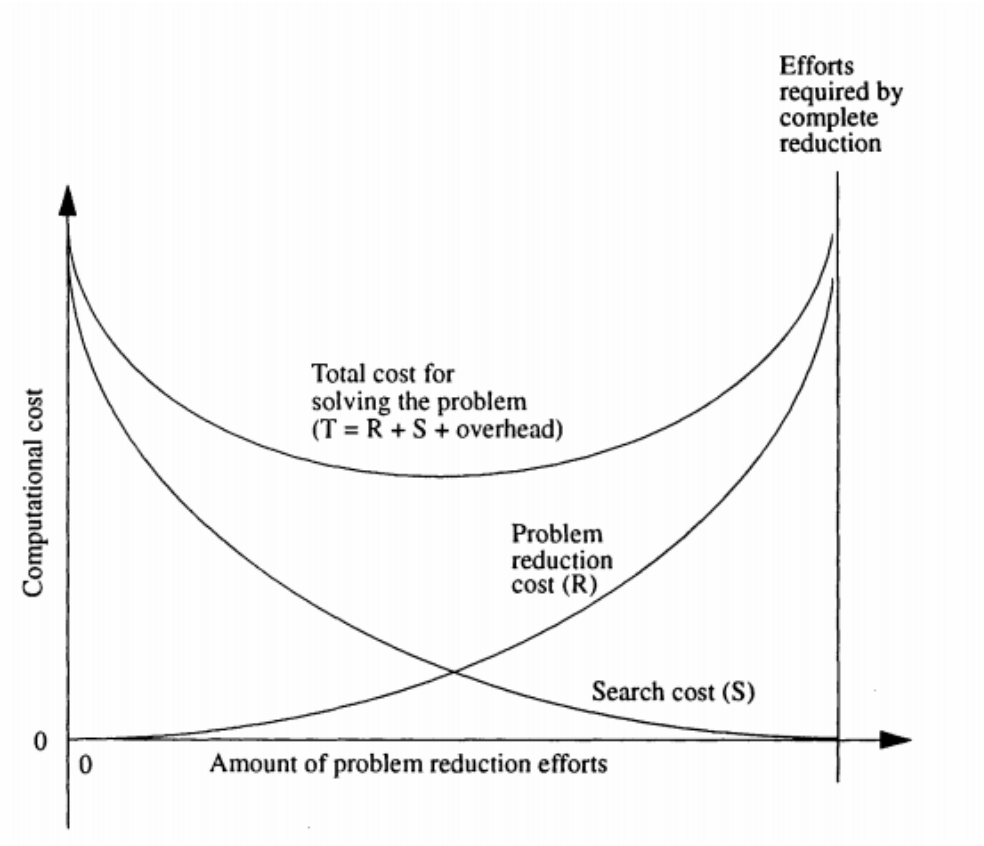
Searching for solution labels
simple backtracking



Choice points in searching:

1. Pick a variable to assign next
2. Pick value for that variable
3. Pick a constraint to work on next

Combining problem reduction and search



Cost of problem reduction vs. cost of backtracking.

The more effort one spends on problem reduction, the less effort one needs in searching

[Tsang]

Solution synthesis

Synthesis techniques constructively generate legal compound labels rather than eliminating redundant labels or redundant compound labels.

One can see solution synthesis as a special case of problem reduction in which the *n*-constraint for a problem with *n* variables is constructed, and all the *n*-assignments which violate some constraints are removed.

Alternatively, solution synthesis can be seen as "searching" multiple partial compound labels in parallel.

Problem characteristics

- Number of solutions required (one / all)
- Problem size (n. of variables and constraints)
- Type of variables and constraints (assume symbolic values, binary constraints)
- Structure of the constraints graph (connectivity, tree form …)
- Tightness of the problem (measured by the number of solution tuples over the number of all distinct compound labels for all variables)
- Quality of solutions (preference among solutions, COP)
- Partial solutions (if no solution exist, find “best” partial solution)

Your turn

Watch the video by Tsang:
https://www.youtube.com/watch?v=wrs6Lvo5LZM&feature=youtu.be

Formalize one of the following as CSP:

- *Sudoku*
- *Zebra puzzle or* **Einstein's Puzzle** [https://en.wikipedia.org/wiki/Zebra_Puzzle]
- *Car sequencing*
- *Scene labelling*

Conclusions

- We introduces CSP and the language to formally define them.
- Several examples representing a number of problems that can be formalized according to this model.
- Types of problems and characteristics that influence the solution.

Next, the techniques for solving CSP:

1. Problem reduction techniques
2. Making the search more efficient
3. Exploiting the problem structure

References

Stuart J. Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach* (3rd edition). Pearson Education 2010 [Cap 6 – CSP]

Edward Tsang, *Foundations of Constraints Satisfaction.*

Handbook of Constraint Programming, Edited by F. Rossi, P. van Beek and T. Walsh. Elsevier 2006.