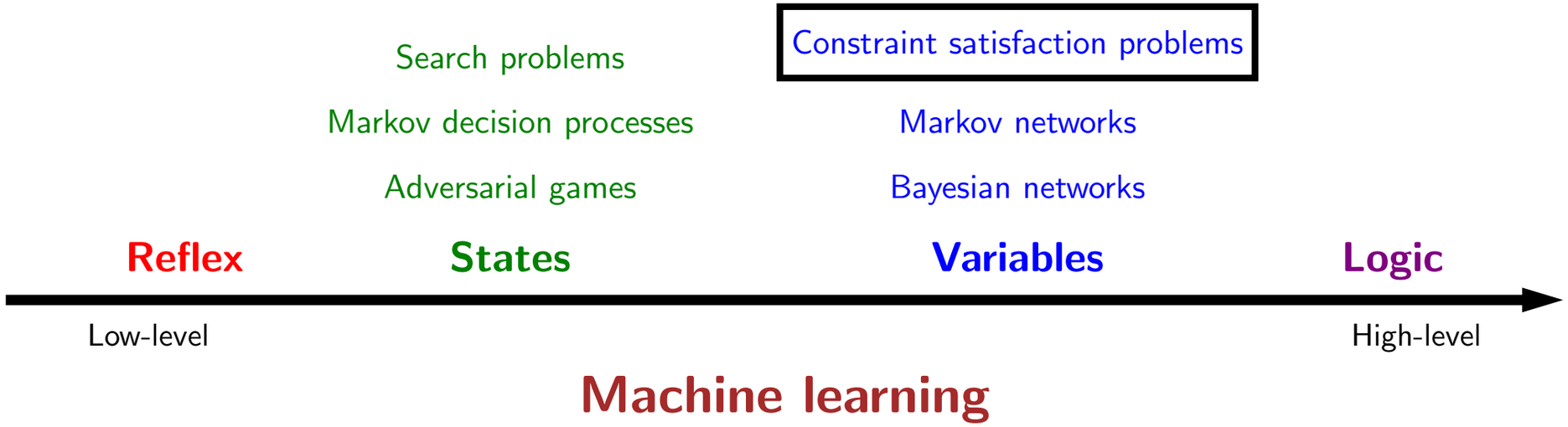
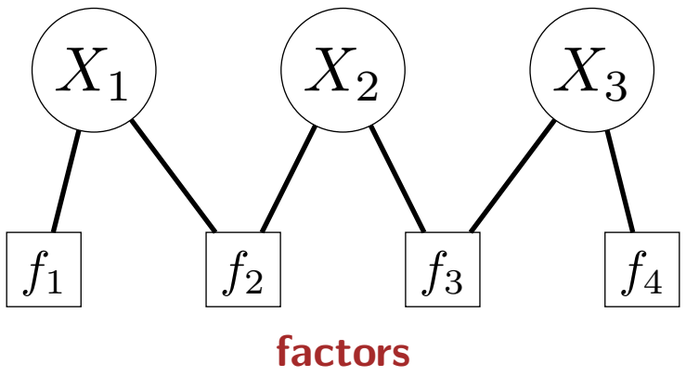
CSPs: overview

Course plan


Search problems
Markov decision processes
Adversarial games
Constraint satisfaction problems
Markov networks
Bayesian networks
Reflex
States
Variables
Logic
Low-level
High-level
Machine learning

Factor graphs

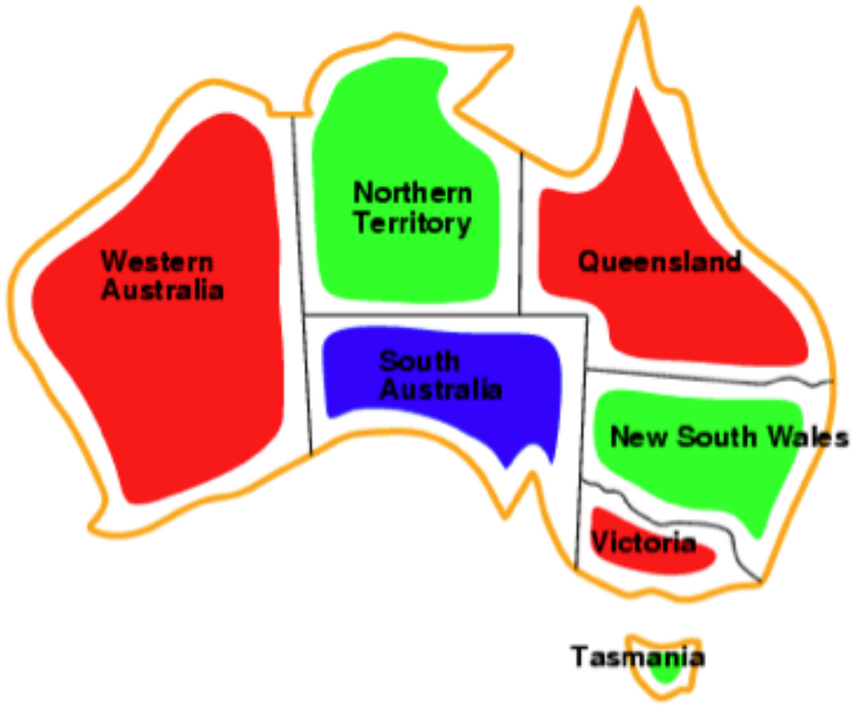


Objective: find the best assignment of values to the variables

Map coloring

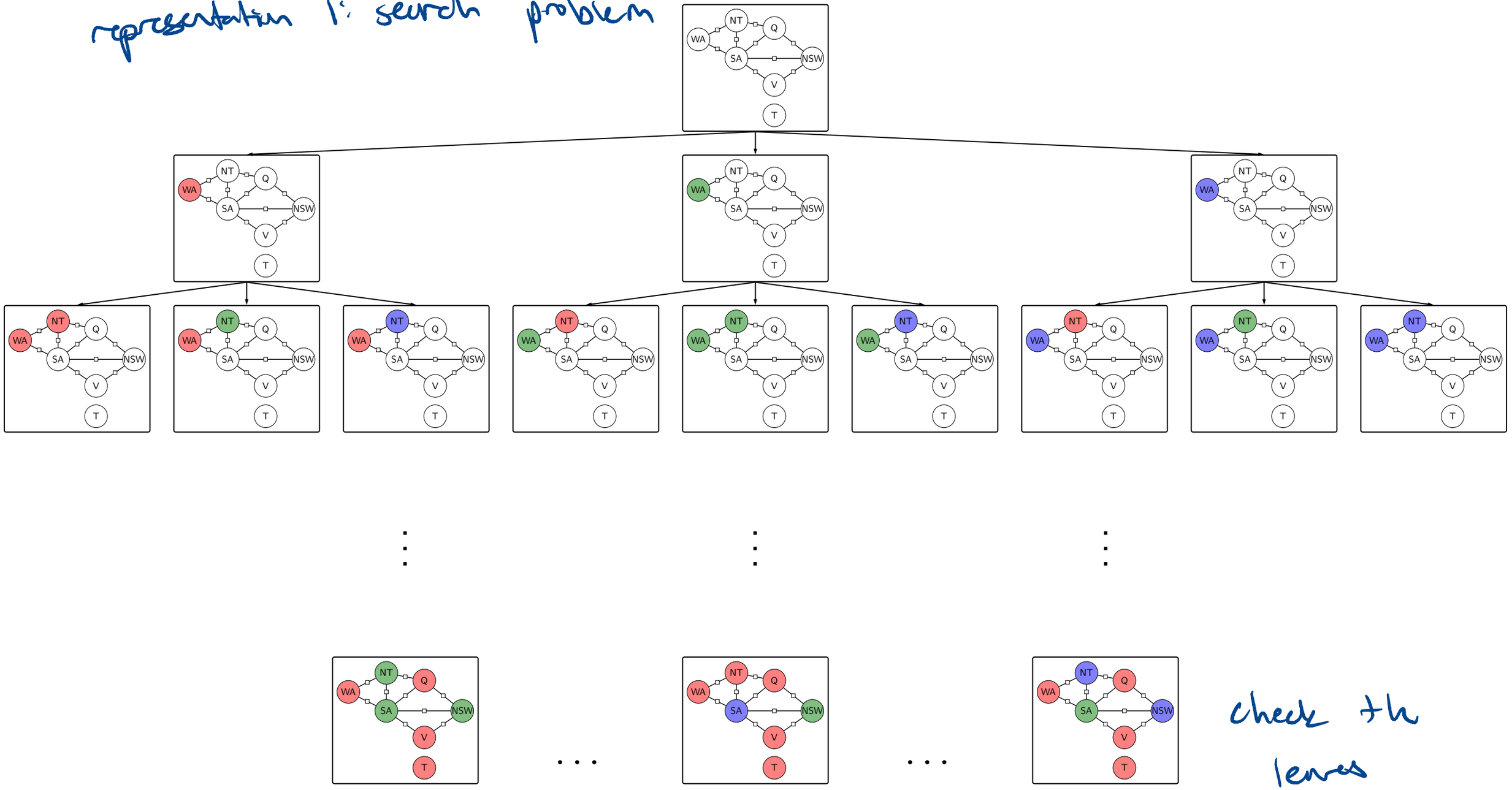
Question: how can we color each of the 7 provinces {red,green,blue} so that no two neighboring provinces have the same color?

Map coloring



(one possible solution)

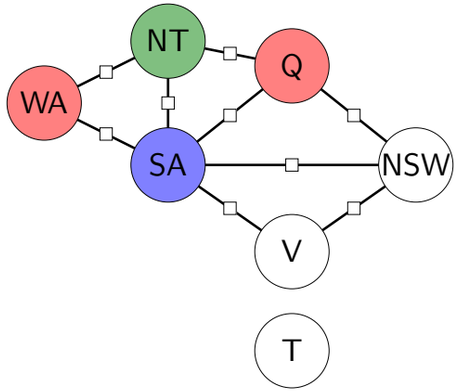
representation 1: search problem



0 0 1

check the leaves

As a search problem



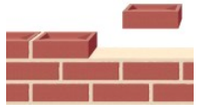
- State: partial assignment of colors to provinces
- Action: assign next uncolored province a compatible color

What's missing? There's more problem structure!

- Variable ordering doesn't affect correctness, can optimize
- Variables are interdependent in a local way, can decompose

we can start anywhere

e.g. T is independent, very easy to solve



Variable-based models

Special cases:

- Constraint satisfaction problems
- Markov networks
- Bayesian networks

Key idea: variables

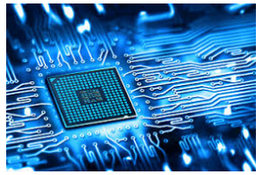
- Solutions to problems $\Rightarrow$ assignments to variables (modeling).
- Decisions about variable ordering, etc. chosen by inference.

Higher-level modeling language than state-based models

Applications

Delivery/routing: how to assign packages to trucks to deliver to customers

Sports scheduling: when to schedule pairs of teams to minimize travel



Formal verification: ensure circuit/program works on all inputs

Roadmap

Modeling

Definitions

Examples

find exact solutions

approximate solutions for efficiency

Backtracking (exact) search

Dynamic ordering

Arc consistency

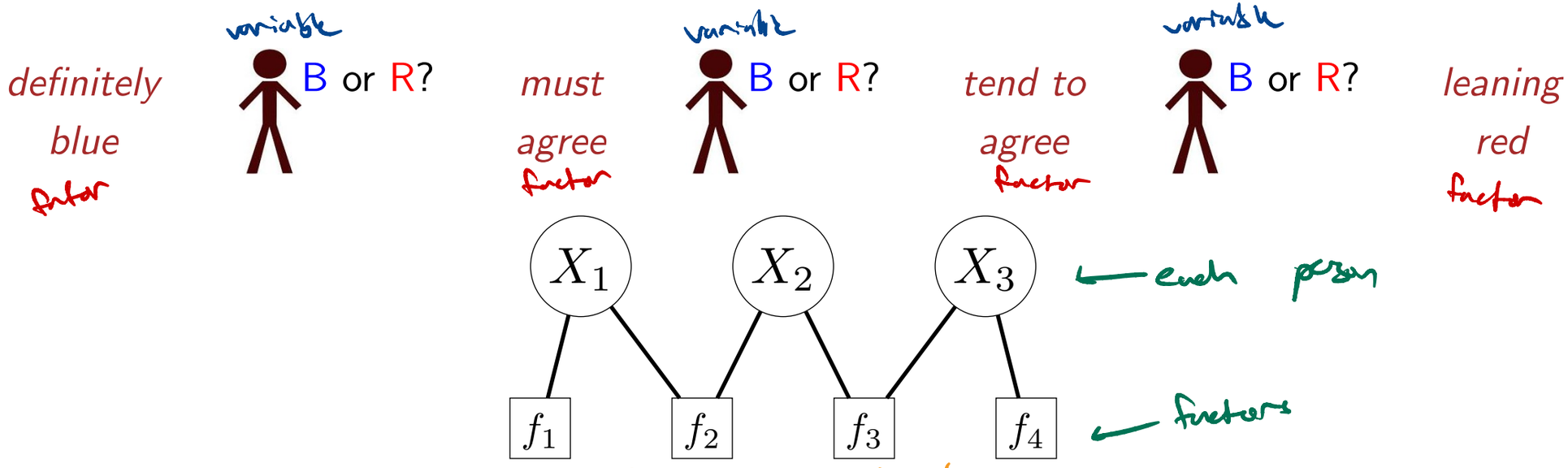
Approximate search

Beam search

Local search

CSPs: definitions

Factor graph example: voting



x_1	$f_1(x_1)$
R	0
B	1

x_1	x_2	$f_2(x_1, x_2)$
R	R	1
R	B	0
B	R	0
B	B	1

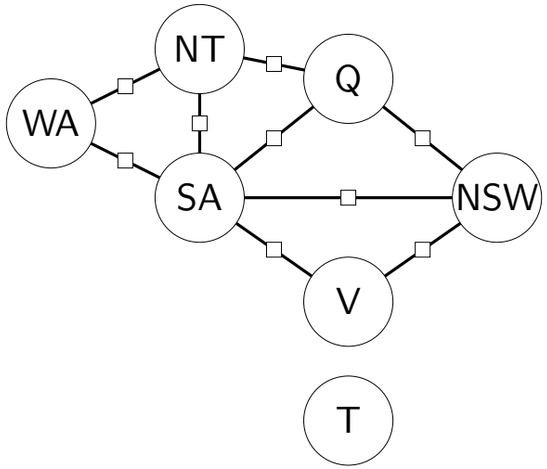
x_2	x_3	$f_3(x_2, x_3)$
R	R	3
R	B	2
B	R	2
B	B	3

x_3	$f_4(x_3)$
R	2
B	1

$f_1(x_1) = [x_1 = \text{B}]$ $f_2(x_1, x_2) = [x_1 = x_2]$ $f_3(x_2, x_3) = [x_2 = x_3] + 2$ $f_4(x_3) = [x_3 = \text{R}] + 1$

[demo]

Example: map coloring



Variables:

$$X = (\text{WA}, \text{NT}, \text{SA}, \text{Q}, \text{NSW}, \text{V}, \text{T})$$

$$\text{Domain}_i \in \{\text{R}, \text{G}, \text{B}\}$$

Factors:

$$f_1(X) = [\text{WA} \neq \text{NT}]$$

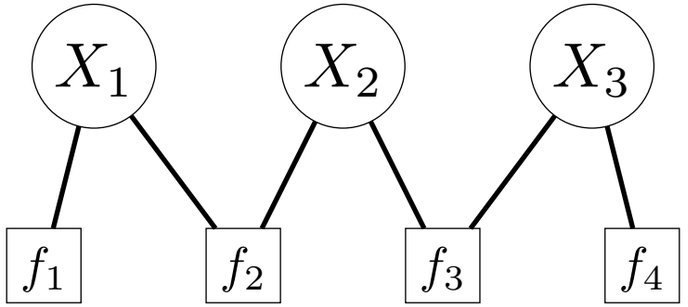
$$f_2(X) = [\text{NT} \neq \text{Q}]$$

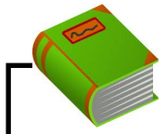
...

color

constraints, can't be same color

Factor graph





Definition: factor graph

Variables:

value of X_i

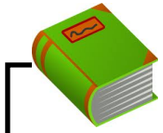
each var has domain

$$X = (X_1, \ldots, X_n), \text{ where } X_i \in \text{Domain}_i$$

Factors:

$$f_1, \ldots, f_m, \text{ with each } f_j(X) \geq 0$$

Factors



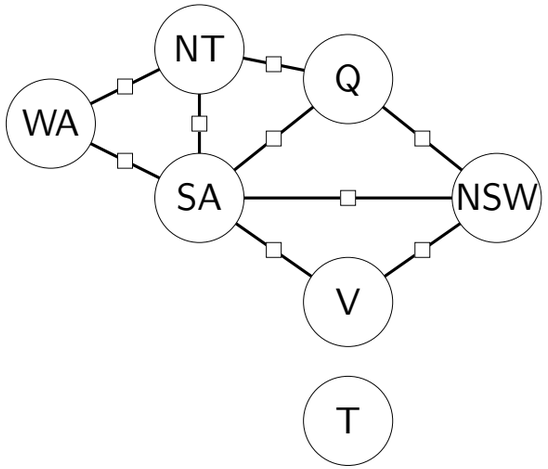
Definition: scope and arity

Scope of a factor f_j: set of variables it depends on.

Arity of f_j is the number of variables in the scope.

Unary factors (arity 1); **Binary** factors (arity 2).

Constraints are factors that return 0 or 1.



Example: map coloring

Scope of $f_1(X) = [\text{WA} \neq \text{NT}]$ is $\{\text{WA}, \text{NT}\}$

f_1 is a binary constraint

Assignment weights example: voting

unary constraint

x_1	$f_1(x_1)$
R	0
B	1

binary constraint

x_1	x_2	$f_2(x_1, x_2)$
R	R	1
R	B	0
B	R	0
B	B	1

x_2	x_3	$f_3(x_2, x_3)$
R	R	3
R	B	2
B	R	2
B	B	3

x_3	$f_4(x_3)$
R	2
B	1

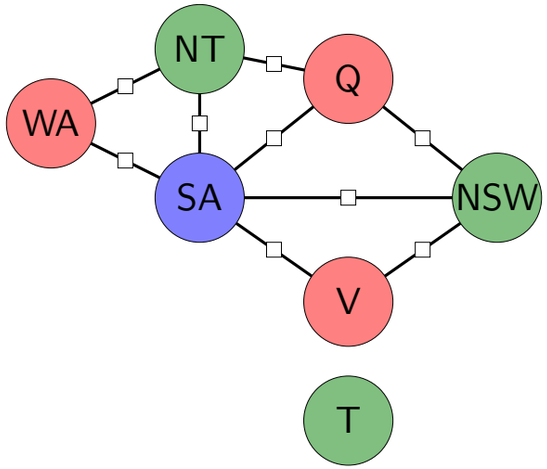
x_1	x_2	x_3	Weight
R	R	R	$0 \cdot 1 \cdot 3 \cdot 2 = 0$
R	R	B	$0 \cdot 1 \cdot 2 \cdot 1 = 0$
R	B	R	$0 \cdot 0 \cdot 2 \cdot 2 = 0$
R	B	B	$0 \cdot 0 \cdot 3 \cdot 1 = 0$
B	R	R	$1 \cdot 0 \cdot 3 \cdot 2 = 0$
B	R	B	$1 \cdot 0 \cdot 2 \cdot 1 = 0$
B	B	R	$1 \cdot 1 \cdot 2 \cdot 2 = 4$
B	B	B	$1 \cdot 1 \cdot 3 \cdot 1 = 3$

→ product of factors on that assignment to variables

← best weight, return this

[demo]

Example: map coloring



Assignment:

$$x = \{\text{WA} : \text{R}, \text{NT} : \text{G}, \text{SA} : \text{B}, \text{Q} : \text{R}, \text{NSW} : \text{G}, \text{V} : \text{R}, \text{T} : \text{G}\}$$

Weight:

$$\text{Weight}(x) = 1 \cdot 1 \cdot 1 \cdot 1 \cdot 1 \cdot 1 \cdot 1 \cdot 1 \cdot 1 = 1$$

Assignment:

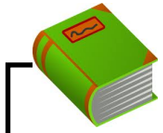
$$x' = \{\text{WA} : \text{R}, \text{NT} : \text{R}, \text{SA} : \text{B}, \text{Q} : \text{R}, \text{NSW} : \text{G}, \text{V} : \text{R}, \text{T} : \text{G}\}$$

Weight:

$$\text{Weight}(x') = 0 \cdot 0 \cdot 1 \cdot 1 \cdot 1 \cdot 1 \cdot 1 \cdot 1 \cdot 1 = 0$$

as soon as a constraint is violated, the weight is zeroed

Assignment weights



Definition: assignment weight

Each **assignment** $x = (x_1, \ldots, x_n)$ has a **weight**:

$$\text{Weight}(x) = \prod_{j=1}^{m} f_j(x)$$

product of all factors

An assignment is **consistent** if $\text{Weight}(x) > 0$.

Objective: find the maximum weight assignment

$$\arg\max_x \text{Weight}(x)$$

A CSP is **satisfiable** if $\max_x \text{Weight}(x) > 0$.

satisfiable if exists a consistent assignment

Constraint satisfaction problems

Boolean satisfiability (SAT):

variables are booleans, factors are logical formulas $[X_1 \vee \neg X_2 \vee X_5]$

Linear programming (LP):

variables are reals, factors are linear inequalities $[X_2 + 3X_5 \leq 1]$

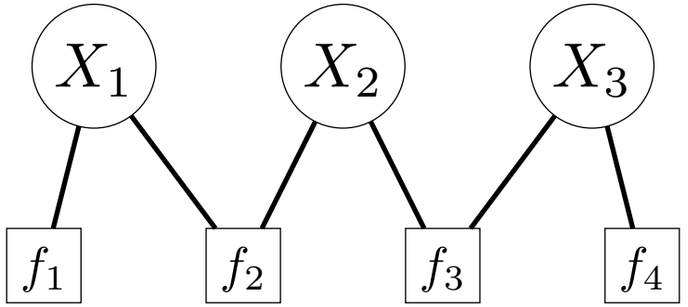
Integer linear programming (ILP):

variables are integers, factors are linear inequalities

Mixed integer programming (MIP):

variables are reals and integers, factors are linear inequalities

Summary



Variables, factors: specify locally

$$\text{Weight}(\{X_1 : \text{B}, X_2 : \text{B}, X_3 : \text{R}\}) = 1 \cdot 1 \cdot 2 \cdot 2 = 4$$

Assignments, weights: optimize globally

CSPs: examples

Example: LSAT question

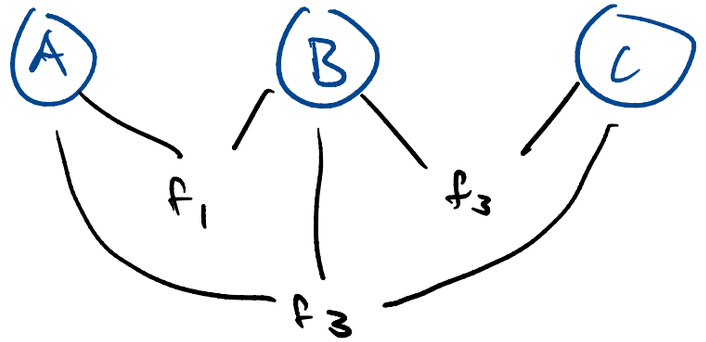
Three sculptures (A, B, C) are to be exhibited in rooms 1, 2 of an art gallery.

The exhibition must satisfy the following conditions:

- Sculptures A and B cannot be in the same room.
- Sculptures B and C must be in the same room.
- Room 2 can only hold one sculpture.

[demo]

domain of variables is {1, 2, 3}

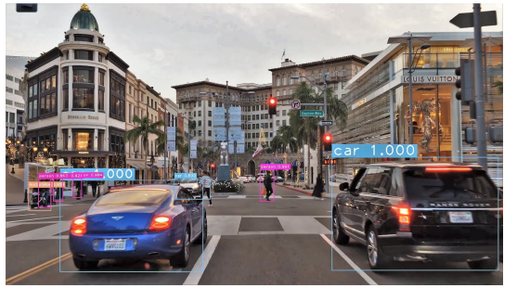


$f_1 = [A \neq B]$

$f_2 = [B = C]$

$f_3 = [A \neq 2 \ \& \ B \neq 2 \ \& \ C \neq 2]$

Example: object tracking

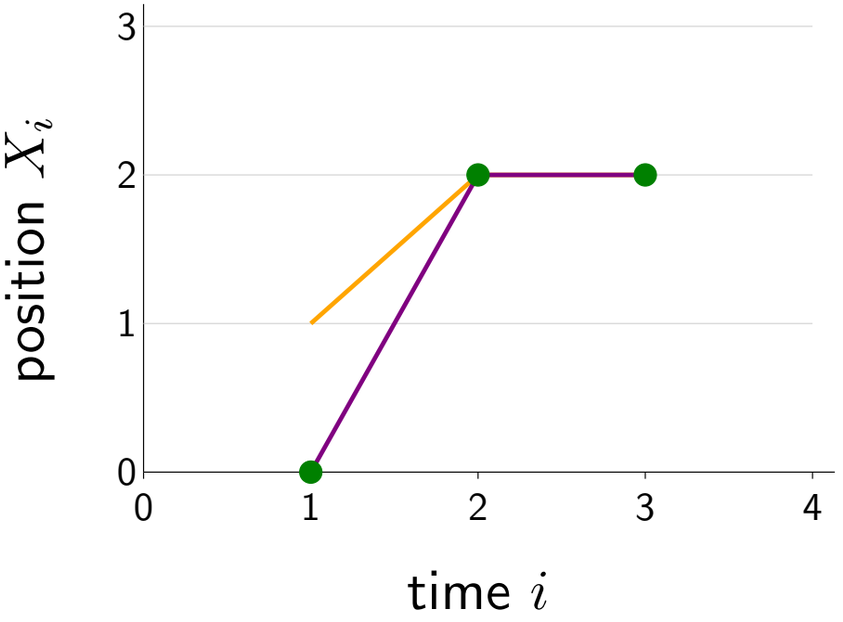


Problem: object tracking

(O) Noisy sensors report positions: $0, 2, 2$.

(T) Objects can't teleport.

What trajectory did the object take?



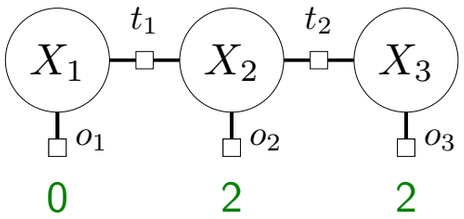
e.g. 1 → 2 → 2

0 → 1 → 2

etc

Example: object tracking CSP

Factor graph:



observed 0, so unlikely high likelihood

noisy

x_1	$o_1(x_1)$
0	2
1	1
2	0

impossible to be super wrong

x_2	$o_2(x_2)$
0	0
1	1
2	2

x_3	$o_3(x_3)$
0	0
1	1
2	2

difference between pos

$\lvert x_i - x_{i+1} \rvert$	$t_i(x_i, x_{i+1})$
0	2
1	1
2	0

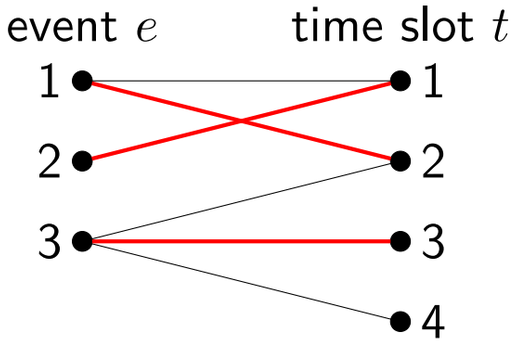
high weight if did not change

unlikely to have moved twice in one time step

[demo]

- Variables $X_i \in \{0, 1, 2\}$: position of object at time i
- Observation factors $o_i(x_i)$: noisy information compatible with position
- Transition factors $t_i(x_i, x_{i+1})$: object positions can't change too much

Example: event scheduling



Problem: Event scheduling

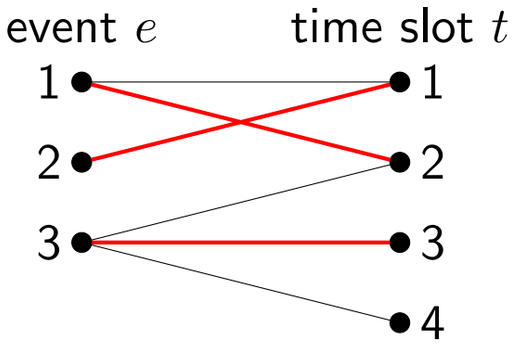
Have E events and T time slots

(C1) Each event e must be put in **exactly one** time slot

(C2) Each time slot t can have **at most one** event

(C3) Event e allowed in time slot t only if $(e, t) \in A$

Example: event scheduling (formulation 1)



Problem: Event scheduling

Have E events and T time slots

(C1) Each event e must be put in **exactly one** time slot

(C2) Each time slot t can have **at most one** event

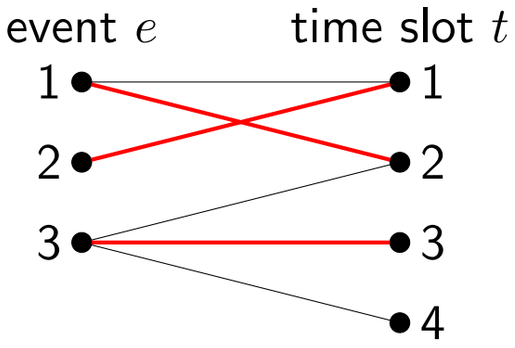
(C3) Event e allowed in time slot t only if $(e, t) \in A$

CSP formulation 1:

- Variables: for each event e, $X_e \in \{1, \ldots, T\}$; satisfies (C1)
- Constraints (only one event per time slot): for each pair of events $e \neq e'$, enforce $[X_e \neq X_{e'}]$; satisfies (C2)
- Constraints (only scheduled allowed times): for each event e, enforce $[(e, X_e) \in A]$; satisfies (C3)

- The first formulation is perhaps the more natural one. We make a variable X_e for each event, whose value will be the time slot that the event is scheduled into. Since each variable can only take on one value, we automatically satisfy (C1), the requirement that every event must be put in exactly one time slot.
- However, we need to make sure no two events end up in the same time slot (C2). To do this, we can create a binary constraint between every pair of distinct event variables X_e and X'_e that enforces their values to be different ($X_e \neq X_{e'}$).
- Finally, to deal with the requirement that an event is scheduled only in allowed time slots (C3), we just need to add a unary constraint for each variable saying that the time slot X_e that's chosen for that event is allowed.
- Note that we end up with E variables with domain size T, and $O(E^2)$ binary constraints.

Example: event scheduling (formulation 2)



Problem: Event scheduling

Have E events and T time slots

(C1) Each event e must be put in **exactly one** time slot

(C2) Each time slot t can have **at most one** event

(C3) Event e allowed in time slot t only if $(e, t) \in A$

CSP formulation 2:

- Variables: for each time slot t, $Y_t \in \{1, \ldots, E\} \cup \{\emptyset\}$; satisfies (C2)
- Constraints (each event is scheduled exactly once): for each event e, enforce $[Y_t = e$ for exactly one $t]$; satisfies (C1)
- Constraints (only schedule allowed times): for each time slot t, enforce $[Y_t = \emptyset$ or $(Y_t, t) \in A]$; satisfies (C3)

- Alternatively, we can take the perspective of the time slots and ask which event was scheduled in each time slot. So we introduce a variable Y_t for each time slot t which takes on a value equal to one of the events or none ($\emptyset$); this automatically takes care of (C2).
- Unlike the first formulation, we don't get for free the requirement that each event is put in exactly one time slot (C1). To add it, we introduce E constraints, one for each event. Each constraint needs to depend on all T variables and check that the number of time slots t which have event e assigned to that slot ($Y_t = e$) is exactly 1.
- Finally, we add T constraints, one for each time slot t enforcing that if there was an event scheduled there ($Y_t \neq \emptyset$), then it better be allowed according to A.
- With this formulation, we have T variables with domain size $E+1$, and E T-ary constraints. We will show shortly that each T-ary constraints can be converted into $O(T)$ binary constraints with $O(T)$ variables. Therefore, the resulting formulation has T variables with domain size $E+1$, $O(ET)$ variables with domain size 2 and $O(ET)$ binary constraints.
- Which one is better? Since $T \geq E$ is required for the existence of a consistent solution, the first formulation is better.
- But if we were to add another constraint relating adjacent time slots (e.g., the courses assigned two adjacent slots should have topic overlap), then the second formulation would make it easier.

Example: program verification

```
def foo(x, y):
    a = x * x
    b = a + y * y
    c = b - 2 * x * y
    return c
```

Specification: c >= 0 for all x and y

CSP formulation: → formulate program as a CSP

- Variables: x, y, a, b, c

$a == x^2$ $b == a + y^2$ $c == b - 2xy$

- Constraints (program statements): $[a = x^2]$, $[b = a + y^2]$, $[c = b - 2xy]$

Note: program (= is assignment), CSP (= is mathematicality equality)

- Constraint (negation of specification): $[c < 0]$

Program satisfies specification iff CSP has no consistent assignment

Summary

- Decide on variables and domains
- Translate each desideratum into a set of factors
- Try to keep CSP small (variables, factors, domains, arities)
- When implementing each factor, think in terms of checking a solution rather than computing the solution