CSPs: overview

Course plan

Search problems

Markov decision processes

Adversarial games

$\boxed{\text{Constraint satisfaction problems}}$

Markov networks

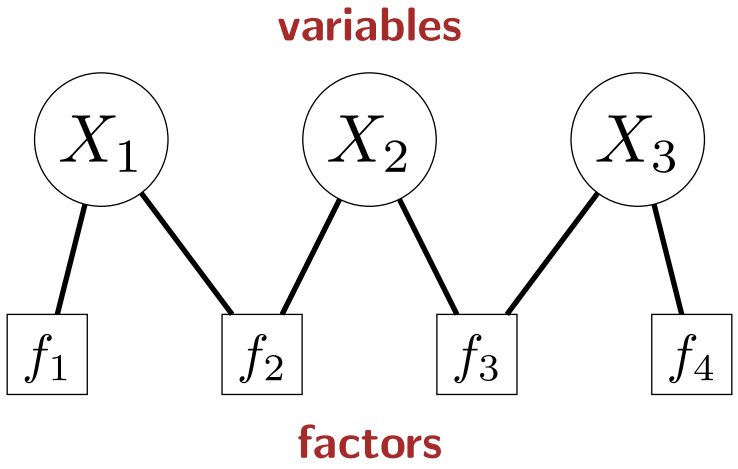
Bayesian networks

Reflex **States** **Variables** **Logic**

"Low-level intelligence" "High-level intelligence"

Machine learning

Factor graphs

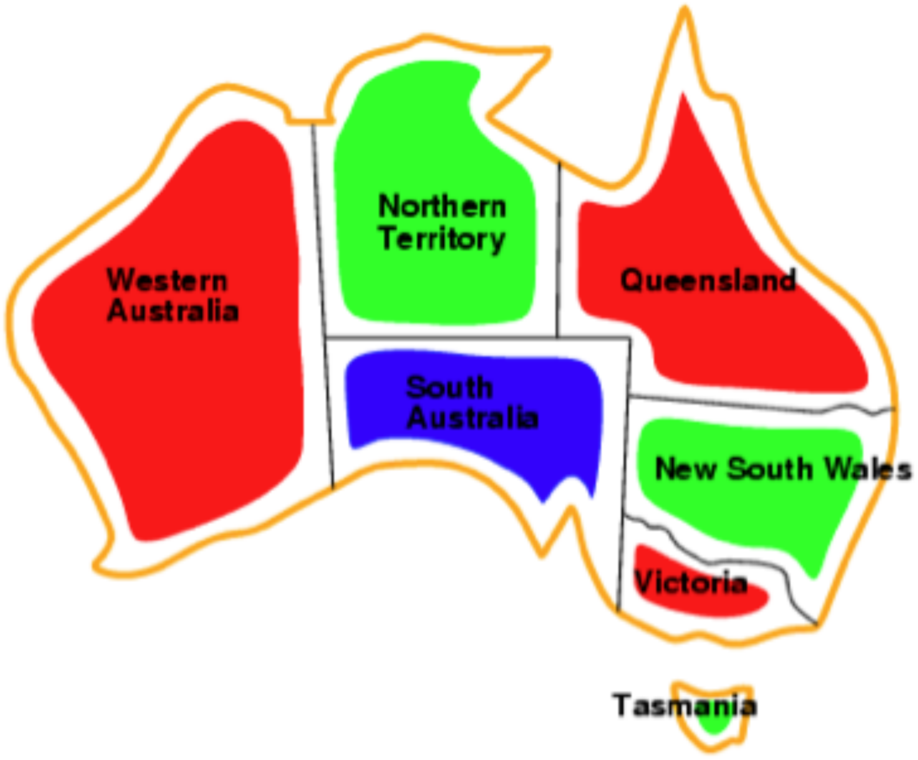


Objective: find the best assignment of values to the variables

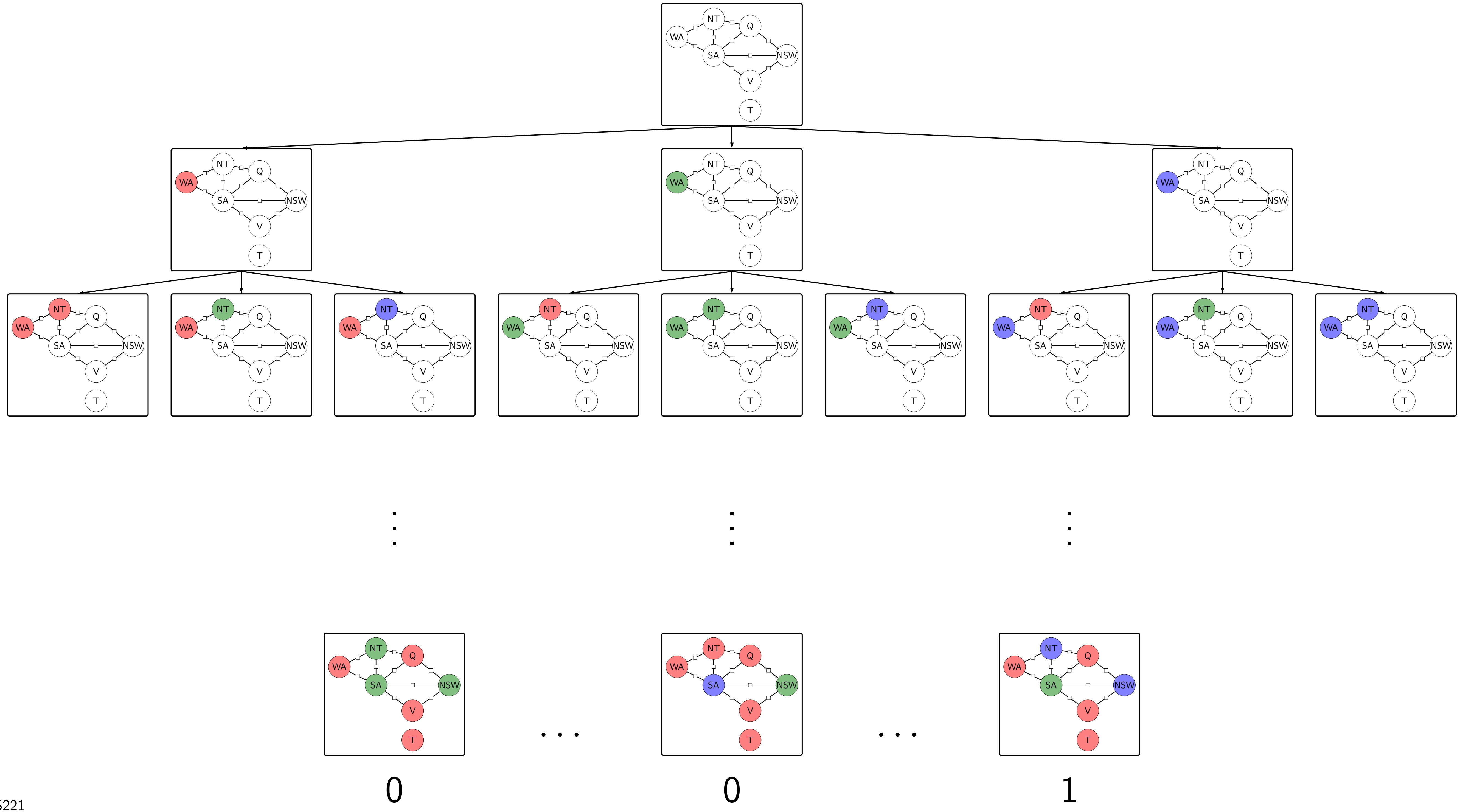
Map coloring

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

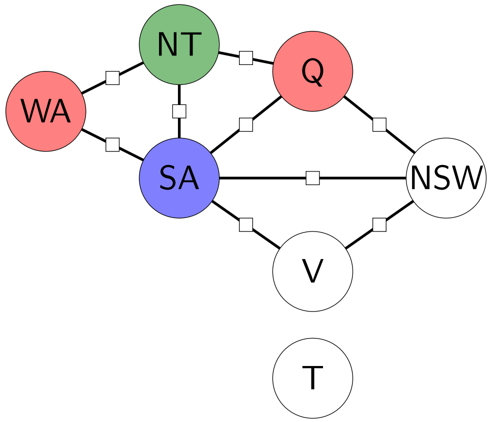
Map coloring



(one possible solution)


WA
NT
Q
SA
NSW
V
T
0
0
1

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

Variable-based models

Special cases:

- Constraint satisfaction problems
- Markov networks
- Bayesian networks

Key idea: variables

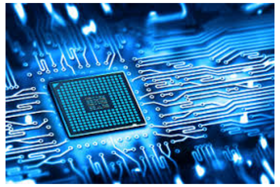
- Solutions to problems $\Rightarrow$ assignments to variables (modeling).
- Decisions about variable ordering, etc. chosen by inference.

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

Backtracking (exact) search

Dynamic ordering

Arc consistency

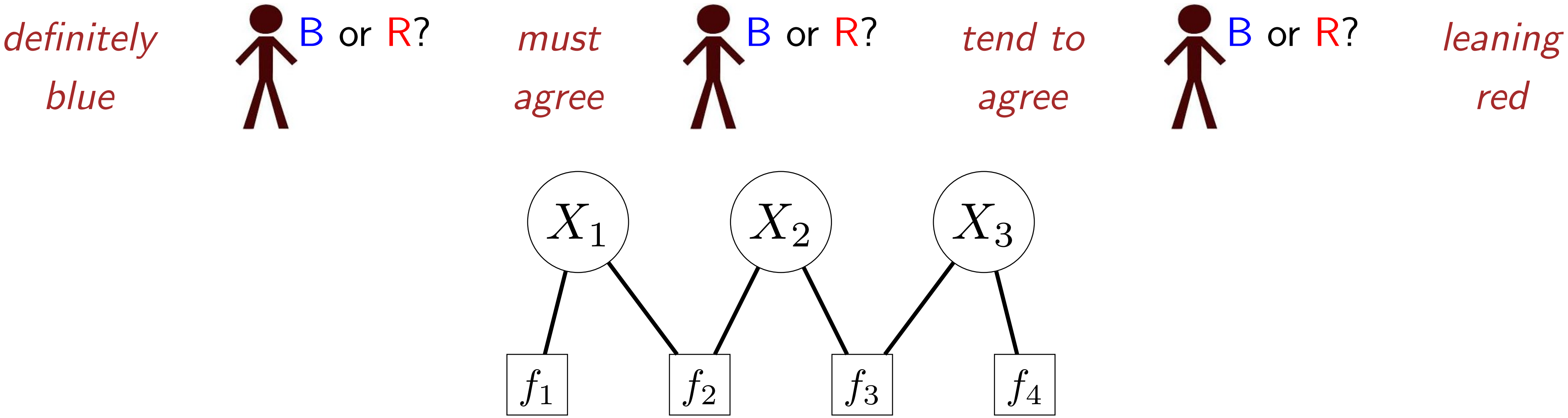
Approximate search

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

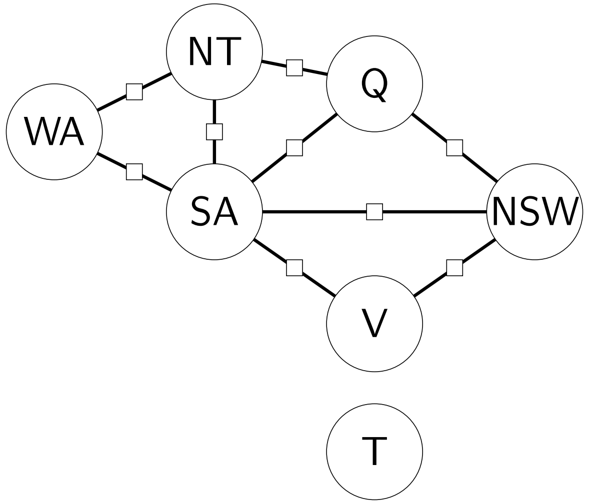
x_2	x_3	$f_3(x_2, x_3)$
R	R	3
R	B	2
B	R	2
B	B	3

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:

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

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

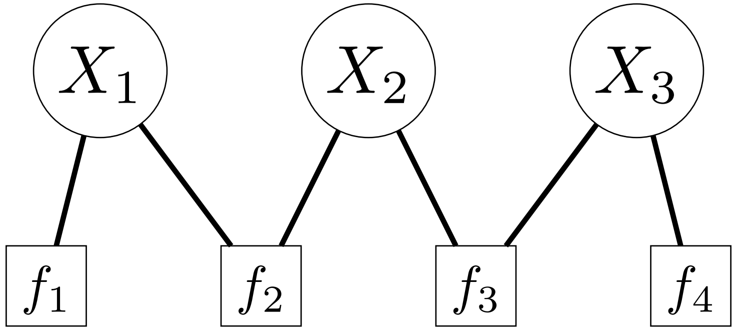
Factors:

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

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

...

Factor graph



Definition: factor graph

Variables:

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

Factors:

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

Factors

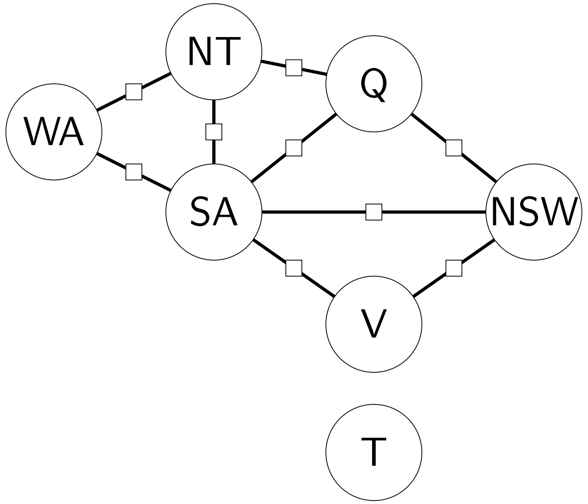
Definition: scope and arity

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

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

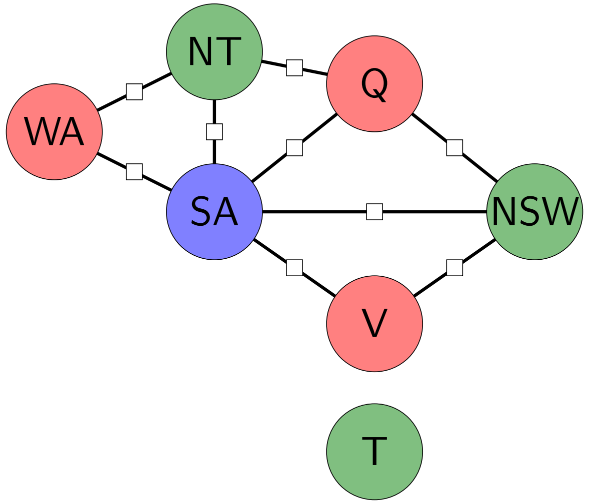
x_2	x_3	$f_3(x_2, x_3)$
R	R	3
R	B	2
B	R	2
B	B	3

x_3	$f_4(x_3)$
R	2
B	1

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	$1 \cdot 1 \cdot 3 \cdot 1 = 3$

[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:

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

Assignment weights

Definition: assignment weight

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

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

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 $\max_x \text{Weight}(x) > 0$.

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 $[X_2 + 3X_5 \leq 1]$

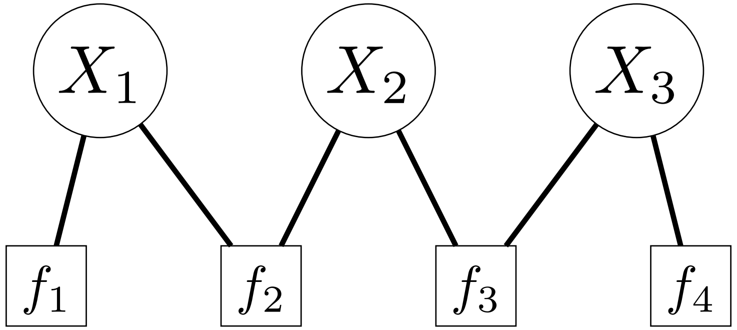
Integer linear programming (ILP):

variables are integers, factors are linear inequalities

Mixed integer programming (MIP):

variables are reals and integers, factors are linear inequalities

Summary



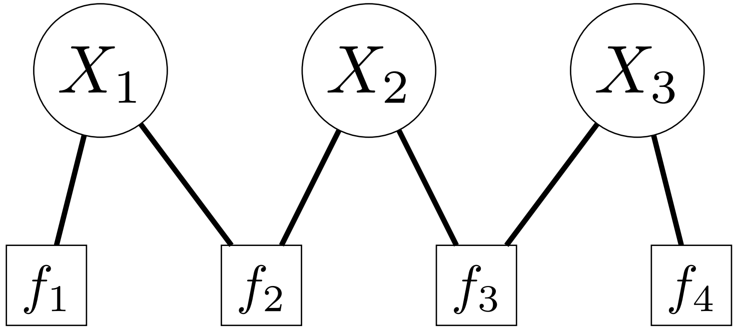
Variables, factors: specify locally

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

Assignments, weights: optimize globally

CSPs: dynamic ordering

Review: CSPs



Definition: factor graph

Variables:

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

Factors:

$f_1, \dots, f_m$, with each $f_j(X) \geq 0$

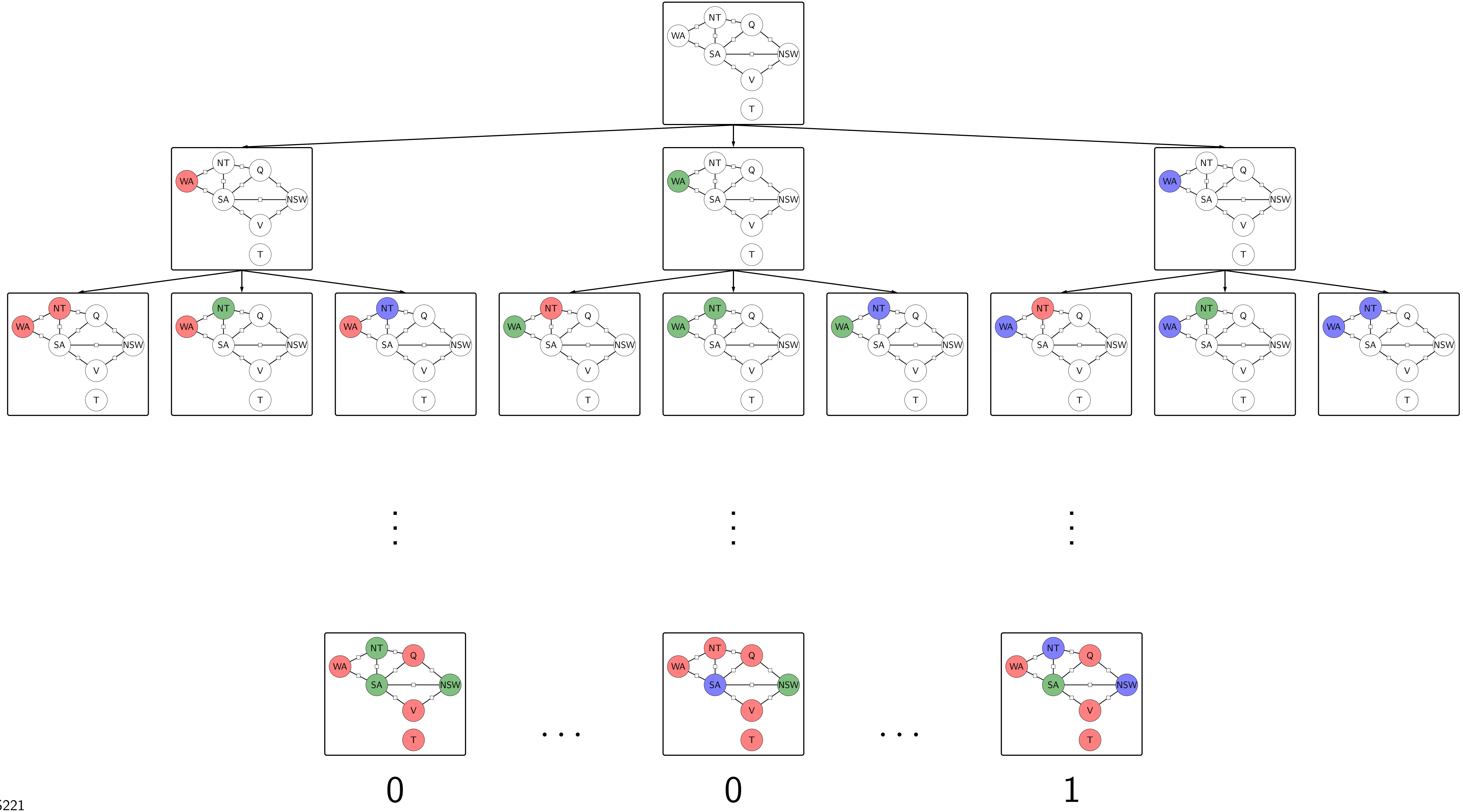
Definition: assignment weight

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

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

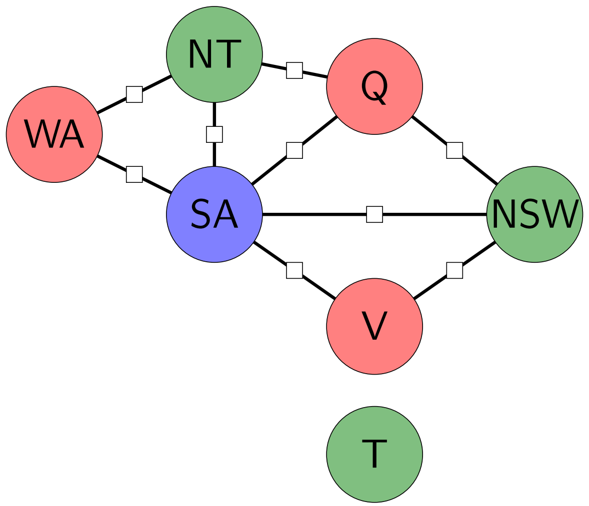
Objective:

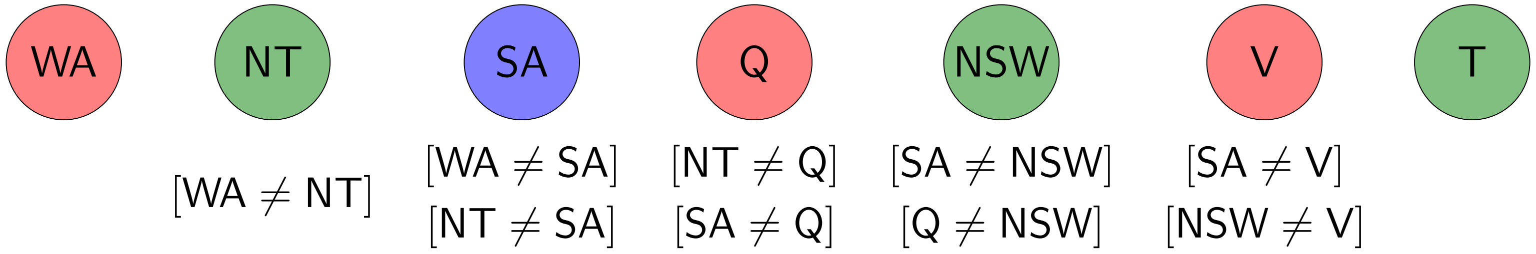
$$\arg\max_x \text{Weight}(x)$$


WA
NT
Q
SA
NSW
V
T
0
0
1

Partial assignment weights

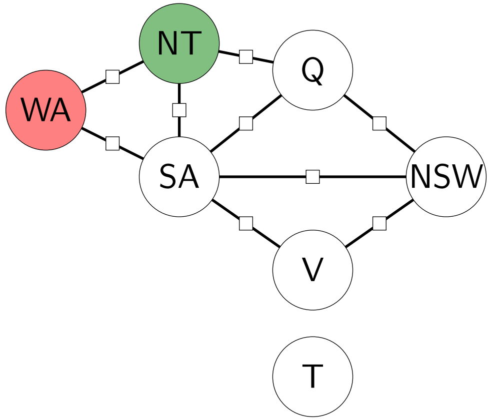
Idea: compute weight of partial assignment as we go

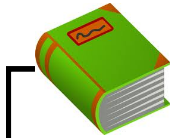




Dependent factors

- Partial assignment (e.g., $x = \{\text{WA} : \text{R}, \text{NT} : \text{G}\}$)





Definition: dependent factors

Let $D(x, X_i)$ be set of factors depending on X_i and x but not on unassigned variables.

$$D(\{\text{WA} : \text{R}, \text{NT} : \text{G}\}, \text{SA}) = \{[\text{WA} \neq \text{SA}], [\text{NT} \neq \text{SA}]\}$$

Backtracking search

Algorithm: backtracking search

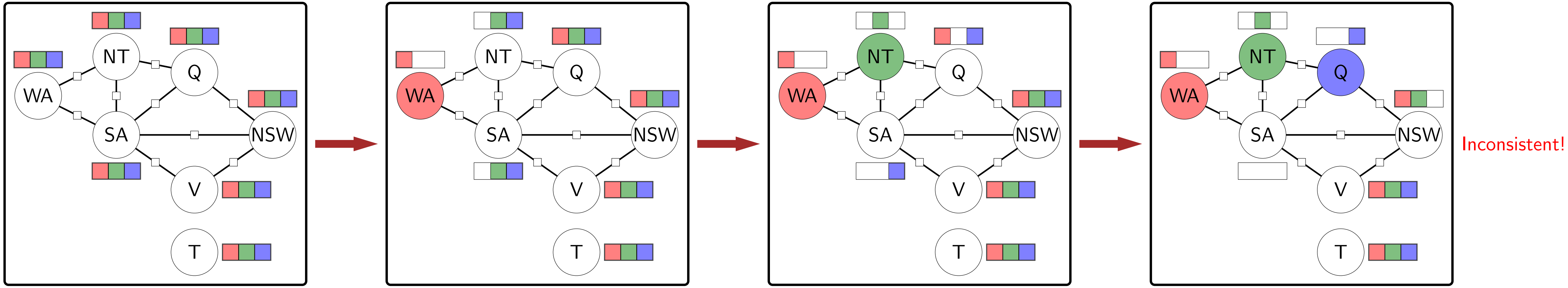
Backtrack(x, w, Domains):

- If x is complete assignment: update best and return
- Choose unassigned **VARIABLE** X_i
- Order **VALUES** Domain_i of chosen X_i
- For each value v in that order:
  - $\delta \leftarrow \prod_{f_j \in D(x, X_i)} f_j(x \cup \{X_i : v\})$
  - If $\delta = 0$: continue
  - $\text{Domains}' \leftarrow$ Domains via **LOOKAHEAD**
  - If any $\text{Domains}'_i$ is empty: continue
  - Backtrack($x \cup \{X_i : v\}, w\delta, \text{Domains}'$)

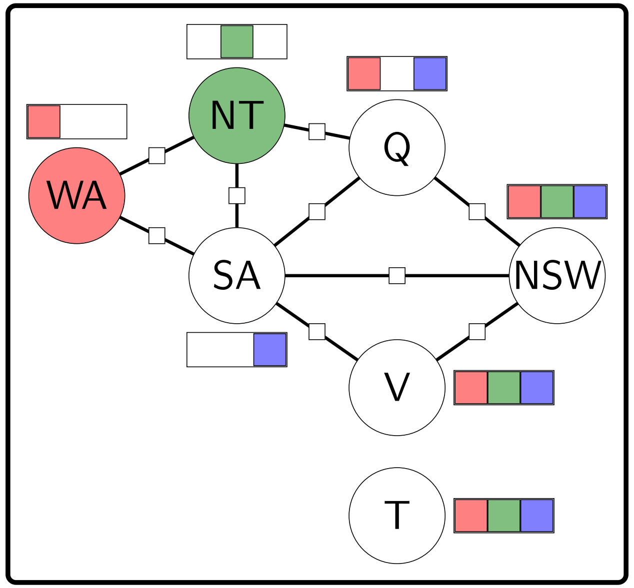
Lookahead: forward checking

Key idea: forward checking (one-step lookahead)

- After assigning a variable X_i, eliminate inconsistent values from the domains of X_i's neighbors.
- If any domain becomes empty, return.



Choosing an unassigned variable



Which variable to assign next?

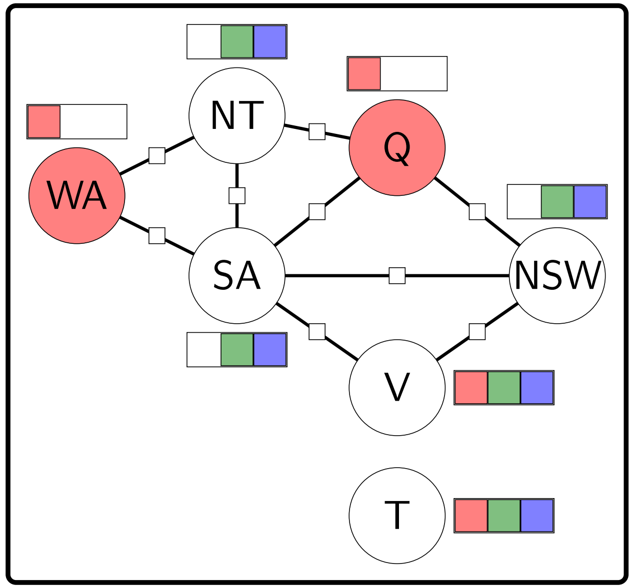
Key idea: most constrained variable

Choose variable that has the smallest domain.

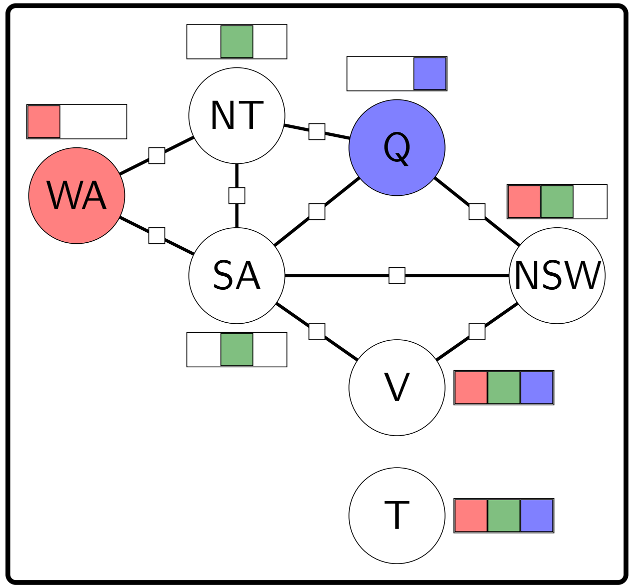
This example: SA (has only one value)

Ordering values of a selected variable

What values to try for Q?



$2+2+2=6$ consistent values

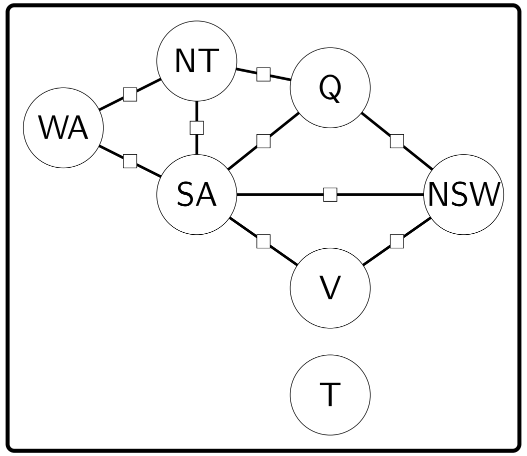


$1+1+2=4$ consistent values

Key idea: least constrained value

Order values of selected X_i by decreasing number of consistent values of neighboring variables.

When to fail?



Most constrained variable (MCV):

- Must assign **every** variable
- If going to fail, fail early $\Rightarrow$ more pruning

Least constrained value (LCV):

- Need to choose **some** value
- Choose value that is most likely to lead to solution

When do these heuristics help?

- Most constrained variable: useful when **some** factors are constraints (can prune assignments with weight 0)

$$[x_1 = x_2] \qquad\qquad [x_2 \neq x_3] + 2$$

- Least constrained value: useful when **all** factors are constraints (all assignment weights are 1 or 0)

$$[x_1 = x_2] \qquad\qquad [x_2 \neq x_3]$$

- Forward checking: needed to prune domains to make heuristics useful!

Summary

Algorithm: backtracking search

Backtrack(x, w, Domains):

- If x is complete assignment: update best and return
- Choose unassigned **VARIABLE** X_i (MCV)
- Order **VALUES** Domain$_i$ of chosen X_i (LCV)
- For each value v in that order:
  - $\delta \leftarrow \prod_{f_j \in D(x, X_i)} f_j(x \cup \{X_i : v\})$
  - If $\delta = 0$: continue
  - Domains$'$ $\leftarrow$ Domains via **LOOKAHEAD** (forward checking)
  - If any Domains$'_i$ is empty: continue
  - Backtrack($x \cup \{X_i : v\}, w\delta$, Domains$'$)

CSPs: arc consistency

Review: backtracking search

Algorithm: backtracking search

Backtrack($x, w,$ Domains):

- If x is complete assignment: update best and return
- Choose unassigned **VARIABLE** X_i (MCV)
- Order **VALUES** Domain$_i$ of chosen X_i (LCV)
- For each value v in that order:
  - $\delta \leftarrow \prod_{f_j \in D(x, X_i)} f_j(x \cup \{X_i : v\})$
  - If $\delta = 0$: continue
  - Domains$'$ $\leftarrow$ Domains via **LOOKAHEAD** (AC-3)
  - If any Domains$'_i$ is empty: continue
  - Backtrack($x \cup \{X_i : v\}, w\delta,$ Domains$'$)

Arc consistency: example

Example: numbers

Before enforcing arc consistency on X_i:

$X_i \in \text{Domain}_i = \{1, 2, 3, 4, 5\}$

$X_j \in \text{Domain}_j = \{1, 2\}$

Factor: $[X_i + X_j = 4]$

After enforcing arc consistency on X_i:

$X_i \in \text{Domain}_i = \{2, 3\}$

X_i	1	2	3	4	5
X_j	1	2			

Arc consistency

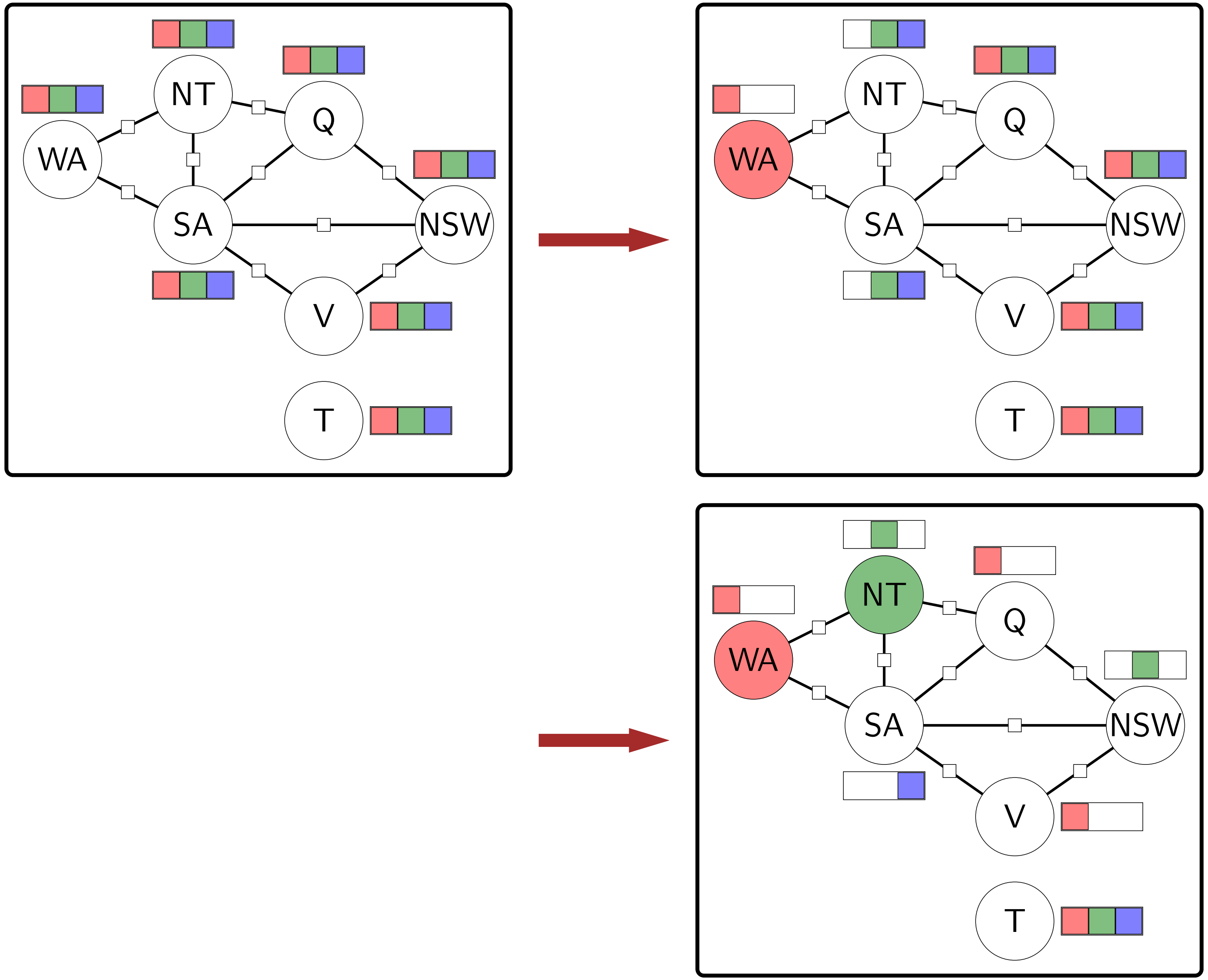
Definition: arc consistency

A variable X_i is **arc consistent** with respect to X_j if for each $x_i \in \text{Domain}_i$, there exists $x_j \in \text{Domain}_j$ such that $f(\{X_i : x_i, X_j : x_j\}) \neq 0$ for all factors f whose scope contains X_i and X_j.

Algorithm: enforce arc consistency

EnforceArcConsistency(X_i, X_j): Remove values from Domain_i to make X_i arc consistent with respect to X_j.

AC-3 (example)



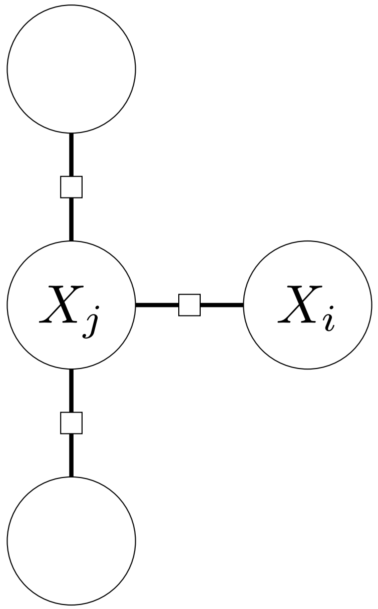
AC-3

Forward checking: when assign $X_j : x_j$, set $\text{Domain}_j = \{x_j\}$ and enforce arc consistency on all neighbors X_i with respect to X_j

AC-3: repeatedly enforce arc consistency on all variables

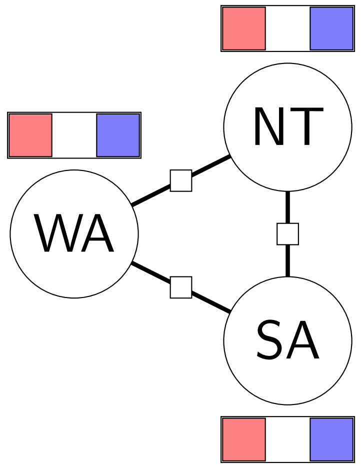
Algorithm: AC-3

$S \leftarrow \{X_j\}$.
While S is non-empty:
  Remove any X_j from S.
  For all neighbors X_i of X_j:
    Enforce arc consistency on X_i w.r.t. X_j.
    If Domain_i changed, add X_i to S.



Limitations of AC-3

- AC-3 isn't always effective:



- No consistent assignments, but AC-3 doesn't detect a problem!

- Intuition: if we look locally at the graph, nothing blatantly wrong...

Summary

- Enforcing arc consistency: make domains consistent with factors

- Forward checking: enforces arc consistency on neighbors

- AC-3: enforces arc consistency on neighbors and their neighbors, etc.

- Lookahead very important for backtracking search!

CSPs: examples

Example: LSAT question

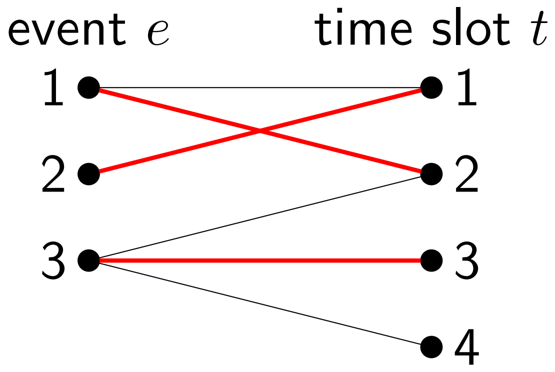
Three sculptures (A, B, C) are to be exhibited in rooms 1, 2 of an art gallery.

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]

Example: event scheduling



Problem: Event scheduling

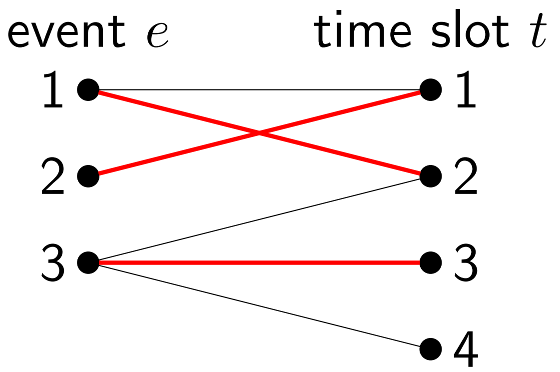
Have E events and T time slots

(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

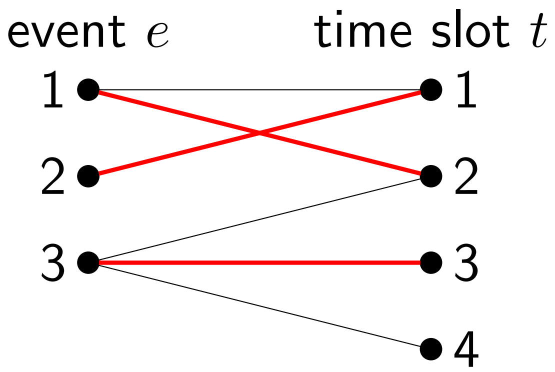
(C2) Each time slot t can have **at most one** event

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

CSP formulation 1:

- Variables: for each event e, $X_e \in \{1, \dots, T\}$; 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)

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

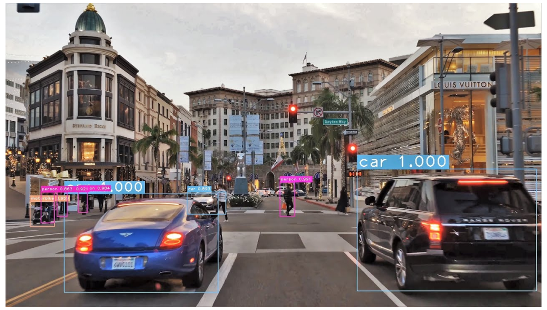
(C2) Each time slot t can have **at most one** event

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

CSP formulation 2:

- Variables: for each time slot t, $Y_t \in \{1, \dots, E\} \cup \{\emptyset\}$; 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)

Example: object tracking

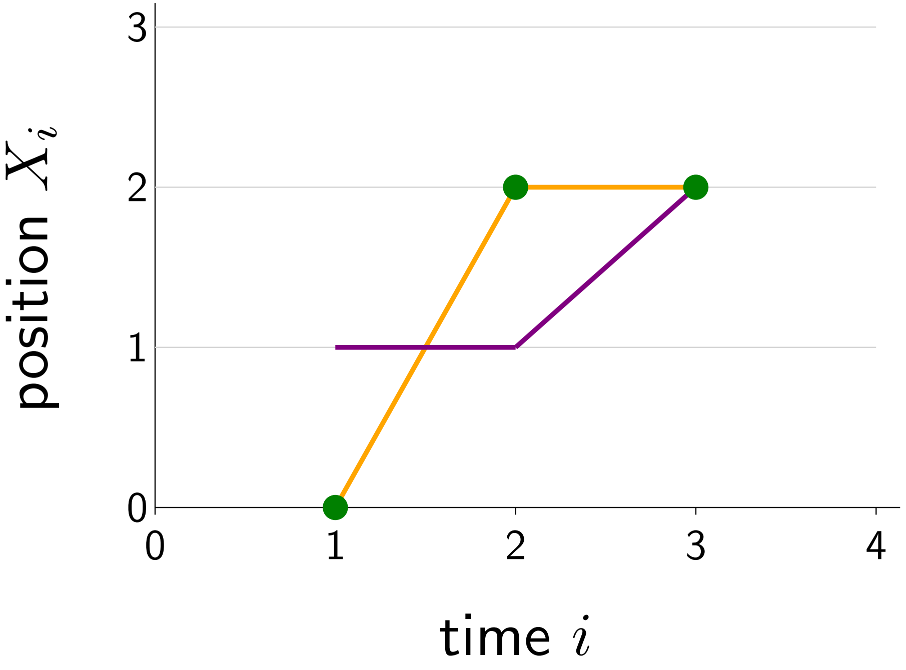


Problem: object tracking

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

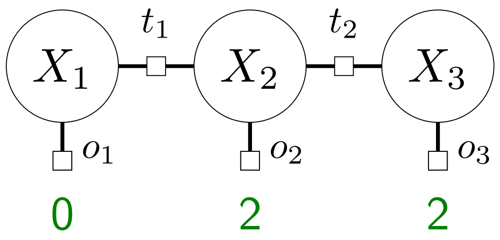
(T) Objects can't teleport.

What trajectory did the object take?



Example: object tracking CSP

Factor graph:



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

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

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

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

[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: 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:

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