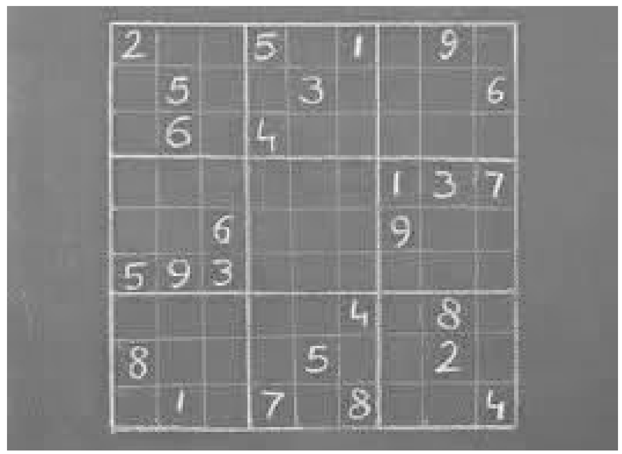
Constraint Satisfaction Problems (CSPs)



Roadmap

Modeling

Definitions

Examples

Backtracking (exact) search

Dynamic ordering

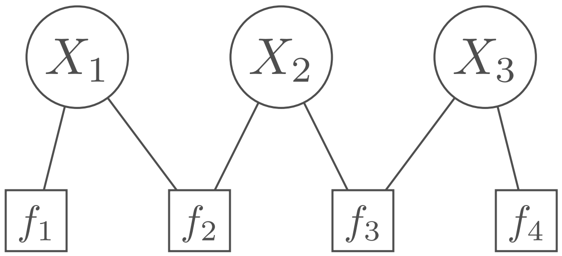
Arc consistency

Approximate search

Beam search

Local search

Review: CSPs



Definition: factor graph

Variables:

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

Factors:

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

Definition: assignment weight

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

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

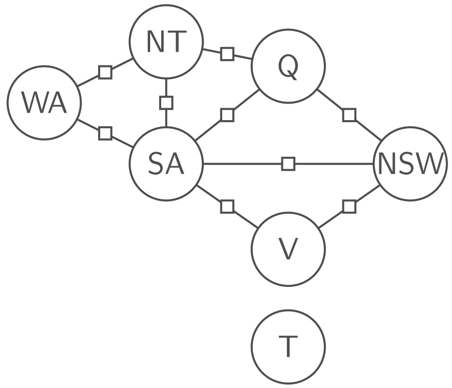
Objective:

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

Map coloring

(one possible solution)

Example: map coloring



Variables:

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

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

Factors:

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

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

...

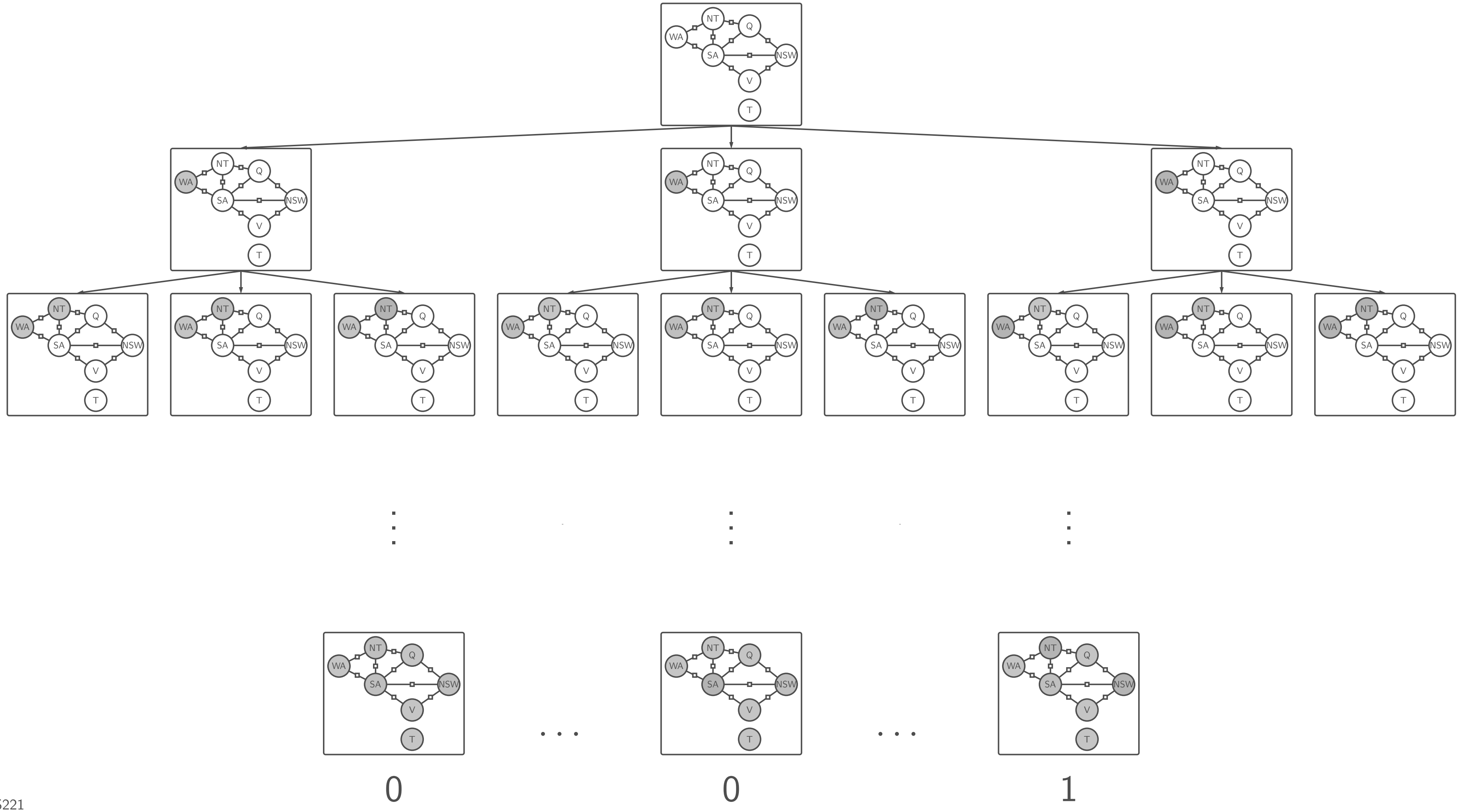
Lecture

Dynamic Ordering

Arc Consistency

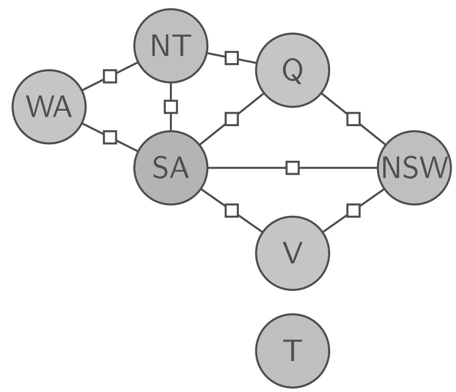
Beam Search

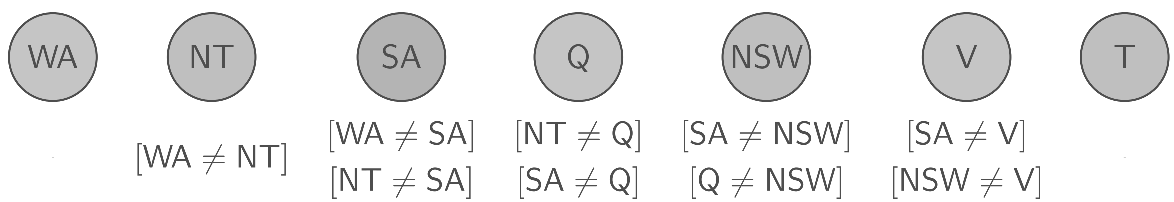
Local Search


WA
NT
Q
SA
NSW
V
T
0
0
1

Partial assignment weights

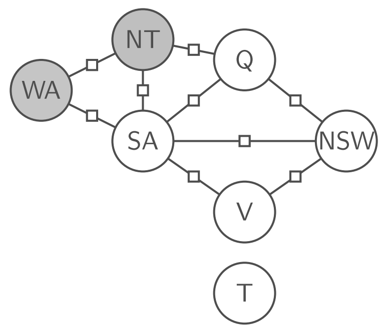
Idea: compute weight of partial assignment as we go





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

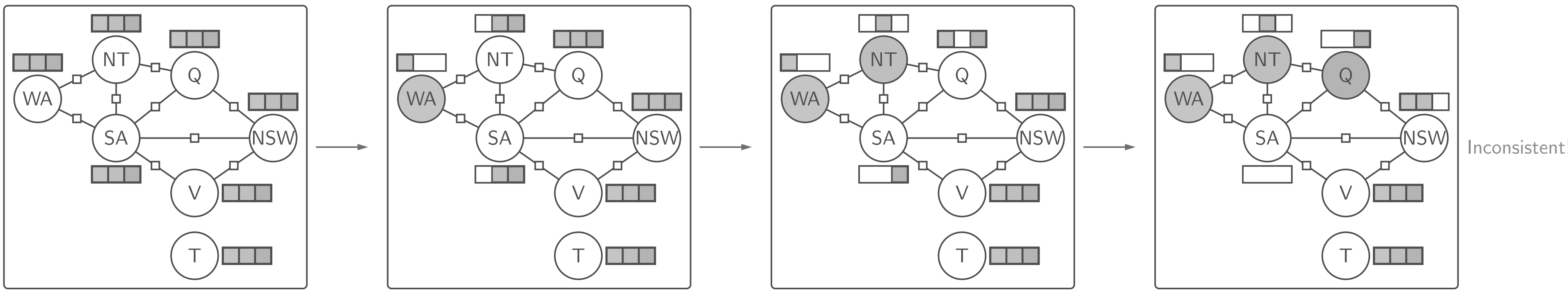
Backtrack($x, w,$ Domains):

- 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
  - Domains$'$ $\leftarrow$ Domains via **LOOKAHEAD**
  - If any Domains$'_i$ is empty: continue
  - Backtrack($x \cup \{X_i : v\}, w\delta,$ Domains$'$)

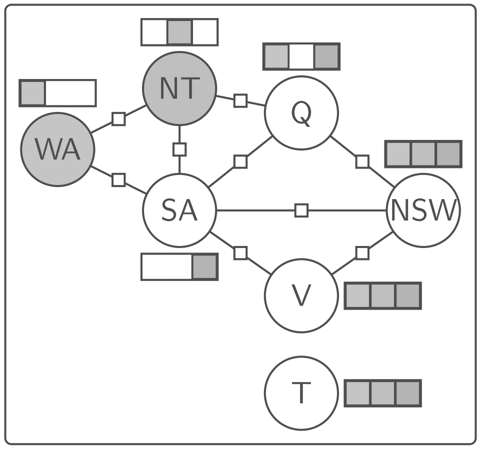
Lookahead: forward checking

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?

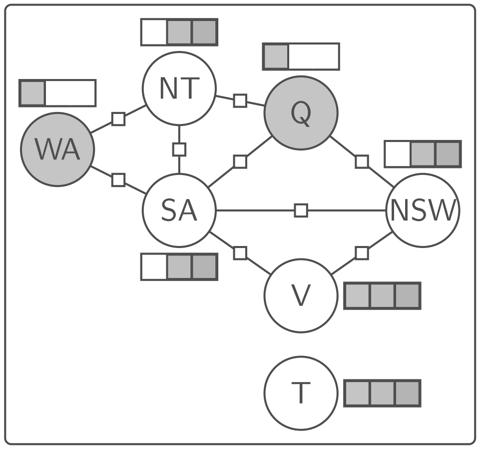
Key idea: most constrained variable

Choose variable that has the smallest domain.

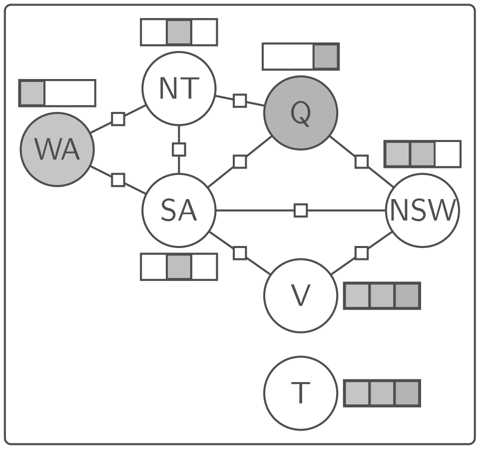
This example: SA (has only one value)

Ordering values of a selected variable

What values to try for Q?



$2+2+2=6$ consistent values

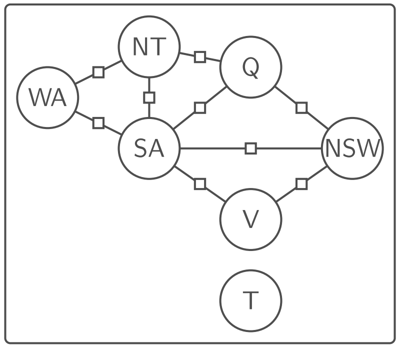


$1+1+2=4$ consistent values

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$'$)

Lecture

Dynamic Ordering

Arc Consistency

Beam Search

Local Search

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

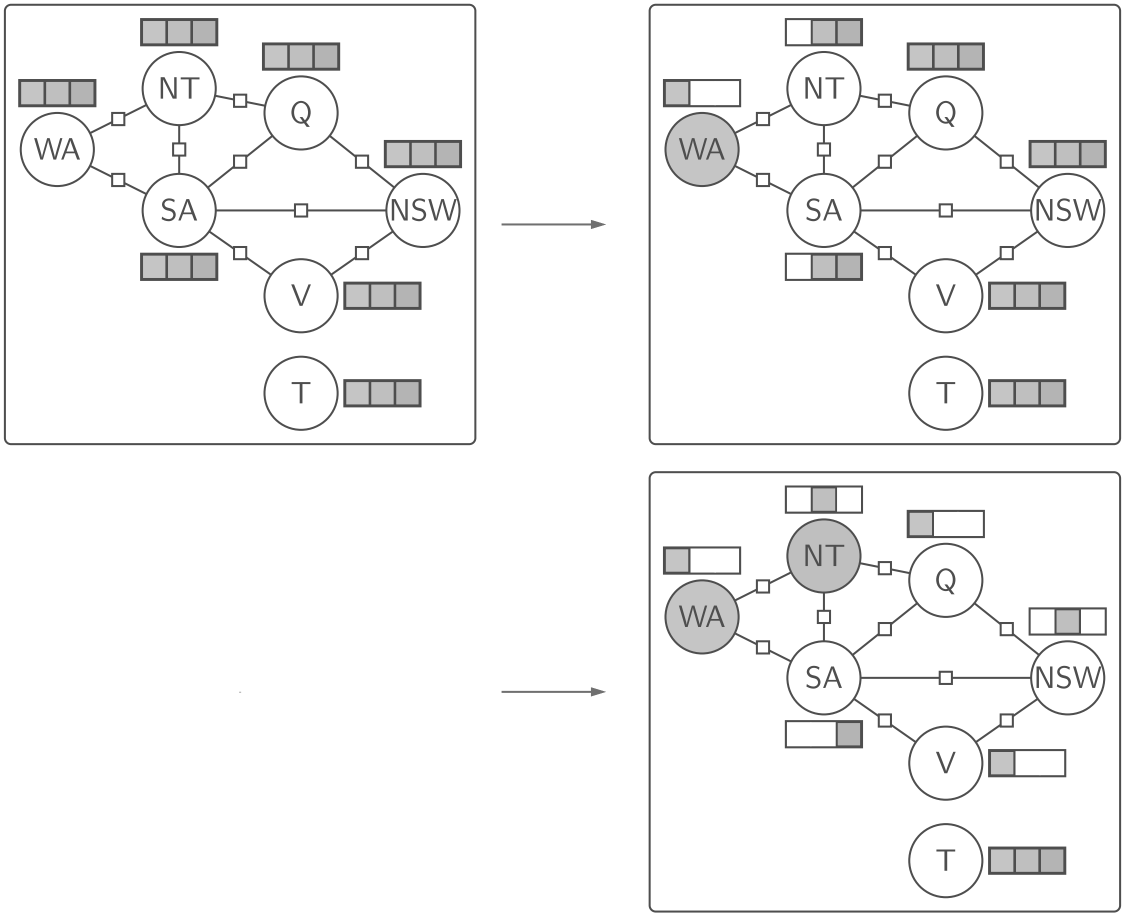
Definition: arc consistency

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)



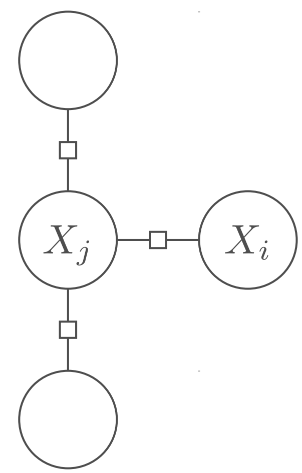
AC-3

Forward checking: when assign $X_j : x_j$, set 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

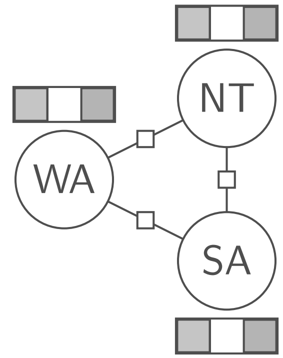
Algorithm: AC-3

$S \leftarrow \{X_j\}$.
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!

Lecture

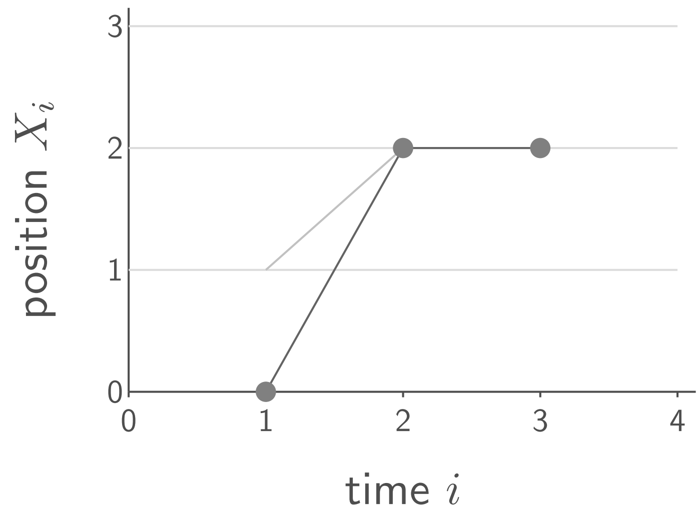
Dynamic Ordering

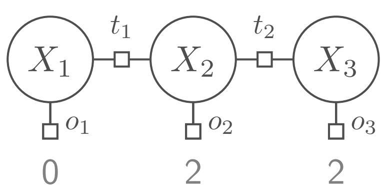
Arc Consistency

Beam Search

Local Search

Example: object tracking





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

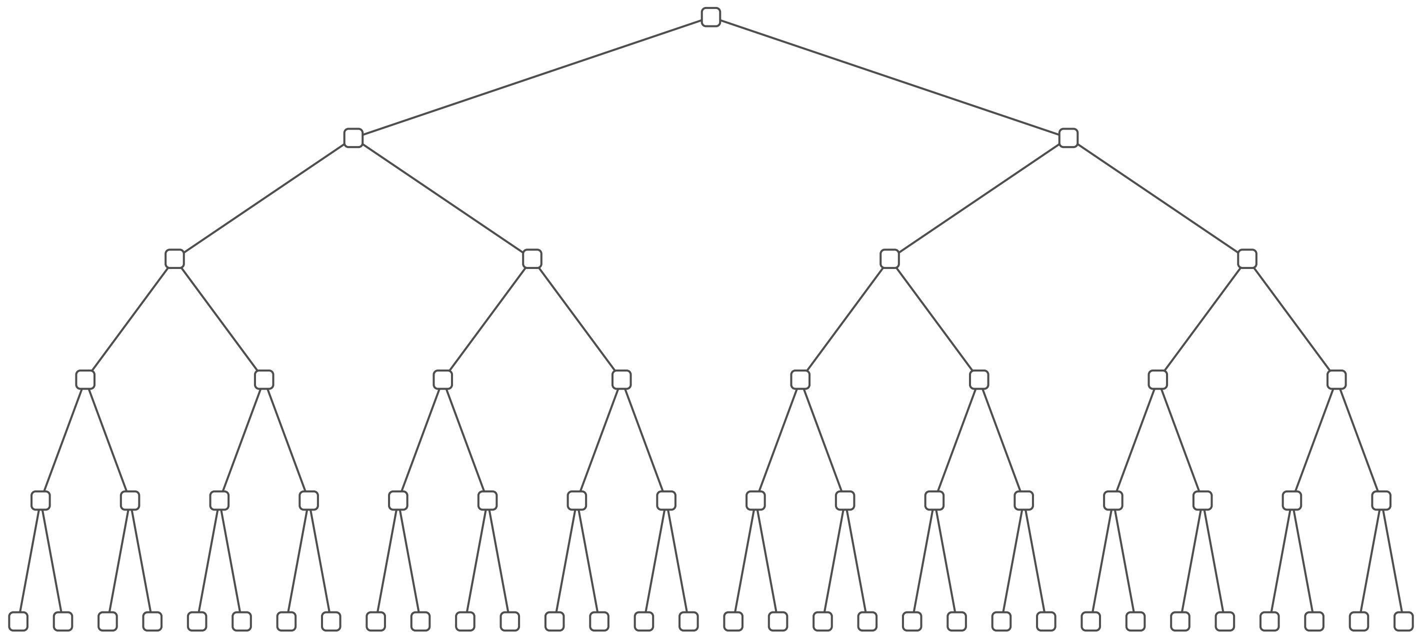
x_2	$o_2(x_2)$
0	0
1	1
2	2

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

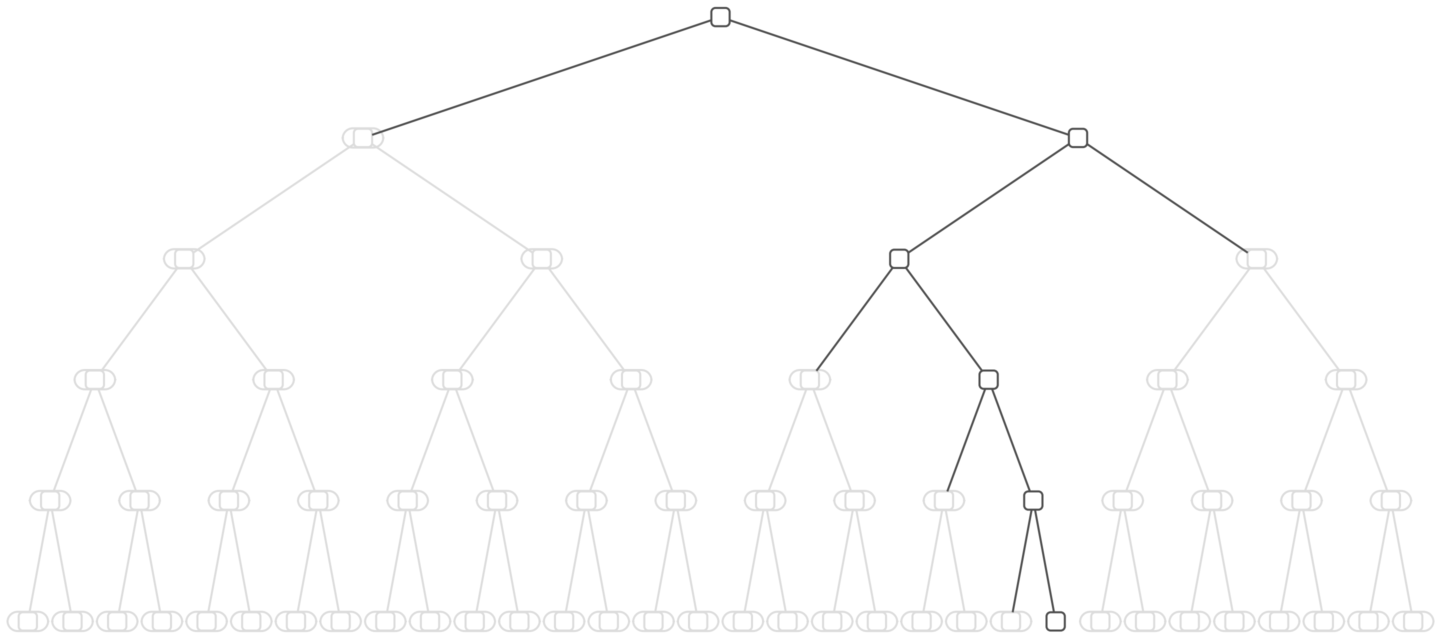
$\|x_i - x_{i+1}\|$	$t_i(x_i, x_{i+1})$
0	2
1	1
2	0

[demo]

Backtracking search



Greedy search



Greedy search

Algorithm: greedy search

Partial assignment $x \leftarrow \{\}$

For each $i = 1, \ldots, n$:

  Extend:

    Compute weight of each $x_v = x \cup \{X_i : v\}$

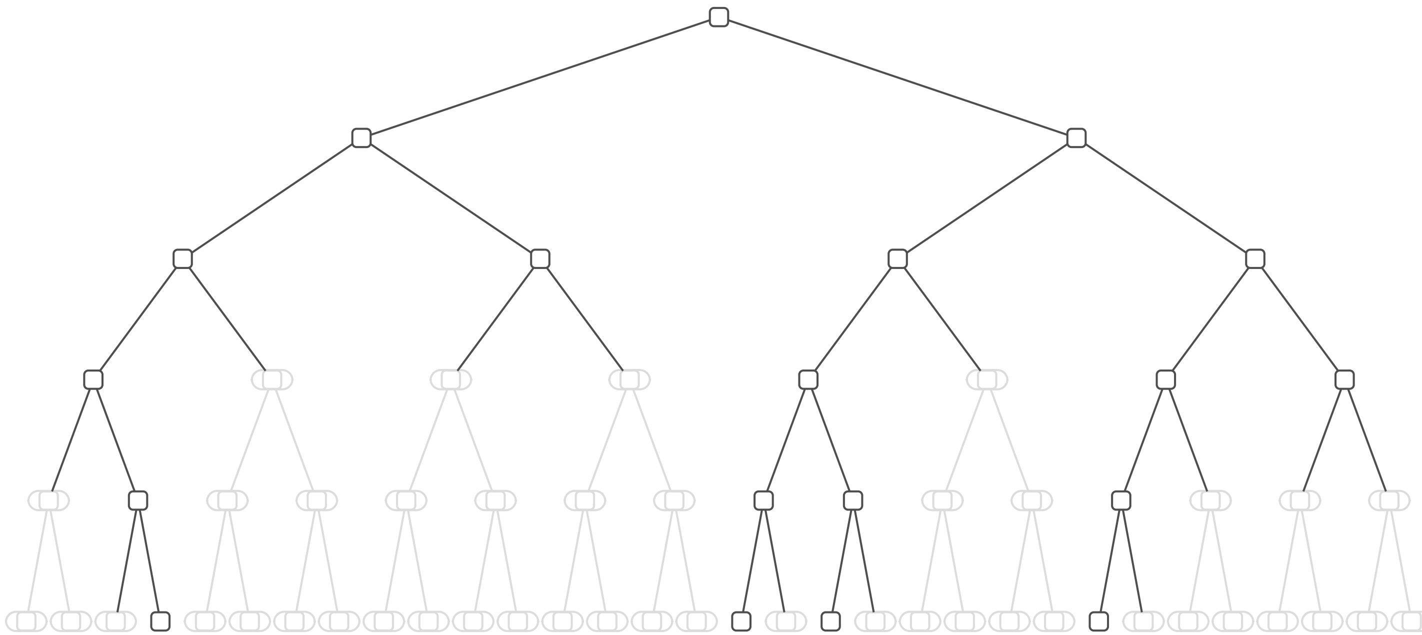
  Prune:

    $x \leftarrow x_v$ with highest weight

Not guaranteed to find maximum weight assignment!

[demo: `beamSearch({K:1})`]

Beam search



Beam size $K = 4$

Beam search

Idea: keep $\leq K$ **candidate list** C of partial assignments

Algorithm: beam search

Initialize $C \leftarrow [\{\}]$

For each $i = 1, \dots, n$:

Extend:

$$C' \leftarrow \{x \cup \{X_i : v\} : x \in C, v \in \text{Domain}_i\}$$

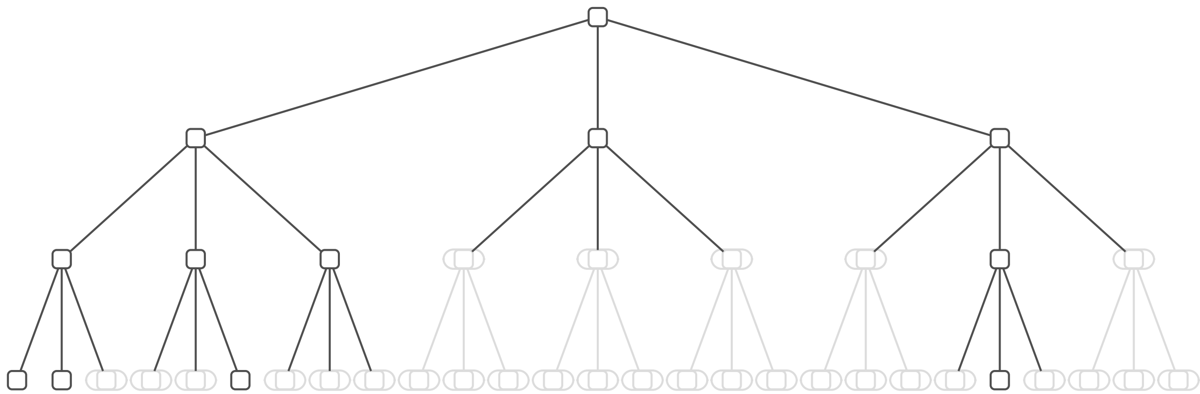
Prune:

$C \leftarrow K$ elements of C' with highest weights

Not guaranteed to find maximum weight assignment!

[demo: `beamSearch({K:3})`]

Time complexity

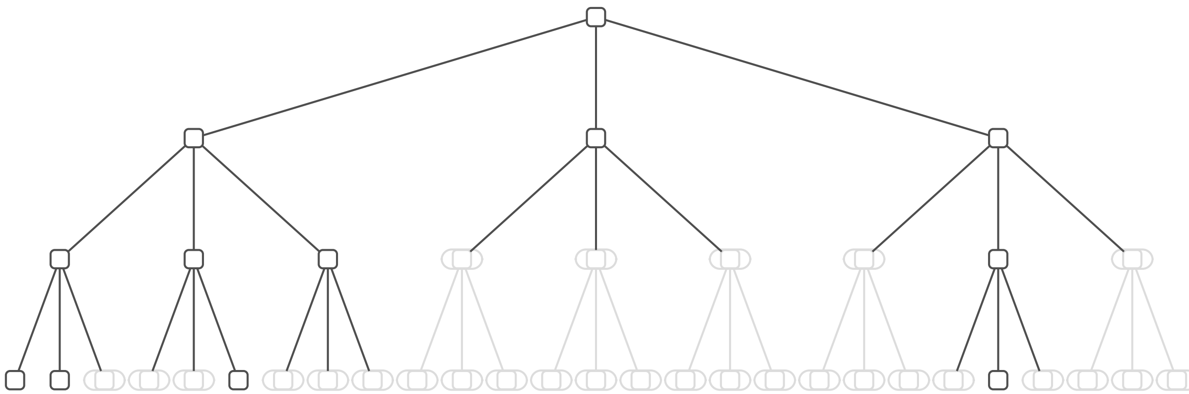


n variables (depth)

Branching factor $b = |\text{Domain}_i|$ $\longrightarrow$ Time: $O(nKb)$

Beam size K

Summary



- Beam size K controls tradeoff between efficiency and accuracy
  - $K = 1$ is greedy search ($O(nb)$ time)
  - $K = \infty$ is BFS ($O(b^n)$ time)

Backtracking search $\simeq$ DFS ; Beam search $\simeq$ Pruned BFS

Lecture

Dynamic Ordering

Arc Consistency

Beam Search

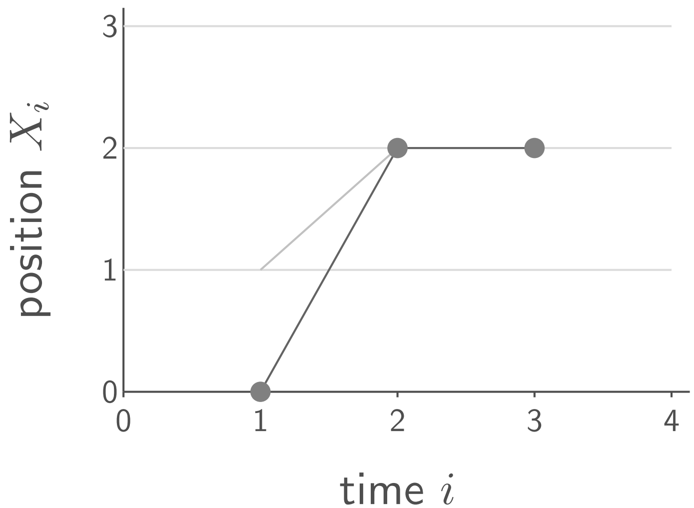
Local Search

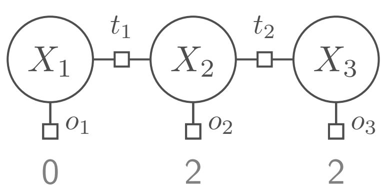
Search strategies

Backtracking/beam search: extend partial assignments

Local search: modify complete assignments

Example: object tracking





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

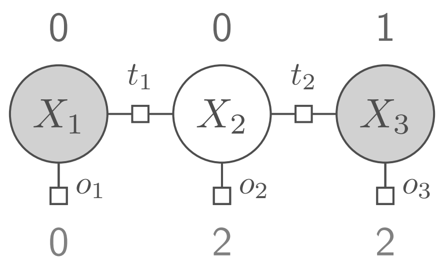
x_2	$o_2(x_2)$
0	0
1	1
2	2

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]

One small step

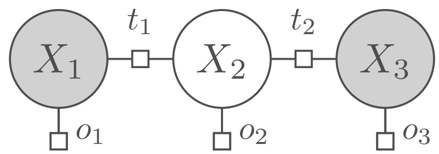


Old assignment: $(0, 0, 1)$; how to improve?

(x_1, v, x_3)	weight
$(0, 0, 1)$	$2 \cdot 2 \cdot 0 \cdot 1 \cdot 1 = 0$
$(0, 1, 1)$	$2 \cdot 1 \cdot 1 \cdot 2 \cdot 1 = 4$
$(0, 2, 1)$	$2 \cdot 0 \cdot 2 \cdot 1 \cdot 1 = 0$

New assignment: $(0, 1, 1)$

Exploiting locality



Weight of new assignment (x_1, v, x_3):

$$o_1(x_1)t_1(x_1, v)o_2(v)t_2(v, x_3)o_3(x_3)$$

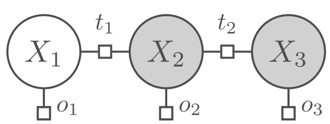
Key idea: locality

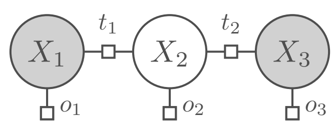
When evaluating possible re-assignments to X_i, only need to consider the factors that depend on X_i.

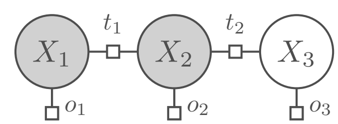
Iterated conditional modes (ICM)

Algorithm: iterated conditional modes (ICM)

Initialize x to a random complete assignment
Loop through $i = 1, \dots, n$ until convergence:
    Compute weight of $x_v = x \cup \{X_i : v\}$ for each v
    $x \leftarrow x_v$ with highest weight



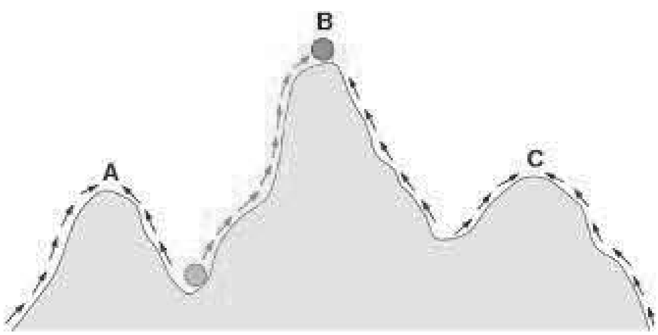




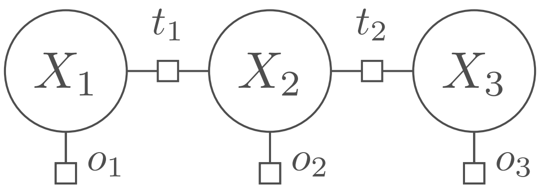
[demo: `iteratedConditionalModes()`]

Convergence properties

- Weight(x) increases or stays the same each iteration
- Converges in a finite number of iterations
- Can get stuck in **local optima**
- Not guaranteed to find optimal assignment!



Summary



Algorithm	Strategy	Optimality	Time complexity
Backtracking search	extend partial assignments	exact	exponential
Beam search	extend partial assignments	approximate	linear
Local search (ICM)	modify complete assignments	approximate	linear*

* time to do $O(1)$ passes

Course plan

Constraint satisfaction problems

Search problems

Markov decision processes

$\boxed{\text{Markov networks}}$

Adversarial games

Bayesian networks

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

Low-level → High-level

Machine learning

Homework

due: the last class, two weeks later

作业 7-周6-周8-课程最终
报告-马里奥玩家