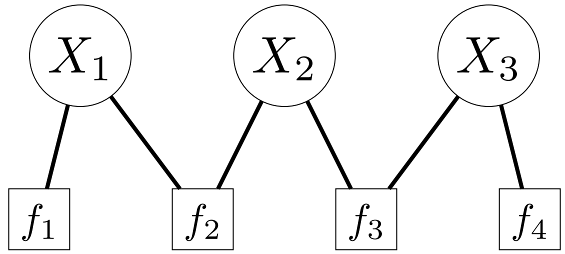
CSPs: overview

Factor graph



Definition: factor graph

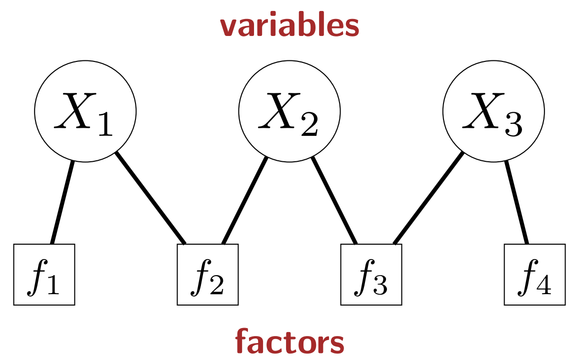
Variables:

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

Factors:

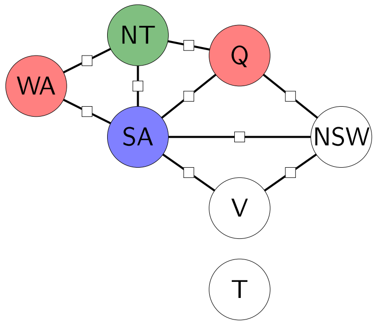
$f_1, \ldots, f_m$, with each $f_j(X) \geq 0$

Factor graphs



Objective: find the best assignment of values to the variables

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

Roadmap

Modeling

Definitions

Examples

Backtracking (exact) search

Dynamic ordering

Arc consistency

Approximate search

Beam search

Local search

Factors

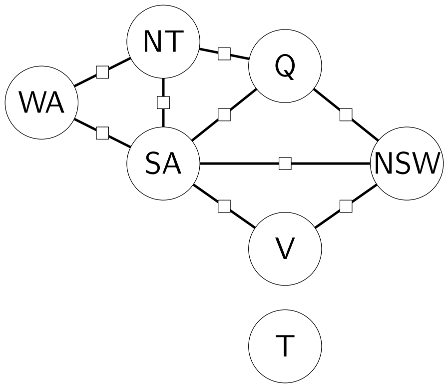
Definition: scope and arity

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

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

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

Roadmap

Modeling

Definitions

Examples

Backtracking (exact) search

Dynamic ordering

Arc consistency

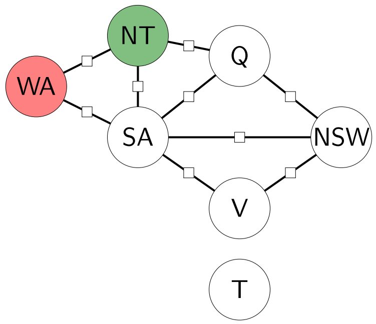
Approximate search

Beam search

Local search

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

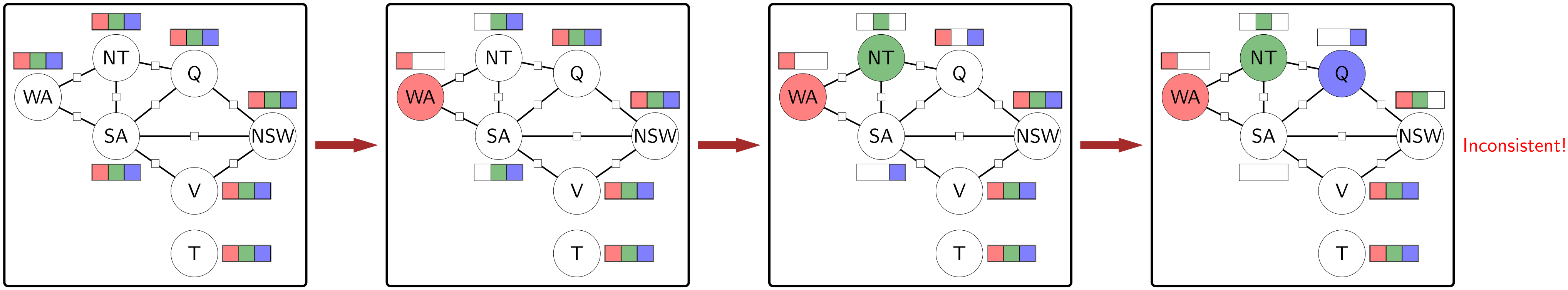
Backtrack(x, w, Domains):

- 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$'$)

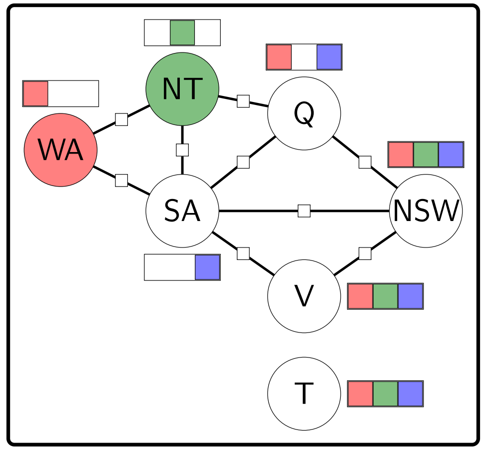
Lookahead: forward checking

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?

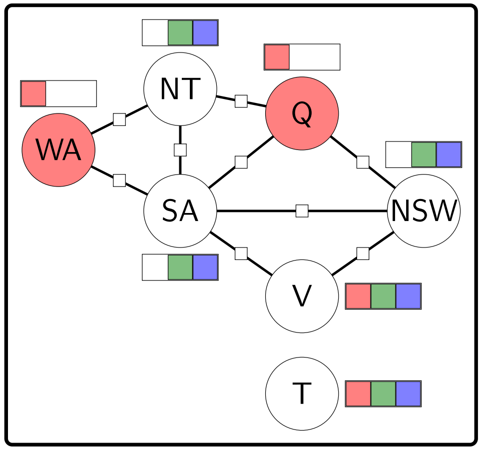
Key idea: most constrained variable

Choose variable that has the smallest domain.

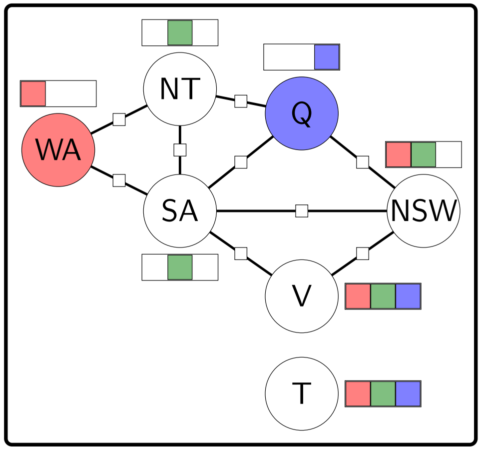
This example: SA (has only one value)

Ordering values of a selected variable

What values to try for Q?



$2+2+2=6$ consistent values

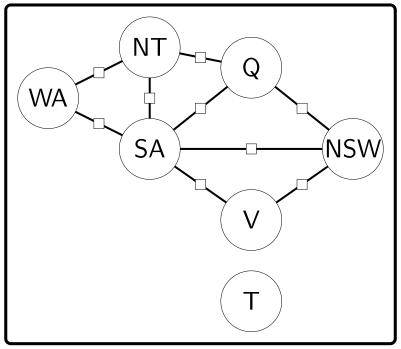


$1+1+2=4$ consistent values

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$'$)

Roadmap

Modeling

Definitions

Examples

Backtracking (exact) search

Dynamic ordering

Arc consistency

Approximate search

Beam search

Local search

Arc consistency

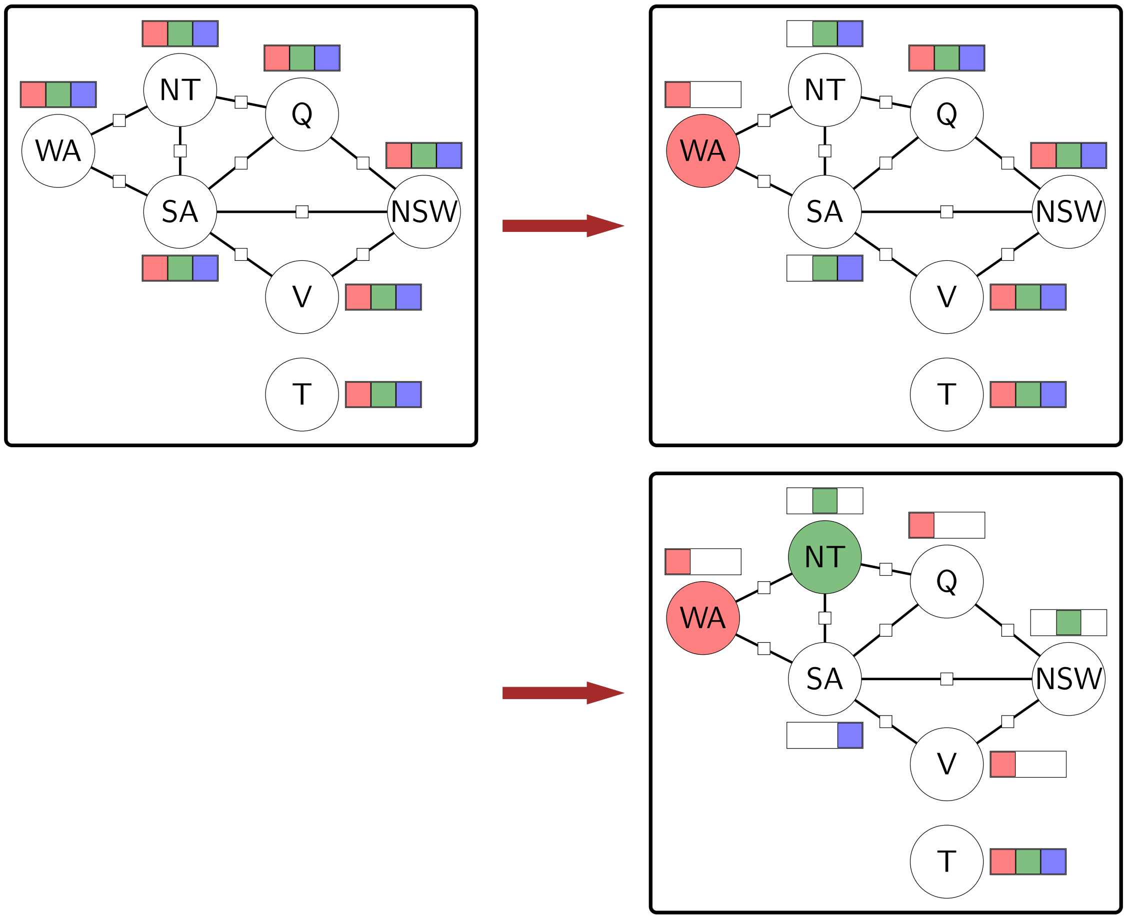
Definition: arc consistency

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)



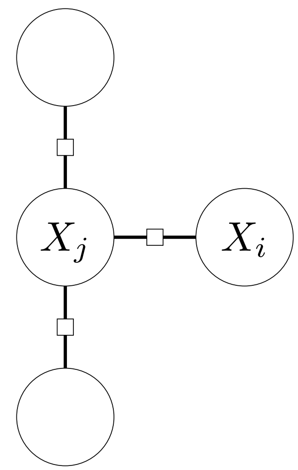
AC-3

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

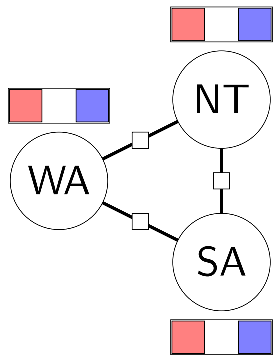
Algorithm: AC-3

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

Roadmap

Modeling

Definitions

Examples

Backtracking (exact) search

Dynamic ordering

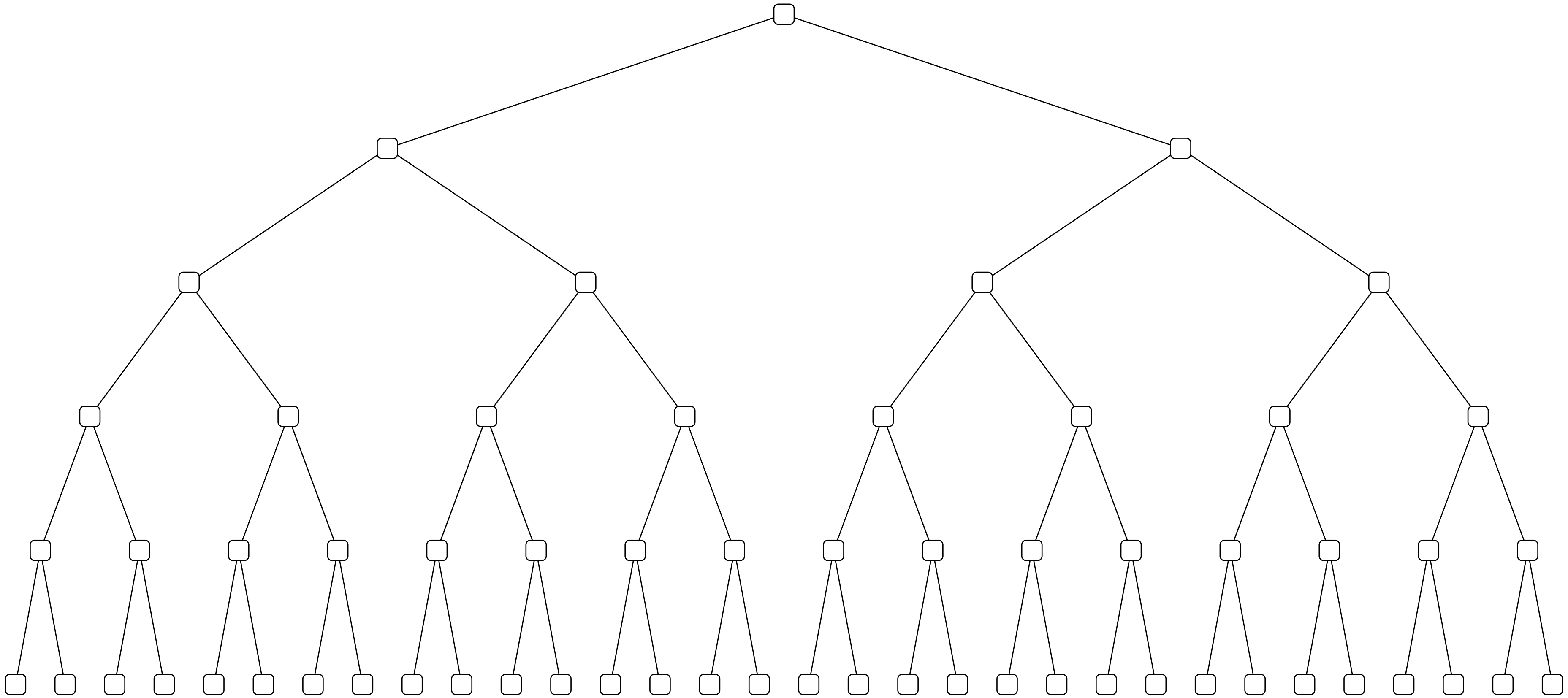
Arc consistency

Approximate search

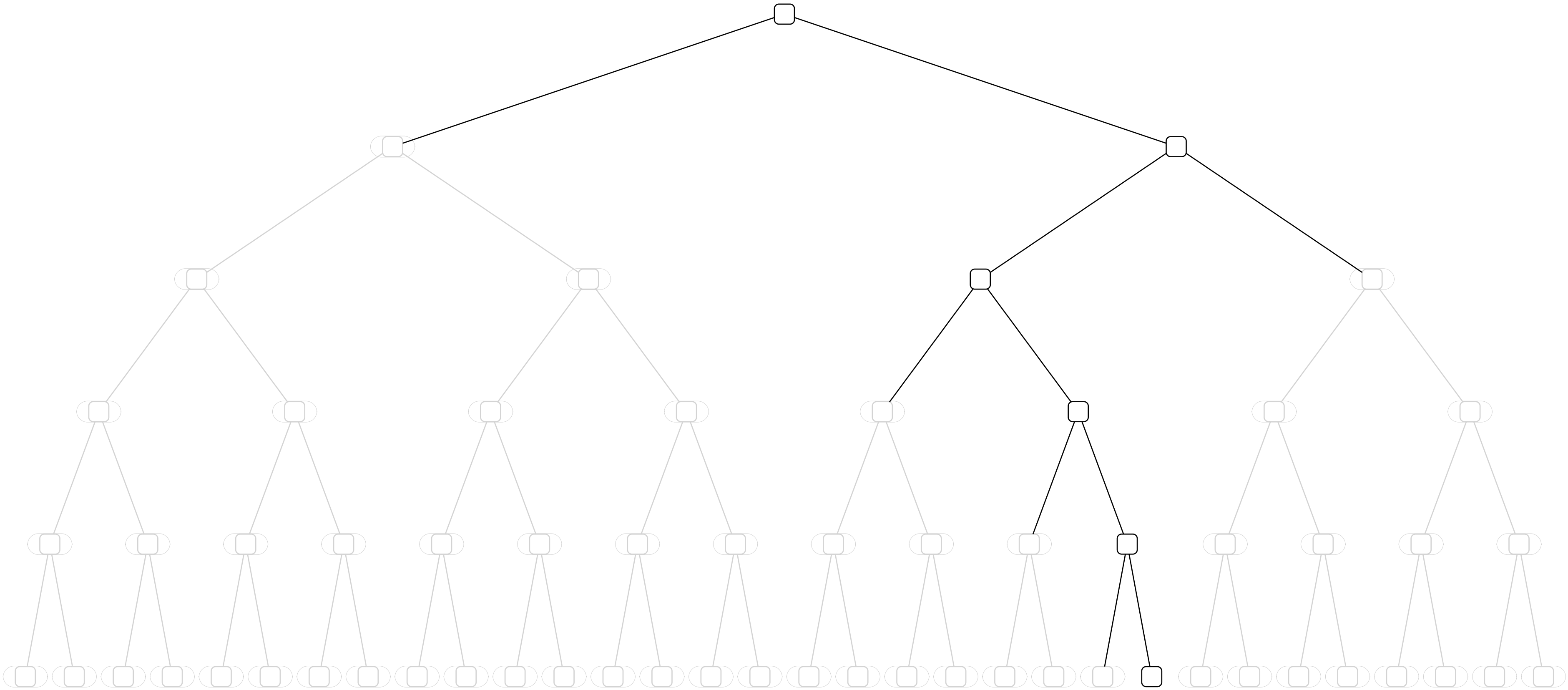
Beam search

Local search

Backtracking search



Greedy search



Greedy search

Algorithm: greedy search

Partial assignment $x \leftarrow \{\}$

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

Extend:

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

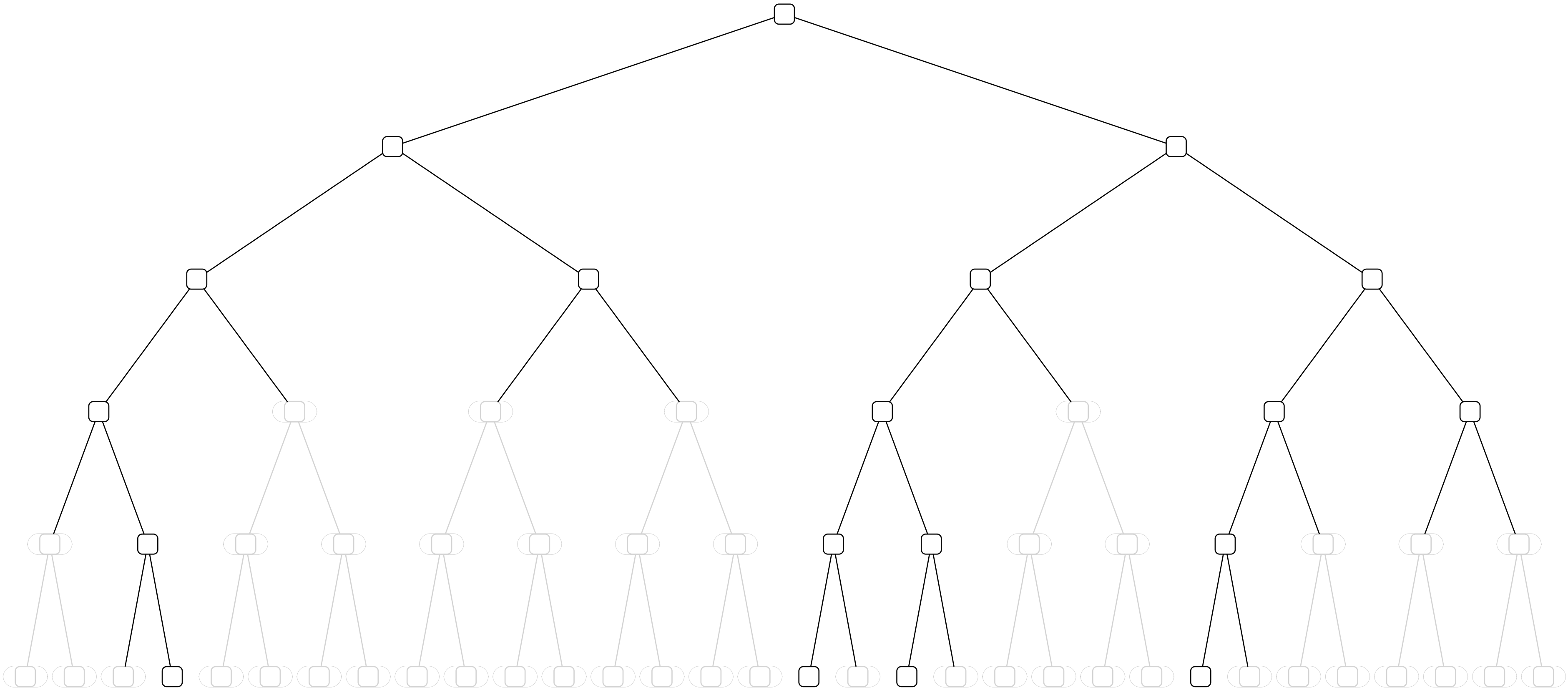
Prune:

$x \leftarrow x_v$ 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 $i = 1, \ldots, n$:

Extend:

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

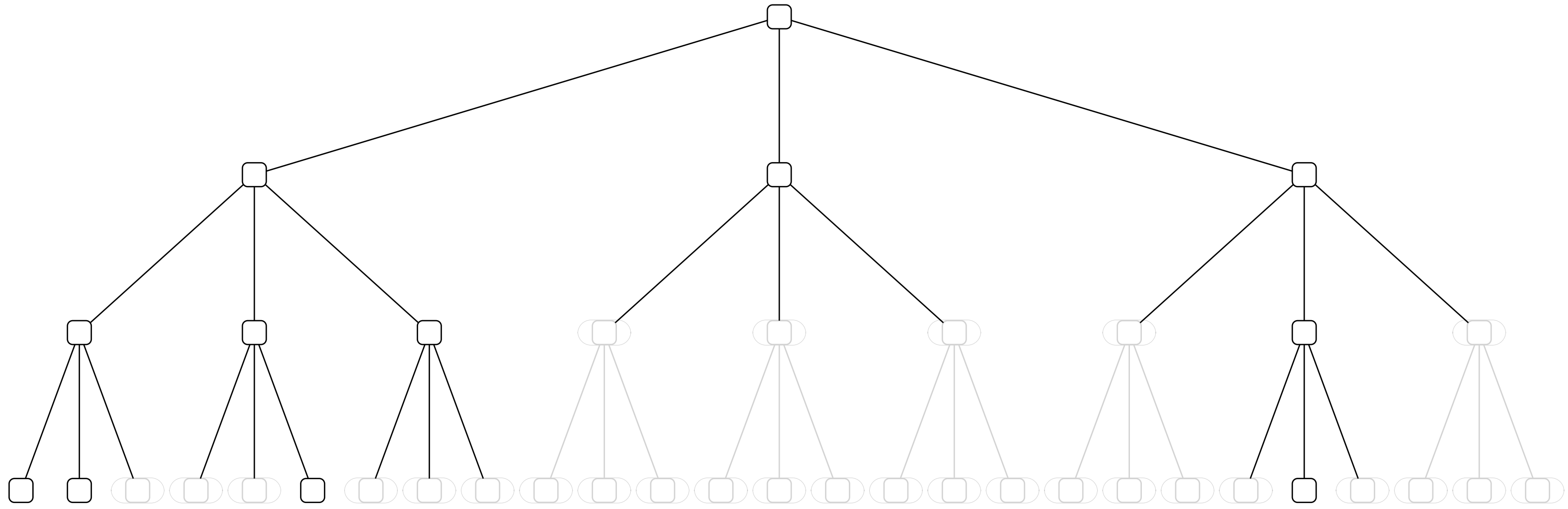
Prune:

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

Not guaranteed to find maximum weight assignment!

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

Time complexity

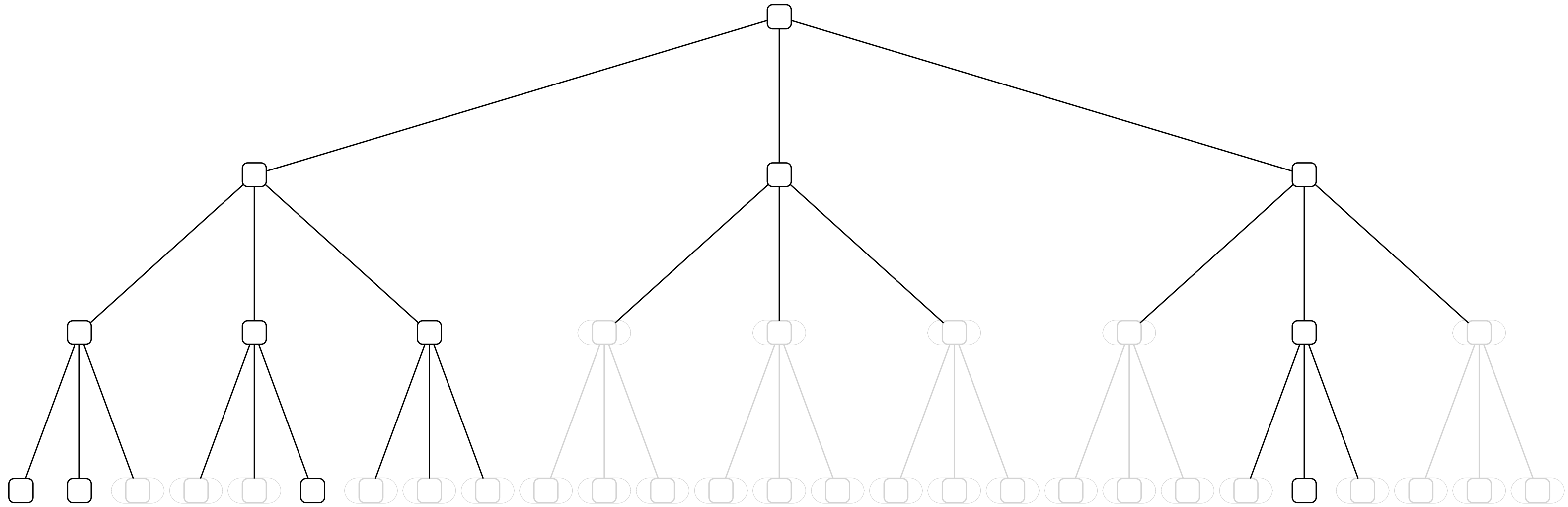


n variables (depth)

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

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)

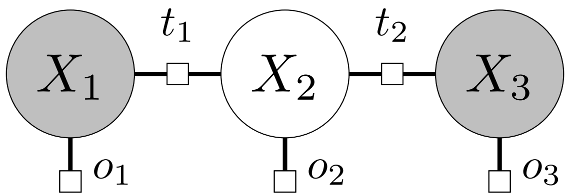
Backtracking search : DFS :: beam search : pruned BFS

Search strategies

Backtracking/beam search: extend partial assignments

Local search: modify complete assignments

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)

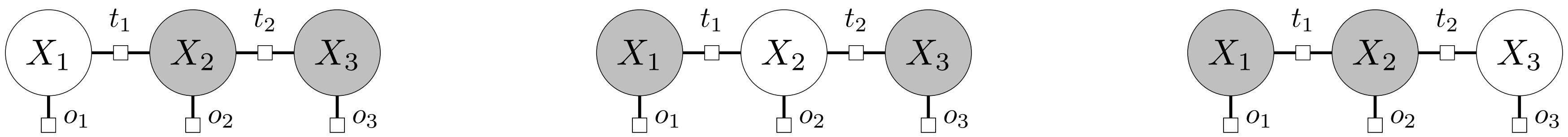
Algorithm: iterated conditional modes (ICM)

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

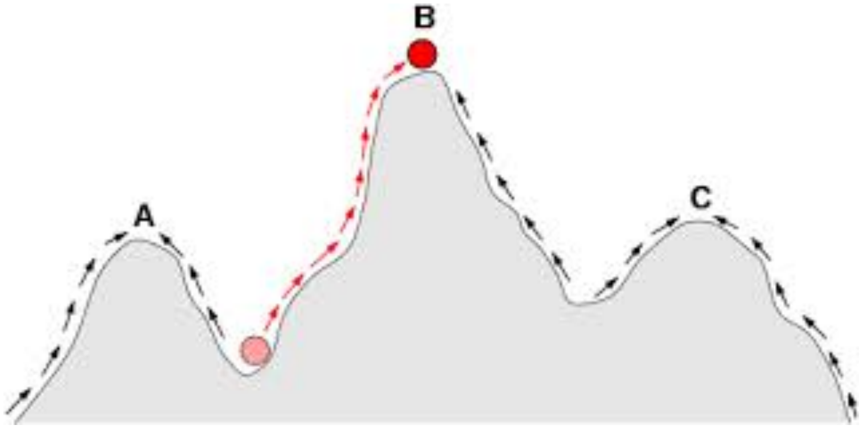
  $x \leftarrow x_v$ with highest weight



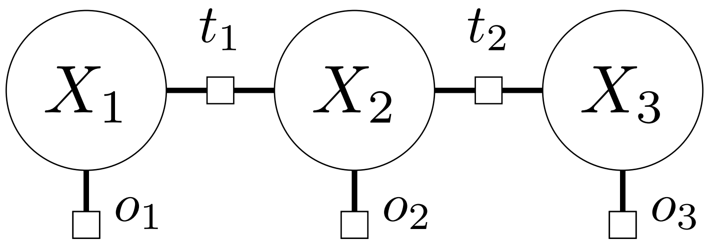
[demo: `iteratedConditionalModes()`]

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!



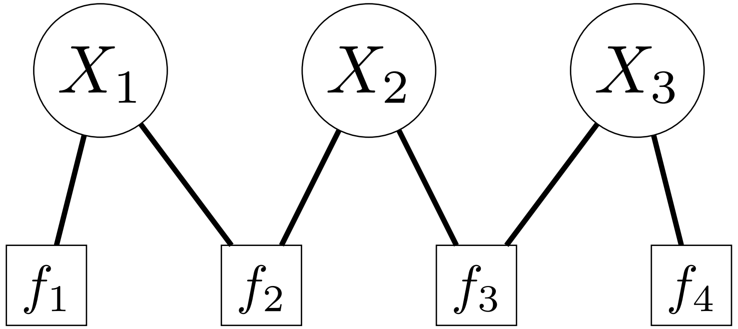
Summary



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

Markov networks: overview

Review: factor graphs



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**:

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

Definition

Definition: Markov network

A Markov network is a factor graph which defines a joint distribution over random variables $X = (X_1, \dots, X_n)$:

$$\mathbb{P}(X = x) = \frac{\text{Weight}(x)}{Z}$$

where $Z = \sum_{x'} \text{Weight}(x')$ is the normalization constant.

x_1	x_2	x_3	Weight(x)	$\mathbb{P}(X = x)$
0	1	1	4	0.15
0	1	2	4	0.15
1	1	1	4	0.15
1	1	2	4	0.15
1	2	1	2	0.08
1	2	2	8	0.31

$$Z = 4 + 4 + 4 + 4 + 2 + 8 = 26$$

Represents uncertainty!

Marginal probabilities

Example question: where was the object at time step 2 (X_2)?

Definition: Marginal probability

The marginal probability of $X_i = v$ is given by:

$$\mathbb{P}(X_i = v) = \sum_{x : x_i = v} \mathbb{P}(X = x)$$

Object tracking example:

x_1	x_2	x_3	Weight(x)	$\mathbb{P}(X = x)$
0	1	1	4	0.15
0	1	2	4	0.15
1	1	1	4	0.15
1	1	2	4	0.15
1	2	1	2	0.08
1	2	2	8	0.31

$$\mathbb{P}(X_2 = 1) = 0.15 + 0.15 + 0.15 + 0.15 = 0.62$$

$$\mathbb{P}(X_2 = 2) = 0.08 + 0.31 = 0.38$$

Note: different than max weight assignment!

Summary

Markov networks = factor graphs + probability

- Normalize weights to get probablity distribution
- Can compute marginal probabilities to focus on variables

CSPs	Markov networks
variables	random variables
weights	probabilities
maximum weight assignment	marginal probabilities

Gibbs sampling

Algorithm: Gibbs sampling

Initialize x to a random complete assignment

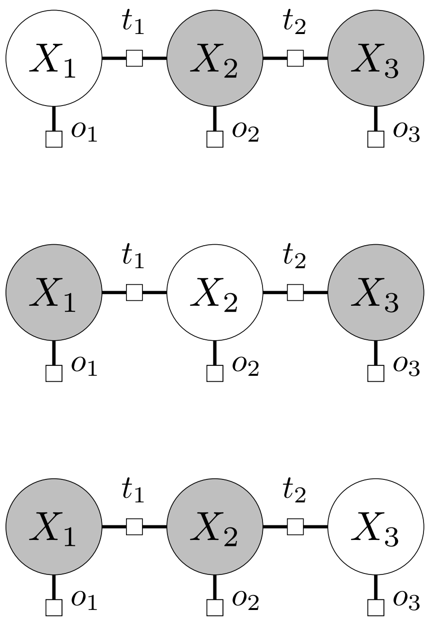
Loop through $i = 1, \dots, n$ until convergence:

  Set $x_i = v$ with prob. $\mathbb{P}(X_i = v \mid X_{-i} = x_{-i})$

    (X_{-i} denotes all variables except X_i)

  Increment $\text{count}_i(x_i)$

Estimate $\hat{\mathbb{P}}(X_i = x_i) = \frac{\text{count}_i(x_i)}{\sum_v \text{count}_i(v)}$



Example: sampling one variable

$\text{Weight}(x \cup \{X_2 : 0\}) = 1$ prob. 0.2

$\text{Weight}(x \cup \{X_2 : 1\}) = 2$ prob. 0.4

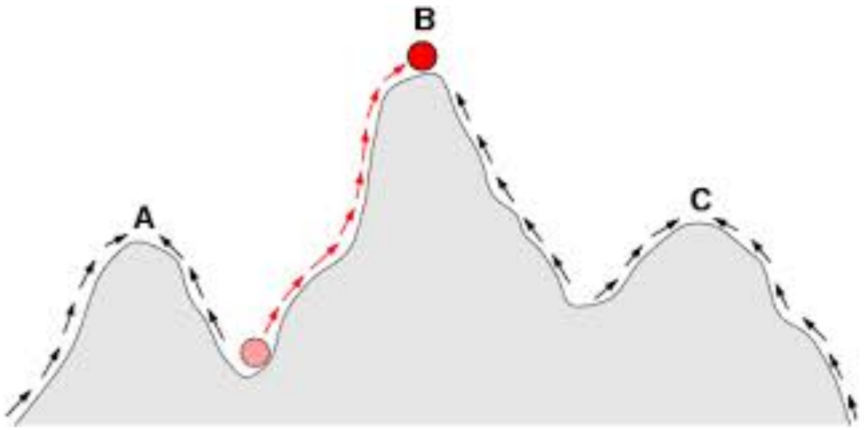
$\text{Weight}(x \cup \{X_2 : 2\}) = 2$ prob. 0.4

[demo]

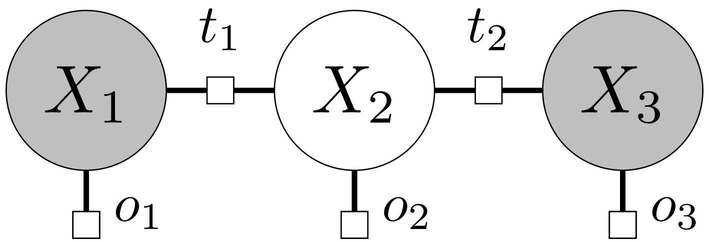
Search versus sampling

Iterated Conditional Modes	Gibbs sampling
maximum weight assignment	marginal probabilities
choose best value	sample a value
converges to local optimum	marginals converge to correct answer*

*under technical conditions (sufficient condition: all weights positive), but could take exponential time



Summary



- **Objective**: compute marginal probabilities $\mathbb{P}(X_i = x_i)$
- **Gibbs sampling**: sample one variable at a time, count visitations
- **More generally**: Markov chain Monte Carlo (MCMC) powerful toolkit of randomized procedures