	完备	最优	Time	Space
BFS	✓	X(除非cost同)	$O(b^s)$	$O(b^s)$
DFS	X	X	$O(b^m)$	$O(bm)$
UCS	✓	✓(cost非负)	$O(b^{C^*/\varepsilon})$	$O(b^{C^*/\varepsilon})$
Greedy S	X	X	\	\
A*	✓	✓(consistent)	\	\

Greedy worst case: badly-guided DFS

Iterative Deepening:

Idea: get DFS's space advantage with BFS's time / shallow-solution advantages

- Run a DFS with depth limit 1. If no solution…
- Run a DFS with depth limit 2. If no solution…
- Run a DFS with depth limit 3. …..

Pancake Heuristic: the number of the largest pancake that is still out of place.

Graph Search: 重要！(不重复探节点)

Idea: never expand a state twice

How to implement:

- Tree search + set of expanded states ("closed set")
- Expand the search tree node-by-node, but…
- Before expanding a node, check to make sure its state has never been expanded before
- If not new, skip it, if new add to closed set

BackTracking:

Backtracking search is the basic uninformed algorithm for solving CSPs

Idea 1: One variable at a time
- Variable assignments are commutative, so fix ordering
- I.e., [WA = red then NT = green] same as [NT = green then WA = red]
- Only need to consider assignments to a single variable at each step

Idea 2: Check constraints as you go
- I.e. consider only values which do not conflict previous assignments
- Might have to do some computation to check the constraints
- "Incremental goal test"

Depth-first search with these two improvements is called *backtracking search*

```
function BACKTRACKING-SEARCH(csp) returns solution/failure
   return RECURSIVE-BACKTRACKING({ }, csp)

function RECURSIVE-BACKTRACKING(assignment, csp) returns soln/failure
   if assignment is complete then return assignment
   var ← SELECT-UNASSIGNED-VARIABLE(VARIABLES[csp], assignment, csp)
   for each value in ORDER-DOMAIN-VALUES(var, assignment, csp) do
      if value is consistent with assignment given CONSTRAINTS[csp] then
         add {var = value} to assignment
         result ← RECURSIVE-BACKTRACKING(assignment, csp)
         if result ≠ failure then return result
         remove {var = value} from assignment
   return failure
```

Filtering:*(forward checking and arc)*

Filtering: Keep track of domains for unassigned variables and cross off bad options

Forward checking: Cross off values that violate a constraint when added to the existing assignment; whenever any variable has no value left, we backtrack

An arc X → Y is consistent iff for *every* x in the tail there is *some* y in the head which could be assigned without violating a constraint

Important: If Y loses a value, then arc X → Y needs to be rechecked!

Arc consistency detects failure earlier than forward checking

```
function AC-3(csp) returns the CSP, possibly with reduced domains
   inputs: csp, a binary CSP with variables {X1, X2, ..., Xn}
   local variables: queue, a queue of arcs, initially all the arcs in csp

   while queue is not empty do                 tail    head
      (Xi, Xj) ← REMOVE-FIRST(queue)
      if REMOVE-INCONSISTENT-VALUES(Xi, Xj) then
         for each Xk in NEIGHBORS[Xi] do
            add (Xk, Xi) to queue

function REMOVE-INCONSISTENT-VALUES(Xi, Xj) returns true iff succeeds
   removed ← false
   for each x in DOMAIN[Xi] do
      if no value y in DOMAIN[Xj] allows (x,y) to satisfy the constraint Xi ↔ Xj
         then delete x from DOMAIN[Xi]; removed ← true
   return removed
```

An arc X → Y is consistent iff for *every* x in the tail there is *some* y in the head which could be assigned without violating a constraint

k-consistency:

As an interesting parting note about consistency, arc consistency is a subset of a more generalized notion of consistency known as **k-consistency**, which when enforced guarantees that for any set of k nodes in the CSP, a consistent assignment to any subset of $k-1$ nodes guarantees that the k^{th} node will have at least one consistent value. This idea can be further extended through the idea of **strong k-consistency**. A graph that is strong k-consistent possesses the property that any subset of k nodes is not only k-consistent but also $k-1, k-2, \dots, 1$ consistent as well. Not surprisingly, imposing a higher degree of consistency on a

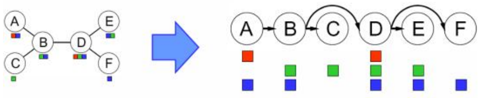
Ordering:

Value Ordering: Least Constraining Value
- Given a choice of variable, choose the *least constraining value*

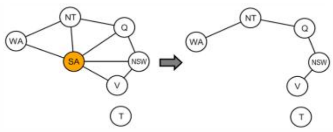
Variable Ordering: Minimum remaining values (MRV):
- Choose the variable with the fewest legal left values in its domain
- Also called "most constrained variable"

Structure:

- Order: Choose a root variable, order variables so that parents precede children



- Remove backward: For i = n : 2, apply RemoveInconsistent(Parent(X_i),X_i)
- Assign forward: For i = 1 : n, assign X_i consistently with Parent(X_i)



- Cutset: a set of variables s.t. the remaining constraint graph is a tree
- Cutset conditioning: instantiate (in all ways) the cutset and solve the remaining tree-structured CSP
  - Cutset size c gives runtime O((d^c) (n-c) d^2), very fast for small c

Cutset 删去之后，剩下的是森林也可以

Iterative Algorithm for CSP:

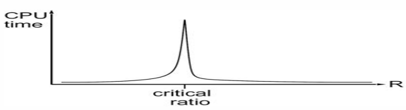
Idea:
- Take a complete assignment with unsatisfied constraints
- *Reassign* variable values to minimize conflicts

Algorithm: While not solved,
- Variable selection: randomly select any conflicted variable
- Value selection: min-conflicts heuristic:
  - Choose a value that violates the fewest constraints

Given random initial state, can solve n-queens in almost constant time for arbitrary n with high probability (e.g., n = 10,000,000)!

The same appears to be true for any randomly-generated CSP *except* in a narrow range of the ratio

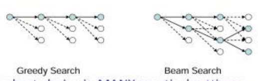
$$R = \frac{\text{number of constraints}}{\text{number of variables}}$$



Local Search: improve a single option until you can't make it better; Generally much faster and more memory efficient (but incomplete and suboptimal); Hill-Beam-Annealing-Genetic:

Simple, general idea:
- Start wherever
- Repeat: move to the best neighboring state
- If no neighbors better than current, quit

Like greedy hill climbing search, but keep K states at all times:

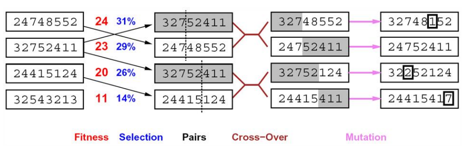


The best choice in MANY practical settings

Idea: Escape local maxima by allowing downhill moves
- Pick a random move
- Always accept an uphill move
- Accept a downhill move with probability $e^{-\Delta E / T}$
- But make the probability smaller (by decreasing T) as time goes on

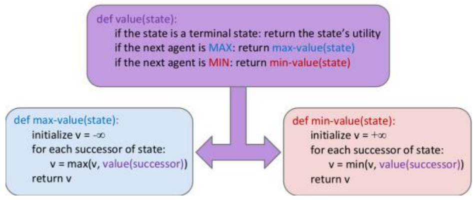
Theoretical guarantee
- If T decreased slowly enough, will converge to optimal state!

Sounds like magic, but reality is reality:
- The more downhill steps you need to escape a local optimum, the less likely you are to ever make them all



- Genetic algorithms use a natural selection metaphor
  - Keep the best (or sample) N states at each step based on a fitness function
  - Pairwise crossover operators, with optional mutation to give variety

Minimax:



How efficient is minimax?
- Just like (exhaustive) DFS
- Time: $O(b^m)$
- Space: O(bm)

Ideal function: returns the actual minimax value of the position

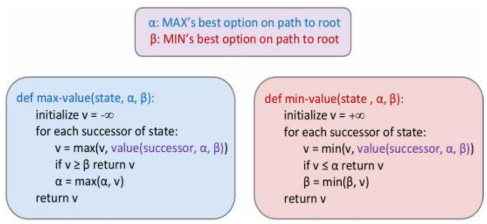
A simple solution in practice: weighted linear sum of features:

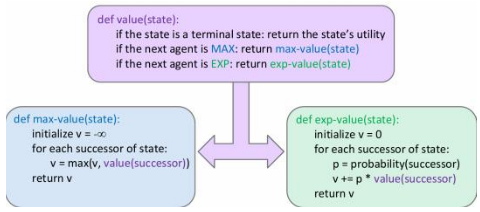
$$Eval(s) = w_1 f_1(s) + w_2 f_2(s) + \ldots + w_n f_n(s)$$

Alpha-Beta Pruning:

General configuration (MIN version)
- We're computing the MIN-VALUE at some node *n*
- We're looping over *n*'s children, so *n*'s estimate is decreasing
- Let *a* be the best value that MAX can get at any choice point along the current path from the root
- **If *n* becomes worse than *a*, then we can stop considering *n*'s other children**
- Reason: if *n* is eventually chosen, then the nodes along the path shall all have the value of *n*, but *n* is worse than *a* and hence the path shall not be chosen at the MAX

Perhaps the simplest check is as follows: pruning of children of a minimizer node m is possible (for some assignment to the terminal nodes), when both of the following conditions are met: (i) the value of another child of m has already been determined, (ii) somewhere on the path from m to the root node, there is a maximizer node M for which an alternative option has already been explored. The pruning will then happen if any such alternative option for the maximizer had a higher value than the value of the "another child" of m for which the value was already determined.





- soundness (可靠性): derivations produce only entailed sentences
- completeness (完备性): derivations can produce all entailed sentences

Forward, backward chaining are linear-time, complete for Horn clauses

Resolution is complete for propositional logic

Propositional logic lacks expressive power

Resolution&TruthTable:

complete&sound for prop.logic

The best way to prove *KB* |= α ?
- Proof by contradiction, i.e., show *KB*∧¬α is unsatisfiable

1. Convert *KB*∧¬α to CNF
2. Repeatedly apply the resolution rule to add new clauses, until one of the two things happens
   a) Two clauses resolve to yield the empty clause, in which case *KB* entails α
   b) There is no new clause that can be added, in which case *KB* does not entail α

Horn logic: only (strict) Horn clauses are allowed
- A Horn clause has the form:
  P1 ∧ P2 ∧ P3 ... ∧ Pn ⇒ Q
  or alternatively
  ¬P1 ∨ ¬P2 ∨ ¬P3 ... ∨ ¬Pn ∨ Q
  where Ps and Q are non-negated proposition symbols (atoms)
- n can be zero, i.e., the clause contains a single atom

Modus Ponens

$$\frac{\alpha_1, \ldots, \alpha_n, \quad \overbrace{\alpha_1 \wedge \ldots \wedge \alpha_n}^{\text{premises}} \Rightarrow \overbrace{\beta}^{\text{conclusion}}}{\beta}$$

Only compl&sound for Horn Logic

FC&BC: not com&sou for prop.logic&FOL

Inference algorithms (for Horn logic)
- Forward chaining, backward chaining
- These algorithms are very natural and run in linear time

FC is data-driven, automatic, unconscious processing,
- e.g., object recognition, routine decisions
- May do lots of work that is irrelevant to the goal

BC is goal-driven, appropriate for problem-solving,
- e.g., Where are my keys? How do I get into a PhD program?
- Complexity of BC can be much less than linear in size of KB

FOL syntax: (and every variable must be bound)

Atomic sentence = *predicate* (*term*$_1$,...,*term*$_n$)
or *term*$_1$ = *term*$_2$

Term = *constant* or *variable*
or *function* (*term*$_1$,...,*term*$_n$)

Complex sentences are made from atomic sentences using connectives

Typically, ∧ is the main connective with ∃
Common mistake: using ⇒ as the main connective with ∃:
∃*x At*(*x*,*STU*) ⇒ *Smart*(*x*)
is true if there is anyone who is not at STU!

Typically, ⇒ is the main connective with ∀
Common mistake: using ∧ as the main connective with ∀:
∀*x At*(*x*,*STU*) ∧ *Smart*(*x*)
means "Everyone is at STU and everyone is smart"

For any sentence α, variable *v* and ground term *g*: (Term without variables)

$$\frac{\forall v\ \alpha}{\text{Subst}(\{v/g\}, \alpha)}$$ ← Substitute *v* with *g* in *α*

Every instantiation of a universally quantified sentence is entailed by it
UI can be applied multiple times to add new sentences

For any sentence α, variable v, and constant symbol k that does not appear elsewhere in the knowledge base:

$$\frac{\exists v\ \alpha}{\text{Subst}(\{v/k\}, \alpha)}$$

EI can be applied once to replace an existential sentence

Unification finds substitutions that make different expressions identical

Resolution:

Full first-order version:

$$\frac{l_1 \vee \cdots \vee l_k, \qquad m_1 \vee \cdots \vee m_n}{(l_1 \vee \cdots \vee l_{i-1} \vee l_{i+1} \vee \cdots \vee l_k \vee m_1 \vee \cdots \vee m_{j-1} \vee m_{j+1} \vee \cdots \vee m_n)\theta}$$

where $\text{Unify}(l_i, \neg m_j) = \theta$.

Generalized Modus Ponens (GMP)

$$\frac{p_1', p_2', \ldots, p_n', (p_1 \wedge p_2 \wedge \ldots \wedge p_n \Rightarrow q)}{q\theta}$$ where $p_i'\theta = p_i\theta$ for all *i*

GMP: incomplete for FOL
–Not every sentence can be converted to Horn form
GMP: complete for FOL KB of definite clauses

FC and BC are complete for Horn KBs but are incomplete for general FOL KBs:

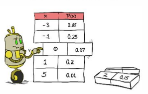
Every variable is conditionally independent of its non-descendants given its parents
Conditional independence semantics <=> global semantics
A variable's Markov blanket consists of parents, children, children's other parents
Every variable is conditionally independent of all other variables given its Markov blanket

Inference by Enumeration

- General case:
  - Evidence variables: $E_1 \ldots E_k = e_1 \ldots e_k$
  - Query variable: Q
  - Hidden variables: $H_1 \ldots H_r$

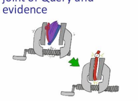
  } $X_1, X_2, \ldots X_n$ *All variables*
- We want: $P(Q|e_1 \ldots e_k)$

- Step 1: Select the entries consistent with the evidence



$P(Q, e_1 \ldots e_k) = \sum_{h_1 \ldots h_r} P(\underbrace{Q, h_1 \ldots h_r, e_1 \ldots e_k}_{X_1, X_2, \ldots X_n})$

- Step 2: Sum out H to get joint of Query and evidence



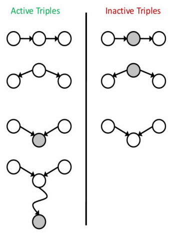
- Step 3: Normalize

$$\times \frac{1}{Z}$$

$Z = \sum_q P(Q, e_1 \cdots e_k)$

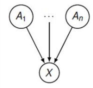
$P(Q|e_1 \cdots e_k) = \frac{1}{Z} P(Q, e_1 \cdots e_k)$

- Question: X, Y, Z are non-intersecting subsets of nodes. Are X and Y conditionally independent given Z?
- A triple is active in the following three cases
  - Causal chain A → B → C where B is unobserved (either direction)
  - Common cause A ← B → C where B is unobserved
  - Common effect (aka v-structure) A → B ← C where B *or one of its descendents* is observed
- A path is active if each triple along the path is active
- A path is blocked if it contains a single inactive triple
- If all paths from X to Y are blocked, then X is said to be "**d-separated**" from Y by Z
- If d-separated, then X and Y are conditionally independent given Z



A directed, acyclic graph

Conditional distributions for each node given its ***parent variables*** in the graph
- ***CPT***: conditional probability table: each row is a distribution for child given a configuration of its parents
- Description of a noisy "causal" process



$P(X|A_1, \cdots, A_n)$

A Bayes net = Topology (graph) + Local Conditional Probabilities

General formula for sparse BNs

- Suppose
  - *n* variables
  - Maximum domain size is *d*
  - Maximum number of parents is *k*
- Full joint distribution has size $O(d^n)$
- Bayes net has size $O(n \cdot d^{k+1})$
  - Linear scaling with *n* as long as causal structure is local

Full joint distribution tables answer every question, but:
- Size is exponential in the number of variables
- Need gazillions of examples to learn the probabilities
- Inference by enumeration (summing out hiddens) is too slow

Bayesian networks:
- Express all the conditional independence relationships in a domain
- Factor the joint distribution into a product of small conditionals
- Often reduce size from exponential to linear
- Faster learning from fewer examples
- Faster inference (linear time in some important cases)

A Bayesian network encodes a joint distribution with a directed acyclic graph
- A CPT captures uncertainty between a node and its parents

A Markov network (or Markov random field) encodes a joint distribution with an undirected graph
- A potential function captures uncertainty between a clique of nodes

Moralize: 如果两个节点指向同一个子节点，那么转化为无向图时两点间连线；这样可以将 BN 转化为 MN；同时***挑出所有涉及变量及其祖先***并 moralize 后得到的图也可用来判断 BN 两点间是否(条件)独立：无向图中删除条件节点及其边，如果两点相连，那么就不独立！因此：不是所有 BN 都可转为 MN ***且包含所有的(条件)独立信息***。(但编码的分布一样)

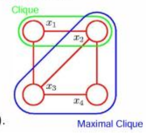
Markov network = undirected graph + potential functions
- For each clique (or max clique), a potential function is defined
  - A potential function is not locally normalized, i.e., it doesn't encode probabilities
- A joint probability is proportional to the product of potentials

$$p(\mathbf{x}) = \frac{1}{Z} \prod_C \psi_C(\mathbf{x}_C)$$

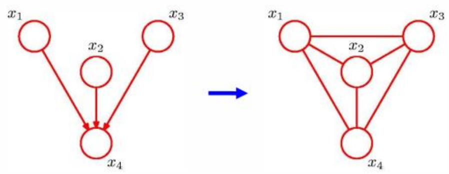
where $\psi_C(\mathbf{x}_C)$ is the potential over clique C and

$$Z = \sum_{\mathbf{x}} \prod_C \psi_C(\mathbf{x}_C)$$

is the normalization coefficient (aka. partition function).



- Additional links (moralization)



$$p(\mathbf{x}) = p(x_1)p(x_2|x_1)\,p(x_3|x_2) \cdots p(x_N|x_{N-1})$$

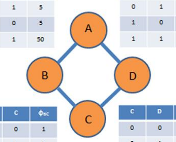
$$p(\mathbf{x}) = \frac{1}{Z}\,\psi_{1,2}(x_1, x_2)\,\psi_{2,3}(x_2, x_3) \cdots \psi_{N-1,N}(x_{N-1}, x_N)$$

A	B	C	D	$\phi_{AB}\phi_{BC}\phi_{CD}\phi_{AD}$
0	0	0	0	250
0	0	0	1	37500
0	0	1	0	50000
0	0	1	1	625000
0	1	0	0	1125
0	1	0	1	168750
0	1	1	0	50000
0	1	1	1	625000
1	0	0	0	250
1	0	0	1	375
1	0	1	0	50000
1	0	1	1	6250
1	1	0	0	112500
1	1	0	1	168750
1	1	1	0	5000000
1	1	1	1	625000

Z = 7520750

A	B	ϕ_{AB}
0	0	50
0	1	5
1	0	5
1	1	50

A	D	ϕ_{AD}
0	0	5
0	1	50
1	0	50
1	1	5



B	C	ϕ_{BC}
0	0	1
0	1	5
1	0	45
1	1	50

C	D	ϕ_{CD}
0	0	1
0	1	15
1	0	40
1	1	50

Bayesian Network → Markov Network

- Steps
  1. Moralization
  2. Construct potential functions from CPTs
- The BN and MN encode the same distribution

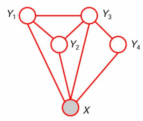
An extension of MN (aka. Markov random field) where everything is conditioned on an input

$$P(y|x) = \frac{1}{Z(x)} \prod_C \psi_C(y_C, x)$$

where $\psi_C(y_C, x)$ is the potential over clique C and

$$Z(x) = \sum_y \prod_C \psi_C(y_C, x)$$

is the normalization coefficient.



Which logic is BN/MN more similar to: PL? FOL?
- Boolean nodes represent propositions
- No explicit representation of objects, relations, quantifiers

BN/MN can be seen as a probabilistic extension of PL
PL can be seen as BN/MN with deterministic CPTs/potentials

Generative models
- A generative model represents a joint distribution $P(X_1, X_2, \ldots, X_n)$
- Both BN and MN are generative models

Discriminative models
- In some scenarios, we only care about predicting queries from evidence
- A discriminative model represents a conditional distribution $P(Y_1, Y_2, \ldots, Y_n|X)$
- It does not model $P(X)$