Propositional Logic

Knowledge base (domain-specific content)
Inference engine (domain-indep. algo.)
Syntax: what sentences are allowed
Semantics: what sen. are t/f in each model
- conjunction: disjunction:
- implication: biconditional:

valid: T in all models
satisfiable: T in some models
unsatis.: T in no models
S is valid iff. not S is unsatis.
Entailment: a|=b (a T, b also T)
Proof: a|-b (a demonstration of entail. from a to b)
- model checking
- application of inference rules
  - axiom: a sen. known to be T
  - rule of inference

Sound inference: everything can be proved is in fact entailed
Complete inference: everything that is entailed can be proved
CNF (Conjunctive Normal Form)
- conjunction of disjunction of literals

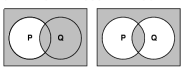
Resolution inference rule (for CNF) (sound and complete)
Inference in propo. logic is in general NBC

Suppose l_i is $\neg m_j$

$$\frac{l_1 \vee \dots \vee l_k, \qquad m_1 \vee \dots \vee m_n}{l_1 \vee \dots \vee l_{i-1} \vee l_{i+1} \vee \dots \vee l_k \vee m_1 \vee \dots \vee m_{j-1} \vee m_{j+1} \vee \dots \vee m_n}$$

P	Q	$\neg P$	$P \wedge Q$	$P \vee Q$	$P \Rightarrow Q$	$P \Leftrightarrow Q$
false	*false*	*true*	*false*	*false*	*true*	*true*
false	*true*	*true*	*false*	*true*	*true*	*false*
true	*false*	*false*	*false*	*true*	*false*	*false*
true	*true*	*false*	*true*	*true*	*true*	*true*

$(\alpha \wedge \beta) \equiv (\beta \wedge \alpha)$ commutativity of $\wedge$
$(\alpha \vee \beta) \equiv (\beta \vee \alpha)$ commutativity of $\vee$
$((\alpha \wedge \beta) \wedge \gamma) \equiv (\alpha \wedge (\beta \wedge \gamma))$ associativity of $\wedge$
$((\alpha \vee \beta) \vee \gamma) \equiv (\alpha \vee (\beta \vee \gamma))$ associativity of $\vee$
$\neg(\neg \alpha) \equiv \alpha$ double-negation elimination
$(\alpha \Rightarrow \beta) \equiv (\neg \beta \Rightarrow \neg \alpha)$ contraposition
$(\alpha \Rightarrow \beta) \equiv (\neg \alpha \vee \beta)$ implication elimination
$(\alpha \Leftrightarrow \beta) \equiv ((\alpha \Rightarrow \beta) \wedge (\beta \Rightarrow \alpha))$ biconditional elimination
$\neg(\alpha \wedge \beta) \equiv (\neg \alpha \vee \neg \beta)$ de Morgan
$\neg(\alpha \vee \beta) \equiv (\neg \alpha \wedge \neg \beta)$ de Morgan
$(\alpha \wedge (\beta \vee \gamma)) \equiv ((\alpha \wedge \beta) \vee (\alpha \wedge \gamma))$ distributivity of $\wedge$ over $\vee$
$(\alpha \vee (\beta \wedge \gamma)) \equiv ((\alpha \vee \beta) \wedge (\alpha \vee \gamma))$ distributivity of $\vee$ over $\wedge$



P ⇒ Q P⇔Q

- 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

$$\frac{\overbrace{\alpha 1, \dots, \alpha n, \quad \alpha 1 \wedge \dots \wedge \alpha n}^{\text{premises}} \Rightarrow \overbrace{\beta}^{\text{conclusion}}}{\beta}$$

  - Modus Ponens is sound and complete for Horn logic

- 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

Partially observable Pac-man:

- NxN world for T time steps => $N^2T + N^2 + 4T + 4T = O(N^2T)$ symbols
- **Sensor model**
- **Transition model**
- A state symbol gets its value according to a successor-state axiom
  $X_t \Leftrightarrow [X_{t-1} \wedge \neg$(some $action_{t-1}$ made it false)] ∨
  $[\neg X_{t-1} \wedge$ (some $action_{t-1}$ made it true)]

- Full first-order version:

$$\frac{\ell_1 \vee \cdots \vee \ell_k, \qquad m_1 \vee \cdots \vee m_n}{(\ell_1 \vee \cdots \vee \ell_{i-1} \vee \ell_{i+1} \vee \cdots \vee \ell_k \vee m_1 \vee \cdots \vee m_{j-1} \vee m_{j+1} \vee \cdots \vee m_n)\theta}$$

where $\mathrm{Unify}(\ell_i, \neg m_j) = \theta$.

- The two clauses are assumed to be standardized apart so that they share no variables.

First Order Logic:

- Logical symbols
  - Connectives ¬, ∧, ∨, ⇒, ⇔
  - Quantifiers ∀, ∃
  - Variables x, y, a, b, ...
  - Equality =
- Non-logical symbols (ontology)
  - Constants KingArthur, 2, ShanghaiTech, ...
  - Predicates Brother, >, ...
  - Functions Sqrt, LeftLegOf, ...

Atomic sentence = *predicate* $(term_1,\dots,term_n)$
or $term_1 = term_2$

Term = *constant* or *variable*
or *function* $(term_1,\dots,term_n)$

- Sentences are true with respect to a model, which contains
  - Objects and relations among them
  - Interpretation specifying referents for
    - constant symbols → objects
    - predicate symbols → relations
    - function symbols → functional relations
- An atomic sentence *predicate*$(term_1,\dots,term_n)$ is true iff the objects referred to by $term_1,\dots,term_n$ are in the relation referred to by *predicate*

∀x *P* is true in a model m iff *P* is true with *x* being each possible object in the model ⇒
∃x *P* is true in a model m iff *P* is true with *x* being some possible object in the model ∧

- A variable is free in a formula if it is not quantified
  - e.g., ∀x *P(x,y)*
- A variable is bound in a formula if it is quantified
  - e.g., ∀x ∃y *P(x,y)*
- In a FOL sentence, every variable must be bound.

Universal instantiation (UI)

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

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

- Every instantiation of a universally quantified sentence is entailed by it

Unification:
- Unify(α,β) = θ if αθ = βθ
- To unify Knows(John,x) and Knows(y,z),
  θ = {y/John, x/z } or θ = {y/John, x/John, z/John}
  - The first unifier is more general than the second.
- There is a single most general unifier (MGU) that is unique up to renaming of variables.
  - MGU = { y/John, x/z }

FOL Inference:
- Horn logic (the FOL case)
  - Forward chaining
  - Backward chaining
- General FOL
  - Resolution

Generalized Modeus Ponens (GMP)

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

Forward chaining:
- Sound and complete for first-order Horn clauses
- FC terminates for first-order Horn clauses with no functions (Datalog) in finite number of iterations
- In general, FC may not terminate if α is not entailed
  - This is unavoidable: entailment with Horn clauses is also semi-decidable

Backward chaining:
- Depth-first recursive proof search: space is linear in size of proof
- Avoid infinite loops by checking current goal against every goal on stack
- Avoid repeated subgoals by caching previous results
- Widely used for logic programming

Logic programming:
- Basis: backward chaining with Horn clauses
  - Program = set of Horn clauses
  - Inference: depth-first, left-to-right backward chaining
- Additions:
  - Built-in predicates for arithmetic etc., e.g., X is Y*Z+3
  - Built-in predicates that have side effects (e.g., input and output predicates, assert/retract predicates)
- Closed-world assumption ("negation as failure")

Resolution:
- Inference algorithm: applying resolution steps to CNF(KB and not α)
- Resolution is sound and complete for FOL

Conversion to CNF

∀x [∀y *Animal(y)* ⇒ *Loves(x,y)*] ⇒ [∃y *Loves(y,x)*]

1. Eliminate biconditionals and implications
   ∀x [¬∀y ¬*Animal(y)* ∨ *Loves(x,y)*] ∨ [∃y *Loves(y,x)*]
2. Move ¬ inwards: ¬∀x p ≡ ∃x ¬p, ¬∃x p ≡ ∀x ¬p
   ∀x [∃y ¬(¬Animal(y) ∨ Loves(x,y))] ∨ [∃y Loves(y,x)]
   ∀x [∃y ¬¬Animal(y) ∧ ¬Loves(x,y)] ∨ [∃y Loves(y,x)]
   ∀x [∃y Animal(y) ∧ ¬Loves(x,y)] ∨ [∃y Loves(y,x)]
3. Standardize variables: each quantifier should use a different variable
   ∀x [∃y Animal(y) ∧ ¬Loves(x,y)] ∨ [∃z Loves(z,x)]
4. Skolemize: a more general form of existential instantiation. Each existential variable is replaced by a Skolem function of the enclosing universally quantified variables:
   ∀x [Animal(F(x)) ∧ ¬Loves(x,F(x))] ∨ Loves(G(x),x)
5. Drop universal quantifiers:
   [Animal(F(x)) ∧ ¬Loves(x,F(x))] ∨ Loves(G(x),x)
6. Distribute ∨ over ∧ :
   [Animal(F(x)) ∨ Loves(G(x),x)] ∧ [¬Loves(x,F(x)) ∨ Loves(G(x),x)]

Decision theory = utility theory + probability theory
Maximize expected utility : $a^* = \mathrm{argmax}_a \sum_s P(s \mid a)\ U(s)$

Bayes' Rule: P(x|y)=P(y|x)P(x)/P(y)

- X is conditionally independent of Y given Z $X \perp\!\!\!\perp Y \mid Z$

if and only if:
∀x,y,z P(x | y,z) = P(x|z)

or, equivalently, if and only if
∀x,y,z P(x,y | z) = P(x|z)P(y|z)

Bayesian networks
Syntax:
- nodes: variables (with domain)
- arcs: interactions (directed)
- no cycle
- no interactions between vars.: absolute indep.
- Bayes net = Topology

Number of free parameters in each CPT:
- Parent domain sizes $d_1,\dots,d_k$
- Child domain size d
- Each table row must sum to 1

$(d-1)\ \Pi_i\ d_i$

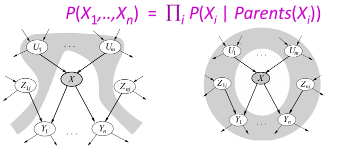
Conditional distributions for each node given its *parent variables* in the graph

- ***CPT***: conditional probability table: each row is a distribution for child given a configuration of its parents

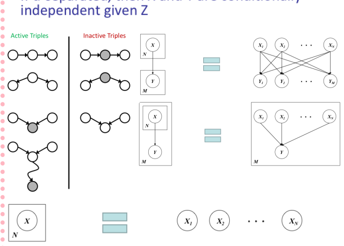
General formula for sparse BNs
- suppose;
  - n vars
  - max domain size: d
  - max # of parents: k
- Full joint distribution has size $O(d^n)$
- Bayes net has size $O(n \cdot d^{k+1})$
  - Linear scaling with *n* as long as causal structure is local

Bayesian networks global semantics
- Bayes nets encode joint distributions as product of conditional distributions on each variable:

$P(X_1,..,X_n) = \prod_i P(X_i \mid Parents(X_i))$



- 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



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

Markov Networks
- Encode a joint distribution with an undirected graph
- MN = undirected graph + potential funcs.

Generative models: repre. a joint distri. (both BN and MN)
Discriminative models: repre. condo. distri. (doesn't model P(X))
- 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)

Structured prediction framework:

$$\hat{\mathbf{Y}} = \arg\max_{\mathbf{Y}} F(\mathbf{X},\mathbf{Y},\mathbf{W})$$

$$F(\mathbf{X},\mathbf{Y},\mathbf{W}) = \sum_i \phi(x_i, y_i; \mathbf{w}_u)$$ Unary potential
$$+\sum_{i,j}\psi(\mathbf{x}_{ij}, y_i, y_j; \mathbf{w}_p)$$ Pairwise potential
$$+\sum_c \psi_c(\mathbf{x}_c, \mathbf{y}_c; \mathbf{w}_c)$$ Higher-order potential

- Input $\mathbf{X} = \{x_i\}_{i=1}^n, \quad x_i \in \mathbf{R}^d$
- Output $\mathbf{Y} = \{y_i\}_{i=1}^n, \quad y_i \in \mathbf{L}, \mathbf{L} = \{1,\cdots,K\}$
- Structured prediction model

- A probabilistic view – (conditional) random field
  - Conditional probability

$$P(\mathbf{Y}|\mathbf{X};\mathbf{W}) = \frac{1}{Z_{\mathbf{X},\mathbf{W}}}\exp\{F(\mathbf{X},\mathbf{Y},\mathbf{W})\}$$

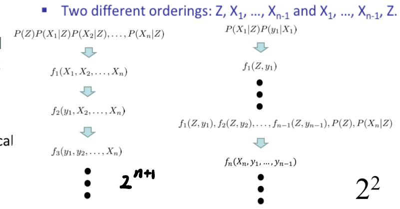
  - Label prediction: MAP estimation

- Joint distribution: P(X,Y)
  - Entries P(x,y) for all x, y
  - Sums to 1
- Selected joint: P(x,Y)
  - A slice of the joint distribution
  - Entries P(x,y) for fixed x, all y
  - Sums to P(x)
- Single conditional: P(Y | x)
  - Entries P(y | x) for fixed x, all y
  - Sums to 1
- Family of conditionals: P(X |Y)
  - Multiple conditionals
  - Entries P(x | y) for all x, y
  - Sums to |Y|
- Specified family: P(y | X)
  - Entries P(y | x) for fixed y, but for all x
  - Sums to ... who knows!

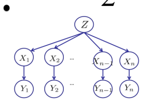
Inference by Enumeration in BN = Multiple Join + Multiple Eliminate

Variable Elimination
- Inference by Enumeration so slow?
  - join up the whole joint dis. before summing out the hidden vars.
- V.E. is NP-hard, but much faster then IbE.

Variable Elimination Ordering
- Query: $P(X_n|y_1,\dots,y_n)$
- Two different orderings: $Z, X_1, \dots, X_{n-1}$ and $X_1, \dots, X_{n-1}, Z$.



- The size of the largest factor determines the time and space complexity of VE
- [F] Does there always exist an ordering that only results in small factors?



Approximate Inference
Prior Sampling:
- For i=1, 2, ..., n (in topological order)
  - Sample X_i from $P(X_i \mid parents(X_i))$
- Return $(x_1, x_2, \dots, x_n)$

Search:

General Tree Search:
- Fringe (frontier)
- Expansion
- Expansion strategy
- b: branching factor
- m: max depth
- # of nodes: 1+b+...+b^m=O(b^m)

DFS:
- If m is finite, time cost: O(b^m)
- space fringe takes: only has siblings on path to root, O(bm)
- complete: m could be inf, only if prevent cycles
- optimal: just find "leftmost" sol

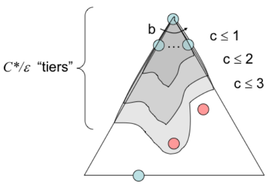
BFS:
- s: depth of shallowest sol
- search time: O(b^s)
- space fringe takes: the last tier, O(b^s)
- complete: s is finite is sol exist, so y
- optimal: only if costs are all 1

Iterative Deeping:

Idea: get DFS's space advantage with BFS's time / shallow-sol advantages (generally most work happens in the lowest level searched)
- Run a DFS with depth limit 1. If no solution…
- Run a DFS with depth limit 2. If no solution…

Cost-Sensitive Search:

Uniform Cost Search:
- Strategy: expand a cheapest node first
- Fringe: priority queue
- space fringe take: last tier, O(b^{C*/epsilon})
- complete: assume best sol has finite cost and min arc cost is positive, y
- optimal: y
- # of node expanded:
- If a sol costs C* and arcs cost at least epsilon, then effective depth: O(b^{C*/epsilon})

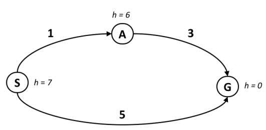


- bad:
  - explores options in every direction
  - no infor. about goal location

Informed Search:

Greedy Search:
- Strategy: expand a node you think is closet to goal state
- worst-case: like badly-guided DFS

A* Search:
- Uniform-cost orders by path cost, or backward cost g(n)
- Greedy orders by goal proximity, or forward cost h(n)
- A* Search orders by the sum: f(n)=g(n)+h(n)
- 但是是否optimal跟对heuristic的估计有关, e.g. not optimal



Admissible Heuristics:
- A heuristic **h** is **admissible** (optimistic) if: 0 <= h(n) <= h*(n)
- h*(n) is the true cost to a nearest goal

Optimality of A* Tree Search:
- Assume:
  - A is an opt. goal node
  - B is a subopt. goal node
  - h is admissible
- Claim:
  - A will exit the fringe before B
- Pf:
  - Imaging B is on the fringe
  - Some ancestor n of A is on the fringe, too
  - Claim: n will be expanded before B (As B is subopt., g(A)<g(B))
    - f(n) < or <= f(A)
    - f(A) < f(B)
  - f(n) <= f(A) < f(B)
  - All ancestors of A expanded before B
  - A expands before B
  - A* search is optimal

UCS vs A* Contours 概要:
- UCS expands equally in all directions
- A* expands mainly toward the goal, but hedge its bets to ensure optimality

Creating Heuristics:
- often, admi. heuristic are sol. to relaxed probs., where new actions are avaliable

Graph Search（对应的是state search）:
- Idea: never expand a state twice
- Implement: tree search + set of expanded states ("closed set")

Consistency of Heuristic:
- Admissibility: heuristic cost <= actual cost to goal
- Consistency: heuristic "arc" cost <= actual cost for each arc
- Con. implies admi.
- A* graph is opt. if heuristic is con.

Constraint Satisfaction Problems (CSPs):
- A special subset of search probs.
- State: **variables X_i** with values from **domain D**
- Goal test is a **set of constraints** specifying allowable combinations of values for subsets of variables
- CSPs are specialized for identification probs.
- **Discrete vars.:**
  - **Finite domains:**
    - Size d means O(d^n) complete assignments
  - **Inf. domains:**
    - Linear constraints solvable, nonlinear undecidable
- **Continuous vars.:**
  - Linear constraints solvable in polynomial time by LP methods
- **Unwary constraints:** involve a single var., e.g.: SA != green
- **Binary constraints:** involve pairs of variables, e.g. : SA != WA
- **Higher-order constraints:** involve 3 or more vars.

Standard Search Formulation:
- States defined by the values assigned so far (partial assignments)

Backtracking Search = DFS + variable-ordering + fail-on-violation

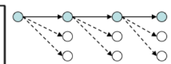
Filtering: Forward Checking

Consistency of A Single Arc
- x->y (delete from the tail)

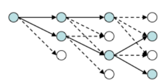
```
function AC-3( csp) returns the CSP, possibly with reduced domains
   inputs: csp, a binary CSP with variables {X1, X2, ..., Xn}
   local variables: queue, a queue of arcs, initially all the arcs in csp

   while queue is not empty do
      (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
```



Greedy Search



Beam Search

- Runtime: $O(n^2d^3)$, can be reduced to $O(n^2d^2)$

K-Consistency
- 1-Consistency (Node Consistency): Each single node's domain has a value which meets that node's unary constraints
- 2-Consistency (Arc Consistency): For each pair of nodes, any consistent assignment to one can be extended to the other
- K-Consistency: For each k nodes, any consistent assignment to k-1 can be extended to the kth node.

Strong K-Consistency (without backtracking)

Ordering:

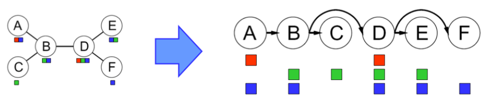
Var. Ordering: Min remaining values (MRV, also "most constrained var.")

Var. Ordering: Least Constraining Value

Structure:

Prob. Structure:
- Extreme case: indep. subprob.
- Indep. subprobs. are identifiable as connected components of constraint graph
- Suppose a graph of n variables can be broken into subproblems of only c variables:
  - Worst-case solution cost is $O((n/c)(d^c))$, linear in n

Tree-Structured CSPs:
- Thm: if the constraint graph has no loops, the CSP can be solved in. O(n*d^2) time
- Algorithm for tree-structured CSPs:
  - 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)
- Runtime: $O(n\ d^2)$

- 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 $O((d^c)(n\text{-}c)\ d^2)$, very fast for small c
- Finding the smallest cutset is NP-hard'

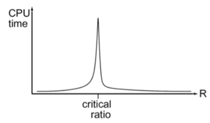
Iterative Algos:

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

Performance:
- 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):
- State: a complete assignment
- Successor func: local changes
- Generally much faster and more memo. efficient (incomplete & subopt.)

Hill Climbing:

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

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

Adversarial:

Minimax Efficiency:
- Time: $O(b^m)$
- Space: O(bm)

Depth-limited search:
- Evaluation Funcs: (ideal) return the actual minimax value of the position, weighted linear sum of features

$$Eval(s) = w_1f_1(s) + w_2f_2(s) + \ldots + w_nf_n(s)$$

Alpha-Beta Pruning:

α: MAX's best option on path to root
β: MIN's best option on path to root

```
def max-value(state, α, β):
    initialize v = -∞
    for each successor of state:
        v = max(v, value(successor, α, β))
        if v ≥ β return v
        α = max(α, v)
    return v
```

```
def min-value(state , α, β):
    initialize v = +∞
    for each successor of state:
        v = min(v, value(successor, α, β))
        if v ≤ α return v
        β = min(β, v)
    return v
```

- Good child ordering improves effectiveness of pruning
- With "perfect ordering":
  - Time complexity drops to $O(b^{m/2})$

Expectimax Search (compute the avg score under opt. play):

```
def value(state):
    if the state is a terminal state: return the state's utility
    if the next agent is MAX: return max-value(state)
    if the next agent is EXP: return exp-value(state)
```

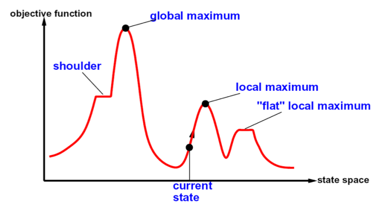
```
def max-value(state):
    initialize v = -∞
    for each successor of state:
        v = max(v, value(successor))
    return v
```

```
def exp-value(state):
    initialize v = 0
    for each successor of state:
        p = probability(successor)
        v += p * value(successor)
    return v
```

Rejection Sampling:
- Input: evidence $e_1,..,e_k$
- For i=1, 2, …, n
  - Sample X_i from $P(X_i \mid parents(X_i))$
  - If x_i not consistent with evidence
    - Reject: Return, and no sample is generated in this cycle
- Return $(x_1, x_2, …, x_n)$

Likelihood Weighting:
- Input: evidence $e_1,..,e_k$
- w = 1.0
- for i=1, 2, …, n
  - if X_i is an evidence variable
    - x_i = observed value$_i$ for X_i
    - Set w = w * P(x_i | Parents(X_i))
  - else
    - Sample x_i from P(X_i | Parents(X_i))
- return $(x_1, x_2, …, x_n)$, w

Gibbs Sampling (consistent):
- Generate each sample by making a random change to the preceding sample
  - Evidence variables remain fixed. For each of the non-evidence variable, sample its value conditioned on all the other variables



Markov Chain Monte Carlo (MCMC)
- MCMC is a family of randomized algorithms for approximating some quantity of interest over a very large state space
  - Markov chain = a sequence of randomly chosen states ("random walk"), where each state is chosen conditioned on the previous state
  - Monte Carlo = an algorithm (usually based on sampling) that is likely to find a correct answer
- MCMC = sampling by constructing a Markov chain
- Gibbs, Metropolis-Hastings, Hamiltonian, Slice, etc.

Metropolis-Hastings
- Repeat
  1. Draw a sample from a proposal distribution $g(x'|x)$
     - $g(x'|x)$ is typically easy to sample from
  2. Accept this sample with probability
     $$\min\left(1, \frac{P(x')g(x|x')}{P(x)g(x'|x)}\right)$$
- Gibbs is a special case of Metropolis-Hastings in which the acceptance rate is always 1