-Local beam search with k = 1 is hill-climbing search.

-Local beam search with k = ∞. The idea is that if every successor is retained (because k is unbounded), then the search resembles breadth-first search in that it adds one complete layer of nodes before adding the next layer.

-Starting from one state, the algorithm would be essentially identical to breadth-first search except that each layer is generated all at once.

-Simulated annealing with T = 0 at all times: ignoring the fact that the termination step would be triggered immediately, the search would be identical to first-choice hill climbing because every downward successor would be rejected with probability 1.

-Genetic algorithm with population size N = 1: if the population size is 1, then the two selected parents will be the same individual; crossover yields an exact copy of the individual; then there is a small chance of mutation. Thus, the algorithm executes a random walk in the space of individuals.

-Percept: the agent's perceptual inputs

-Percept sequence: the complete history of everything the agent has perceived

-Agent function: maps any given percept sequence to action

[f:p\* -> a]

-Agent program: runs on physical architecture to produce f (architecture + program)

-Rational Agent: Agent = something that acts. Rational: doing the right thing

-An agent is anything that can be viewed as perceiving its environment through sensors and acting upon that environment through actuators

-Fully observable (vs. partially observable): An agent's sensors give it access to the complete state of the environment at each point in time

-Deterministic (vs. stochastic): next state of environment determined by current state of agent action. If deterministic except for the actions of other agent then the environment is strategic

-Episodic (vs. sequential): Agent's experience is divided into atomic "episodes". Choice of action in each episode depends only on the episode itself

-Static (vs. dynamic): the environment is unchanged while an agent is deliberating. Semi dynamic if the environment itself doesn’t change with time but the agent's performance score does

-Discrete (vs. continuous): A limited number of distinct, clearly defined percepts and actions

-Single agent (vs. multi-agent): An agent operating by itself in an environment. Competitive vs. cooperative

-Goal formulation: based on current situation and agent’s performance measure

-Problem formulation: deciding what actions and states to consider, given a goal

-Problem can be defined into 5 components: Initial State, Actions, Transition model (state space), Goal Test, Path Cost

Search strategies evaluation:

-Completeness: Does it always find a solution if one exists?

-Optimality: Does it always find a least-cost solution?

-Time complexity: number of nodes generated

-Space complexity: maximum number of nodes in memory

b: max branching factor of the search tree

d: depth of the least cost solution

m: maximum depth of the state space

Search cost (time), total cost (time + space)

General uninformed search strategies:

-Breadth first search; fifo queue frontier

-Uniform cost search;priority queue frontier ordered by path cost g(n)

-Depth first search; lifo queue / recursive frontier

-Depth limited search

-Iterative deepening search

**Analysis** DFS Depth

Uniform Tree Limited

BFS| Cost Search | Search |DFS Graph Search |IDS| Search

Complete | Yes | Yes, cost >= E | No | No(infinite) Yes(finite) | Yes | no |

Time | b ^ (d+1) | b ceiling (C\*/E) | b ^ m | b ^ m | b ^ d | b ^ l |

Space | b ^ (d+1) | b ceiling (C\*/E) | bm | not linear | b d | bl |

Optimal | Yes | yes, increase g(n) | No | No | Yes | not |

BFS.space. keep every node in memory

BFS.optimal.if step costs are identical and non-decreasing function

Uniform cost search. C\* = cost of optimal solution

Backtracking search is a variant of DFS.

Each partially expanded nodes remember successors

Memory requirement is O(m) vs O(bm)

Depth Limited Search is DFS with limit l

IDS (Interative deeping search) = DFS + BFS

IDS.space. b d. tree search version

IDS.optimal. if step cost are identical and non-decreasing function

**Informed Search Analysis**

-Best First search. Expand most desirable unexpanded node. Frontier data structure in a decreasing order of desirability

-h(n) = estimated cheapest path from node n to goal node. if 0 then goal. fn = hn

Greedy best first search = hsld = straight line distance

Greedy Best First A\*

Complete | no, get stuck | yes |

Time | b ^ m | exponential |

Space | b ^ m | all in ram |

Optimal | no | yes |

-A heuristic is consistent if for every node n, every successor n' of n generated by any action a, h(n) ≤ c(n,a,n') + h(n')

-f(n) is non-decreasing along any path

Variant of Hill Climbing

-stochastic hill climbing. random. uphill moves. converges slowly. better solution

-first choice hill climbing. Successor random until better than current.

-random restart hill climbing. Complete with probability approaching 1. series of hill climbing until goal found

very effective for n queen problem

Genetic Algorithm. start with population. evaluation function(fitness function). Higher values for better states produce next generation by selection, crossover, and mutation

**Validity**

-A sentence is valid if it is true in all models, also known as tautology e.g., True, A ∨ ¬A, A ⇒ A, (A ∧ (A ⇒ B)) ⇒ B

-Validity is connected to inference via the Deduction Theorem

KB ╞ α if and only if (KB ⇒ α) is valid

-A sentence is satisfiable if it is true in some model e.g., A ∨ B, C

-A sentence is unsatisfiable if it is true in no models e.g., A ∧ ¬A

-Satisfiability is connected to inference via the following:

KB ╞ α if and only if (KB ∧ ¬α) is unsatisfiable thus proof by contradiction

**CSP – min conflicts, backtracking, or local search**

\_ Since we can formulate CSP problems as standard search problems, we can apply search algorithms from chapter 3,4

\_ If breadth-first search were applied, branching factor? *nd* tree size? *nd* \* (*n*-1)*d* \* … \* *d* = *n*! \* *dn* leaves

\_ Complete assignments? *dn*

\_ A crucial property to all CSPs: commutativity

\_ the order of application of any given set of actions has no effect on the outcome

\_ Variable assignments are commutative, 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 node \_ *b* = *d* and there are *d n* leaves

\_ **Backtracking search** is used for a depth-first search that chooses values for one variable at a time and backtracks when a variable has no legal values left to assign

**Minimum remaining values** (MRV)

\_ choose the variable with the fewest “legal” values

\_ also called most constrained variable or fail-first heuristic

\_ does not help in choosing first variable

**Most constraining variable**:

\_ selecting the variable that has the largest number of constraints on other unassigned variables

\_ also called degree heuristics

\_ Tie-breaker among MRV

Given a variable, choose the **least constraining value**:

\_ the one that rules out the fewest values in the remaining variables

\_ Combining these heuristics makes 1000 queens feasible

**Forward Checking**

-Keep track of remaining legal values for unassigned variables

-Terminate search when any variable has no legal values

-Forward checking propagates information from assigned to unassigned variables, but doesn't provide early detection for all failures

**Constraint propagation** repeatedly enforces constraints locally by propagating implications of a constraint of one variable onto other variables

**AC-3 Time complexity** *O* (*n* 2*d* 3) - Simplest form of propagation makes each arc consistent X *Y* is consistent iff for every value *x* of *X* there is some allowed *y*

\_ If *X* loses a value, neighbors of *X* need to be rechecked

\_ Arc consistency detects failure earlier than forward checking

\_ Can be run as a preprocessor or after each assignment

Hill-climbing, simulated annealing typically work

with "complete" states, i.e., all variables assigned

\_ To apply to CSPs:\_ allow states with unsatisfied constraints

\_ Operators reassign variable values

\_ Variable selection: randomly select any conflicted variable

\_ Value selection by min-conflicts heuristic:\_ choose value that violates the fewest constraints

\_ i.e., hill-climb with *h*(*n*) = total number of violated constraints

**Adversarial Search**

Optimal play for MAX assumes that MIN also plays optimally. For every game tree the utility obtained by max using mini-max decisions against a suboptimal will not be lower than the utility obtained playing against an optimal min.

\_ A complete depth-first search? Yes

\_ Time complexity? O(bm)

\_ Space complexity? O(bm) (depth-first exploration)

\_ For chess, b ≈ 35, m ≈ 100 for "reasonable" games exact solution completely infeasible

**Alpha Beta**

The number of game states with mini-max search is exponential in the # of moves thus need to prune away branches that cannot possibly influence the final decision

Consider a node n such that Player has a choice of moving to

\_ If Player has a better choice m either at the parent of n or at any choice point further up, then n will never be reached in actual play

\_ α-β pruning gets its name from the two parameters that describe bounds on the backed-up values

α = the value of the best (highest-value) choice we have found so far at any choice point along the path for MAX

\_ β = the value of the best (lowest-value) choice we have found so far at any choice point along the path for MIN

\_ α-β search updates the values of α and β as it goes along and prunes the remaining branches at a node as soon as the value of the current node is worse than the current α or β for MAX or MIN respectively

\_ Pruning does not affect final result

\_ The effectiveness of alpha-beta pruning is highly dependent on the order of successors

\_ It might be worthwhile to try to examine first the successors that are likely to be best

\_ With "**perfect ordering**," time complexity = O(bm/2)

-> effective branching factor becomes square root of b

\_ For chess, 6 instead of 35

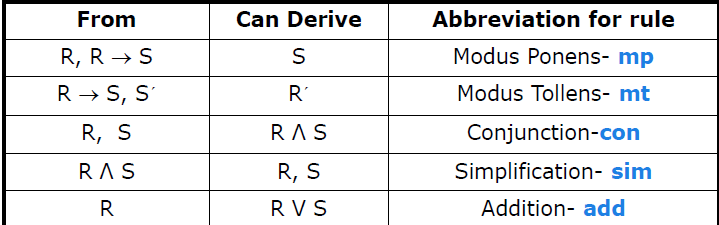
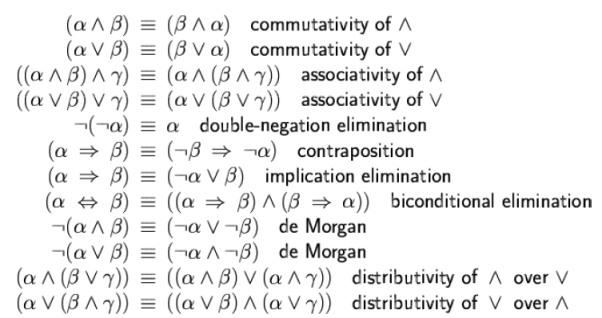
\_ it can look ahead roughly twice as far as mini-max in the same amount of time

\_ Ordering in chess: captures, threats, forward moves, and then backward moves

**-If successors are examined in random order rather than best-first**, the complexity will be roughly O(b3m/4)

-Adding dynamic move-ordering schemes, such as trying first the moves that were found to be best last time, brings us close to the theoretical limit

-The best moves are often called killer moves (killer move heuristic)

****

**Conjunctive Normal Form** (CNF): conjunction of disjunctions of literals clauses E.g., (A ∨ ¬B) ∧ (B ∨ ¬C ∨ ¬D)

**Horn Form** KB = conjunction of Horn clauses

Horn clause = disjunction of literals of which at most one is positive E.g., (¬L1, 1 V ¬Breeze V B1, 1)

-Every horn clause can be written as an implication whose premise is a conjunction of positive literals and whose conclusion is a single positive literal E.g., (L1,1 ^ Breeze) ⇒ B1,1

**Definite clause**: horn clauses with exactly one positive literal. The positive literal is called the head and the negative literals form the body. Definite clauses form the basis for logic programming. Can be used for FC and BC. Linear time

**Forward vs. Backward Chaining**

\_ 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**

\_ Some students took French in spring 2001.

\_ Every student who takes French passes it.

\_ Only one student took Greek in spring 2001.

\_ The best score in Greek is always higher than the best score in French.

-∃x Student(x) ^ Takes (x, F, S2001)

-∀, s Student(x) ^ Takes(x, F, S) -> Passes(x, F, S)

-∃x Student(x) ^ Takes(x, G, S2001) ^ ∀y Student(y) ^! (y=x) ->! Takes(y, G, S2001)

-∀s ∃x ∀y Score(x, G, S2001) > Score(y, F, S2001)

**Unification**

p q θ

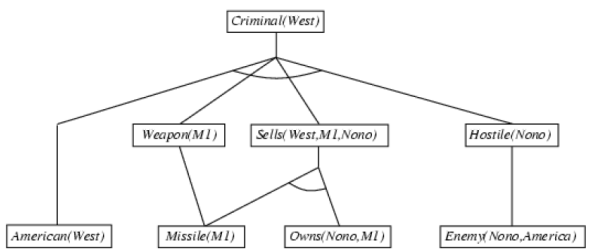
Knows(John,x) Knows(John,Jane) {x/Jane}

Knows(John,x) Knows(y,OJ) {x/OJ,y/John}

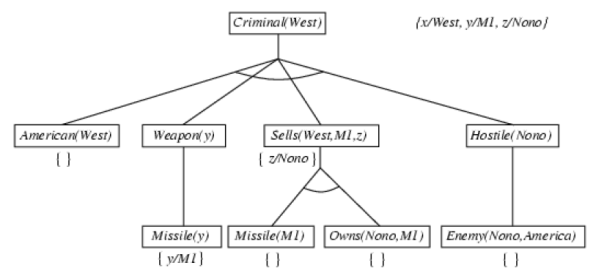
Knows(John,x) Knows(y,Mother(y)) {y/John,x/Mother(John)}

Knows(John,x) Knows(x,OJ) {fail}

Forward Chaining



Backward Chaining



**Planning**: the task of coming up with a sequence of actions that will achieve a goal

\_ Search-based problem-solving agent

\_ Logical planning agent

\_ Complex/large scale problems?

**Planning vs. Problem Solving**

Planning agent is very similar to problem solving agent

\_ Constructs plans to achieve goals, and then executes them

\_ Planning agent is different from problem solving agent in:

\_ Representation of goals, states, actions

\_ Use of explicit, logical representations

\_ Way it searches for solutions

Planning systems do the following:

\_ divide-and-conquer

\_ relax requirement for sequential construction of solutions

**Shakey World**

Init:

INROOM (S, R1); STATUS (D1, OPEN); INROOM (B1, R2)

CONNECTS (D1, R1, R2); CONNECTS (D1, R2, R1)

Plan:

GOTODOOR (S, D1); GOTHRUDOOR (S, D1)

Goal:

INROOM (S, R2); INROOM (B1, R2)

**Total-Order Planning**

\_ Forward/backward state-space searches are forms of totally ordered plan search

\_ explore only strictly linear sequences of actions directly connected to the start or goal

cannot take advantages of problem decomposition

**Partial-Order Planning**\_ Idea:

\_ works on several subgoals independently

\_ solves them with subplans

\_ combines the subplans

flexibility in ordering the subplans

\_ \_ least commitment strategy:

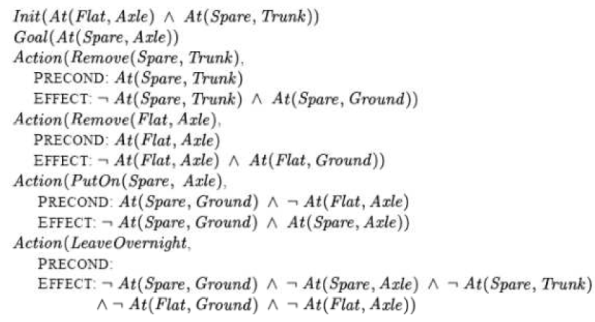
\_ delaying a choice during search

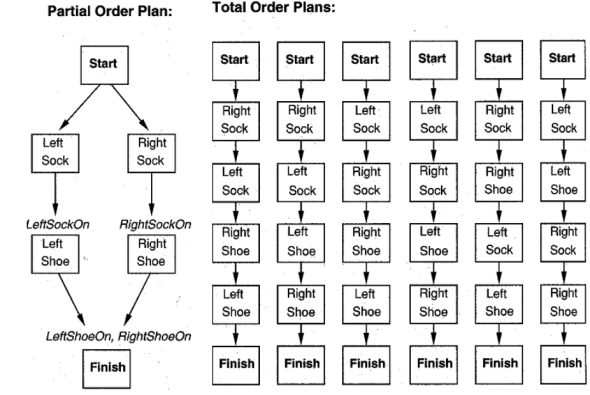
\_ Example, leave actions unordered, unless they must be sequential

The causal links lead to early pruning of portions of the search space because of irresolvable conflicts

\_ The solution is a partial-order plan

\_ linearizations produce flexible plans





**Progression**

\_ A plan is a sequence of STRIPS operators

\_ From initial state, search forward by selecting operators whose preconditions can be unified with literals in the state

\_ New state includes positive literals of effect; the negated literals of effect are deleted

\_ Search forward until goal unifies with resulting state

\_ This is state-space search using STRIPS operators

**Regression**

\_ A plan is a sequence of STRIPS operators

\_ The goal state must unify with at least one of the positive literals in the operator’s effect

\_ Its preconditions must hold in the previous situation, and these become sub goals which might be satisfied by the initial conditions

\_ Perform backward chaining from goal

\_ Again, this is just state-space search using STRIPS operators

**Sensor less planning**: constructs sequential plans to be executed without perception

**Conditional planning**: (contingency planning): constructs a conditional plan with different branches for different contingencies that could happen

**Execution monitoring and re-planning**: uses preceding techniques to construct a plan, but monitors the execution process and re-plan when necessary

**Continuous planning**: a planner designed to persist over a lifetime

**AND-OR Graph**

\_ For conditional planning, we modify the mini max algorithm

\_ MAX and MIN nodes become OR and AND nodes

\_ the plan needs to take some action at every state, but must handle every outcome for the action it takes

\_ The algorithm needs to return a conditional plan rather than just a single move

\_ at an OR node, the plan is just the action selected

\_ at an AND node, the plan is a nested series of if-then else steps specifying sub plans for each outcome

The algorithm terminates in every finite state space, because every path must reach a goal, a dead end, or a repeated state. --- noncyclic solution

To create **conditional plans**, we need conditional steps

if < *test* > then *plan* \_ *A* else *plan* \_ *B*

if *AtL* ∧*CleanL* then *Right* else *Suck*

\_ By nesting conditional steps, plans become trees

\_ In general, conditional plan should work *regardless of which action outcomes actually occur* -- games against nature

Consider the following vacuum cleaner:

The initial state has the robot in the right square of a clean world. The environment is fully observable. The goal state has the robot in the left square of a clean world.

It sometimes deposits dirt when it moves to a clean destination square and sometimes deposits dirt if Suck is applied to a clean square.

**Uncertainty**

**Inference using full join distribution** (table with probabilities)

Toothache !toothache

Catch !catch catch !catch

Cavity 0.108 .012 .072 .008

!cavity .016 .064 .144 .576

P (toothache) = .108 + .016 + .012 + .064

P (Cavity) = <true, false> = <cavity, !cavity> = <.2, .8>

P (toothache | cavity) = <P (toothache | cavity), P (!toothache, cavity)>

=<P (toothache^cavity)/P(cavity) , P(!toothache^cavity)/P(cavity)>

=(.108+.012)/.02 , (.072+.008)/.02

=<.6, .4>

P(cavity|toothache v catch)

=<P(cavity^toothache^catch)/P(toothache v catch), ….>

=(.108+.012+.072)/.416

**Independence**: a and b are independent

P(a|b) = P(a) or P(b|a) = P(b)

P(a^b) = P(a) \* P(b)

P(a,b,c,d(independent)) = P(a,b,c)P(d(independent))

P(d(independent)|a,b,c)\*P(a,b,c) = P(d(independent))

**Condition Probability**

P(a|b) = P(a^b) / P(b)

**Product Rule**

P(a^b) = P(a|b)\*P(b)

P(b^a) = P(b|a)\*P(a)

**Random** variables are upper case

Include domain, <a, b> where a + b = 1

**Value** variables are lower case

**Bayes’ Rule**

P(a|b)\*P(b) = P(b|a)\*P(a)

-> P(a|b) = ( P(b|a)\*P(a) ) / P(b) or P(b|a) = ( P(a|b)\*P(b) ) / P(a)

**Marginalization Rule**

P(t) = P(t|d)\*P(d)+P(t|!d)\*P(!d)

P(t|!d) = 1 - P(!t|!d)

P(!d) = 1 - P(d)

**Resolution**

To show that KB╞ α, we show that (KB ^ ￢α) is unsatisfiable, proof by contradiction

\_ First, (KB ^ ￢α) is converted into CNF

\_ Then the resolution rule is applied to the resulting clauses

Each pair that contains complementary literals is resolved to produce a new clause, which is added to the set if it’s not present

\_ Process continues until:

\_ no new clauses can be added, thus KB does not entail α

\_ two clauses resolve to yield {}, thus KB entails α

\_ {} is a disjunction of no disjuncts is equivalent to False, thus the contradiction

