Introduction

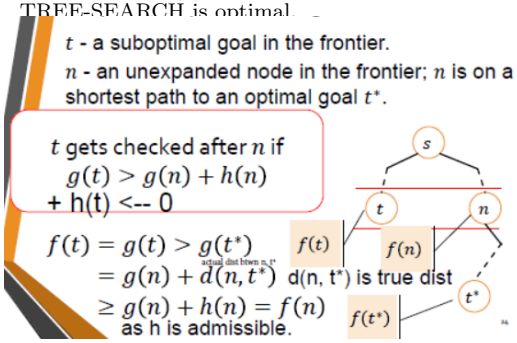
- **Performance measure**: An **external**, objective measure of how good the AI is.
- **Utility Function**: An **internal** metric to represent how well an agent is doing.
- **Fully/partially observable**: Sensors provide access to the complete state of environment at each point in time.
- **Deterministic/stochastic**: The next state of the environment is completely determined by current state and action executed by the agent.
- **Episodic/sequential**: The choice of current action does not depend on past episodes, and does not affect future decisions.
- **Static/dynamic**: The environment is unchanged while agent is thinking.
- **Discrete/continuous**: There is a finite number of distinct states, percepts and actions.
- **Single/multi agent**: An agent operating by itself in an environment.
- **Simple reflex agent**: Passive (acts only when observes a percept)
  state update only based on percept.
- **Model-based Reflex agent**: Passive.
  +State update based on percept, current state, most recent action and model of environment.
- **Goal-based agent**: +Active (acts to achieve goals)
  State update based on percept, state, action, model.
- **Utility-based agent**: +Active, has utility function and aims to maximise it.
  State update based on percept, state, action, model.

Uninformed Search

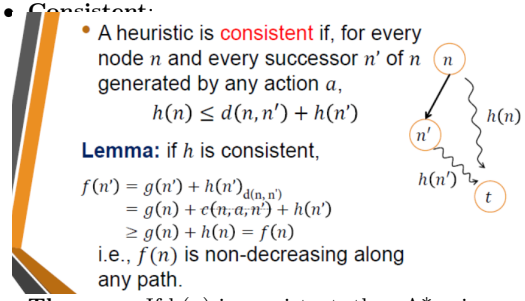
- **State**: Represents a physical configuration of the world.
- **Nodes**: Part of search tree/graph. Includes state, parent action and cost.
- **Tree Search**: Start at source node, keep searching until goal. Add neighbours to frontier at each iteration. **Can repeat states**.
- **Frontier**: Nodes we've seen but not yet explored.
- **Graph Search**: Similar to tree search, but do not revisit nodes. CP: Use memoization tables.
- **Search evaluation criteria**:
  1. Completeness: always find a solution if one exists
  2. Optimality: Able to find least-cost solution
  3. Time complexity: Num of nodes generated
  4. Space complexity: Num of nodes in memory at any time
- **Parameters**:
  b: max # of successor nodes of any node.
  d: depth of shallowest **goal** node m: max depth of search tree
- **Uninformed Search**: Only uses problem input to search.
- **Uniform-cost Search**: Dijkstra's algorithm.
  Each round, we can get ϵ distance closer to the goal, so num steps $= \lfloor \frac{C^*}{\epsilon} \rfloor$ where C^* is optimal cost. At step k, there are at most b^k nodes in PQ.
  1. BFS, IDS complete if b is finite
  2. UCS complete if b is finite and step cost $\geq \epsilon$.
  3. BFS, IDS optimal if all step costs equal
  4. IDS optimal only if depth increases by 1 each time.

Informed Search

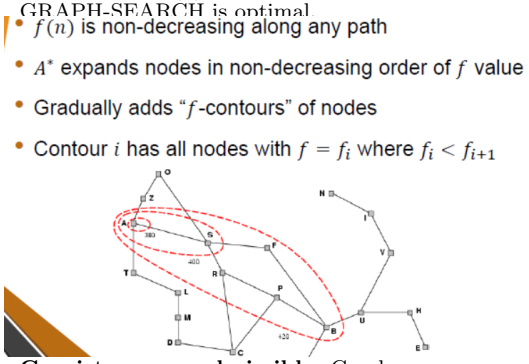
- **Greedy best-first search**: Explore nodes that appear closest to goal nodes.
  Complete, not optimal, $O(b^m)$ time and space worst case complexity although heuristic can reduce complexity. Reduces to BFS in worst case. **Ignores cost to get to node, hence might take expensive routes that look close**. f(n) is estimated cost of path **from** n to goal.
- **A* Search**: Evaluation function
  $f(n) = g(n) + h(n)$ where $g(n)$ is cost start $\rightarrow$ n, $h(n)$ is estimate n $\rightarrow$ goal.
  So $f(n)$ is estimated cost of cheapest path **through** n to goal. Time: $O(b^{h'(S_0) - h(S_0)})$ where h' actual cost of root $\rightarrow$ goal.
  Space: Max size $O(b^m)$.
- **Theorem**: If h(n) is admissible, then A* using TREE-SEARCH is optimal.



- **Consistent**:



- **Theorem**: If h(n) is consistent, then A* using GRAPH-SEARCH is optimal.



- **Consistency vs admissible**: Can be proven consistent $\rightarrow$ admissible. Other way not true. Admissible cannot guarantee optimality on A* GRAPH-SEARCH because GRAPH-SEARCH discards new paths to repeated state. Consistent: always follow optimal path.
- **Dominance**: If $h_2(n) \geq h_1(n)$ for all n (both admissible), then h_2 dominates h_1. It follows that h_2 incurs lower search cost.
  In other words, the closer heuristic is to actual answer, the lower search cost involved. Unless heuristic over-estimates, then it's bad.
- **Local Search**: Find best move at each step, forget about the rest. Saves memory, but may run into local optima.

Adversarial Search

- **Problem Formulation**: A game is defined by 7 components: initial state s_0, states, players, actions, transitions/result, terminal test, utility function.
- **Winning/Non-losing Strategy**: A strategy is called a winning/NL strategy (for player 1) if for any strategy by player 2, player 1 wins / player 2 does not win.
- **Minimax algorithm**: MAX player selects move that maximises utility at each choice, MIN player selects move that minimises utility (the more negative, the better) at each choice.
  Complete: Yes, if the game tree is finite.
  Returns a sub-perfect Nash Equilibrium: best action at every choice node. Optimal: Yes
  Time: $O(b^m)$
  Space: $O(bm)$
  Minimax is comparable to running a DFS.
- **Alpha-Beta pruning**: Prune subtrees that never affect decision.
- **Heuristic evaluation function**: Alpha-Beta pruning needs the entire search tree. Infeasible. Use cutoff test to heuristically evaluate. Should be strongly correlated with actual chances of winning.

Prove that the minimax algorithm shown in class outputs a subgame-perfect Nash equilibrium.

Solution: The proof is by induction on the height of the game tree, h. If $h = 1$ then the source node is a terminal node and there is nothing to prove. Suppose that for all trees at height $h-1$ the minimax algorithm computes a subgame-perfect Nash equilibrium, and consider a node v at height h. We need to show that the minimax algorithm strategy will be a Nash equilibrium for the subtree induced by the node v. A bit of notation: given a node u, let $T(u)$ be the subtree rooted at u, and let $C(u)$ be the children of u. Assume that v is a MAX node (if it's a MIN node the proof is similar). The minimax algorithm has, by assumption, selected strategies in the subtrees rooted in the children of v such that no player can improve their welfare by switching strategies (they are Nash equilibria). Let s_1^* be the strategy outputted by the minimax algorithm for the MAX player, and let s_2^* be the strategy outputted by the minimax algorithm for the MIN player. Let s_1 be some other strategy by player 1. We need to show that $u_1(v, s_1^*, s_2^*) \geq u_1(v, s_1, s_2^*)$, where $u_1(v, \cdot, \cdot)$ is the utility that player 1 receives from the game subtree rooted at the node v. Suppose that at node v the minimax algorithm chooses the action $c^* \in C(v)$. Then we know that for any other child $c \in C(v)$, $u_1(c^*, s_1^*, s_2^*) \geq u_1(c, s_1^*, s_2^*)$. Thus, by induction hypothesis, we have that for all $c \in C(v)$:

$$u_1(v, s_1^*, s_2^*) = u_1(c^*, s_1^*, s_2^*) \geq u_1(c, s_1^*, s_2^*) \geq u_1(c, s_1, s_2^*)$$

The last inequality holds by the induction hypothesis - the strategy pair s_1^*, s_2^* induces a Nash equilibrium on the subtree rooted at c. Our claim immediately follows: some child node c_0 is the one chosen by s_1, and the payoff from playing s_1 is thus $u_1(v, s_1, s_2^*) = u_1(c_0, s_1, s_2^*) \leq u_1(c_0, s_1^*, s_2^*) \leq u_1(c^*, s_1^*, s_2^*) = u_1(v, s_1^*, s_2^*)$.

Prove that α-β pruning does not remove any strategies that are played in a Nash equilibrium of an extensive form game; can α-β pruning be used to find a subgame-perfect Nash equilibrium? Prove or provide a counterexample.

Solution: Suppose α-β pruning removes a subtree, say a subtree rooted in a node u_1 that's the child of a node v. Suppose that v is a MAX node; then there is some other node u_2 who's a child of v and the payoff to the MAX player from choosing u_2 is at least as high as that of choosing u_1 (that's the only reason we'll prune u_1 in the first place). In other words, let x be the payoff that player 1 gets from choosing the node u_1 from v; then it must be the case that x is no more than the payoff to player 1 from playing u_2.

Therefore, pruning u_1 will never prune a strategy that's played in a Nash equilibrium of the game overall. However, since we do not explore the entire tree, we do not specify a complete strategy, and in particular, we cannot be specifying a sub-perfect Nash equilibrium strategy.

Constraint Satisfaction Problem

- **Complete assignment**: All variables are set
- **Consistent assignment**: No violation of constraints
- **Binary CSP**: All constraints involve only 2 variables
- **Constraint Graph**: A graph where nodes are variables and links are constraints
- **Types of variables**: Can be discrete or continuous
- **Types of constraints**:
  1. Precendence: $A + k \leq B$
  2. Disjunctive: $A = jorA = k$
  1. Unary: involving single variable: $A = k$ or $A \neq k$
  2. Binary: $A \neq B$
  3. Higher order: $A + B + C \leq k$
- **Arc consistency propagation / AC3 algorithm**: After removing some values, propagate the updated set again; there might be new "impossibles" as a result. Similar to playing Sudoku. TIme complexity: $O(n^2 d^3)$. At most n^2 constraints, at most d values to delete, so each arc inserted d times. Checking consistency takes $O(d^2)$ time.
- **Drawbacks**: Easy to propagate, difficult to implement backtrack correctly.
- **CSP with tree**: If CSP constraint creates a tree, then we can compute a satisfying assignment in $O(nd^2)$ time. Idea: O(n) constraints, and tree means no cycles $\implies$ no need to propagate in cycles / only need to propagate within subtree.
- **Algorithm**: Arbitrarily root the tree. Starting from leaves, make parents arc-consistent with children, i.e. ensure choice of parents don't cause children to become impossible. (No need vice versa). Then, if successful, start assigning from root down. If multiple choices, any is OK.
- **Local Search**: Min-conflict algorithm: Do hill-climbing by selecting value which results in least conflict.

CSP

- **Modeling** m models α if α is true under m.
- **Entailment**: Entailment means that one thing follows from another. For example, α = q is prime entails β = q is odd or q = 2. If $\alpha \vDash \beta, M(\alpha) \subseteq M(\beta)$.
- **Model checking**: If KB entails α, i.e. KB is a subset of $M(\alpha)$ then we can infer/prove α is true by model checking.
- **Sound**: "Don't infer nonsense". $KB \vdash_A \alpha$ implies $KB(actually) \vDash \alpha$.
- **Complete**: "If it's implied, it will be inferred." $KB \vDash \alpha$ implies $KB \vdash_A \alpha$
- **Satisfiability**: $KB \vDash \alpha$ if and only if $(KB \wedge \neg\alpha)$ is unsatisfiable.
- **Horn Clause**: Has at most 1 positive literal, e.g. $\neg A \wedge \neg B \wedge C$. Can transform to $(A \vee B) \implies C$. Then can draw AND-OR graph for forward/backward chaining.
- **WalkSAT**: Tests for satisfiability. If return true, satisfiable. If return false, either unsatisfiable, or needs more time. Algorithm: Choose a random unsatisfied clause. With probability p flip truth value of a random symbol, otherwise flip symbol that maximises number of satisfied clauses in the formula. Repeat the above for max_k times. If all satisfied, return true. Otherwise return false.

- For every rule c, let count(c) be the number of symbols in c's premise.
- For every symbol s, let inferred(s) be initially $False$
- Let agenda be a queue of symbols (initially containing all symbols known to be true.
- While agenda $\neq \emptyset$:
  - pop a symbol p from $agenda$; if it is q we're done
  - Set $inferred(p) = True$
  - For each clause $c \in KB$ such that p is in the premise of c, decrement $count(c)$. If $count(c) = 0$, add c's conclusion to $agenda$.

- **Convert KB to CNF:**

"Everyone who loves all animals is loved by someone"

$$\forall x: (\forall y: \text{Animal}(y) \Rightarrow \text{Loves}(x, y)) \Rightarrow \exists y: \text{Loves}(y, x)$$

- **Eliminate implications: $A \Rightarrow B$ becomes $\neg A \vee B$**

$$\forall x: \neg(\forall y: \neg\text{Animal}(y) \vee \text{Loves}(x, y)) \vee \exists y: \text{Loves}(y, x)$$

- **De Morgan's rule:**
  $\neg\forall x: P \equiv \exists x: \neg P$ and $\neg\exists x: P \equiv \forall x: P$

$$\forall x: (\exists y: \neg(\neg\text{Animal}(y) \vee \text{Loves}(x, y))) \vee \exists y: \text{Loves}(y, x)$$
$$\forall x: (\exists y: \neg\neg\text{Animal}(y) \wedge \neg\text{Loves}(x, y)) \vee \exists y: \text{Loves}(y, x)$$
$$\forall x: (\exists y: \text{Animal}(y) \wedge \neg\text{Loves}(x, y)) \vee \exists y: \text{Loves}(y, x)$$

- **Standardize variables:**

$$\forall x: (\exists y: \text{Animal}(y) \wedge \neg\text{Loves}(x, y)) \vee \exists z: \text{Loves}(z, x)$$

- **Skolemize existential quantifiers:**
  replace with functions depending on external universal quantifier. Cannot just apply existential instantiation (y and z may depend on x)!

$$\forall x: (\text{Animal}(F(x)) \wedge \neg\text{Loves}(x, F(x))) \vee \text{Loves}(G(x), x)$$

$F(x)$ and $G(x)$ are called Skolem Functions.

- **Drop Universal Quantifiers:**

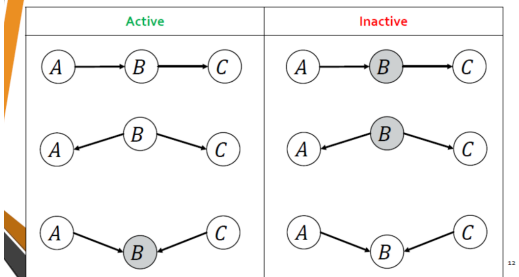
$$(\text{Animal}(F(x)) \wedge \neg\text{Loves}(x, F(x))) \vee \text{Loves}(G(x), x)$$

- **Distribute $\vee$ over $\wedge$:**

$$(\text{Animal}(F(x)) \wedge \text{Loves}(G(x), x)) \vee (\neg\text{Loves}(x, F(x)) \wedge \text{Loves}(G(x), x))$$

- **Independence given variable**: Shaded = given

  - A path is **active** iff every triple on path is active
  - **One** inactive triple ⇒ path is **inactive**!



- **Proof consistent**:
$h(n) = |h_{SLD}(sibiu) - h_{SLD}(n)|$
Need to prove that for every successor n' of n,
$h(n') \leq h(n) + d(n, n')$
By triangular inequality,
$h_{SLD}(n') \leq h_{SLD}(n) + d(n, n')$
$\implies h_{SLD}(n') - h_{SLD}(Sibiu) \leq h_{SLD}(n) - h_{SLD}(Sibiu) + d(n, n')$
$h_{SLD}(n) \leq h_{SLD}(n') + d(n, n')$
$\implies h_{SLD}(n) + h_{SLD}(Sibiu) \leq h_{SLD}(n') + h_{SLD}(Sibiu) + d(n, n')$
$\implies h_{SLD}(Sibiu) - h_{SLD}(n') \leq h_{SLD}(Sibiu) - h_{SLD}(n) + d(n, n')$
So $|h_{SLD}(Sibiu) - h_{SLD}(n')| \leq |h_{SLD}(Sibiu) - h_{SLD}(n)| + d(n, n')$
$h(n) \leq h(n') + d(n, n')$