Midterm

Lisp

- = is only for numbers
- equal is to check if the contents are the same
- eql is to check if they are actually the same object in memory
- AND - if T, then returns value of the final expression
  if F, then returns NIL
- OR - returns the first expression that is true
- (cond (bexp exp, ... exp$_n$) ...) [bexp → exp$_i$]
- let - creates local bindings for local variables [variables assigned in parallel]
- let* - same as let, but assigns variables sequentially instead

Search

- Three components - Initial State, Final State, Actions, (Heuristic)
  - Actions have a particular cost, want to satisfy some conditions/constraints
  - Actions are deterministic (1 action → 1 state) or non-deterministic
- Successor Function - takes a state, tells us possible next states based on a legal action
- Search Process
  1. Generate Children
  2. Expand - Check if you reached the goal yet
     If not at goal yet, generate children

  Fringe/Frontier - set of children (leaf nodes) at the end of the visited path
- Evaluation of Search Strategies
  1. completeness - does it always find a solution if one exists?
  2. time complexity - number of nodes generated
  3. space complexity - max number of nodes in memory to find a solution
  4. optimality - does it find the least-cost solution?
- Branching Factor (b) - max number of children per parent
- Depth (d) - depth of least cost solution
- Max depth of Search Tree (m) [Note: this could be infinite if no solution]

UI ■ BFS : $O(b^d)$ time/space complexity, Complete, Optimal

UI ■ DFS : $O(b^m)$ time, $O(bm)$ space, not complete, not optimal

UI ■ Limited DFS : limit depth to L so we don't search infinitely
↳ Not complete if solution below depth limit, complete if $d \le L$
$O(b^L)$ time, $O(bL)$ space, not optimal (same as DFS)

UI ■ Iterative Deepening (ID) Search - Limited DFS, gradually increase depth until solution

↳ $O(b^d)$ time, $O(bd)$ space, complete, optimal

g(n): actual path real dist UI ■ Uniform-Cost Search (UCS) - actions have varying costs [think Dijkstra's]

- ε: smallest cost attained by any action, C^*: depth of optimal solution

↳ $O(b^{C^*/\varepsilon})$ time/space complexity, complete, optimal

h(n): estimated path crow dist to goal ↳ Heuristic I ① Best-First Search (Greedy Search, Informed) - UCS but with straight-line distances

- straight-line distances from current point to end point, good for estimates (not always optimal)
- travel to the option that has smallest straight-line distance to goal

f(n): heuristic I ② A^* Search: $f(n) = g(n) + h(n)$ - A^* is optimal if heuristic is admissible

- travel to the node with the smallest $f(n)$

Uninformed (UI) search: only have access to problem definition, no extra information

Informed (I) search: have access to problem definition, along with heuristic function

Admissible Heuristic: estimated cost ≤ cost of optimal path from n to goal [$h(n) \le h^*(n)$]

- basically - admissible heuristic cost should never overestimate the cost of an actual node
- if there are two admissible heuristics, choose the one that dominates (greater)

↳ $H_1 = 7$, $H_2 = 18$ $H_2 > H_1$ → choose H_2, it will solve more efficiently

↳ H_1 and H_2, neither one dominates → use $H_3(n) = \max(h_1(n), h_2(n))$

CSP

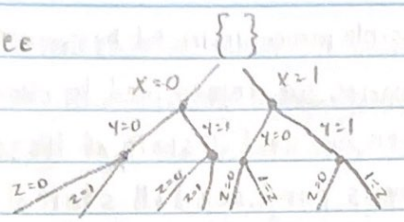
- Given a set of variables, values, and constraints, find a solution to the problem
  - Unary/Binary constraints - one variable involved, two variables involved
  - High-order constraints - more than 2 variables involved
  - Hard constraints - absolutely can not violate these
  - Soft constraints - not all constraints equally weighted, can violate some if needed
- SAT - given a statement with different variables and values, return a list of value assignments to the variables so the statement returns True

CSP as Search

- Initial State - empty variable assignment {}
- Final State Test
- Actions
- State - partial variable assignment
- Successor Function - generates next states based on current state and legal actions

CSP Search Tree

[DFS here]



Tree Depth : n

\# of values : d [0,1]

\# of leaves : d^n

\# of distinct states : d^n

Improvements of DFS for CSP

① Backtrack Search - DFS, but only generate feasible children

↳ "feasible" - no single constraint is violated

• Ex: N-Queens Problem - Place 1 Queen in a spot, then place 2nd Queen. If 2nd Queen is invalid, "backtrack" and place it again in different spot

② Value Ordering

• change order of values [i.e x=1 on left branch, x=0 on right branch]

• this permutes the leaf nodes, can help find solution quicker

• value ordering heuristic - assign value for the variable with the most restraints (variable with fewest legal values) first

③ Variable Ordering

• change order of variables [i.e instead of x→y→z (root … leaf), do x→z→y (root … leaf)]

• this prunes the tree so there's less to search through

• variable ordering heuristic - assign value for the variable with the least restraints (variable with most legal values) first

④ Forward Checking

• Maintain a set of legal values for each variable

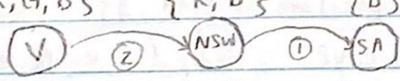
• When you assign a value to one variable, update legal values of other variables accordingly

• declare "bad state" when some variable has no legal values left

⑤ Arc Consistency

• Create state diagram for all variables → All arcs should be consistent

{R,G,B} {R,B} {B} ↳ each value has an option to go to



ARC 2 : {R→B} {G→R, G→B} {B→R} [R,G,B have options → consistent!]

ARC 1 : {R→B} {B→??} [B has no option, ARC 1 is inconsistent!]

• make ARC 1 consistent by removing B from NSW [this makes ARC 2 inconsistent]

• make ARC 2 consistent again by removing R from V

• Complexity : variable has at most "d" values [cardinality], n variables ⇒ $O(n^2 d^2 d)$ (# of arcs, can't check 1 arc, max # of times we might re-check an arc)

2P Games

Deterministic - Player's possible moves restricted by current state of the game

Chance - Player's possible moves are determined by chance (ie dice roll)

Perfect Information - Both players can see full state of the game

Imperfect Information - Players can't see full state of the game

Game Tree - Maps out the possibilities for the game state based on each player's turn - used to try and find the optimal move that gives you the best chance of winning

- ply - one person's turn move - both persons' turns

Minimax Algorithm - used to find the optimal move for a player, assuming the opponent is playing perfectly [uses backtracking DFS logic]

- Δ takes max value among child nodes, ∇ takes min value among child nodes

↳ $O(b^m)$ time, $O(bm)$ space, complete, optimal

Alpha-Beta Pruning - prune off branches that don't need to be explored because we already have a better alternative

↳ $O(b^{d/2})$ time - better than BFS!

KRR

Logical Connectives $\wedge$ (and) $\vee$ (or) $\neg$ (not) $\Rightarrow$ (imply) $\Leftrightarrow$ (equivalent)

Conjunction (AND): $S_1 \wedge S_2$ [conjuncts]

Disjunction (OR): $S_1 \vee S_2$ [disjuncts]

Implication: $S_1 \Rightarrow S_2$ [antecedent $\Rightarrow$ consequent]

Contrapositive: $[P \Rightarrow Q] = [\neg Q \Rightarrow \neg P]$

U — CNF: Clause - disjunction of literals

CNF is conjunction of clauses EX: $(A \vee \neg B) \wedge (B \vee \neg C \vee \neg D)$

U T — DNF: term - conjunction of literals

DNF is disjunction of terms EX: $(A \wedge B) \vee (A \wedge \neg C) \vee (A \wedge \neg D)$

T — Horn: clause with at most one positive literal

Note: this is a subset of CNF EX: $(A \vee \neg B \vee \neg C)$

U — NNF: Connectives: $\wedge, \vee, \neg$ $\neg$ only next to variables

EX: $(((\neg X \vee \neg Y) \wedge Z) \vee W) \vee \neg X$

Universal (U) - Normal form can express any sentence

Tractable (T) - Can solve SAT in linear time

U T — DNNF: decomposable NNF

Implies $X \Rightarrow Y$

X	Y	⇒
T	T	T
T	F	F
F	T	T
F	F	T

$w \models \alpha$ → [sentence α holds in world w]
↳ the variable assignments in w satisfy α

$M(\alpha) = \{w_1, w_2\}$ → [The models of α are w_1, w_2]
↳ w_1, w_2 satisfy α

- $\alpha \Leftrightarrow \beta$ [$M(\alpha) = M(\beta)$]
- α is inconsistent / unsatisfiable / contradictory [$M(\alpha) = \{\}$]
- α, β are mutually exclusive [$M(\alpha) \cap M(\beta) = \emptyset$] $\cap$ = intersection
- $\alpha \Rightarrow \beta$ [$M(\alpha) \subseteq M(\beta)$] $\subseteq$ = "is a subset of"

Complete term - term containing all of the variables
Valid - a sentence is valid if $M(\alpha) = W$ W = all worlds

Logical Inference Knowledge Base (KB) : $\Delta = \{\alpha_1, \alpha_2, \ldots \alpha_n\}$
Query (α) - a sentence we're investigating
Check whether $\Delta \models \alpha$ [Does Δ imply α?]

① Inference Method 1 - Enumerating Models
- create Truth Table (T/F) for all of the variables
- create columns for the sentences / Knowledge base, these are the outputs
- fill out contents of truth table.
  - $M(\Delta)$ represents worlds that satisfy statements in KB
  - $M(\alpha)$ represents worlds that satisfy the query α
  - If $M(\Delta) \subseteq M(\alpha)$, then $\Delta \models \alpha$.

Refutation Theorem: (α holds in Δ) iff ($M(\Delta \wedge \neg\alpha) = \{\}$)

② Inference Method 2 - Inference Rules
Modus Ponens : $\frac{\alpha,\ \alpha \Rightarrow \beta}{\beta}$
OR - Introduction : $\frac{\alpha,\ \beta}{\alpha \vee \beta}$ AND - Introduction : $\frac{\alpha,\ \beta}{\alpha \wedge \beta}$

$\Delta \vdash_R \alpha$ → [α can be inferred from Δ using inference rules R]

(12) • Soundness : If $\Delta \vdash_R \alpha$, Then $\Delta \models \alpha$
↳ If we can derive α from Δ using inference rules R, then α logically follows from Δ.

(21) • Completeness : If $\Delta \models \alpha$, Then $\Delta \vdash_R \alpha$
↳ If α logically follows from Δ, then we can derive α from Δ using inference rules R.

EX: $\Delta = \{A \wedge B,\ A \wedge B \Rightarrow D\}$
$R = \frac{\alpha,\ \alpha \Rightarrow \beta}{\beta}$
$\Delta' = \{A \wedge B,\ A \wedge B \Rightarrow D,\ D\}$

Resolution - convert a KB into CNF, cancel out variables between disjunctions to create new disjunctions

- Refutation complete - Resolution leads to contradiction if CNF is unsatisfiable
- Unit Resolutions (Linear Time) faster than normal Resolution (Exponential Time)
  - ↳ however, unit resolutions aren't refutation complete [def: using rules with one variable to cancel out]

① List Given Statements: Convert KB and claim $\neg\alpha$ into CNF, list each individual disjunction as a rule

② Cancel out variables from given rules to create new rules until no more rules can be applied (each variable is by itself in a rule)

- If there was a contradiction - $\Delta \wedge \neg\alpha$ is unsat, but we prove α
- If there was no contradiction - $\Delta \wedge \neg\alpha$ is sat, but we can't prove α

Conversion to CNF

① Get rid of all connectives except $\wedge, \vee, \neg$

- $\alpha \Rightarrow \beta \longrightarrow \neg\alpha \vee \beta$
- $\alpha \Leftrightarrow \beta \longrightarrow (\alpha \Rightarrow \beta) \wedge (\beta \Rightarrow \alpha) \longrightarrow (\neg\alpha \vee \beta) \wedge (\neg\beta \vee \alpha)$

② Push negations inwards

- $\neg(\alpha \wedge \beta) \rightarrow (\neg\alpha \vee \neg\beta)$ • $\neg(\alpha \vee \beta) \rightarrow (\neg\alpha \wedge \neg\beta)$

③ Distribute $\vee$ over $\wedge$

- $(\alpha \wedge \beta) \vee \gamma \longrightarrow (\alpha \vee \gamma) \wedge (\beta \vee \gamma)$

③ Inference Method 3: Reduction to SAT

- $\Delta \models \alpha$ iff $\Delta \wedge \neg\alpha$ is unsat
  - $\Delta \Rightarrow \alpha$ [$\Delta \wedge \neg\alpha$ is unsat]
  - $\Delta \Leftrightarrow \alpha$ [$\Delta \Rightarrow \alpha$; $\alpha \Rightarrow \Delta$]
  - $\Delta \wedge \neg\alpha$ is unsat ; $\neg\Delta \wedge \alpha$ is unsat
  - Δ is valid [$\neg\Delta$ is unsat]
  - Δ and α mutually exclusive [Δ and α unsat]
- Complete Methods - will return if something is SAT or not
  - DPLL - specialized backtracking algorithm, basically uses Resolution to determine if we can prune the tree
- Incomplete Methods - Local Search
  - Start with a state. if it's a solution, stop there.
  - If it's not a solution, change value of one variable (neighbor) Repeat until solution.

(min-conflicts) (hill-climbing)

- idea is to go to a neighbor that is closer to a solution, keep repeating
  - local minimum - all neighbors are worse, restart at different point
  - all neighbors are basically same as current state - go to a random one
- Local Search has no memory, possible to repeat states
- Simulated Annealing - traverse to neighbor based on probability

  S : current state   n : neighbor ΔE = violations(n) - violations(s)

  If $\Delta E < 0$: Traverse to that n

  Else : traverse to n based on probability $\left[\frac{1}{e^{\Delta E/T}}\right]$

  (the more violations a neighbor has, the less likely I go to it)

  T : as I get deeper into the graph, T increases

  ↳ more cautious about choosing bad neighbors the further we go

④ Inference Method 4 : Tractable Circuits

- #SAT - doesn't solve SAT, counts the number of satisfiable assignments
- NNF Circuit

solve #SAT in linear time

- Decomposability - direct children of an AND gate can't use the same variables (negations ok)
- Determinism - all combinations of pairs of children for an OR gate must cancel each other out
  - OR gates must have at most 1 true input under any circuit input
- Smoothness - direct children of an OR gate must use the same variables (negations ok)

How many assignments satisfy the problem (# SAT)?

① Assign all leaves to be 1   [literal T = 1, literal F = 0]

② Traverse upwards, multiplying at AND, adding at OR gates

③ Result at the root is the result of #SAT

α-β pruning

① Do the first triangle normally, no pruning involved, Save that value as X.

② If triangle to the right is Δ:

look through its children, if there's a child $\geq$ X then choose that one and prune all the ones after it (bigger than smallest one)

If triangle to the right is ∇:

look through its children, if there's a child $\leq$ X then choose that one and prune all the ones after it (smaller than biggest one).

③ Continue on the row above until finished.