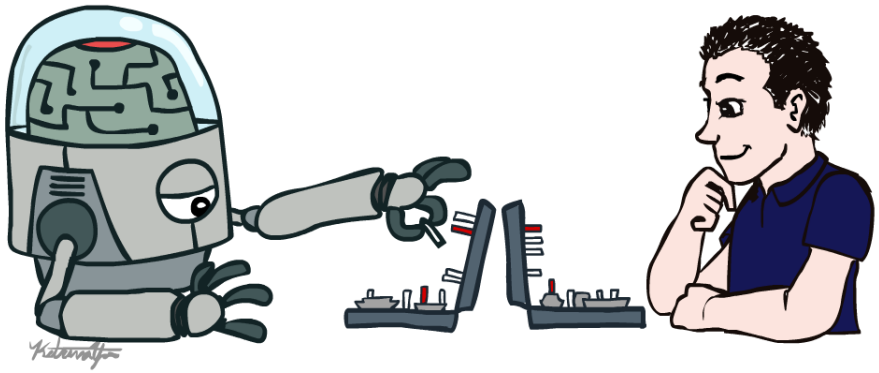
CSE 3521:
Introduction to Artificial Intelligence



[Many slides are adapted from the UC Berkeley. CS188 Intro to AI at UC Berkeley and previous CSE 3521 course at OSU.]

THE OHIO STATE UNIVERSITY

Final Exam Format

- 60 minutes
  - Available for 24 hrs
  - i.e, you can take it anytime during the day

- Carman Quiz
  - Mixed format: MCQ, T/F, fill-in-the-blanks, ….

- If you need any special accommodation email me **ASAP**
  - tabassum.13@osu.edu

PEAS

PEAS

- **P**erformance – measuring the agent's success
- **E**nvironment – what populates the problem's world?
- **A**ctuators – what can the agent act with?
- **S**ensors – how can the agent perceive the world?

Peas: Examples

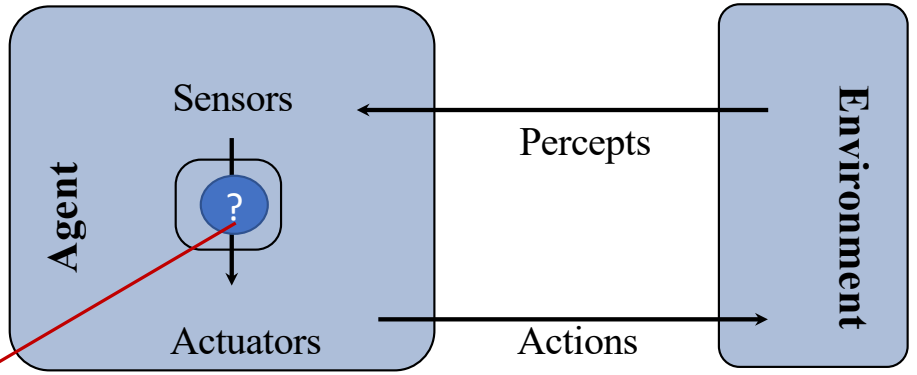
Agent Type	**Perf. Measure**	**Environment**	**Actuators**	**Sensors**
Medical diagnosis system	Healthy patient, minimize costs/lawsuits	Patient, hospital, staff	Display questions, tests, diagnoses, treatments, referrals	Keyboard entry of symptoms, findings, patient's answers
Satellite image analysis system	Correct image classification	Downlink from orbiting satellite	Display classification of scene	Color pixel arrays (cameras)
Part-picking robot	Percentage of parts in correct bins	Conveyor belt with parts, bins	Jointed arm and hand	Camera, joint angle sensors
Refinery controller	Maximize purity, yield, safety	Refinery, operators	Valves, pumps, heaters, displays	Temperature, pressure, chemical sensors
Interactive English tutor	Maximize student's score on test	Set of students, testing agency	Display exercises, suggestions, corrections	Keyboard entry

Rational Agent

What makes an AI agent

- **Agent** – an entity that perceives its environment through sensors, and acts on it with effectors (actuators).

- Percepts are constrained by Sensors + Environment
- Actions are constrained by Actuators + Environment

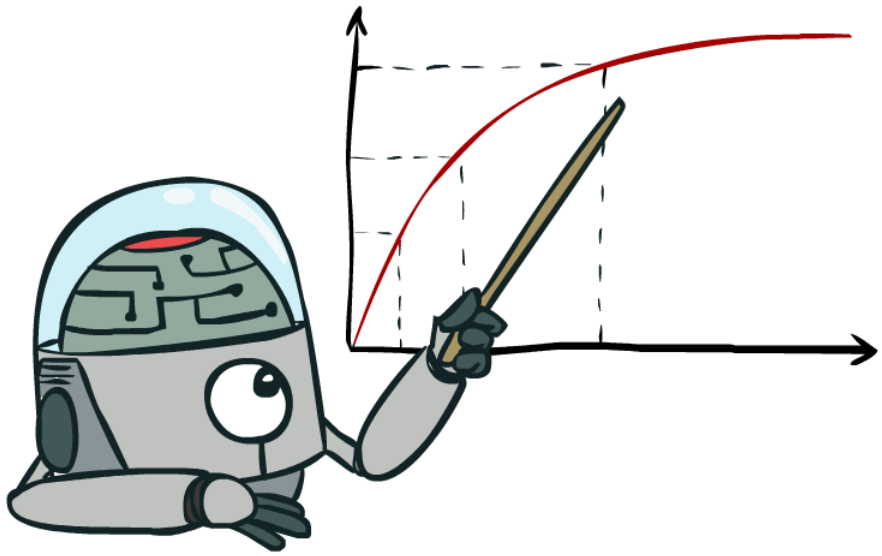


Agent Function (policy) – how does it choose the action?

What is a rational AI agent?

- A **rational agent** always acts to **maximize its expected performance measure**, given current **percept/state**

- Rationality ≠ omniscience
  - There is “uncertainty” in the environment.
  - That is why we emphasize “expected”.



Kinds of Environments

- **Six** common properties to distinguish environments (not exhaustive)
  - Fully observable vs Partially observable
  - Single agent vs Multiagent
  - Deterministic vs Stochastic
  - Episodic vs Sequential
  - Static vs Dynamic
  - Discrete vs Continuous

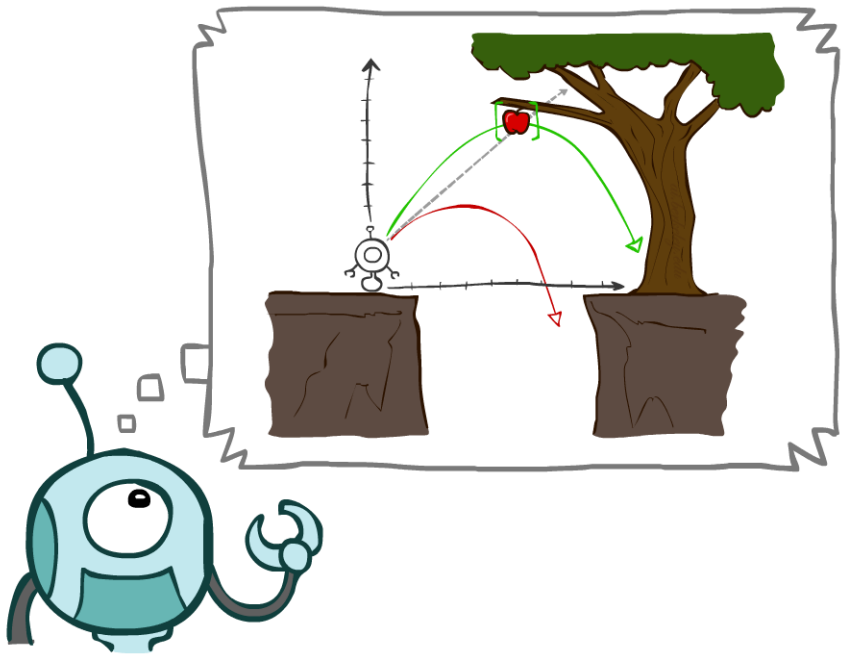
Examples

	Crossword puzzle	Taxi Driving
Observability	Fully	Partially
Deterministic vs Stochastic	Deterministic	Stochastic
Episodic vs Sequential	Sequential	Sequential
Static vs Dynamic	Static	Dynamic
Discrete vs Continuous	Discrete	Continuous
Single vs Multi Agent	Single	Multi

Search

Search

- Types
  - Uninformed Search Methods
    - Depth-First Search
    - Breadth-First Search
    - Uniform-Cost Search
  - Informed Search Methods
    - Greedy Search
    - A* Search



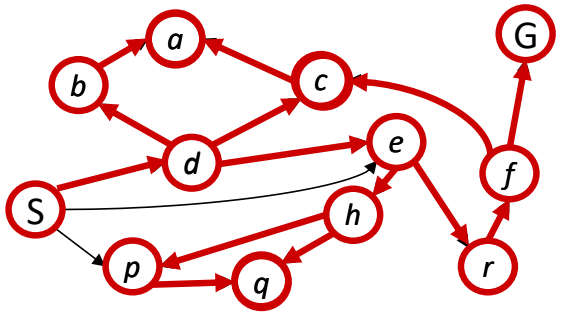
Search Problems

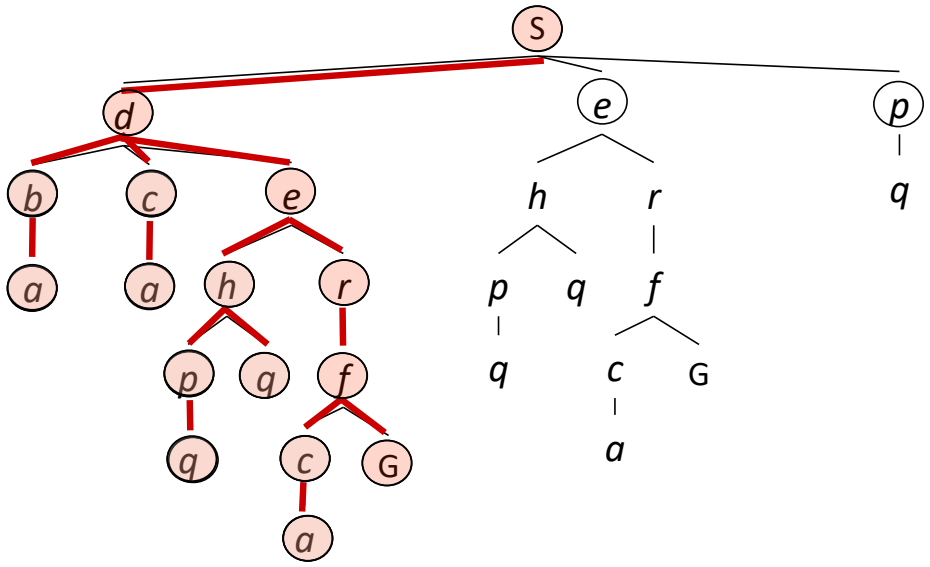
- A search problem consists of:
  - A state space
  - A successor function (with actions, costs)
  - A start state and a goal test
- A solution is a sequence of actions (a plan) which transforms the start state to a goal state

Depth-First Search

Strategy: expand a deepest node first

Implementation: Fringe is a LIFO stack

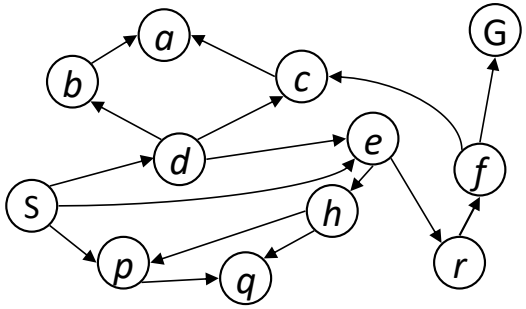


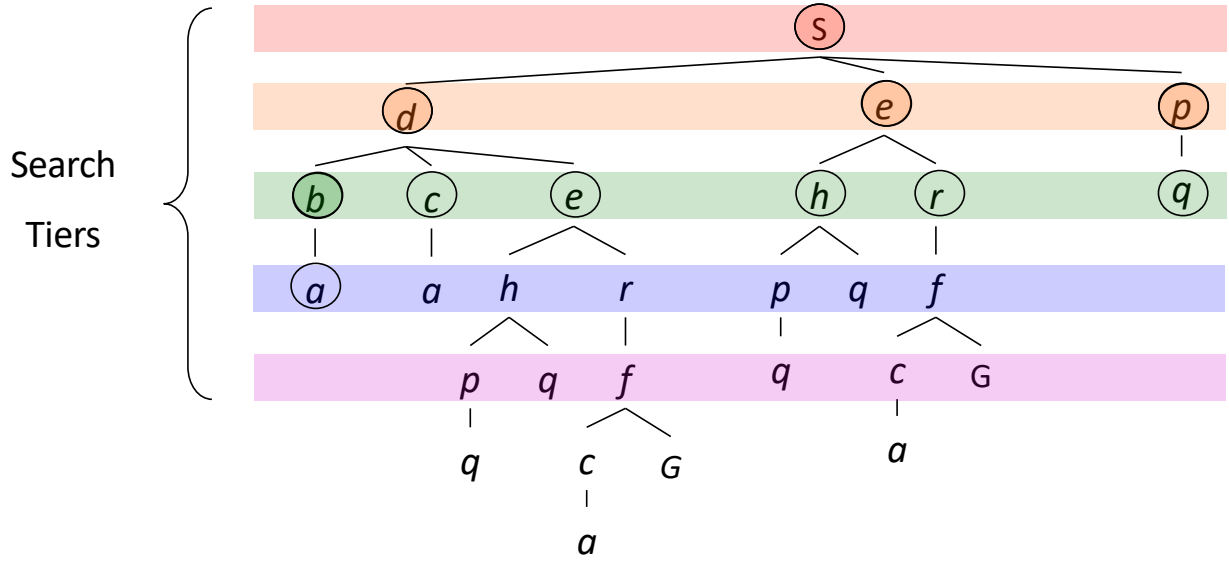


Breadth-First Search

Strategy: expand a shallowest node first

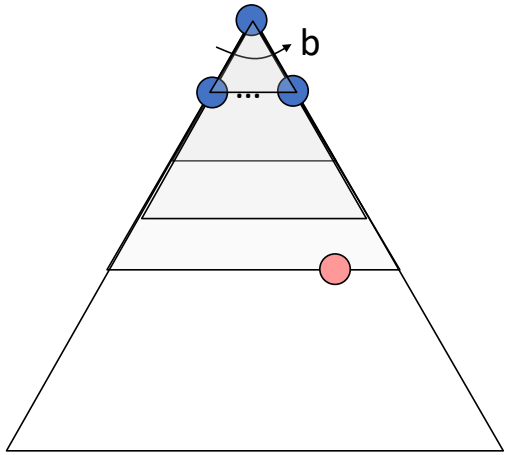
Implementation: Fringe is a FIFO queue





Iterative Deepening Search

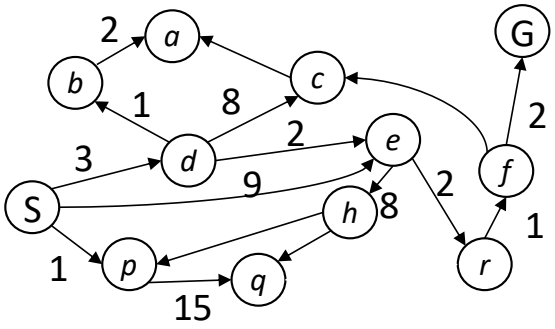
- Idea: get DFS's space advantage with BFS's time / shallow-solution advantages
  - Run a DFS with depth limit 1. If no solution…
  - Run a DFS with depth limit 2. If no solution…
  - Run a DFS with depth limit 3. …..
- Isn't that wastefully redundant?
  - Generally most work happens in the lowest level searched, so not so bad!
- A Preferred method with large search space and depth of solution not known

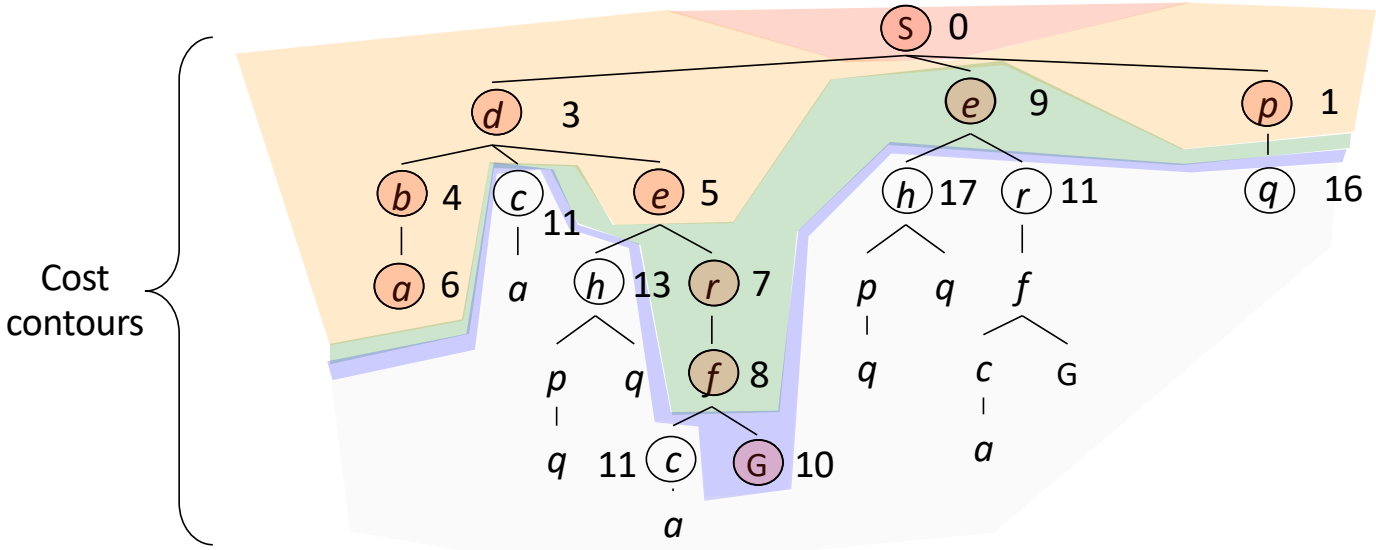


Uniform Cost Search (USC)

Strategy: expand a cheapest node first:

Fringe is a priority queue (priority: cumulative cost)



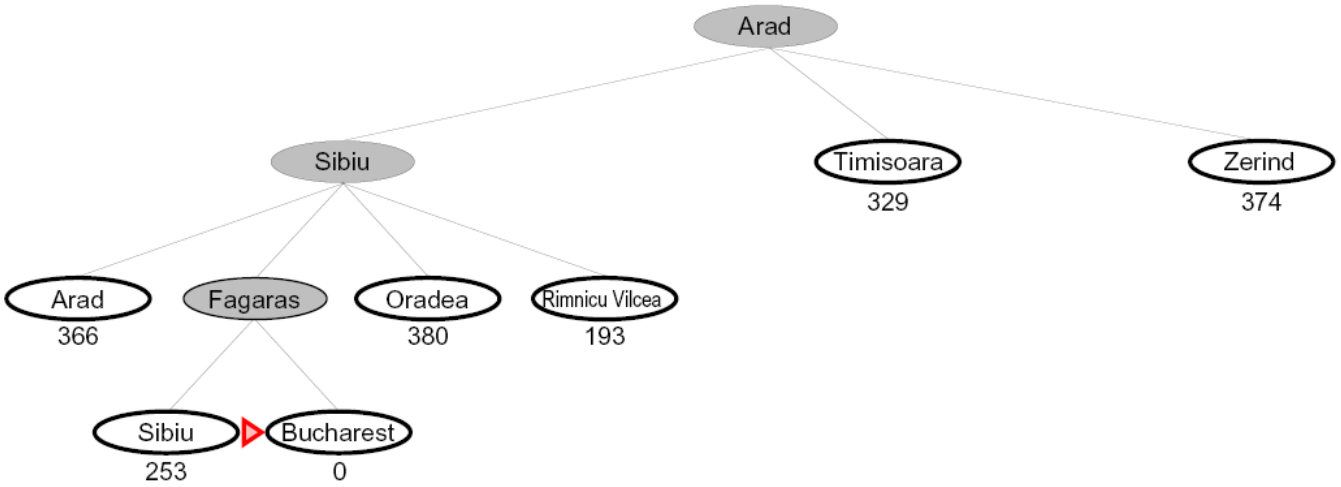


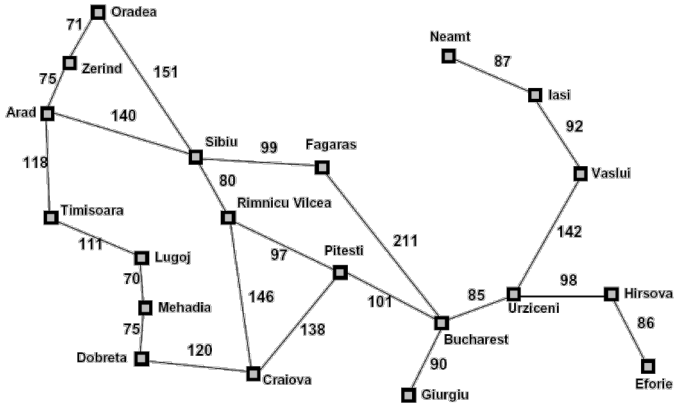
Search Heuristics

- A heuristic is
  - A function that *estimates* how close a state is to a goal
  - Designed for a particular search problem
  - Examples: Manhattan distance, Euclidean distance for pathing
    - not the exact “path” distance

Greedy Search

- Expand the node that seems closest…

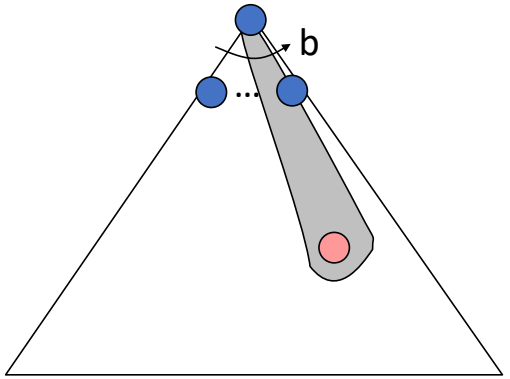




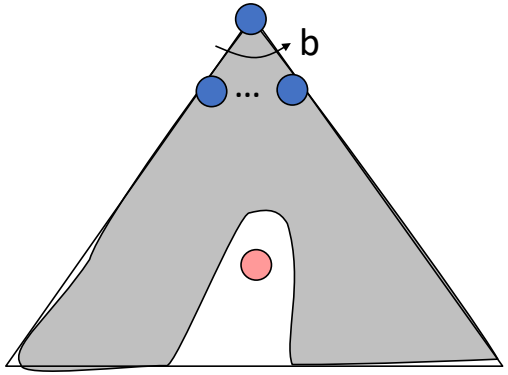
- What can go wrong?
  - Does not guarantee the optimal solution

Greedy Search

- Strategy: expand a node that you think is closest to a goal state
  - Heuristic: estimate of distance to nearest goal for each state



- A common case:
  - Best-first takes you straight to the (wrong) goal



- Worst-case: like a badly-guided DFS

A* Search

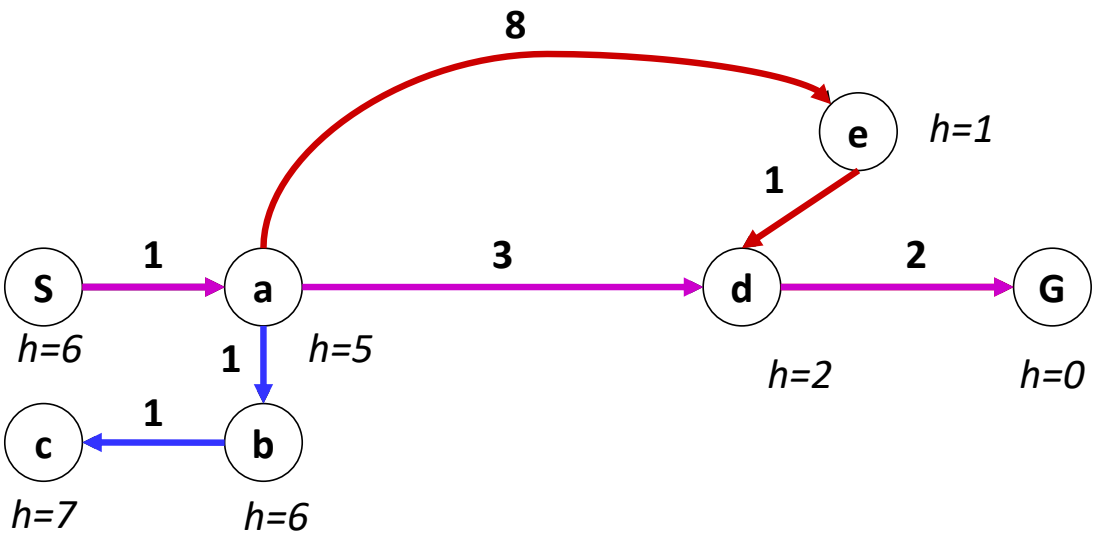
UCS

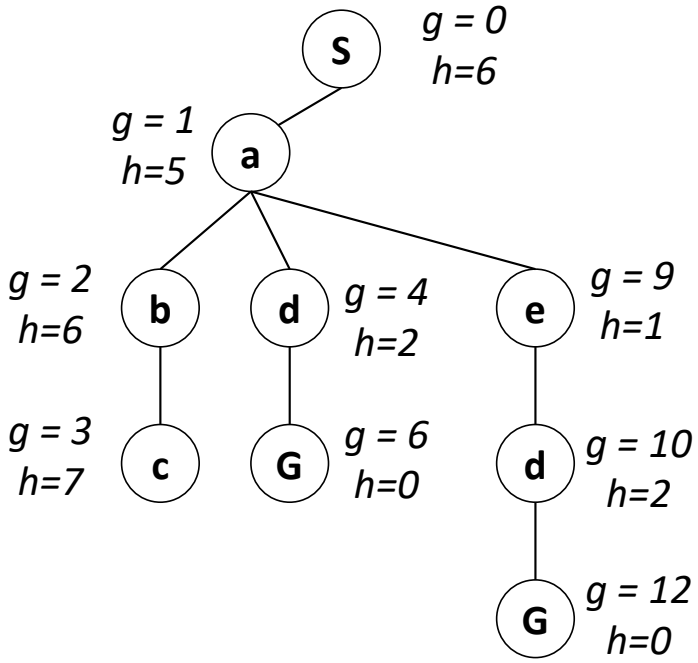
Greedy

A*

A* is a Combination of UCS and Greedy

- 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)

Propositional Logic

Logic

- For logical agents, knowledge is definite
  - Each proposition is either “True” or “False”
- Logic has advantage of being simple representation for knowledge-based agents
  - But limited in its ability to handle uncertainty
- We will examine propositional logic and first-order logic

Logical Agent

- Need agent to represent beliefs
  - "There is a pit in (2, 2) or (3, 1)"
  - "There is no Wumpus in (2, 2)"
- Need to make inferences
  - If available information is correct, draw a conclusion that is guaranteed to be correct
- Need representation and reasoning
  - Support the operation of knowledge-based agent

Knowledge Representation

- For expressing knowledge in computer-tractable form
- Knowledge representation language defined by
  - **Syntax**
    - Defines the possible well-formed configurations of sentences in the language
  - **Semantics**
    - Defines the "meaning" of sentences (need interpreter)
    - Defines the truth of a sentence in a world (or model)

The Language of Arithmetic

- Syntax: "$x + 2 \geq y$" is a sentence

  "$x2 + y >$" is not a sentence

- Semantics: $x + 2 \geq y$ is true iff the number $x + 2$ is no less than the number y

  $x + 2 \geq y$ is True in a world where x=7, y=1

  $x + 2 \geq y$ is False in a world where x=0, y=6

Inference

- Sentence is valid iff it is true under all possible interpretations in all possible worlds
  - Also called tautologies
  - "There is a stench at (1,1) or there is not a stench at (1,1)"
  - "There is an open area in front of me" is not valid in all worlds

- Sentence is satisfiable iff there is some interpretation in some world for which it is true
  - "There is a wumpus at (1,2)" could be true in some situation
  - "There is a wall in front of me and there is no wall in front of me" is unsatisfiable

Propositional Logic: Syntax

- Syntax of propositional logic defines allowable sentences
- Atomic sentences consists of a single proposition symbol
  - Each symbol stands for proposition that can be True or False
- Symbols of propositional logic
  - Propositional symbols: *P*, *Q*, … (e.g., "Today is Tuesday")
  - Logical constants: *True*, *False*
- Making complex sentences
  - Logical connectives of symbols: $\wedge$, $\vee$, $\Leftrightarrow$, $\Rightarrow$, $\neg$
  - Also have parentheses to enclose each sentence: (…)
- Sentences will be used for inference/problem-solving

Propositional Logic: Syntax

- *True*, *False*, S_1, S_2, … are sentences
- If S is a sentence, $\neg S$ is a sentence
  - Not (negation)
- $S_1 \wedge S_2$ is a sentence, also $(S_1 \wedge S_2)$
  - And (conjunction)
- $S_1 \vee S_2$ is a sentence
  - Or (disjunction)
- $S_1 \Rightarrow S_2$ is a sentence (e.g., “Today is Tuesday” implies “Tomorrow is Wednesday”)
  - Implies (conditional)
- $S_1 \Leftrightarrow S_2$ is a sentence
  - Equivalence (biconditional)

Propositional Logic: Semantics

- Semantics defines the rules for determining the truth of a sentence
  - With respect to a particular model)
    - $\neg S$ is true iff S is false
    - $S_1 \wedge S_2$ is true iff S_1 is true and S_2 is true
    - $S_1 \vee S_2$ is true iff S_1 is true or S_2 is true
    - $S_1 \Rightarrow S_2$ is true iff S_1 is false or S_2 is true
      (is false iff S_1 is true and S_2 is false)
      (if S_1 is true, then claiming that S_2 is true, otherwise make no claim)
    - $S_1 \Leftrightarrow S_2$ is true iff $S_1 \Rightarrow S_2$ is true and $S_2 \Rightarrow S_1$ is true (S_1 same as S_2)

Propositional Inference: Enumeration Method

- Truth tables can test for valid sentences
  - True under all possible interpretations in all possible worlds
- For a given sentence, make a truth table
  - Columns as the combinations of propositions in the sentence
  - Rows with all possible truth values for proposition symbols
- If sentence true in every row, then valid

Example

- Test $(P \wedge H) \Rightarrow (P \vee \neg H)$

P	H	$P \wedge H$	$\neg H$	$(P \vee \neg H)$	$(P \wedge H) \Rightarrow (P \vee \neg H)$
False	False	False	True	True	True
False	True	False	False	False	True
True	False	False	True	True	True
True	True	True	False	True	True

Inference Rules for Prop. Logic

- Modus Ponens
  - From implication and premise of implication, can infer conclusion

$$\frac{\alpha \Rightarrow \beta, \quad \alpha}{\beta}$$

Inference Rules for Prop. Logic

- And-Elimination
  - From conjunction, can infer any of the conjuncts

$$\frac{\alpha_1 \wedge \alpha_2 \wedge \cdots \wedge \alpha_n}{\alpha_i}$$

Inference Rules for Prop. Logic

- And-Introduction
  - From list of sentences, can infer their conjunction

$$\frac{\alpha_1, \alpha_2, \ldots, \alpha_n}{\alpha_1 \wedge \alpha_2 \wedge \cdots \wedge \alpha_n}$$

Inference Rules for Prop. Logic

- Or-Introduction
  - From sentence, can infer its disjunction with anything else

$$\frac{\alpha_i}{\alpha_1 \vee \alpha_2 \vee \cdots \vee \alpha_n}$$

Inference Rules for Prop. Logic

- Double-Negation Elimination
  - From doubly negated sentence, can infer a positive sentence

$$\frac{\neg\neg\alpha}{\alpha}$$

Inference Rules for Prop. Logic

- Unit Resolution
  - From disjunction, if one of the disjuncts is false, can infer the other is true

$$\frac{\alpha \vee \beta, \quad \neg\beta}{\alpha}$$

Inference Rules for Prop. Logic

- Resolution
  - Most difficult because β cannot be both true and false
  - One of the other disjuncts must be true in one of the premises
    - (implication is transitive)

$$\frac{\alpha \vee \beta, \quad \neg\beta \vee \gamma}{\alpha \vee \gamma}$$

$$\neg\beta \quad \textbf{OR} \quad \beta$$

$$\frac{\alpha \vee \beta, \neg\beta}{\alpha} \quad \textbf{OR} \quad \frac{\neg\beta \vee \gamma, \beta}{\gamma}$$

$$\alpha \quad \textbf{OR} \quad \gamma$$

First Order Logic

Syntax of FOL: Basic Elements

- Constant symbols for specific objects
  KingJohn, 2, *OSU*, …
- Variables
  x, *y*, *a*, *b*, …
- Predicate properties (unary) / relations (pairwise or more)
  Smart(), Brother(), Married(), >, …
- Functions (return objects)
  Sqrt() , *LeftTo()*, *FatherOf(),* …
- Connectives
  $\wedge$ $\vee$ $\neg$ $\Rightarrow$ $\Leftrightarrow$
- Quantifiers
  $\forall$ $\exists$
- Equality
  =

Quantifiers

- Currently have logic that allows objects
- Now want to express properties of entire collections of objects
  - Rather than enumerate the objects by name
- Two standard quantifiers
  - Universal $\forall$
  - Existential $\exists$

Properties of Quantifiers

- Important relations
  - $\exists x \;\; P(x) \;=\; \neg \forall x \;\; \neg P(x)$
  - $\forall x \;\; P(x) \;=\; \neg \exists x \; \neg P(x)$
  - $P(x) \Rightarrow Q(x)$ is same as $\neg P(x) \vee Q(x)$
  - $\neg \; (P(x) \wedge Q(x))$ is same as $\neg P(x) \vee \neg Q(x)$

Universal Quantifiers

- $\forall x \, \forall y$ is same as $\forall y \, \forall x \, (\forall x,y)$
- $\exists \, x \, \exists \, y$ is same as $\exists \, y \, \exists \, x \, (\exists \, x,y)$
- $\exists \, x \, \forall y$ is <u>not same</u> as $\forall y \, \exists \, x$
  - $\exists \, y \; Person(y) \wedge (\forall x \; Person(x) \Rightarrow Loves(x,y))$
    - "There is someone who is loved by everyone"
  - $\forall x \; Person(x) \Rightarrow \exists y \; Person(y) \wedge Loves(x,y)$
    - "Everybody loves somebody"
      (not guaranteed to be the same person)

How to do inference in FOPC

- Reduction of first-order inference to propositional inference
- First-order inference algorithms
  - Generalized Modus Ponens
  - Forward chaining ***
  - Backward chaining ***
  - Resolution-based theorem proving ***

Reduction to Propositional Inference

- Universal Quantifiers ($\forall$)
  - Recall: Sentence must be true *for all* objects in the world (all values of variable)
  - So substituting any object must be valid (Universal Instantiation, UI)
- Example
  - $\forall x \;\; Person(x) \Rightarrow Likes(x, IceCream)$
    - Substituting: (1), $\{x/Jack\}$
  - $Person(Jack) \Rightarrow Likes(Jack, IceCream)$

Reduction to Propositional Inference (con't)

- Existential Quantifiers (∃)
  - Recall: Sentence must be true *for some* object in the world (or objects)
  - Assume we know this object and give it an arbitrary (unique!) name (Existential Instantiation, EI)
  - Known as Skolem constant (SK1, SK2, …)
- Example
  - $\exists x \;\; Person(x) \wedge Likes(x, IceCream)$
    - Substituting: (1), $\{x/SK1\}$
  - $Person(SK1) \wedge Likes(SK1, IceCream)$
- We don't know who "SK1" is (and usually can't), but we know they must exist

Reduction to Propositional Inference (con't)

- Multiple Quantifiers
  - No problem if same type ($\forall x,y$ or $\exists x,y$)
  - Also no problem if: $\exists x\ \forall y$
    - There must be some *x* for which the sentence is true with every possible *y*
    - Skolem constant still works (for *x*)
- Problem with $\forall x\ \exists y$
  - For every possible *x*, there must be some *y* that satisfies the sentence
  - Could be different *y* value to satisfy for each *x*!
  - Solution Skolem function

Reduction to Propositional Inference (con't)

- Example
  - $\forall x \exists y$ *Person*$(x) \Rightarrow$ *Loves*(x,y)
    - Substitute: (1), {$y/$*SK1*(x)}
  - $\forall x$ *Person*$(x) \Rightarrow$ *Loves*$(x,$*SK1*$(x))$
    - Then: (2), {x/*Jack*}
  - *Person*$($*Jack*$) \Rightarrow$ *Loves*$($*Jack*,*SK1*$($*Jack*$))$

- *SK1*(x) is *effectively* a function which returns a person that *x* loves. But, again, we can't generally know the specific value it returns.

Reduction to Propositional Inference (con't)

- Internal Quantifiers
  - Previous rules only work if quantifiers are external (left-most)
  - Consider: $\forall x\ (\exists y\ \mathit{Loves}(x,y)) \Rightarrow \mathit{Person}(x)$
  - "For all *x*, if there is some *y* that *x* loves, then *x* must be a person"
  - A Skolem function limits the values *y* could take (to one) and we can't know what it is.
- Need to move the quantifier outward
  - $\forall x\ (\exists y\ \mathit{Loves}(x,y)) \Rightarrow \mathit{Person}(x)$
  - $\forall x\ \neg(\exists y\ \mathit{Loves}(x,y)) \vee \mathit{Person}(x)$ (convert to $\neg,\vee,\wedge$)
  - $\forall x\ \forall y\ \neg\mathit{Loves}(x,y) \vee \mathit{Person}(x)$ (move $\neg$ inward)
  - $\forall x\ \forall y\ \mathit{Loves}(x,y) \Rightarrow \mathit{Person}(x)$
- Now we can see that we can actually substitute *anything* for *y*
- May need to rename variables before moving quantifier left
- Once have non-quantified sentences (from quantified sentences using UI, EI), possible to reduce first-order inference to propositional inference

Forward and Backward Chaining

- Have language representing knowledge (FOL) and inference rules (Generalized Modus Ponens)
  - Now study how a reasoning program is constructed
- Generalized Modus Ponens can be used in two ways:
  - Start with sentences in KB and generate new conclusions (forward chaining)
    - **"Used when a new fact is added to database and want to generate its consequences"**

      or
  - Start with something want to prove, find implication sentences that allow to conclude it, then attempt to establish their premises in turn (backward chaining)
    - **"Used when there is a goal to be proved"**

Forward Chaining

- Forward chaining normally triggered by addition of new fact to KB (using TELL)
- When new fact *p* added to KB:
  - For each rule such that *p* unifies with a premise
    - If the other premises are known, then add the conclusion to the KB and continue chaining
  - Premise: Left-hand side of implication
    - Or, each term of conjunction on left hand side
  - Conclusion: Right-hand side of implication
- Forward chaining uses unification
  - Make two sentences (fact + premise) match by substituting variables (if possible)
- Forward chaining is data-driven
  - Inferring properties and categories from percepts

Backward Chaining

- Backward chaining designed to find all answers to a question posed to KB (using ASK)
- When a query *q* is asked:
  - If a matching fact *q′* is known, return the unifier
  - For each rule whose consequent *q′* matches *q*
    - Attempt to prove each premise of the rule by backward chaining
- Added complications
  - Keeping track of unifiers, avoiding infinite loops
- Two versions
  - Find any solution
  - Find all solutions

Resolution

- Uses proof by contradiction
  - Referred to by other names
    - Refutation
    - Reductio ad absurdum
- Inference procedure using resolution
  - To prove *P*:
    - Assume *P* is FALSE
    - Add $\neg P$ to KB
    - Prove a contradiction
  - Given that the KB is known to be True, we can believe that the negated goal is in fact False, meaning that the original goal must be True