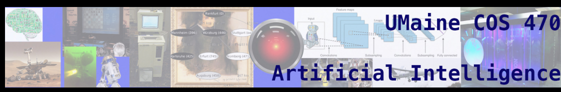


Search

"How do I find a solution to a problem?"

The problem

- Give some description of a goal, how to achieve it?
- Think of agent
  - Being in some state
    - Chess: board position
  - Knowing what properties a goal state should have
    - Chess: king in check, nowhere to move that isn't in check
  - Knowing how to move from state to state
    - Chess: rules of the game
- State space: composed of all possible states
- State space search:

From a start state, find a path to a goal state

What can be represented in this formalism?

- Anything that has objects, properties, relationships , ways to get from state to state
- Chess: board, pieces, properties of pieces, locations of pieces, moves…
- Theorem proving: axioms, rules of inference, theorem, objects
- Route planning: locations, roads, direction of travel of roads, speed limits, …

What can be represented in this formalism?

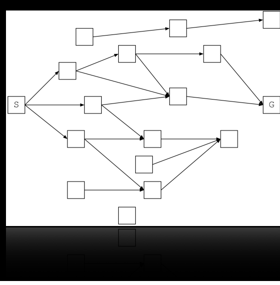
- Mobile robot: robots, other objects, locations, locations of objects, actions robot can take, sensor data, …
- Medicine: patient, providers, equipment, pathogens, drugs, drug–patient effects, drug–pathogen effects, …
- Natural language understanding: words, phrases, definitions, grammar rules,…

Domains

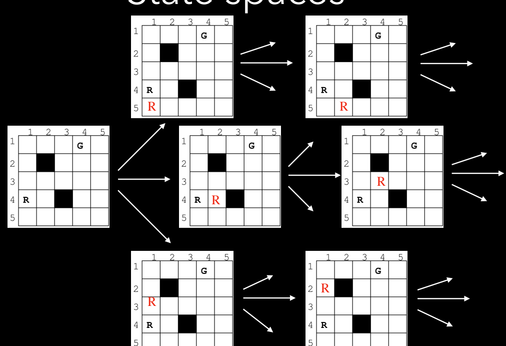
- Domain: the world of the agent
- Domain knowledge: what agent can know about world
  - Chess: moves, pieces' worth, …
  - Mobile robot: actions, action results, sensors, temperature, objects can't occupy same space, …
  - Medicine: anatomy, physiology, pathology, pharmacology, …
- Agent may know domain knowledge, or could be built in to problem/solution formalization

State spaces

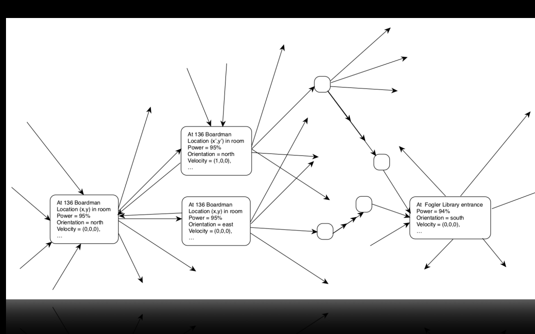
- Model state space with graph
- Often a digraph – e.g., one-way streets, irreversible reactions, …



State spaces



State spaces



State space search formalism

- Problem: state space + start state + goal state description
  - Description may completely or partially describe goal
  - May have 0, 1, or more actual goal states
- Solution: path from start state to goal state
- Search: process of finding the path
  - Start at start state
  - Expand the start state by finding its children (neighbors)
  - If goal found, stop
  - Else, systematically expand children, etc.

State representation issues

- What to put in the state?
  - Important stuff
  - Enough to differentiate between states
- What to leave out?
  - The rest
  - Saves space, time during matching

COS 470/570
Introduction to Artificial Intelligence

State representation issues

- Represent entire state?
- Partial state representation?
  - Concentrate search on important features of goal
  - Can match multiple goals
- Concrete or abstract state description?
  - "White king is on K1, black queen is on Q1,…" or "White king is in check, no moves that aren't also in check"
  - Other abstract representations: "understand the utterance", "diagnose the patient's problem", "create a plan to build a house", …
  - Relative or absolute description (of goals, esp.)?
    - "Closest state to the ocean"
    - Diagnosis that covers most symptoms

COS 470/570
Introduction to Artificial Intelligence

State space representation

- Represent entire state space?
  - Pros: can use global knowledge, can guarantee optimal path
  - Cons: map of space may be unavailable, size may be huge (GPS route planning, chess) or infinite (NLP) – too big to store, too big to search efficiently
- Generate states as needed?
  - Need operators to apply to states ⇒ children
  - Pros: cheaper for large search spaces, can deal with huge/infinite search spaces
  - Cons: maybe inefficient for small search spaces, may not be able to "run" operators in reverse

COS 470/570
Introduction to Artificial Intelligence

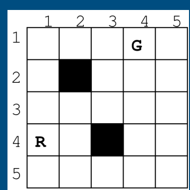
Operators

- Operator – if s, s_i are states:

$$f(s) \rightarrow \{s_1, s_2, \dots\}$$

- Operator set: defines all legal state transitions
- Choosing operator to apply to state:
  - Match description of applicable state(s) to current state
  - Preconditions
- Operator provides description of new state

- Robot world:



- State representation?
- Initial state? Goal state?
- Operators?

Uninformed search

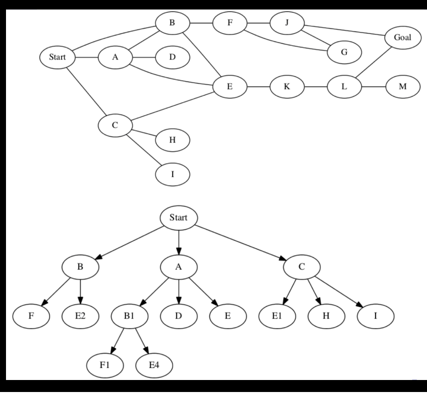
Blind (uninformed) search

- Basically, all the searches you've seen so far
- No information used about topology of state space
- Which node chosen for expansion ⇒ search type

Search trees

- Search space: graph of states
- Search tree:
  - Record/state of search so far
  - Root: node corresponding to initial state
  - Leaves: nodes that are candidates for expansion (frontier)
  - Interior nodes: states that have been seen/expanded
- Nodes ⇔ states
  - Information about the state
  - Bookkeeping information
- Don't want repeated nodes
- Path from root → goal node at leaves = solution

Search space vs search tree



Kinds of search algorithms

- Uninformed search
  - No knowledge about search space guides search
  - Often exponential in worst & average case
    - Not exponential in total number of nodes *n*!
    - Rather, search is exponential in *depth* of search tree
    - If b = branching factor, d = depth, then most are $\mathcal{O}(b^d)$
    - n is also $\mathcal{O}(b^d)$
- Informed (heuristic) search
  - Use heuristic knowledge to choose nodes
  - Heuristics about topology, problem structure, domain, problem solving itself

Kinds of search algorithms

- Searches also differ by general order they expand nodes
- Breadth-first search (BFS)
  - What is it?
  - Implement with what data structure?
- Depth-first search (DFS)
  - What is it?
  - Implement with what data structure?
  - Temporal backtracking
  - Backtracking during search vs during execution

Evaluating search algorithms

- Completeness
- Time/space complexity
- Optimality
  - of quality of solution (path cost, goal selected)
  - of search effort/space
- Complexity of algorithm (to some extent)

Surprise: trade-offs!

Evaluation of BFS

- As a group, answer:
  - Complete?
  - Optimal? If yes, under what condition(s)?
  - Time complexity?
  - Space complexity?

Evaluation of BFS

- Complete?
- Optimal? If yes, under what condition(s)?
- Time complexity:
  - Process each node, each edge, so $\mathcal{O}(|V| + |E|)$
  - Best, average, and worst-case time!
  - Cool – linear…right?
  - Suppose goal is *d* links away from root, and on average branching factor is *b*
  - #nodes seen/expanded? or, better, what is $|E|$? $\mathcal{O}(b^d)$
- Space complexity? $\mathcal{O}(b^d)$

How bad is that, though, really?

- Suppose *b* = 2, process 1 edge/ns

d	Time
2	4 ns
3	8 ns
4	16 ns
5	32 ns
6	64 ns
7	128 ns
8	256 ns
9	512 ns
10	~1 μs
20	~1 ms
30	~1 s

How bad is space complexity?

- Do we need to store all expanded nodes?
  - Could prune branch when we reach known node
  - Problem: how do we know?
- Worst case – search space is a tree
- Suppose only require 1 byte/node
- For *b*, *d* from before:
  - Goal at level 30 ⇒ need 1 GiB
  - Goal at level 85 ⇒ 32 YiB (yobibytes or 39 yottabytes) = 35 trillion TiB!
- If *d* = 266, store 1 bit/atom, would need all the atoms in the universe

Evaluation of DFS

- As a group, answer:
  - Complete?
  - Optimal? If yes, under what condition(s)?
  - Time complexity?
  - Space complexity?

Evaluating DFS

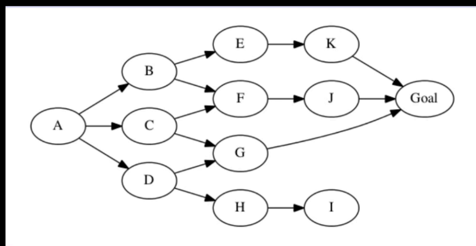
- Complete?
- Optimal? If so, under what condition(s)?
- Time & space complexity?
  - With branching factor *b* and goal depth *d*…
  - What's max #edges traversed in worst case? All of them!
  - What's the max # nodes stored at any one time? $\mathcal{O}(bd)$

BFS vs DFS

	BFS	DFS
Complete?	Yes	Yes
Optimal?	Yes (with uniform costs)	No
Time?	Exponential	Exponential
Space?	Exponential (*b*=2,*d*=80: ~1E12 TiB)	Linear (*b*=2, *d*=80: 2x80=160 B)
Other	Susceptible to infinite *b*	Susceptible to infinite *d*

Can we do better?

- Iterative deepening DFS (IDFS)
  - Use DFS to search a series of depths into the graph
  - Overall, behaves like BFS: expands level at a time



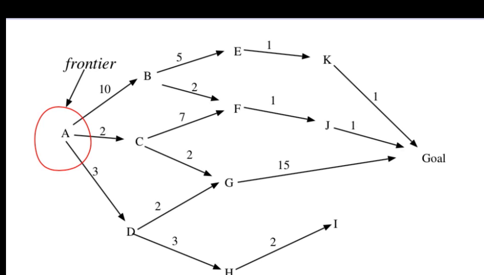
Can we do better?

- Iterative deepening DFS (IDFS)
  - Use DFS to search a series of depths into the graph
  - Overall, behaves like BFS: expands level at a time
  - Properties:
    - Complete?
    - Optimal?
    - Time complexity?
    - Space complexity?

Finding optimal paths

- With uniform costs: BFS works
- But what about weighted graphs?
- Maybe apply idea of BFS to weight maxima rather than edges
- Branch-and-bound search
  - Keep track of a frontier of unexpanded nodes
  - Keep track of cost of path to each node
  - Expand cheapest node among all nodes on the frontier
  - Stop when cost of path to goal < or = to cost to all other nodes on frontier
- Also called, confusingly enough, uniform cost search

Branch-and-bound search



Branch-and-bound search

- Complete?
- Optimal? If so, under what condition(s)?
  - Non-negative cost edges/operators
  - Test for goal prior to expansion
- Time complexity?
  - In worst case: expand everything: exponential w/ *b*, *d*
  - But may not have to expand all nodes at goal depth
  - If not, then effective branching factor < *b*
  - If ϵ = min step (link) cost, *C* = cost of best path: $\mathcal{O}(b^{C/\epsilon})$

What about Dijkstra's algorithm?

- Dijkstra's [1959] finds shortest path to all nodes from start node
  - Can be modified to stop when a goal is found
  - Running time $\mathcal{O}(|E| + |V|\log(|V|))$
- So why not use it?
- Well, we are, sort of – branch-and-bound is variant that stops when goal is found
- Complexity roughly the same: $b^d \leq |E|$

Bi-directional search

- Should mention: can search backward from goal → start
- Why would you?
- Could try searching both ways
  - If searches meet, then done
  - Two searches of depth d/2
  - Time complexity $\mathcal{O}(b^{d/2})$
- Problems:
  - Can't use partial goal descriptions
  - Hard to deal with multiple goal states
  - Have to be able to regress through operators when going backward

Kinds of problems and search

Properties of problems

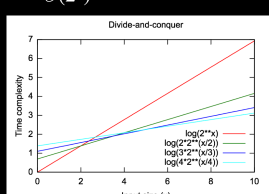
- Problems differ by their characteristics
- Different problems ⇒ different difficulty, different search techniques

Kinds of problems

- Synthesis vs classification
  - Classification (categorization)
    - "What is category does this thing belong to?"
    - Ex: image recognition, NLP, diagnosis, loan decisions
    - Also called analysis problems
  - Synthesis
    - "How do I create *x*?"
    - Ex: automated planning, scheduling, design
    - Also called construction problems
  - Can have elements of both

Kinds of problems

- Decomposable or not?
  - Can problem be decomposed into independent subproblems?
  - If so, maybe amenable to divide-and-conquer techniques
    - Suppose solving problem is $\mathcal{O}(2^n)$ as are the subproblems
    - Break into m pieces
    - Time is now $\mathcal{O}(m2^{n/m})$



  - Some problems: nearly-decomposable

Kinds of problems

- Problems can vary based on step characteristics:
  - Steps are ignorable?
  - Steps are reversible?
  - Steps are recoverable?
  - Steps are irrecoverable?

Kinds of problems

- Problems can vary by domain predictability
  - Predictability of world, agent's own actions
  - Unpredictability ⇐ uncertainty in knowledge, uncertainty in percept, lack of knowledge, stochastic nature of world
  - Frame problem
  - Effect on solution: suboptimal solutions, no/wrong solution, increased time/space complexity
  - Predictable domains ⇒ easier to solve, planning is more useful
    - Toy domains: 8-puzzle, water jug problem,…
    - Games: TTT, chess, …
    - Real world

Kinds of problems

- Problems vary by kind of solution needed
- Absolute solution: find a solution or not
- Optimization problems
  - Find cheapest, shortest, etc. solution
  - Almost always much harder than just finding solution – often NP-hard
  - Optimize over:
    - Result: find best of all goal states, find best of all paths to goal state
    - Solution itself: take least amount of time/effort to find solution

Kinds of problems

- Based on role of knowledge:
  - to generate states
  - to recognize solution
  - to constrain search
- Ex:
  - Find any path to goal in graph: no knowledge needed
  - Find best path to goal in graph: knowledge of weights, heuristics to predict future weights, etc.
  - Planning, robot control: moderate amount of knowledge
  - NLP, medical diagnosis: large amount of knowledge

Kinds of problems

- Whether or not other agents are present
  - Multiagent systems, human user, etc.
- Other agents may
  - be source of uncertainty
  - may actively undo/hinder your work

Kinds of problems

- Single vs multi-state problems
  - State → state
  - {state, state,…} → {state, state,…}
  - Due to uncertainty, ⇒ harder problem
  - Could build contingencies into solution, or interleave action & search
  - Exploration problem – worst of the worst

Search and agents

Where does search fit into an agent?

- Could be used by agent program (e.g., to find a route)
- Could be the agent program:
  - Find best next state, return action to get there
  - Find complete path, return next action each time called

Generic search-based agent

```
function SEARCH-AGENT(p)
    Inputs: a percept p
    Returns: an action
    Static: s: action sequence, initially nil
            state: description of current world state
            g: goal, initially nil
            problem: problem formulation
    state ← UPDATE-STATE(state,p)
    if s = nil then
        g ← FORMULATE-GOAL(state)
        problem ← FORMULATE-PROBLEM(state,g)
        s ← SEARCH(problem)
    action← RECOMMENDATION(s, state)
    s ← REMAINDER(s,state)
    return action
```

Reflex agents and search

- One view: no search at all
- Another: a purely local search
  - Find next best state, return it
  - E.g., Roomba, possibly ants,…
- Pure reflex agents: only current percept
- Model-based reflex agents: percept + memory
- Search occurs in the world itself

Goal-based agents

- Use search to find how to achieve goals
- Search usually:
  - > local
  - "thinks" about the search – not search in real world

Utility-based agents

- Search to find best way to achieve goal
- Optimizers
- May want optimal solution and/or optimal search
- Also "think" about search rather than search in world

Learning agents

- Perform meta-level search
- Start state: current performance module's knowledge
- Goal state: performance module's knowledge that can best achieve agent's goals
- Transitions: tweaks to knowledge
- E.g., reinforcement learner – adjust state ↔ expected reward function (Q-learning)
- E.g., neural networks – adjust weights between nodes down a gradient in an error function