Reflex agent

- have a current recept (maybe memory)
- makes decisions without considering the consequences of their actions

may be rational in cases where you don't need to bother with consequences

Planning agents

- ask "what if", makes hypothesises
- needs a model of how the world works and how it reacts
- must have a goal (test)

optimal vs complete planning

- optimal: achieve goal in minimal cost
- complete: when a solution exist, you find it

planning vs replanning

- planning: entirely ahead of time
- replanning: plans step-by-step

Search problems

they consist of:

objective: cast real life problems to fit this interface

- a state space
- a successor function (with action, cost)
- start state and a goal test

A solution is a sequence of actions (a plan) with transforms the start state to a goal state

Search problems are models.

STATE SPACES

what is in a state space?

A world state includes every last detail of the environment, a search state uses only the necessary details for planning (abstraction)

problem: pathing
- state: (x,y) location
- action: NSEW
- successor: update location only
- goal test: is (x,y) = END

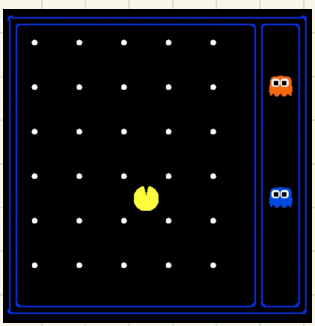
problem: eat all dots
- state: {(x,y), dot booleans}
- action: NSEW
- successor: update location and possibly a dot boolean
- goal: all dots false

state space sizes?

- world state:
  - agent positions: 120 (10x12 grid)
  - food count: 30
  - ghost positions: 12
  - agent facing: NSEW

world state: $120 \times 2^{30} \times 12 \times 12 \times 4$

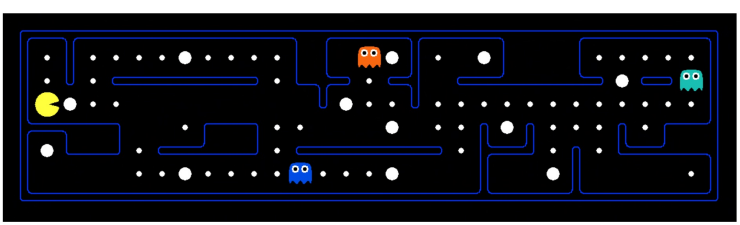
could be or not be food in a place



eat all dots: 120×2^{30} don't care about ghosts or with direction the agent is facing

pathing: 120

as we can se we need much more calculations for eating than simply path



- Problem: eat all dots while keeping the ghosts perma-scared

power pellet remaining locations
timer ghosts
regular pellets (dot booleans)
pacman (x,y)
ghost locations (depends on the world)

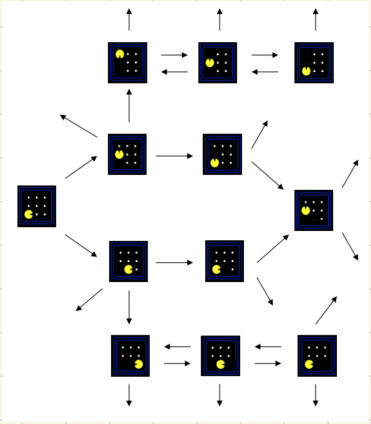
state space graphs

mathematical rappresentation of a search problem

- nodes -> abstracted world configurations
- arcs -> successors
- the goal is a set of goal nodes

each state only occurs only once

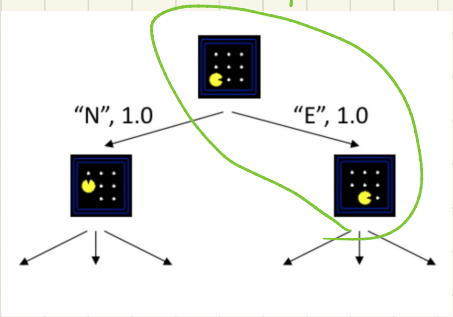
we can rarely use this because they are too big for memory, but they are nice



see as a path

Search trees

- a "what if" tree of plans and their outcomes
- the start is the root
- children are successors
- nodes show states but they are plans to reach those states



for most problems we won't be able to actually build the whole tree

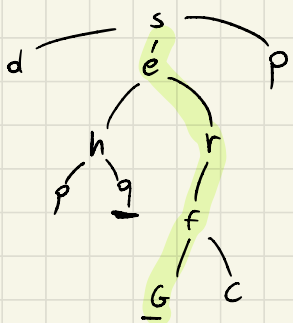
General tree search

treeSearch(problem, strategy) returns a solution or failure
- initialize the search tree with the initial state of the problem

loop do:
- if there are no candidates for expansion return failure
- choose a leaf for expansion according to strategy
- if node contains a goal state return the corresponding solution
  else expand and add the resulting nodes to the search tree

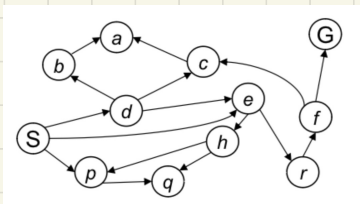
example:

search tree



code

~~s~~
s → d
~~s → e~~
s → p
~~s → e → h~~
~~s → e → r~~
s → e → h → p
~~s → e → h → q~~
~~s → e → r → f~~
s → e → r → f → c
s → e → r → f → g

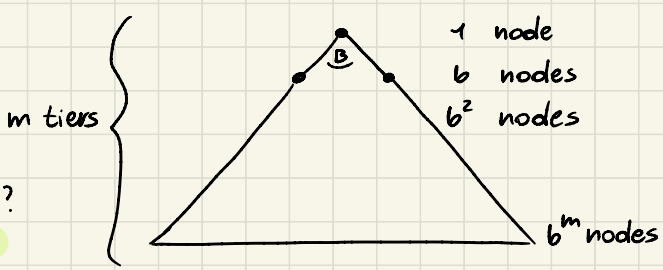


← goal

Properties of search algorithms

- complete: guaranteed to find a solution if there is one?
- optimal: guaranteed to find the least cost path
- time complexity?
- space complexity?

let's analyze with a cartoon of search tree
- b branching factor
- m maximum lenght



number of nodes in total?

$1 + b + b^2 + \dots b^m = O(b^m)$

Uninformed search

Depht first search

strategy: expand a deepest node first

implement via a LIFO stack

- what nodes does DFS search?

  some left prefix of the tree

  could process the whole tree

  so $O(b^m)$

- How much space does the fringe (frontiera) take?

  only has siblings on path to root, so $O(bm)$

- Is it complete?

  m could be infinite, so we must prevent cycles

- Is it optimal?

  no, it finds the "leftmost" solution not considering depht or cost

Breadht first search

- what nodes does BFS expand?

  all nodes above the shallowest solution (level s)

  so $O(b^s)$

- how much space does the fringe take?

  $O(b^s)$

- is it complete?

  s must be finite so yes

- is it optimal?

  only if all costs = 1

BFS VS DFS

- DFS takes less memory
- BFS finds shorter path, doesn't waste time exploring whole tree

would be nice to combine the advantages

Iterative deepening

have DFS's space advantage with BFS's time/shallow solution advantages

- run DFS with depth limit 1, if no solution...
- " " limit 2

iterate like this

there is some waste happening but it's not that bad

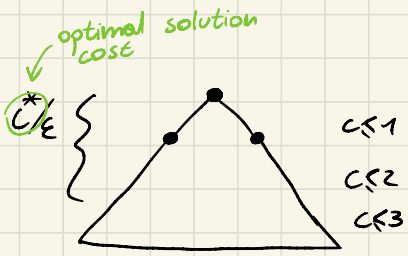
Cost sensitive search

BFS does not take in account the cost of the path

Uniform cost search

strategy: expand to cheapest node first

fringe is a priority queue (priority=cumulative cost)



- what nodes does UCS expand?

  all nodes with cost < C^*, the arcs cost at least ε

  then the "effective depth" is roughly C^*/ε

  takes time $O(b^{C^*/\varepsilon})$

- how much space fringe?

  roughly the last tier so $O(b^{C^*/\varepsilon})$

- Is it complete?

  assuming best sol. has finite cost and minimum arc cost yes

- Is it optimal?

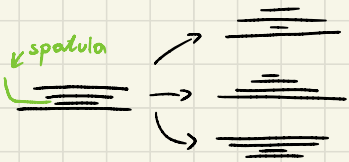
  yes!

Issues

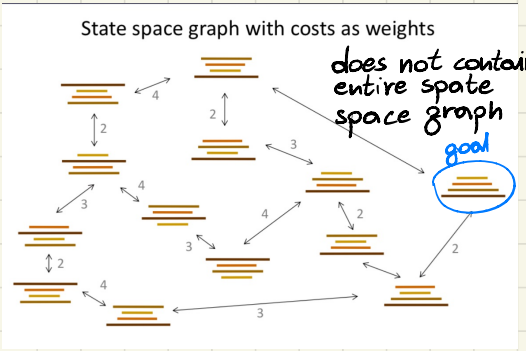
- increasing cost
- explores options in every direction
- no indication about goal location

The only difference for algorithms so far has been fringe strategies

Example: pancake flipping



i can flip pancakes above the spatula tip



how many elements?

$4 \cdot 3 \cdot 2 \cdot 1 = 24$

↑ choices i can make per level

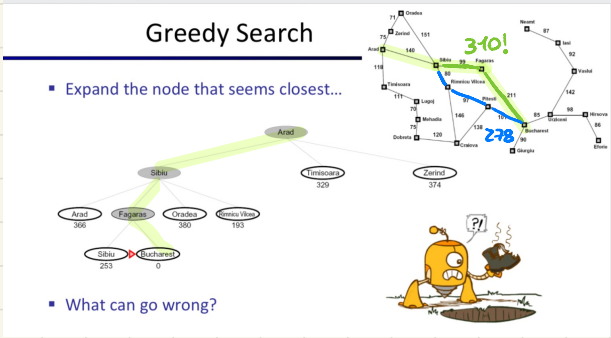
I can the choose a strategy to search the tree.

Informed Search

heuristic -> function that estimates how close a state is to a goal
designed for a particular search problem

greedy search

pick from the fringe the node that seems closest to the goal.



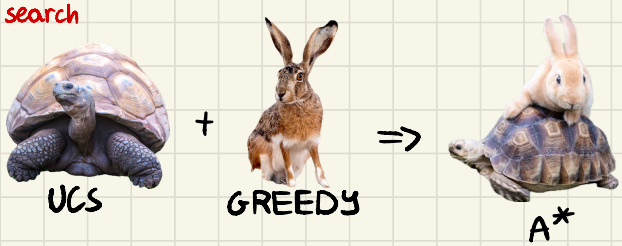
Heuristic: estimate of distance to nearest goal each state

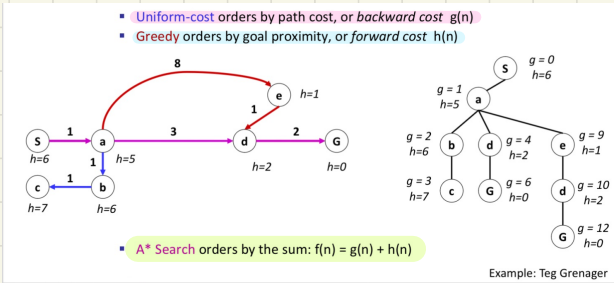
we did not find the shortest path, you just go for the one wich looks the best

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

worst case: behaves like a badly guided DFS

A* search





When should it terminate?
Not when we enque a goal, but when we deque it

it's not guaranteed to be optimal

Admissable Heuristics

Inadmissable (pessimistic) -> heuristics break optimality by blocking good plans on the fringe

Admissable (optimistic) -> heuristics slow down bad plans but never outweight true cost

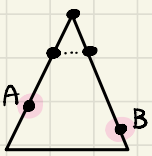
formal definition:

> an heuristic h is admissable if:
>
> $0 \leq h(n) \leq h^*(n)$ — the true cost to a nearest goal

Optimality of A* tree search

assume:

- A is optimal goal
- B is sub-optimal goal
- h is admissable



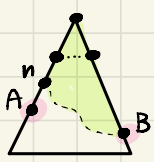
claim:

A will exit the fringe before B

proof

imagine B on the fringe

- some ancestor n of A is on the fringe too (maybe A)
- claim: n will expand before B



1) $f^*(n) \leq f(A)$ — $f^*(n) = g(n) + h(n)$; $f(n) \leq g(A)$ for h to be admissable

2) $f(A) < f(B)$ — $g(A) < g(B)$ because B is sub-optimal

3) n expands before B — $f(n) \leq f(A) < f(B)$

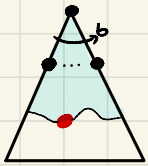
⇓

all ancestors of A expand before B

⇓

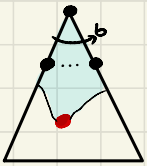
A* search is optimal

UCS



- expands equally in all directions

A*



- expands mainly toward the goal but does hedge its bets for optimality

---

Creating admissable heuristics

- most of the work of solving hard search problems optimally is coming up with admissable heuristics
- often admissable heuristics are solutions to relaxed problems, where new actions are available
- inadmissible heuristics are often useful too

we cannot use the actual cost as heuristic because then we would have already solved the problem.
The closer the estimate to the actual cost, the less nodes I'll have to expand, but usually do more work to compute the heuristic.

Trivial Heuristics

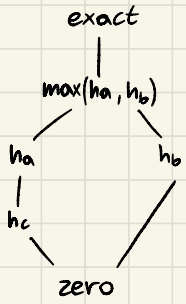
dominance: $h_a \geq h_c$ if $\forall n : h_a(n) \geq h_c(n)$

(semi reticolo, poset)

Heuristics form a semi-lattice

the max of admissible heuristic is admissable

$h(n) = \max(h_a(n), h_b(n))$



GRAPH SEARCH

trees can be redundant

in graph search ya never expand a state twice

implementation:

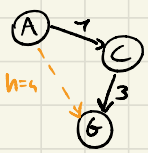
- tree search + set of expanded states (closed set) ← not list bcz it's faster
- expand the search tree node-by-node but check the set before
- if new add to set, if not skip

it's still complete, I only exclude things I have already found

If the heuristic is wrong it can be not optimal

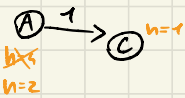
costincency of heuristics

whe have to expand what we have seen before with admissibility



admissability : heuristic cost ≤ actual cost to goal

h(a) ≤ actual cost from A to G



consistency: heuristic "arc" cost ≤ actual cost of each arc

$h(A) - h(C) \leq cost(A \text{ to } C)$

consequences

⇓

the f value never decreses along a path

$h(a) \leq cost(A \text{ to } c) + h(c)$

let's add previous cost

$f(A) = g(a) + h(a) \leq g(A) + cost(A \text{ to } c) + h(c) = f(c)$

⇓

$f(A) \leq f(c)$

f cost goes up while continuing