State Space Search

- A deliberative phase, in which we plan an action sequence, should generally precede an execution phase.
  Why?

- Requires 'mental' representation of
  - initial state of the world
  - action repertoire of the execution agent, i.e. operations/rules/productions specifying preconditions and effects
  - termination test
    e.g. explicit set of possible goal states or a property to be checked for

State space search, cont'd

- Initial state and operators define **state space** of the problem.

- We must search for a *path* in the state space leading from initial state to a goal state, and return the sequence of actions on that path.

- So we have a graph search problem.
  BUT,
  - our graph is implicit in the operators
  - and may be too large to make explicit

Toy example of a state space: the 8-puzzle

—sliding 8 numbered tiles around a 3×3 grid

- Representation of states: 3×3 array of numbers

- Initial state, e.g.:

2	8	3
1	6	4
7		5

- Operators: *if blank is not leftmost, move it left by 1*, etc.

- Goal state, e.g.:

1	2	3
8		4
7	6	5

— this defines a space of $9!$ $(= 362,880)$ states

Toy example of a state space: the water jugs problem

— a 4-gallon and 3-gallon jug with no measuring markers, and a tap
— must get exactly 2 gallons into the 4-gallon jug

- Representation of states: a pair of integers (x, y)
  x is the amount of water in the 4-gallon jug,
  $x \in \{0, 1, 2, 3, 4\}$
  y is the amount of water in the 3-gallon jug,
  $y \in \{0, 1, 2, 3\}$

- Initial state: $(0, 0)$

- Operators:

  1. if $x < 4$ then $(4, y)$
  2. if $y < 3$ then $(x, 3)$
  3. if $x > 0$ then $(0, y)$
  4. if $y > 0$ then $(x, 0)$
  5. if $x + y \geq 4 \wedge y > 0$ then $(4, y - (4 - x))$
  6. if $x + y \geq 3 \wedge x > 0$ then $(x - (3 - y), 3)$
  7. if $x + y \leq 4 \wedge y > 0$ then $(x + y, 0)$
  8. if $x + y \leq 3 \wedge x > 0$ then $(0, x + y)$

- Goal states: $(2, n)$

— this defines a space of only 20 states

Real examples of state spaces

- Route planning
- VLSI layout
- Cargo layout
- Automatic assembly

Searching the state space

- If the current state is not a goal state, *'expand'* the current state, i.e. *generate* its *successor* states
- One successor becomes new current state. Others must be kept in case we want to come back to them
  — keep them on an **agenda**
  (list of unexplored options)
- *Search strategy* (or *control strategy*) determines which state to expand next.

Pseudocode

```
function stateSpaceSearch
         returns a solution path or failure


agenda := list comprising only the initial
          state

while agenda is not empty do

   remove node from front of agenda,
   call it currentNode,
   if currentNode satisfies goal test then
      return path of actions that lead to it
   else
      compute successor of currentNode,
      *place them all on the agenda*
   end while

return failure
```

Search tree

- The algorithm builds a **search tree**
  — distinguish search tree from state space
  e.g. some search strategies may leave parts of state space unexplored
  e.g. some search strategies may re-explore parts of state space

- Avoiding re-exploration:
  When placing successors onto agenda

  - discard successor if same as currentNode
  - discard successor if same as another node on that path
  - discard successor if same as any other previously generated node

  Compare for effectiveness, and time/space overhead

Evaluating a search strategy

- **Soundness**: never claim something is a solution when it isn't

- **Completeness**: never claim there is no solutioin when there is one

- **Time complexity**: how long does it take to find a solution?

- **Space complexity**: how much memory is needed to perfom search?

- **Optimality**: is the highest quality solution found?
  — uses *path cost function*, g
  — $g(n)$ is cost of path to node n (sum of costs of actions on the path)

Roadmap

(1) Uninformed / undirected / blind search
(no guidance on what to expand next)

(a) $g(n) =$ depth(n)
(shallowest solution is optimal)
breadth-first
depth-first
depth-bounded depth-first
iterative deepening

(b) $g(n) =$ arbitrary cost function
(cheapest path is optimal)
least-cost search

(2) Informed / directed / heuristic search
best-first
A^*

Breadth-first search

- Treat agenda as a queue
  — successors placed at back
  $\Rightarrow$ all nodes at depth d in search tree are explored before any at $d + 1$

Depth-first search

- Treat agenda as a stack
  — successors placed at front
  $\Rightarrow$ always expands one of the nodes at the deepest level of the tree (only expands at a shallower level if it has reached a dead-end)
  — can implement recursively. Then, agenda is stored implicitly in the local state of each invocation of the calling stack. Gives (chronological) backtracking.

Depth-bounded search

- Like depth-first search but discards paths whose length has reached some limit, l

Iterative deepening

- Like depth-bounded search, but with successively greater depth limits

```
for l = 0 to infinity do
    depth-bounded search with limit l
```

Least-cost search

- Treat agenda as priority-ordered queue
  — agenda is ordered by cost
  — least cost path is expanded

Comparison

b	if each state has b successors, b is the branching facto
d	the depth of the solution
m	maximum depth of search tree
l	depth-bound

	sound	complete	time	space	optima
breadth-first					
depth-first					
depth-bounded					
it. deepening					
least-cost					

- **Question**: Consider using a breadth-first strategy on the 8-tile problem. Will switching to depth-first search increase or decrease the size of the state space?

- To reduce size of state space, try
  - more abstract state representation
  - eliminate useless operators
  - tighten operator preconditions
  - make operator effects more far-reaching

  (Even some work on doing these things automatically —using machine learning techniques.)

- To reduce size of search tree
  - If not too expensive,
    avoid re-exploration
    (consider canonical representations and data structures such as hash tables to speed up the checking)
  - Where possible use the more focussed search strategies,
    e.g. depth-first, depth-bounded, iterative deepening
    e.g. heuristic search...
    i.e. move away from ‘weak methods’...

Heuristic search

- To focus search towards the goals use an **evaluation function** to choose which node to expand next
  — agenda is a priority-ordered queue, ordered by the evaluation function

Best-first search

- Evaluation function is an **heuristic function**, h
  $h(n) =$ estimated cost of cheapest path from n to a goal

- E.g. for 8-puzzle
  (1) $h(n) =$ no. of tiles out of place
  (2) $h(n) =$ sum of for each tile Manhattan distance between tile and its goal position

A^* search

- Best-first is sound, incomplete, time and space $O(b^m)$ and not optimal

- Can we combine strengths of least-cost (complete, optimal) with those of best-first (focus in 'typical' cases)?

- Order agenda using evaluation function f that combines g and h
  $f(n) = g(n) + h(n)$

- Bringing in g restores completeness

- If h is an **admissible heuristic** (one that never overestimates), then this restores optimality

- The two ideas together give the A^* search strategy

Closing remarks

- If computing h is expensive, might be better to search!

- Even with A^*, space may be a problem so
  — use heuristics not just to order but to prune (loses completeness and optimality)

- Heuristics
  - may be human-designed (hard!)
  - may be discovered automatically using machine learning techniques