COMPSCI4004/COMPSCI5087 AI (H/M)

Week 3: Solving Problems by Searching

Debasis Ganguly [1]

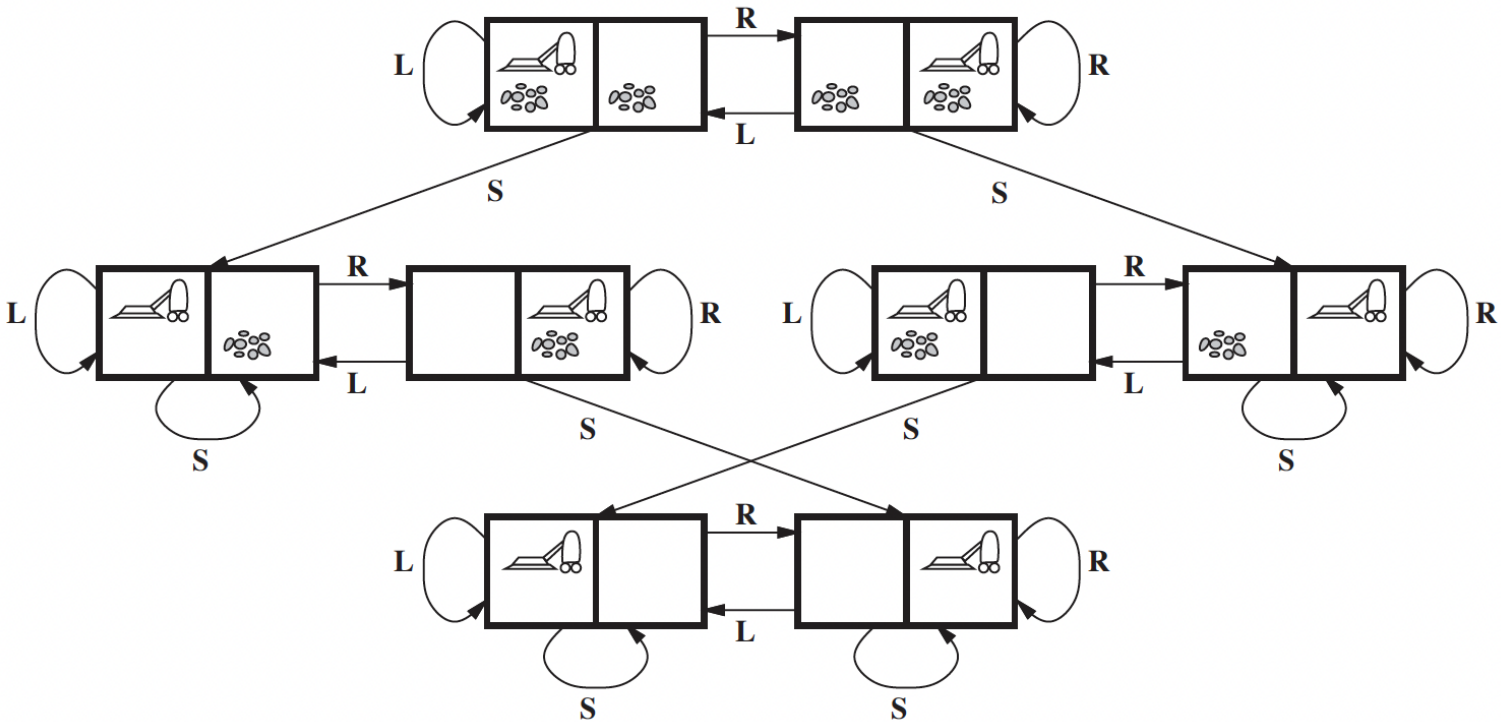
[1]University of Glasgow, Glasgow, UK

November 11, 2025

Road Map

- Last week:
  - Looked at different agent types.
  - Recollect: Goal-based agents.
- This week, we will study:
  - A generic **state-transition based graph framework** to model a common class of problems.
  - Discuss **search strategies on these graphs** to solve the problems.
  - Uninformed Search $\rightarrow$ Informed Search
  - Designing Heuristics Functions
  - Stochastic search for Deterministic problems.
  - Introduce the coursework (which will be released this week).

State-Space



- The simple vacuum world.
- The state-space is **Discrete** and **Deterministic**.

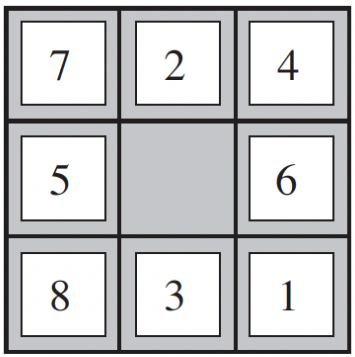
State-Space

- **State**: A state is a particular snapshot of the environment.
- **Initial State**: The point from where we start our 'search' for the solution (the goal state).
- **Actions**: Possible ways to transition across states.
- **Transition Model**: The entire graph. May need to list some exceptions like 'sucking dirt from a clean room has no effect' (can you spot this in the graph?).
- **Path cost**: All actions cost equally (unweighted edges). Is this accurate for the vacuum world?

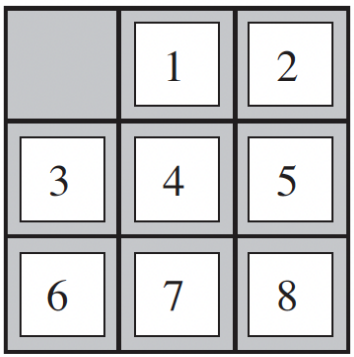
Menti Quiz

Join at menti.com using code "3334 0792".

Some more examples - 8 Puzzle



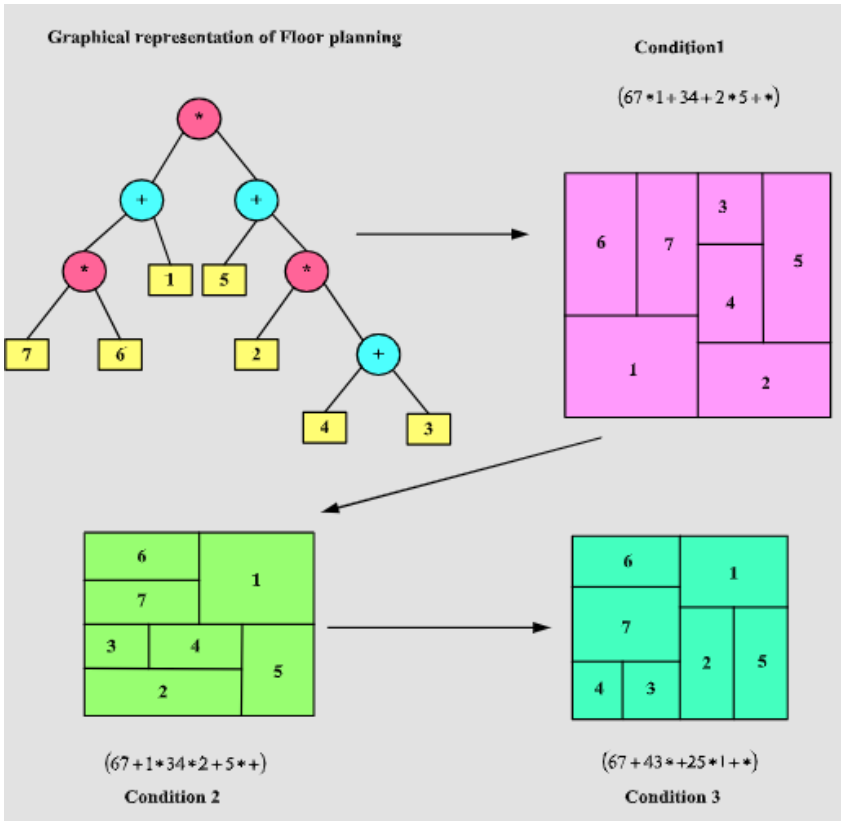
Start State



Goal State

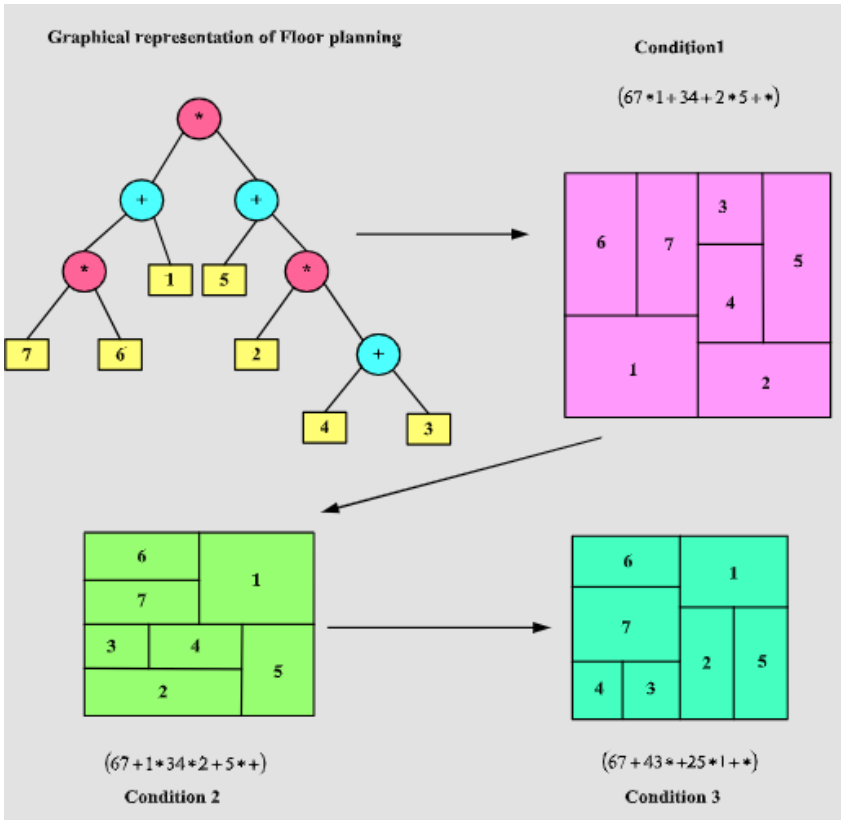
- **Action**: Left, Right, Up or Down. 'Left' could mean that you swap the tile to the 'right' with the blank tile; this means 'physically moving' the adjacent tile to its 'left'.
- **Transition Model**: Note that some actions can't be taken from some states, e.g., "`right`" can't be applied to the 'Goal state' shown.
- **Path cost**: Uniform? Is this realistic?

Some more examples - VLSI Floor Planning



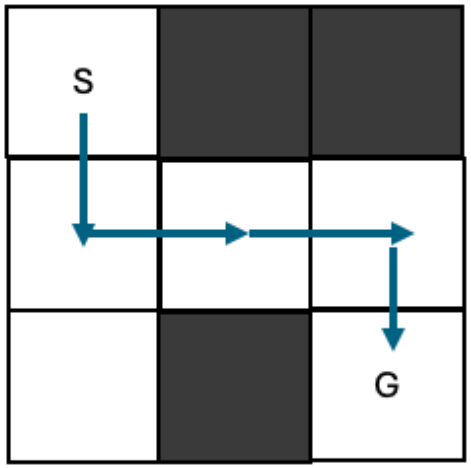
- **State**: A partial floor plan with blocks placed on a grid.
- **Action**: Place a new block, remove a block, transpose a block and so on.
- **Transition Model**: Captures **constraints** like some blocks **need to be adjacent** to others.
- **Path cost**: Uniform?

Some more examples - VLSI Floor Planning



- **State**: A partial floor plan with blocks placed on a grid.
- **Action**: Place a new block, remove a block, transpose a block and so on.
- **Transition Model**: Captures **constraints** like some blocks **need to be adjacent** to others.
- **Path cost**: Uniform?
- May be easier to try out transposing a block as compared to inserting or removing a block?

Maze navigation



- **State**: A partial path navigated through a maze.
- **Action**: Move *left*, *right*, *up* or *down* from a cell if there are no obstacles in the target cell.
- **Transition Model**: Captures **constraints** like obstacles.
- **Path cost**: Uniform? Real life: Need more effort to change a robot's direction than to keep on heading along the same direction?

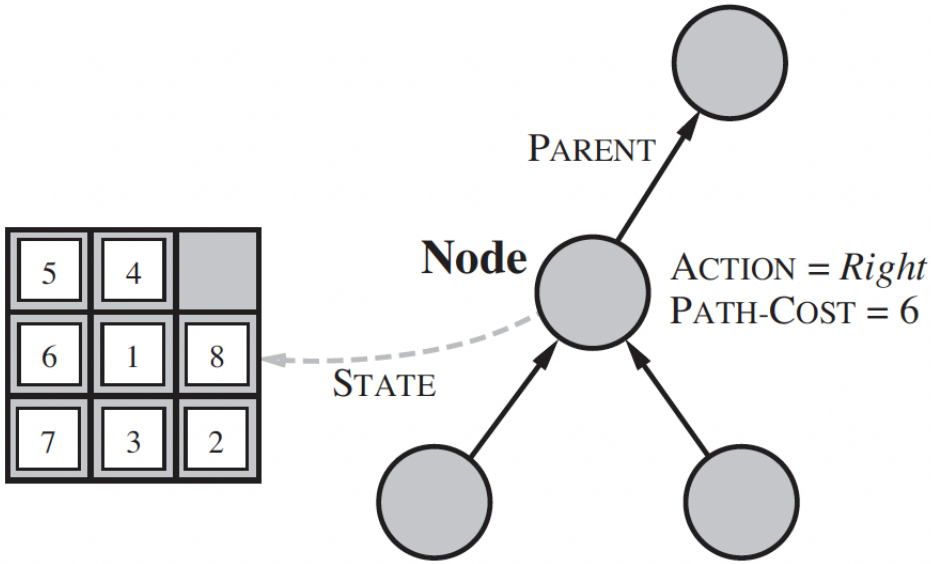
State spaces may not always be finite!

- Generate a target integer with a sequence of $x!$, $\sqrt{x}$ and $\lfloor x \rfloor$ functions, the only value of x being allowed is 4.
- Example solution:
  - $\lfloor \sqrt{\sqrt{\sqrt{\sqrt{\sqrt{(4!)!}}}} \rfloor = \underbrace{5}_{\text{target integer}}$
- What is the state space?
- What is the action space for this problem?

Finding a solution ≡ Searching a path to Goal

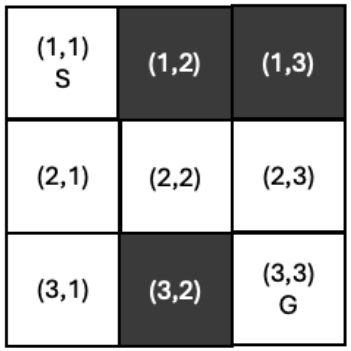
1. Root expanded
2. 1-level successors of root node expanded
3. 2-level successors of root node expanded
4. Note: Realise that we don't need to store the entire graph in memory because we can't do it if the state space is massive.
   - Think about an example state space that is massive.

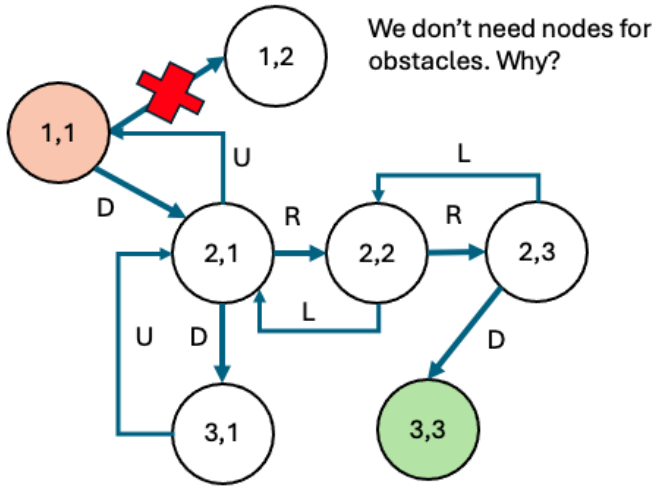
Nodes ≡ States + Book keeping data structures



- Each state is represented by a unique node.
- A node however contains additional generic information.
  - Links to children and parent (helps backtracking)
  - Edge information, e.g., the cost of the edge and the action type.

Transition Graph for Maze Navigation



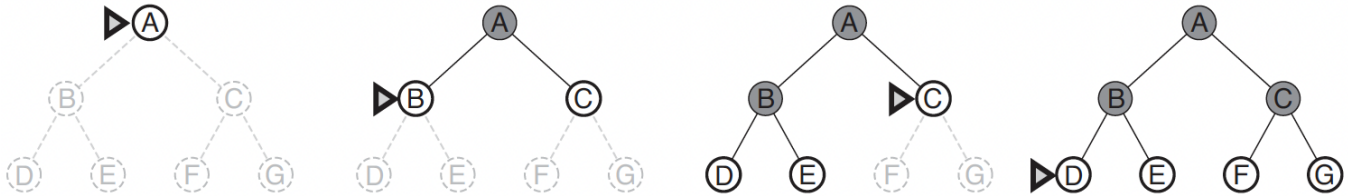


- Find a path (without loops) from the start to one the goal nodes in this graph.
  - In general, there can be multiple goal nodes.

Uninformed (Blind) Search Strategies

- Search strategies that don't have any problem-specific information.
- Capable only of expanding the search tree (traversing the transition graph by generating successors).
- Can only distinguish a goal state from a non-goal one.
- We will study three algorithms under this category:
  - Depth-first Search (DFS) – LIFO
  - Breadth-first Search (BFS) – FIFO
  - Iterative Depth First Search (IDFS).

Breadth First Search



- From a node, generate children (if not already generated before) and add them to a FIFO queue.
- Remove from queue, and repeat unless goal state is reached.
- Hypothesis: Exhaust all k-sequence length action consequences before exploring $(k+1)$-sequence length ones.
- Pros:
  - Guaranteed to find solution if one exists.
  - Even **works on infinite state spaces** (can you see why?)

Breadth First Search

Space Complexity

- Let branching factor (average number of actions that can be taken from a state) be b.
  - For 8-puzzle and maze navigation: $b = 4$ (why?)
  - What about a VLSI floor planner?
- Number of children at depth $k = O(b^k)$.

Time Complexity

- Total #states visited (till depth k) $= b^0 + b^1 + \ldots b^k = O(b^{k+1})$.
  - Is BFS good for 8-puzzle?
  - What about a VLSI floor planner?

Breadth First Search

Space Complexity

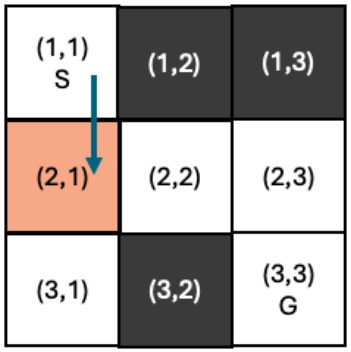
- Let branching factor (average number of actions that can be taken from a state) be b.
  - For 8-puzzle and maze navigation: $b = 4$ (why?)
  - What about a VLSI floor planner?
- Number of children at depth $k = O(b^k)$.

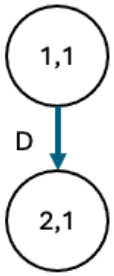
Time Complexity

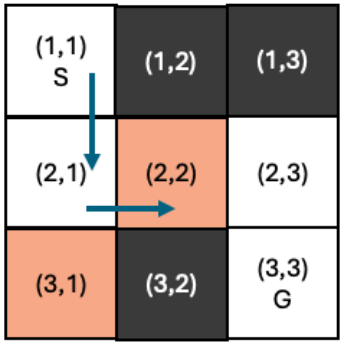
- Total #states visited (till depth k) $= b^0 + b^1 + \ldots b^k = O(b^{k+1})$.
  - Is BFS good for 8-puzzle?
  - What about a VLSI floor planner?
- BFS is excessively slow if the solution occurs somewhere deeper down the search space tree, or in other words you need a relatively **large number of actions to get to the goal state**.

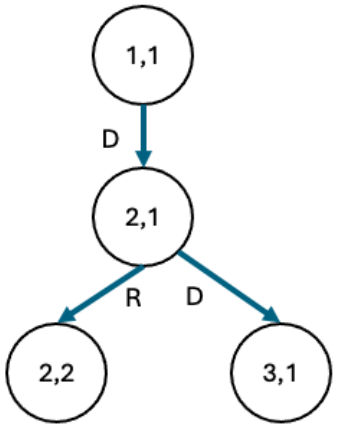
BFS for the Toy Maze Example

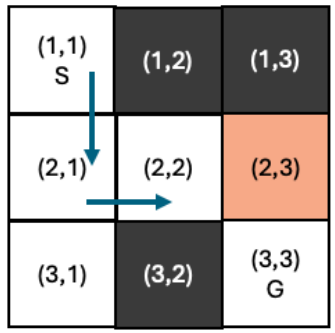
- Actions considered in order: Right, Down, Left, Up.

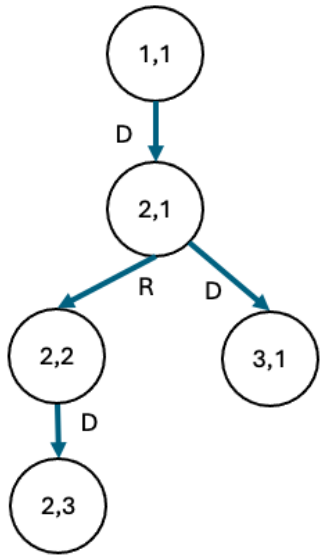








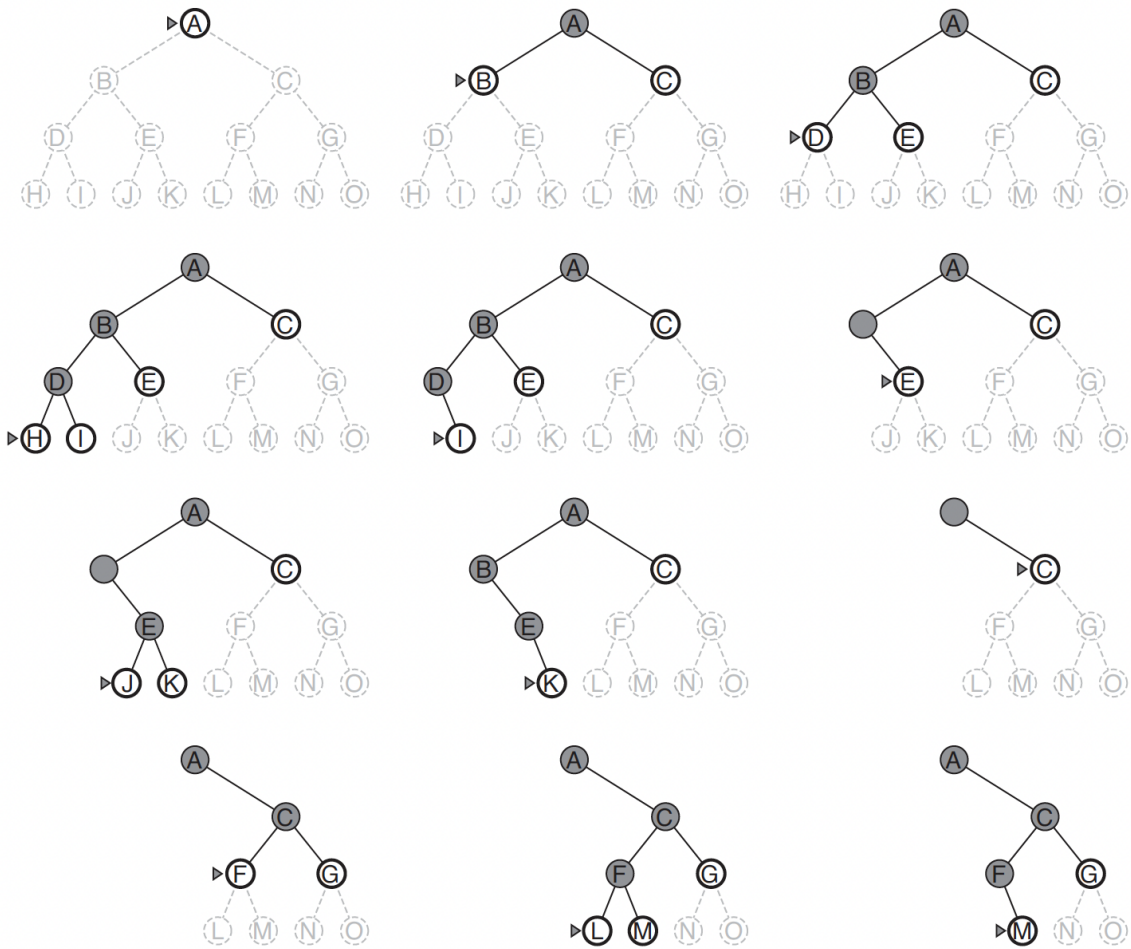




Depth First Search

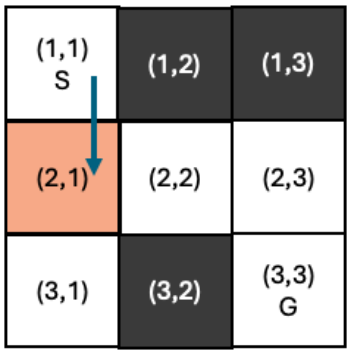
- From a node, generate children (if not already generated before) and add them to a LIFO queue (stack).
- Remove from queue, and repeat unless goal state is reached.
- Hypothesis: Continue expanding along a particular direction.
- Pros:
  - Can find a goal state quicker if the path to the goal state is considerably large (**alleviates the limitation of BFS**).
- Cons:
  - No guarantee to find a solution for infinite state spaces (Why?).

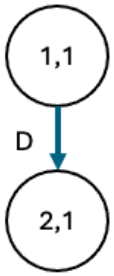
Depth First Search

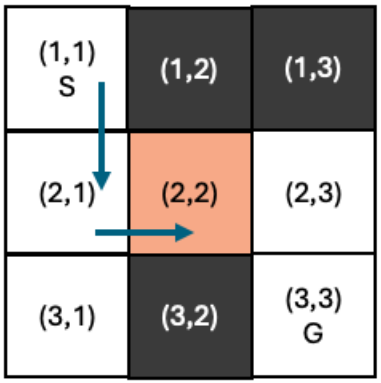


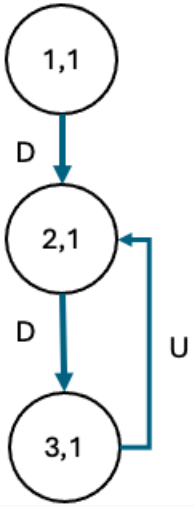
DFS for the Toy Maze Example

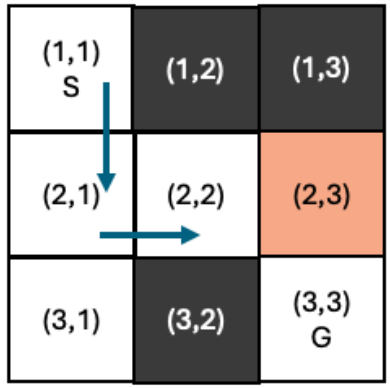
- Actions considered in order: Right, Down, Left, Up.

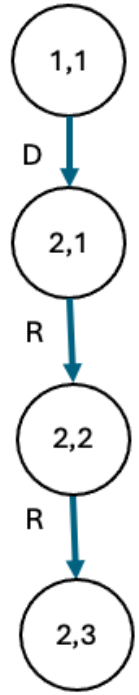




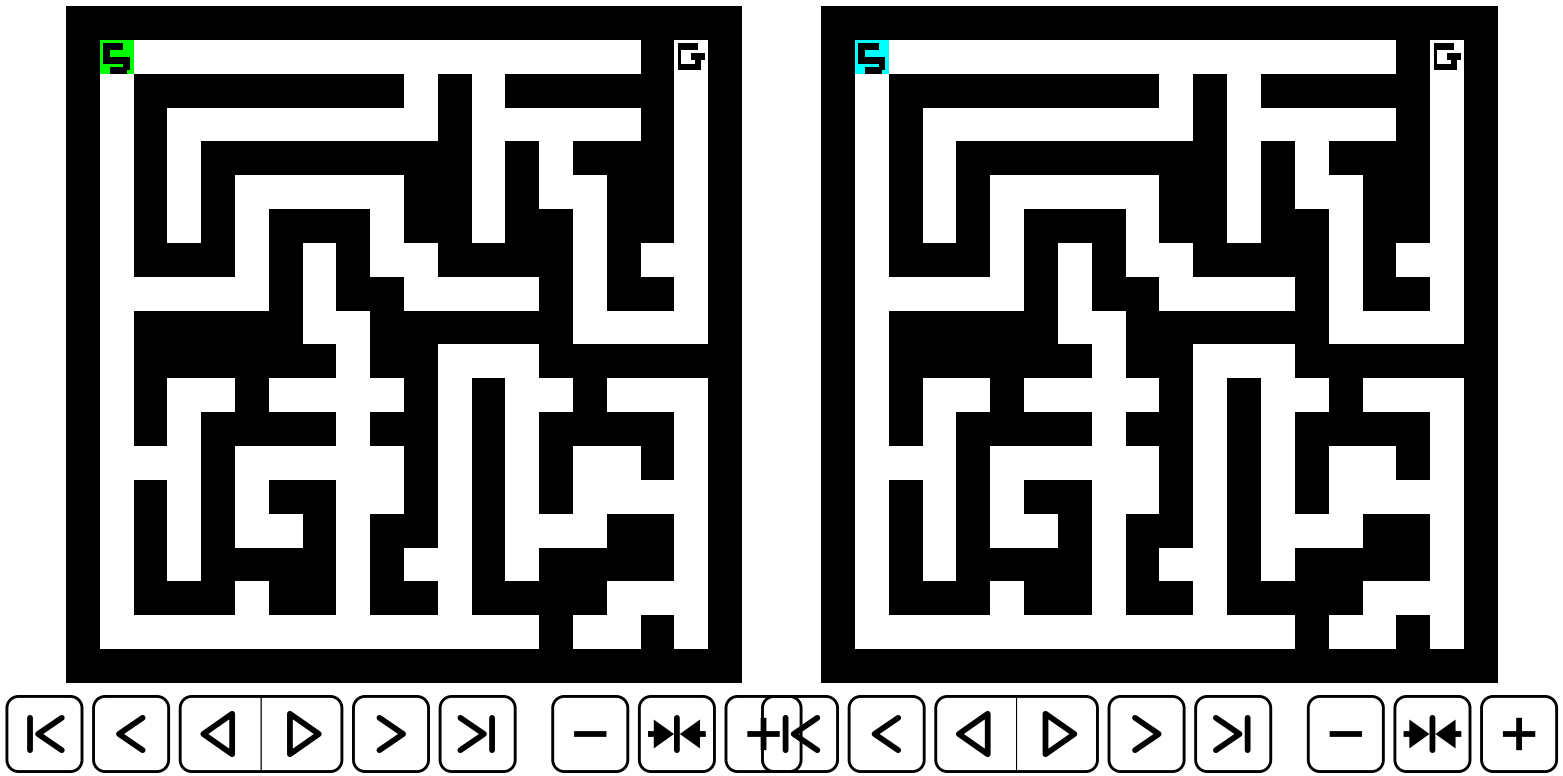








DFS/BFS Animation



Can we do better Uninformed Search?

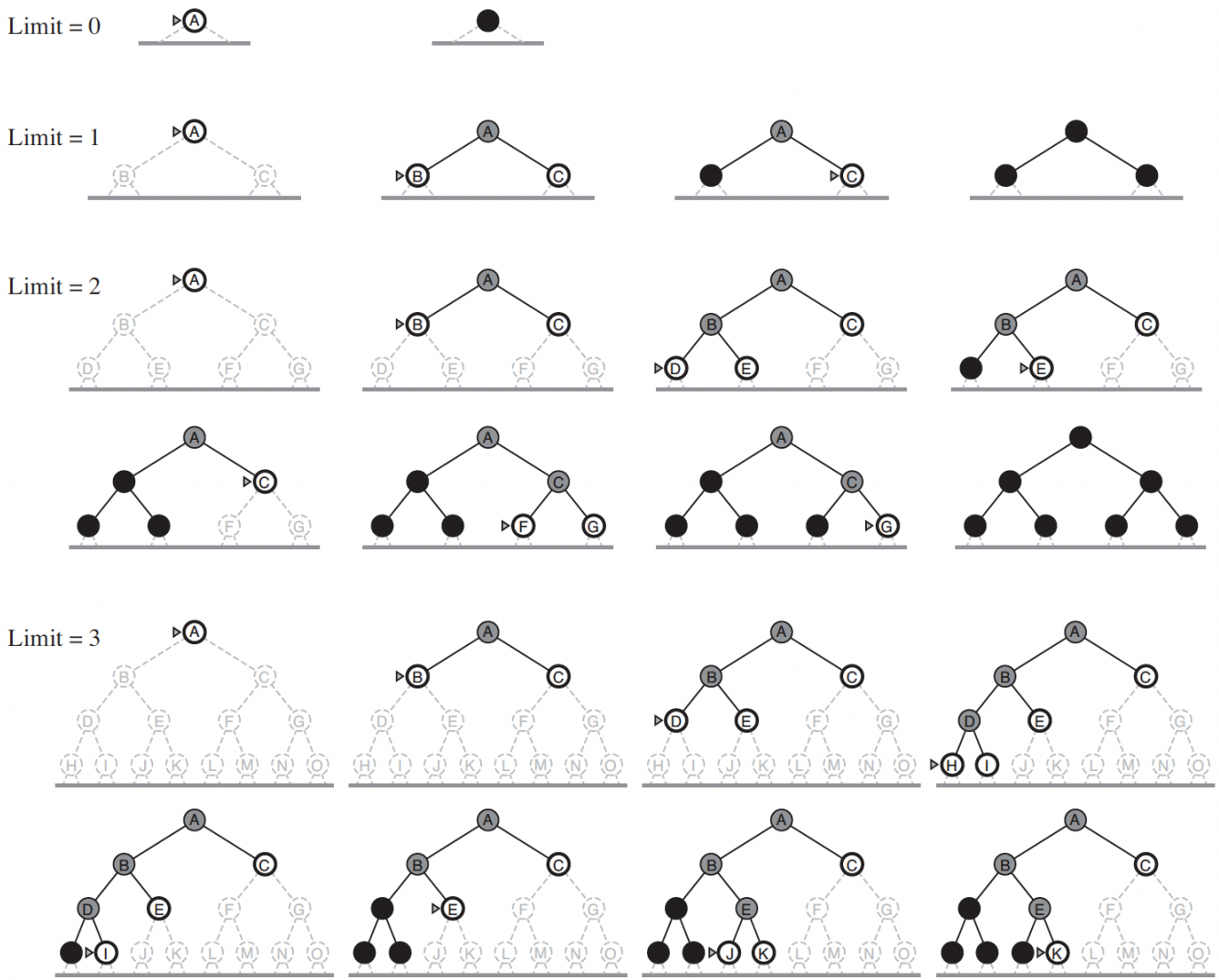
Depth Limited Search

- Run a DFS to a maximum depth, say l.
- Intuition: Forces to backtrack and explore along other paths.
- Doesn't work if the shallowest goal's depth (d) is beyond the depth limit, i.e., $l < d$.
- What could be a reasonable depth limit for 8-puzzle?

Iterative Depth First Search (IDFS)

- Run a DFS to a maximum depth, say l.
- Repeat with $l \leftarrow l + 1$.
- Not sensitive to the choice of l.
- But has to repeat the exploration every time.

IDFS



Space Complexity Comparisons

Number of states required to be stored

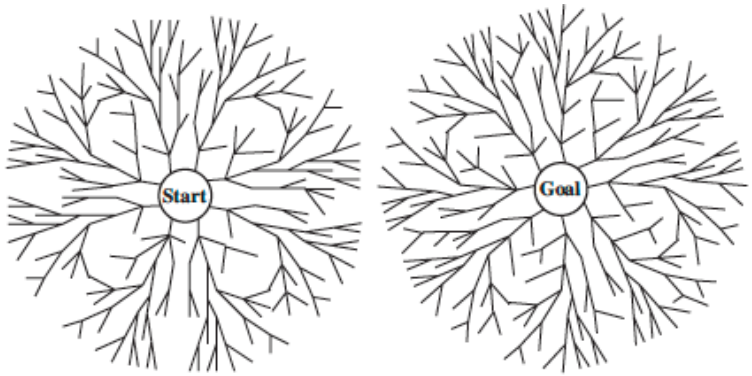
- BFS: $b^0 + b^1 + \ldots + b^d = O(b^{d+1})$
- DFS: $O(bd)$ (only required to save the parent nodes along a path for backtracking)

Space Complexity Comparisons

Number of states required to be stored

- BFS: $b^0 + b^1 + \ldots + b^d = O(b^{d+1})$
- DFS: $O(bd)$ (only required to save the parent nodes along a path for backtracking)
- IDFS: Same as DFS (why?)
  - What about the number of times a node is generated due to the multiple DFS calls in IDFS?
  - $(d)b + (d-1)b^2 + \ldots + \underbrace{(d-i)b^{i-1}}_{\text{(\#times a node at i-th depth is generated)\#nodes at depth i}} + \ldots (1)b^d = O(b^{d+1})$.

Bidirectional Search

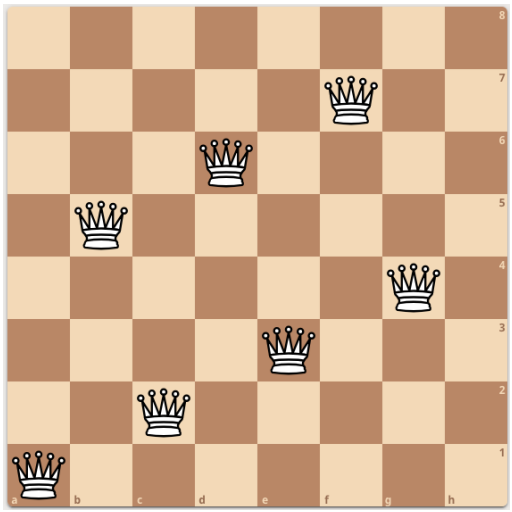


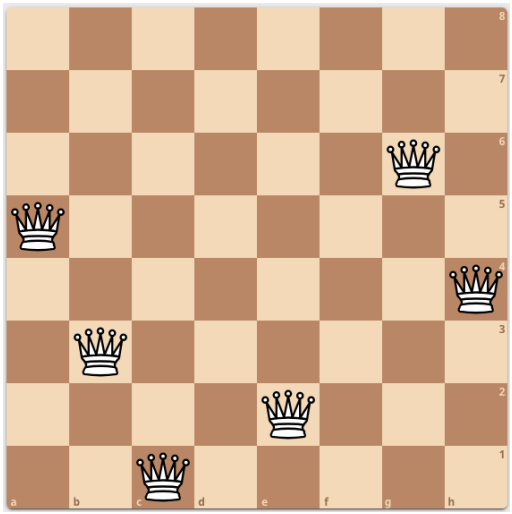
- Search both from the start and goal (usually BFS or IDFS why?), and hope to **meet in the middle**.
- If the goal is d-depth away, #states expanded = $b^{d/2+1} + b^{d/2+1} << b^d$ (for large d).
- Questions:
  - Can we use DFS as the forward or backward searches?
  - What should be done for multiple goal states?

Informed (Heuristics-based) Search Strategies

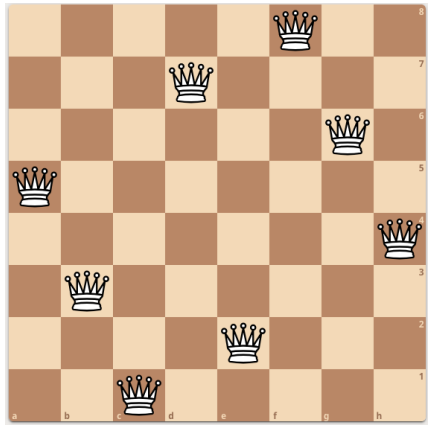
- Search strategies that use **problem-specific knowledge** to **find solutions more efficiently**.
- Relies on two **key cost functions**:
  - Evaluation function $g(n)$ - How good is the current state or how much effort/cost did it take to reach here?
  - Evaluation function $h(n)$ - An estimate on how 'close' is this to the goal state?
  - As these are cost functions, we interpret them as: **lower the better**.

Good $g(n)$ does not imply good $h(n)$!





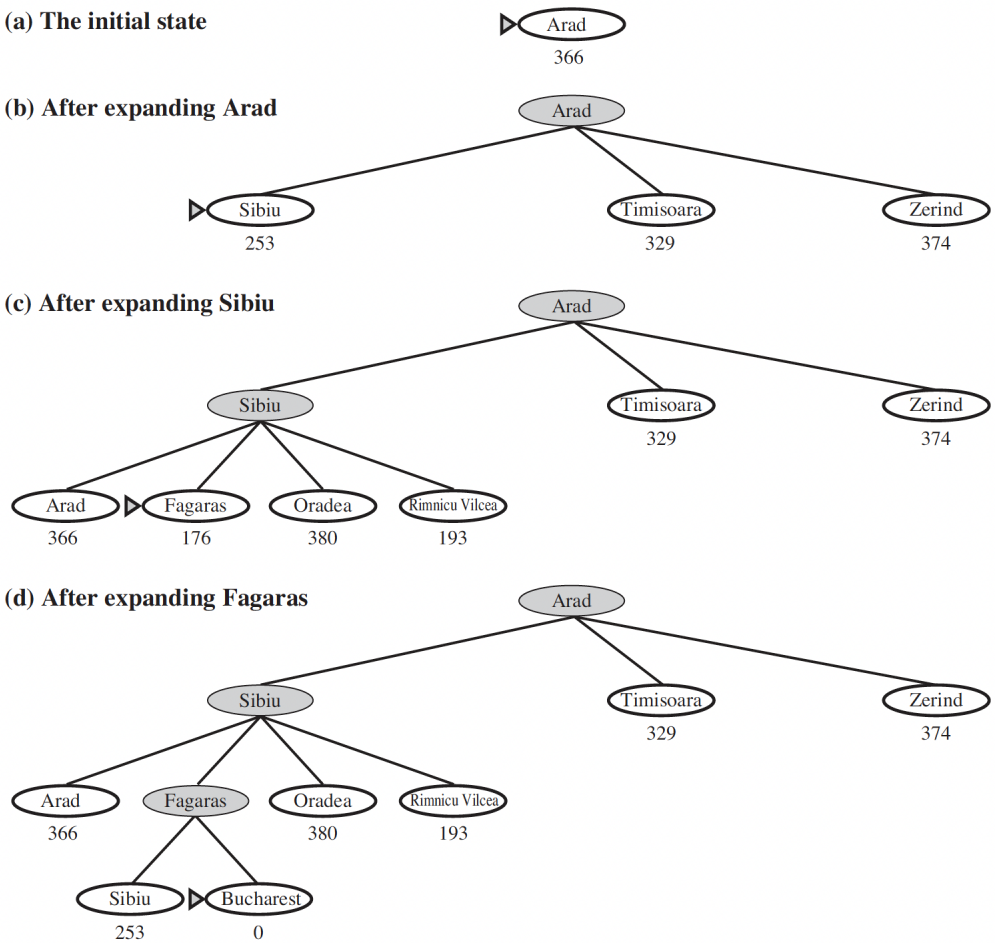
- 7 queens placed: better $g(n)$
- But no way to reach goal state from the position shown.
- 6 queens state admits a solution.



Two key search strategies

- **Greedy Best First** - Expands the node with the minimum $h(n)$, i.e., $f(n) = h(n)$.
- **A* search** - Minimizes the total estimated solution cost. i.e., $f(n) = g(n) + h(n)$.
- Note:
  - $g(n)$ is an exact function
  - $h(n)$ is only an approximation because if we had known the exact path to the goal, then there's nothing left to solve!

Best First Search



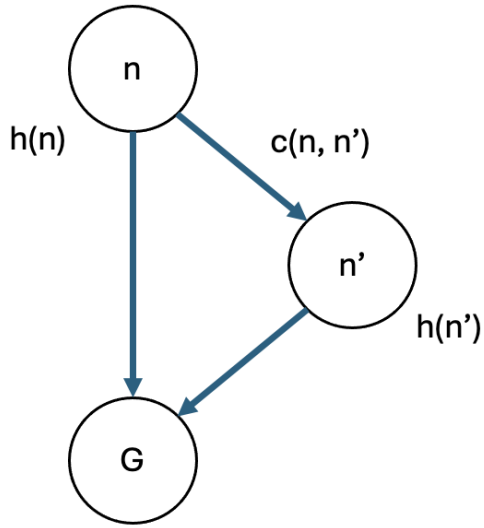
- Combines benefits of both BFS (exploit more, explore less) and DFS (explore more, exploit less).
- Not optimal because it ignores the cost that it took to reach the current node.
  - Example: There may be a shorter route from Timisoara (node not expanded) to Fagaras which is shorter than the distance from Sibiu to Fagaras.

A* Search

- Instead of using $h(n)$ use $g(n) + h(n)$ as the criteria to select a state for expansion.
- $g(n)$ is the cost from the root to node n.
- Balances: current cost (exploitation) + future cost (exploration).

Conditions for Optimality

- **Admissibility**: *Never overestimates* the cost to the goal (underestimation is good; **don't be pessimistic; better be optimistic!**).
- **Consistency**: $\forall a, h(n) \leq c(n, a, n') + h(n')$.



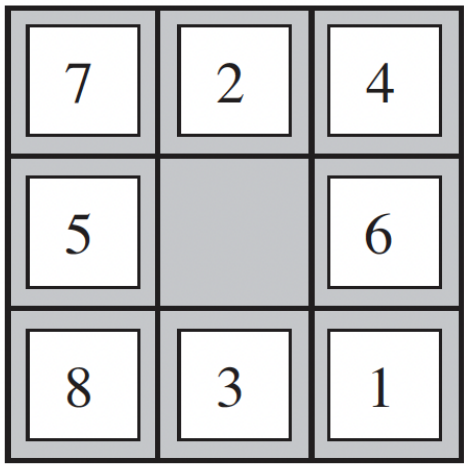
A* Search Implementation

- Like DFS and BFS, implemented by adding nodes to a queue (chidren states as candidates to be expanded later).
- Unlike DFS and BFS, the queue is no longer a LIFO/FIFO. Rather it is a **priority queue**.
- Standard data structure is a heap, with `extract_min` operation's time complexity is $O(\log N)$.
- The priority value is the cost $f(n) = g(n) + h(n)$, i.e., the cost it took to reach that node plus the estimated cost from there to reach the goal.

How to design good heuristics?

- Realise that the performance gains of A* over uninformed search strategies **depends on well-defined heuristics** functions.
- **Good heuristic function**: lets **explore promising paths** that **potentially lead to goal states much faster**.
- There is no single general heuristic function (or a class of such functions) that work for all discrete combinatorial optimization problems.
- Good heuristics are derived (logic and ingenuity) from the **knowledge about a problem task itself**.
- Of course, if you design a heuristics function you need to check if it is **admissible** and **consistent**. Just by being so, it doesn't always mean that it would be efficient in finding a goal state.

8-puzzle



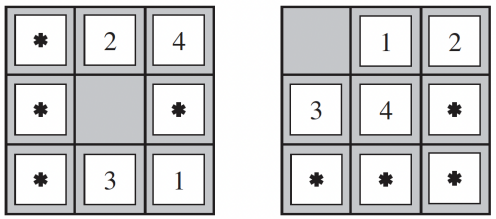
Consider the two heuristics functions:

- h_1: #misplaced tiles.
  - Admissible? Yes
  - You have to place each incorrect tile in its correct position and that will take at least one step.
- h_2: Sum of L1 distance of each tile from its goal position.
  - Admissible? Yes
  - Each tile would need to go to its correct position in at least d steps, where d is the L1 distance between its current position and the goal.

Heuristics by Constraints-Relaxed Sub-problems

- $h_1 \equiv$ Cost to the goal state for the relaxed action space - swap a tile A with tile B.
- $h_2 \equiv$ Cost to the goal state for the relaxed action space - swap a tile A with tile B f they are adjacent.
- In general, #state transitions leading to a goal state of a relaxed sub-problem is a **good heuristic** for **any** actual problem task.
  - No more assumptions specific to a problem task like #misplaced tiles.
  - Just make the action space less constrained (quite easy to do).

Disjoint Pattern Databases for Heuristics



- Split a problem into disjoint subproblems, e.g., place the tiles '1, 2, 3, 4' in order without worrying about the other tiles (suppose they are not numbered).
- #steps to solution from start to goal of this relaxed subproblem is a good heuristic of the 8-puzzle.
  - **Homework**: Can you see why this is admissible/consistent?
- How to compute the heuristics?
  - Do a backward search from the goal state to one of the states of the relaxed subproblem
  - Save the states and the cost in a dictionary.

Finding a Heuristic may not always be easy!

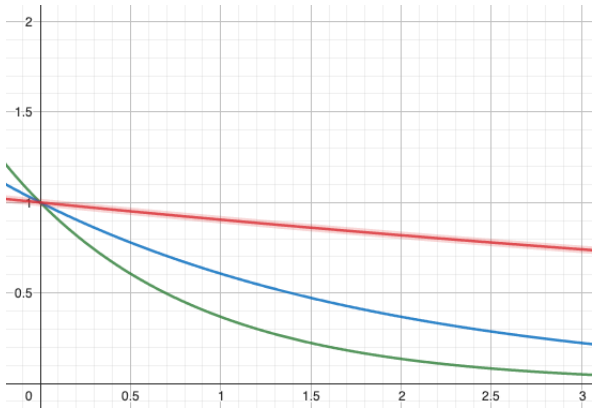
- Generate a target integer with a sequence of $x!$, $\sqrt{x}$ and $\lfloor x \rfloor$ functions, the only value of x being allowed is 4.
- Example: $\lfloor \sqrt{\sqrt{\sqrt{\sqrt{(4!)!}}}} \rfloor = 5$
- Difficult to come up with an admissible heuristic, i.e., an $h(n)$ which never overestimates the cost to goal.

Stochastic Search on Discrete Deterministic Problems

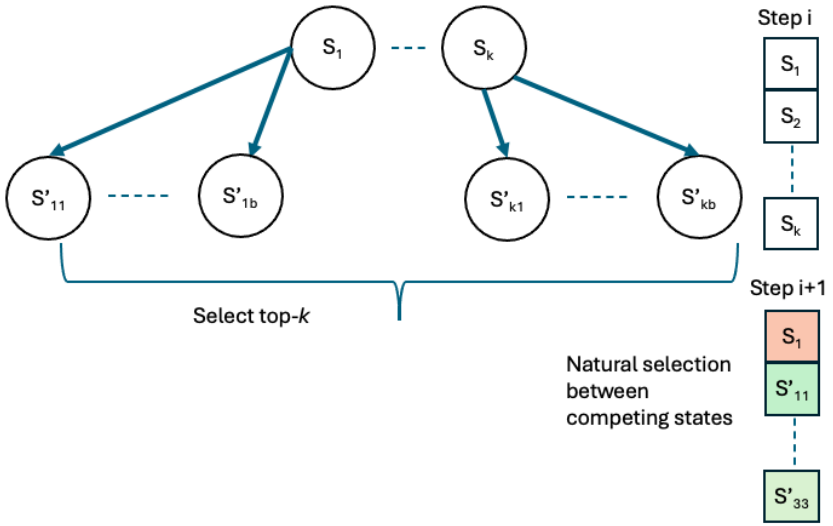
- Looked at deterministic search algorithms that operate on Discrete Deterministic Problems.
- Main limitations -
  - Inefficient if branching factor is too high.
  - Efficiency mainly relies on the heuristics function that estimates the cost to goal.
- Using a **stochastic search** helps address these problems.
- We will cover:
  - Simulated Annealing
  - Beam Search
  - Genetic Algorithm

Simulated Annealing (SA)

- From each state, **randomly choose an action** to generate a new child state.
  - If child state is better (evaluation function $f(n)$): Accept child state
  - If worse: Accept child state with probability $e^{\Delta E/T}$.
  - ΔE: $f(child) - f(eval)$ – amount by which the evaluation function becomes worse.
  - T: Temperature that decreases with time (steps); a possible mapping is $T \leftarrow e^{-i}$ or $1/i$, $i = 1, \ldots, N$
- Advantage: No need to save child states in memory (queue), and no backtracking.



Beam Search

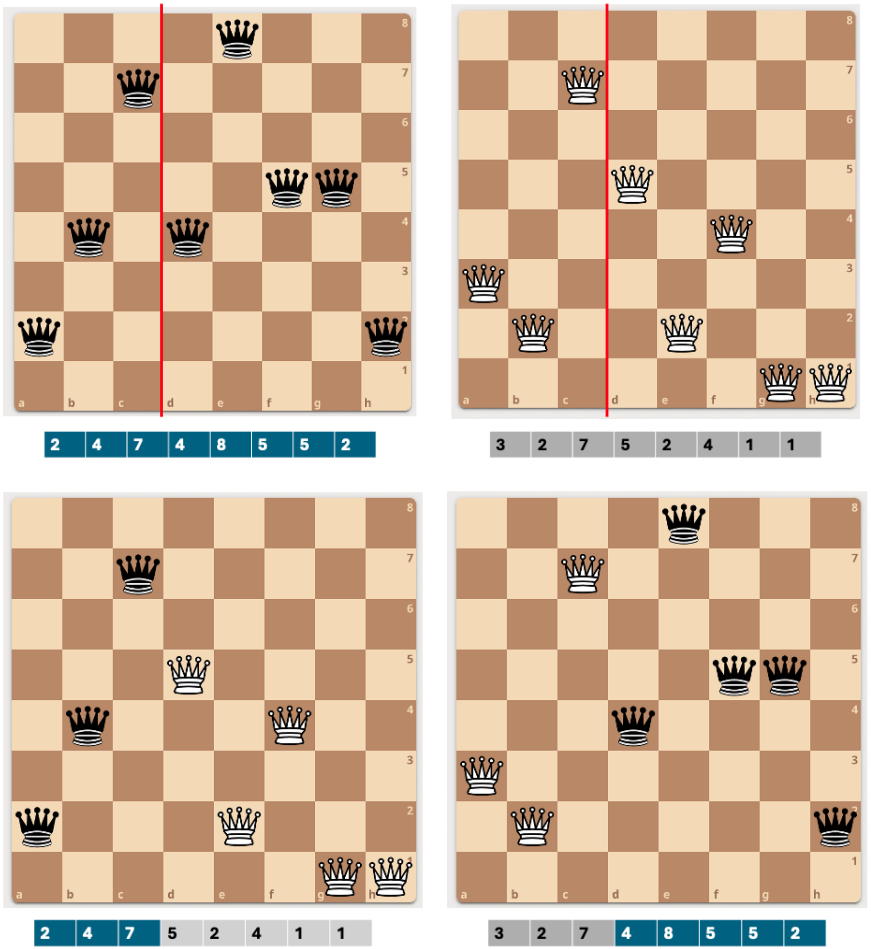


- SA keeps track of **only one state**. We can do better.
- Beam Search (BS) keeps track of k states (k - max-size of the queue).
- For each current state in the buffer S_i $(i = 1, \ldots, k)$
- Generate all successor states $S'_{i,j}$ $(j = 1, \ldots, b)$ – b_i: branching factor from node S_i.
- Select the k best states from this candidate pool and repeat.
- Avg branching factor: kb.

Stochastic Beam Search

- **Stochastic Beam Search (SBS)**: Uses the temperature (similar to SA) to define a probability of selecting a state with a lower 'goodness'.
  - Prob of selecting a worse state in the candidate pool: $\exp(\Delta E \times T)$
- Introduces more diversity in the state space exploration (otherwise local beam search can get focused on one region of the state space).
- Favours more exploration during the initial phase.

Genetic Algorithm



- SBS: Natural selection (asexual reproduction) with mutations.
- Variant of SBS, where successor states are generated by **combining two parent states** rather than by modifying a single state.

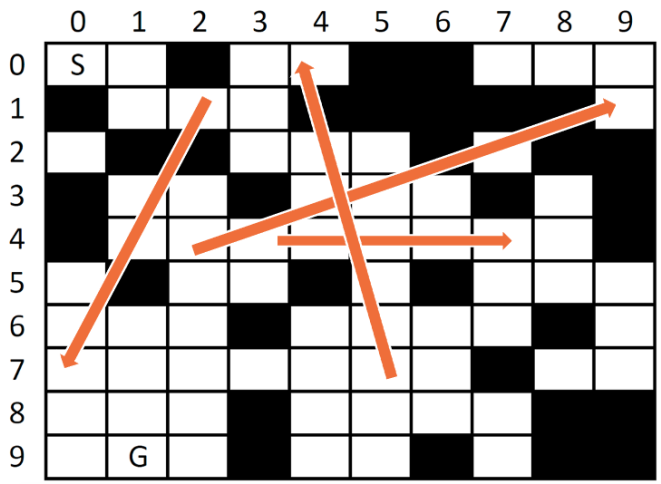
Genetic Algorithm (GA)

- Biological analogy:
  - SBS $\rightarrow$ asexual reproduction.
  - GA $\rightarrow$ sexual reproduction.
- Representation:
  - A population of **individuals** (states)
  - Individual represented by a string, each element a **gene**.
- Fitness (negative cost function):
  - Evaluate fitness function of the individual.
  - i.e. how likely are you (or your offspring) to survive?
  - Fitness function for the 8-queen's problem?

GA (Contd.)

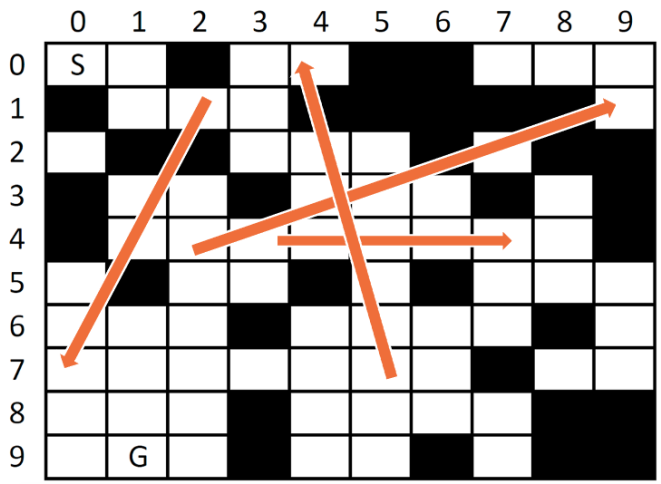
- Selection:
  - Select individuals to survive and also to perish. Different strategies but (usually) some form of stochastic selection depending on the fitness level.
- Crossover:
  - Have survivors mate and **mix genes**.
  - Combines parts of the goodness of the solutions.
    - Recall the relaxed subproblems in A* (Do you see an analogy?)
- Mutation:
  - Enforce some mutation of the genes (random changes in the encoded string), usually by a stochastic process.

A modified maze



- Modified transition graph.
- Right(1, 1) $\mapsto$ (7,0) - instead of (1, 2)
- Some wormholes are rewarding, e.g., (1, 2) is rewarding. Why?
- Some are penalizing - e.g., wormhole at (7,5).
- Which ones do you think work well?
  - DFS
  - BFS
  - IDFS

A modified maze



- Modified transition graph.
- Right(1, 1) $\mapsto$ (7,0) - instead of (1, 2)
- Some wormholes are rewarding, e.g., (1, 2) is rewarding. Why?
- Some are penalizing - e.g., wormhole at (7,5).
- Which ones do you think work well?
  - BFS
  - DFS
  - A*

Course-work Description

- Will be released this week.
  - Tip: Start working on it in an incremental manner.
- Will ask you to generate a random **maze with wormholes** on a **frozen lake** (code provided).
- What would be a performance measure in this environment?
- **Tasks**:
  - Implement your favorite maze navigation algorithm.
  - Explain your decision choices.
  - Conduct experiments and report observations to analyze the effects of you agent's performance measure on the following:
    - Density of wormholes.
    - Density of holes in the lake.
    - Uncertainty of actions, i.e., the slipperiness of the surface?