Problem Solving
- State Space Search

林伯慎 Bor-shen Lin
bslin@cs.ntust.edu.tw
http:www.cs.ntust.edu.tw/~bslin

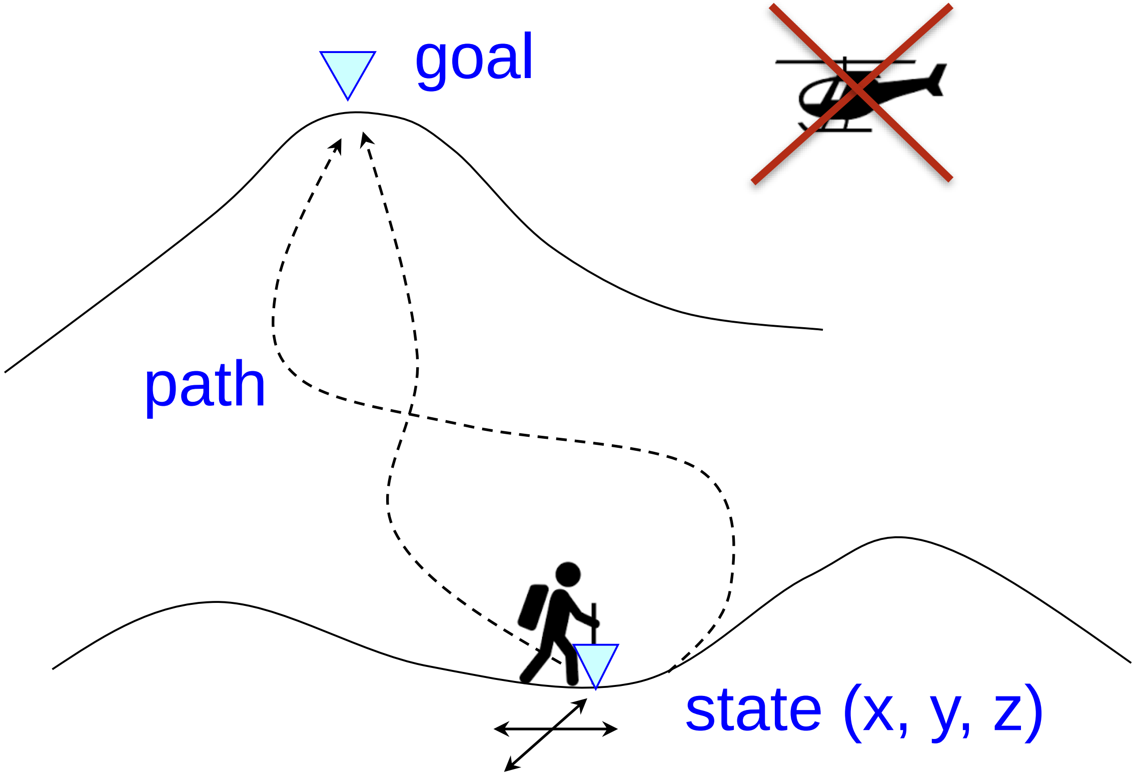
Agenda

- Fundamentals of Search
- Uninformed Search
- Informed Search
- Appendix

Basics of Problem Solving

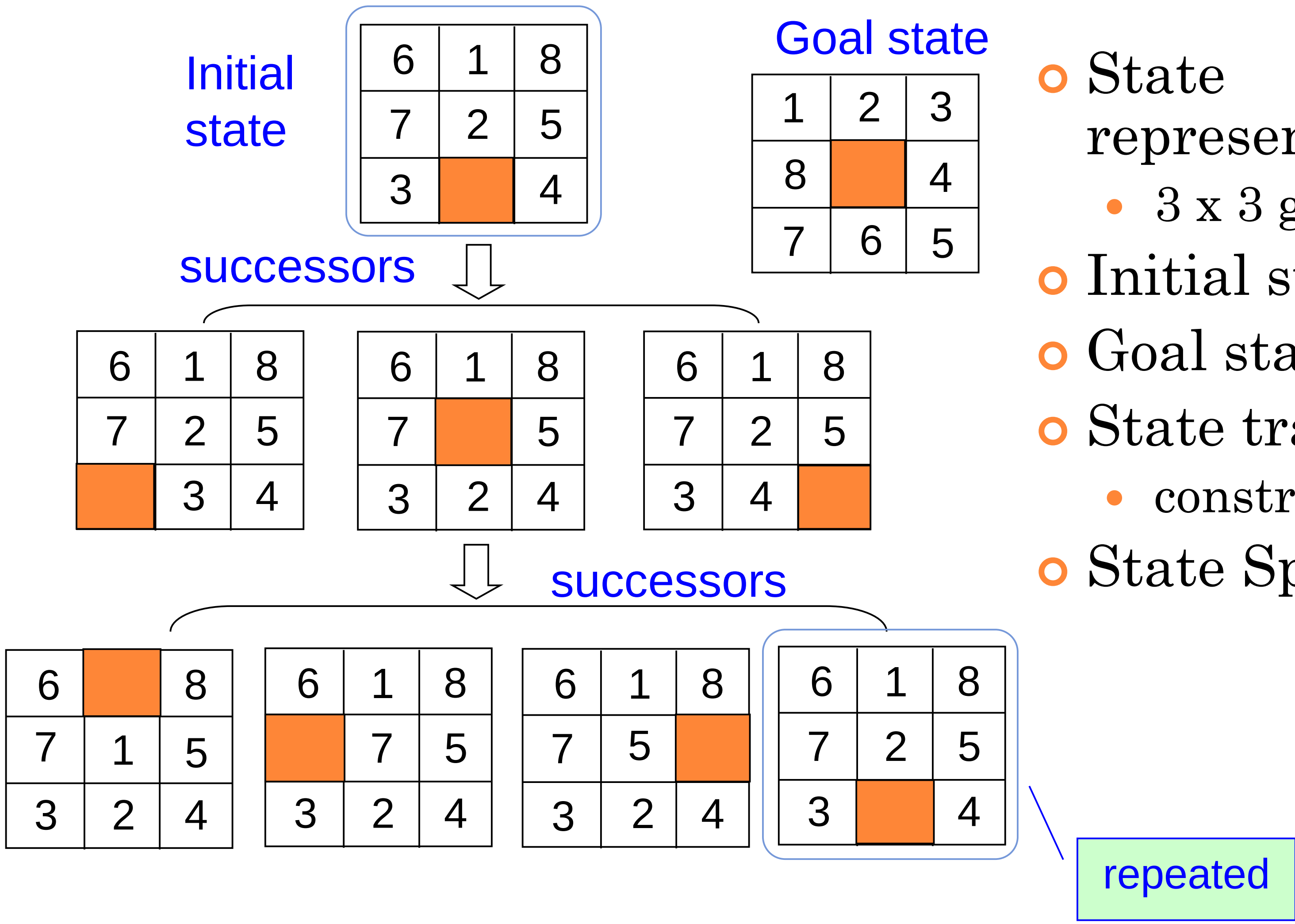
- Problem ~ Goal
  - Computing, proving something, buying things, …
- Solving ~ Search
  - Find a set of actions that can be taken to lead to the goal state
- State
  - Represent where we have reached
- Successors
  - Children states generated from current state
- State space
  - Space consisting of all possible states

Problem: Mountain Climbing



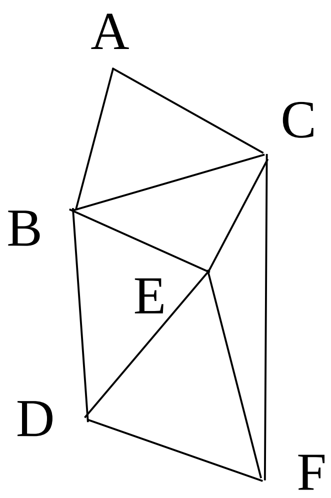
Successor states:
physical constraint

Problem: 8-Puzzle

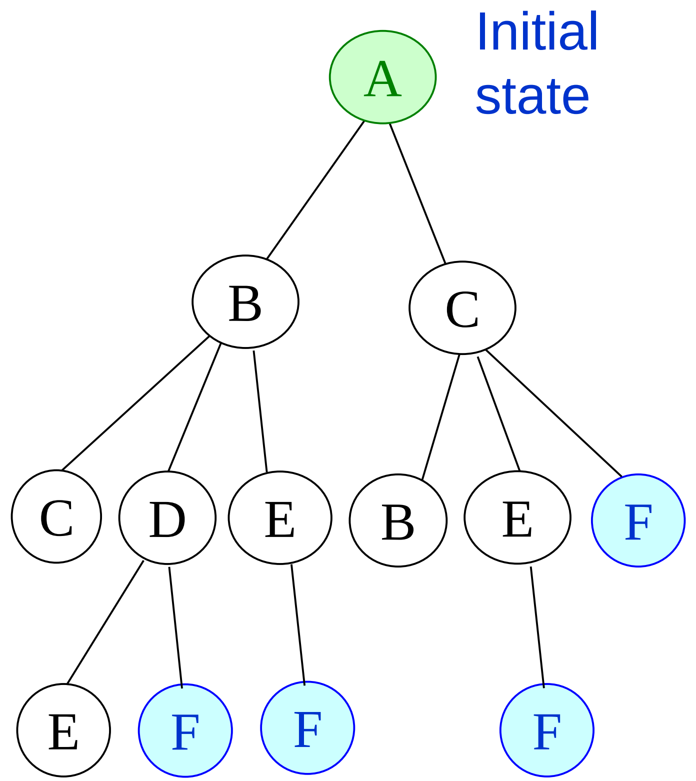


- State representation
  - 3 x 3 grids
- Initial state
- Goal state
- State transition
  - constraints
- State Space: 9!

Shortest Path Problem



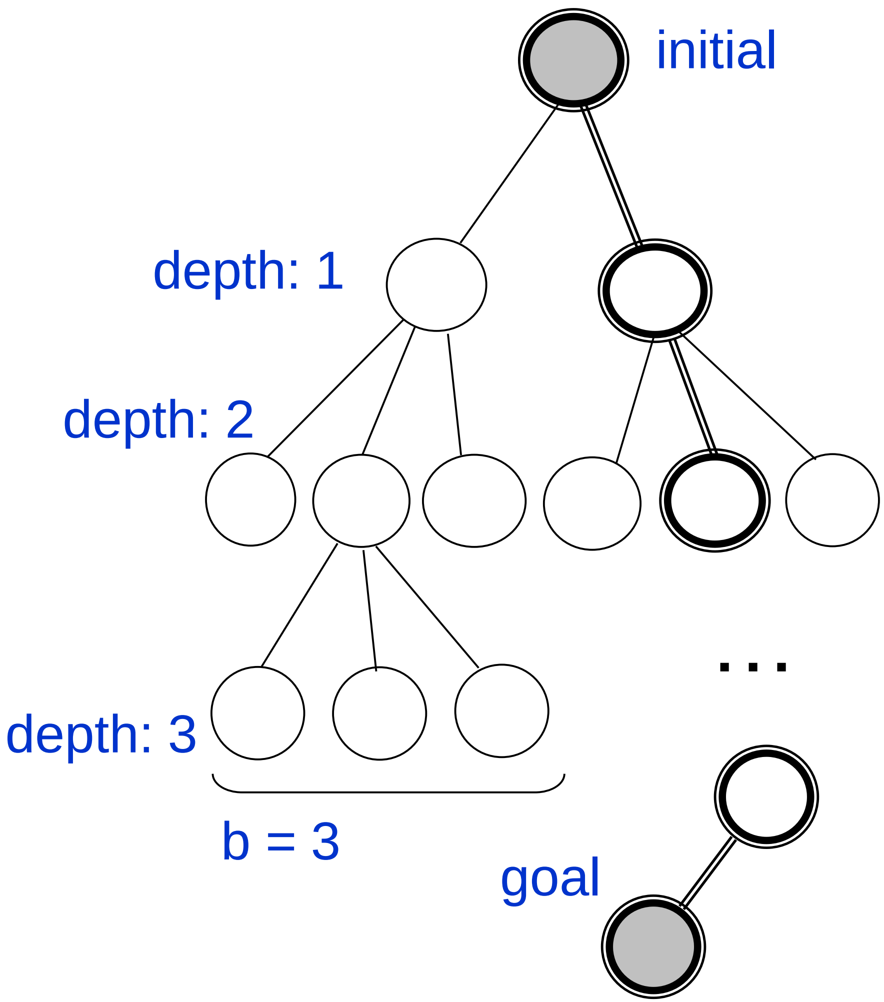
From A to F



Search as Problem Solving

- Problem
  - Modeled and formulated flexibly
- Successor
  - Depends on the rules or physical constraints
- Goal state
  - Might be more than one
  - (e.g. with/without optimization)
- State space
  - Huge or infinite
- Path
  - Multiple choices

Search Tree



- Search Tree
  - Consisting of all states generated during search
- Search State
  - Any node on search tree
- Solution path
  - Any path from initial state to the goal state
- Branching factor b
  - Average number of successors for each node
- Depth of tree node

Search Strategies

- Generate and Test
  - Generate successors
  - Test if goal state is achieved
- Required data structures
  - Open list: frontier nodes
  - Closed list: visited nodes
  - Closed list is usually implemented with set/map for avoiding duplication for the same state
- Type of search strategies
  - Uninformed Search : without information
  - Informed Search : with information

AGENDA

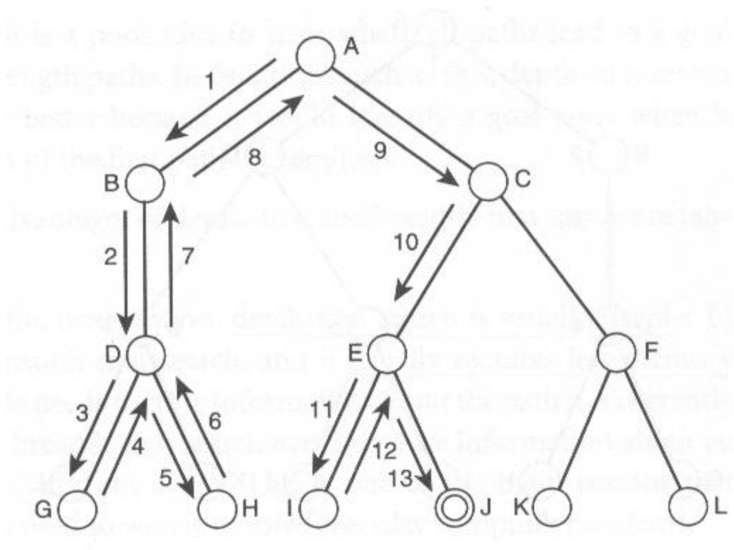
- Fundamentals of Search
- Uninformed Search
  - Depth-first Search (DFS)
  - Breadth-first Search (BFS)
  - Depth-first Iterative Deepening (DFID)
- Informed Search
- Appendix

Uninformed Search

- Without using any prior information
- Terms with similar meaning
  - brute-force search
  - exhaustive search
  - blind search
- Depth-first search (DFS)
- Breadth-first search (BFS)

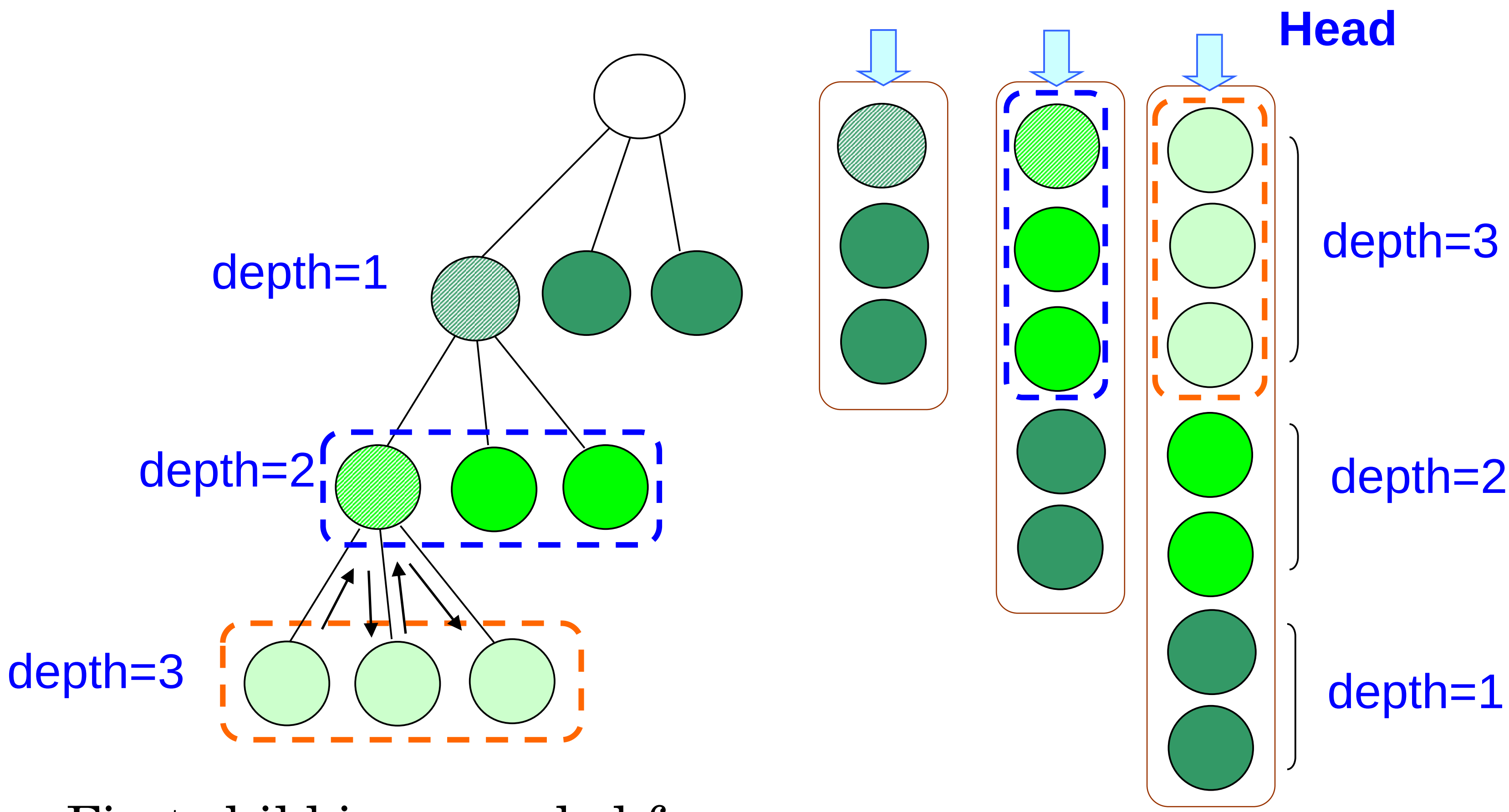
Depth-First Search

- It follows each path to its greatest depth before moving on to the next path
  - When leaf node is reached, trace it back
- A stack is maintained for controlling the search
  - The first entry is popped
  - Spanned nodes are inserted at head of the list



head	[A]	tail	pop A
	[BC]		pop B
	[DC]		pop D
	[GHC]		pop G
	[HC]		pop H
	[C]		pop C
	[EF]		pop E

Concept of Depth First Search



- First child is expanded for successors
- Newest states with highest depth gain higher priorities

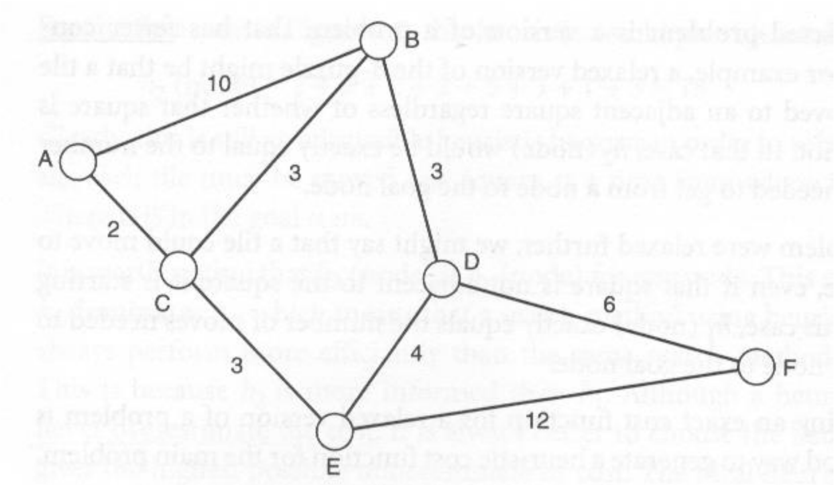
Pseudo Codes for DFS

```
Function depth()
{
   open_list = [ root_node ];
   while(true) {
      state = pop(open_list );
      if(is_goal(state)) then return(state); // goal found
      else add_to_front_of_queue(successors(state));
      if open_list == [ ] then return(null);   // goal not existing
   }
}
```

- Avoid repeated states by closed list
- Implement open list as stack (Last In First Out)

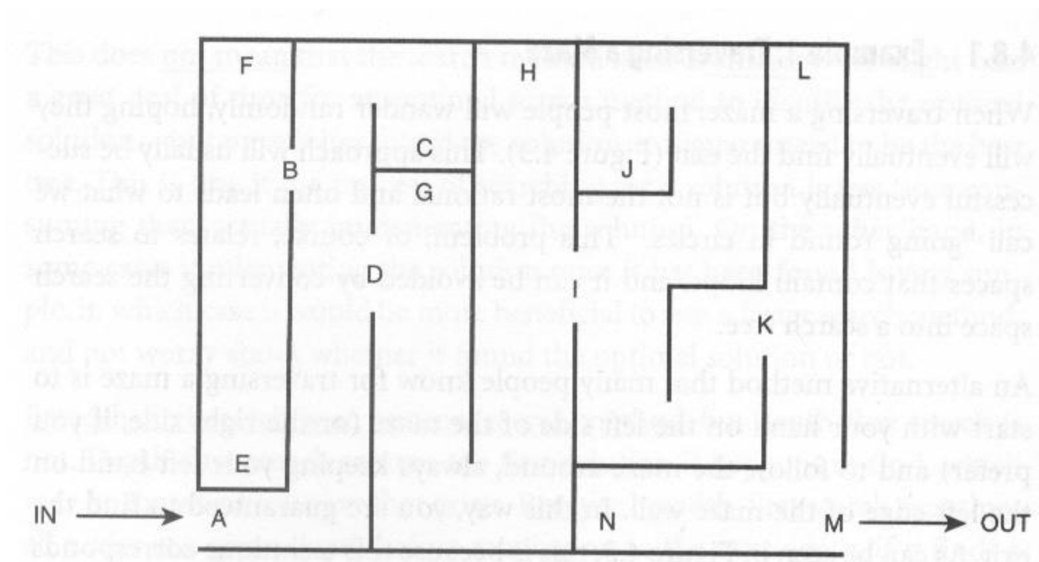
DFS: Practice

- Please draw search tree from A to F using depth first search (e.g. A → B, C & B → C, D)



Discussions on DFS

- DFS is similar to human behaviors
  - Buying things or climbing mountains
- Find an exit in a foggy floor in case of fire
  - Touch a wall on one side



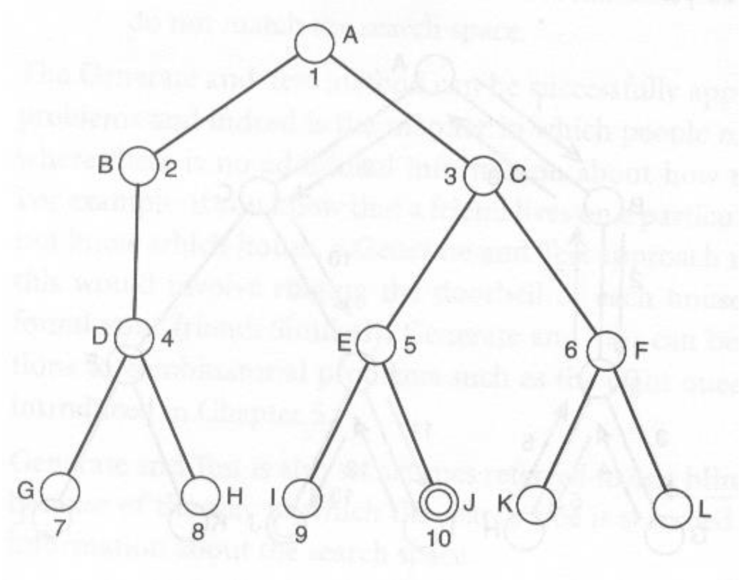
Complexity of DFS

- Time Complexity: $O(b^d)$
- Space Complexity: O(bd)

  Requirement of search space is reduced by use of back tracking

Breadth-First Search

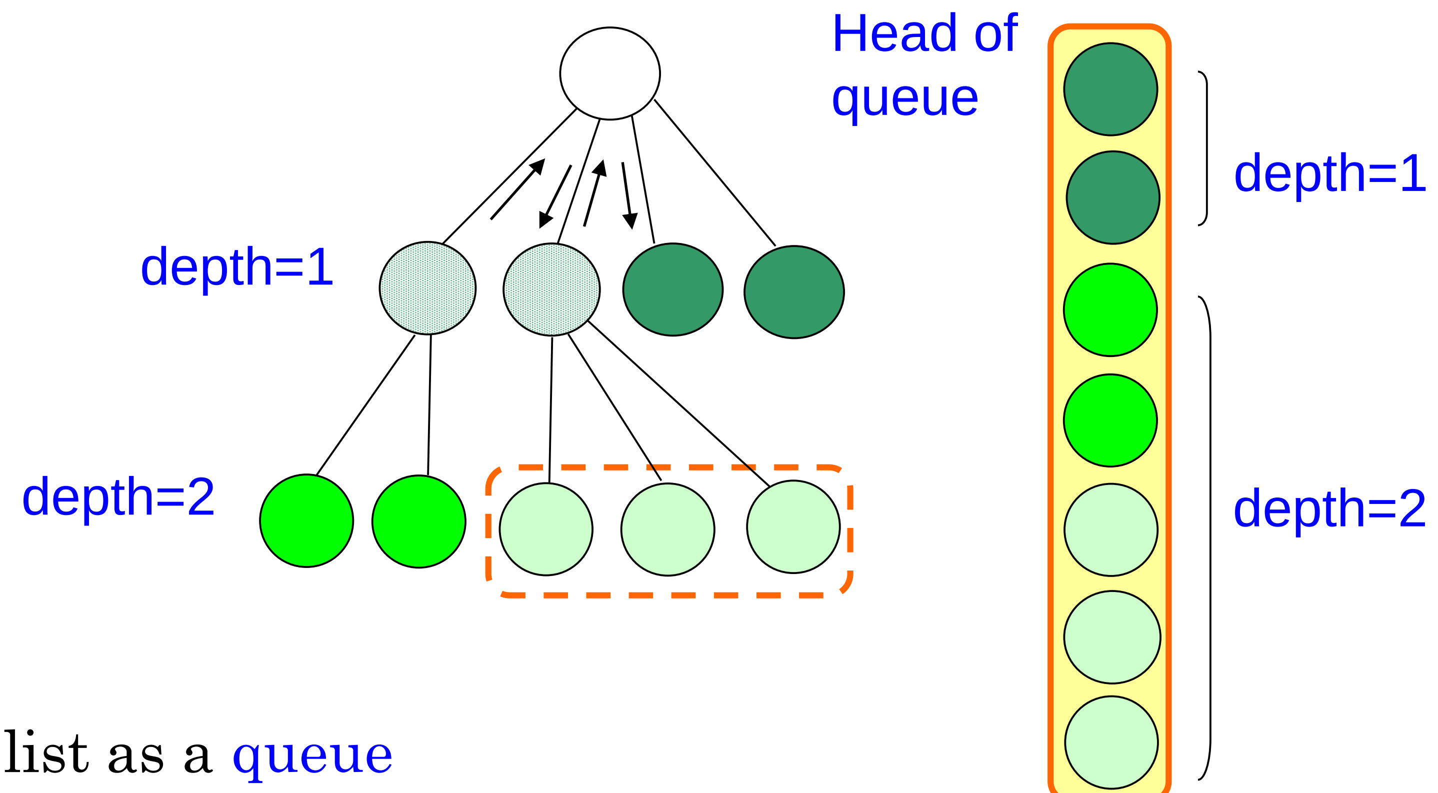
- Examine all nodes one level down from the root node
- More memory usage more than depth-first
  - Depth increases gradually
  - Goals in lower depth will be found first
- A queue is maintained for controlling the search
  - Newly spanned nodes are inserted at back of the queue



[A]	pop A
[BC]	pop B
[CD]	pop C
[DEF]	pop D
[EFGH]	pop E
[FGHIJ]	pop F
[GHIJKL]	…

Queue: larger space $O(b^d)$

Concept of Breadth First Search



- Open list as a queue
- First entry is popped and expanded
- Newest states are inserted at the back of the queue (i.e. with lower priorities)

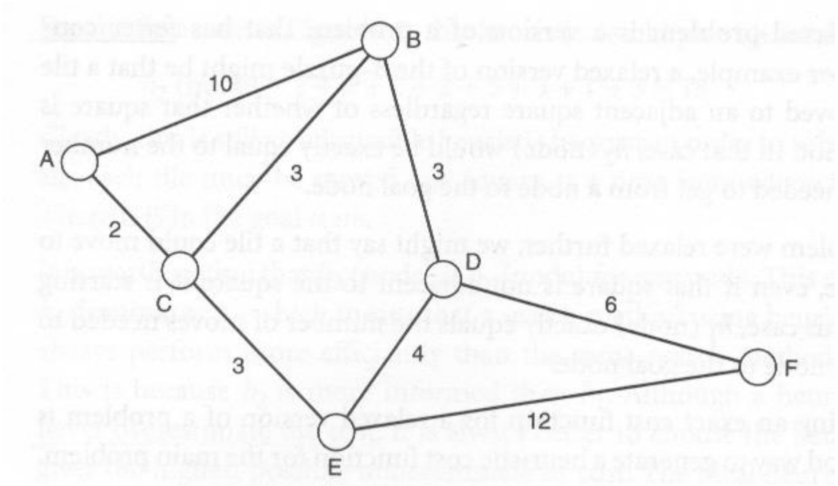
IMPLEMENTATION OF BFS

```
Function breadth()
{
    open_list = [ root_node ]; // root_node as initial state
    while(true) {
        state = pop(open_list );
        if(is_goal(state)) then return(state); // goal found
        else add_to_back_of_queue(successors(state));
        if  open_list == [ ] then  return(null);  // not not existing
    }
}
```

Practice of BFS

- Please draw the search tree from A to F using breadth first search

  (e.g. A → B, C & B → C, D)



Breadth-First Search

- Minimum-depth path is guaranteed
  - Lower-depth first
  - Nodes in shallower layers of search tree get higher priorities than those in deeper layers
- Time Complexity: $O(b^d)$
  - $1+b^1+b^2+\ldots+b^d = (b^{d+1}-1)/(b-1)$
  - Complexity compatible with DFS
- Space Complexity: $O(b^d)$
  - Much higher than DFS

DFS vs. BFS

- Ways of managing open list
  - DFS: inserted at front (as stack)
  - BFS: inserted at back (as queue)
- It is often not easy to consider optimality, time complexity and space complexity simultaneously
  - DFS: lower memory requirement
  - BFS: minimum-length path
- Which is better?
  - It depends on the problems
  - What is the shape of state space
  - Where are the goal states located

DFS VS. BFS

- DFS is more often used: Less memory usage
- Search tree may have very deep path, and goal node is in shallower part
  - Breadth-first good, but depth-first poor
  - 8-Puzzle: very deep path, DFS not good
- All paths are of similar length
  - DFS good (do not try lower level)
- Search tree has high branching factor
  - DFS good (BFS consumes too many memories)

Depth-First Iterative Deepening (DFID)

- Depth-First with depth D
  - DFS : search down to depth=D
  - time complexity DFS(D) = $1+b^1+b^2+\ldots+b^D = (b^{D+1}-1)/(b-1)$
- Depth-First Iterative Deepening (DFID)
  - Increase the depth up to D gradually
  - DFID = DFS(0) + DFS(1) + DFS(2) + … + DFS(d)

    $= (d+1) + b \cdot (d) + b^2 \cdot (d-1) + \ldots + b^{d-1} \cdot (2) + b^d$
  - Time complexity $O(b^d)$
  - Space complexity O(bd)
- Avoid the major defect of DFS (invalid for goal of infinite depth), but with similar space complexity

D=0	1
D=1	1 b
D=2	1 b b^2
…	
D=d	1 b b^2 … b^d

Agenda

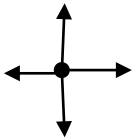
- Fundamentals of Search
- Uninformed Search
- Informed Search
- Appendix

Informed Search

- Use some heuristics
  - The choice of heuristics may influence search efficiency
- Hill Climbing
- Best-first Search
- Algorithm A
- Algorithm A*
- Greedy Search

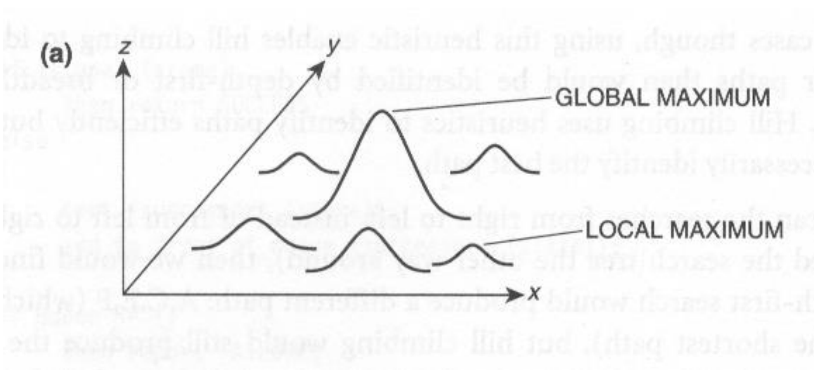
Hill Climbing

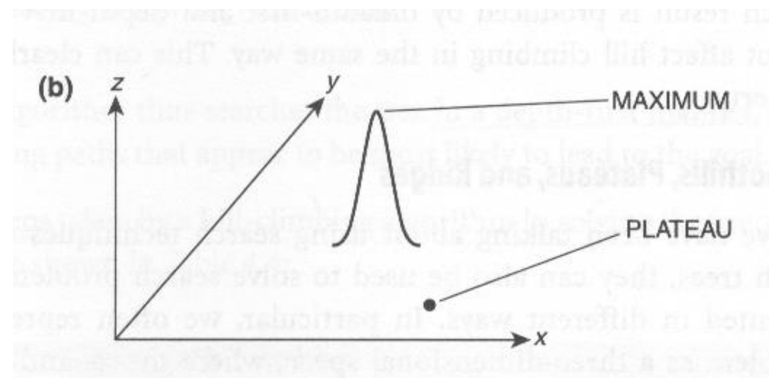
- Fundamental principles
  - Foggy day, no map
  - Move east/west/south/north
  - Evaluation h(x, y) for every move
  - Choose the highest direction
  - Stop if the system cannot move to a higher point



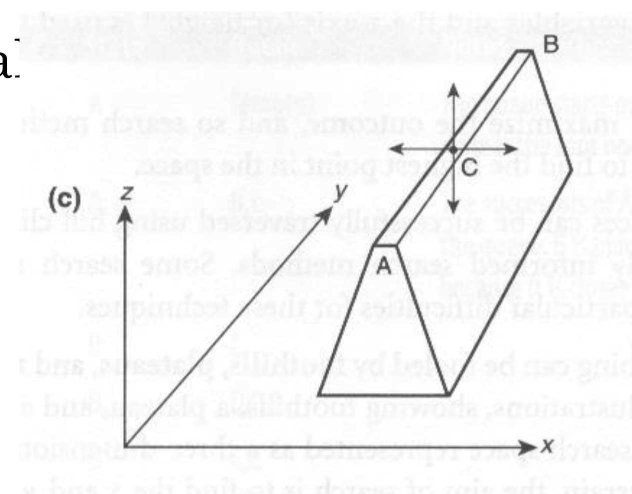
- A kind of greedy algorithm
  - Do not consider all possible states; consider only nearby states (successors)
  - A kind of coarse heuristic
  - Can reach only the local optima.

FOOTHILLS, PLATEAUS, RIDGES

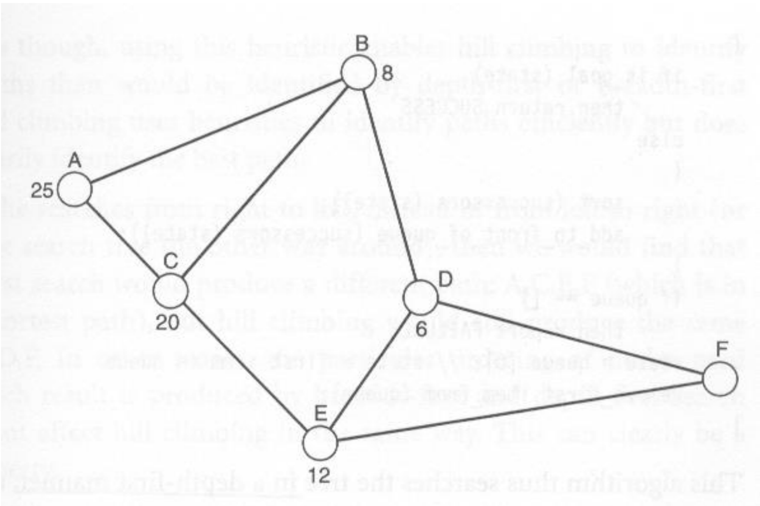




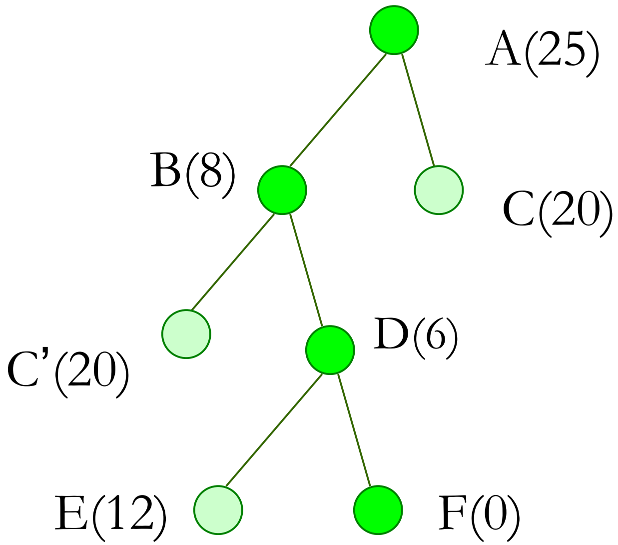
- Foothills
  - Global maximum surrounded by local maxima
- Plateaus
  - A region where all values are the same
- Ridges
  - A long, thin region of high land with low land on either side



Hill Climbing



Heuristic h(n) (state n)

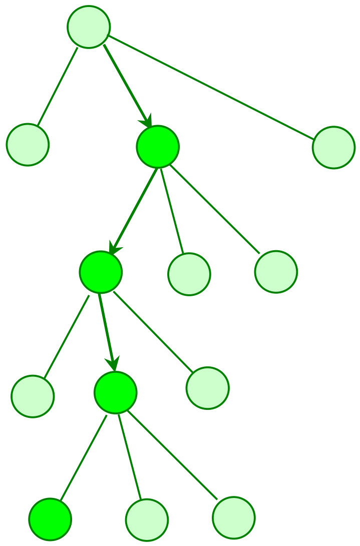


[A(25)]		pop A
[**B(8) C(20)**]		pop B
[**D(6) C'(20)**]	~~C~~	pop D
[**F(0) E(12)**]	~~C' C~~	pop F

- only evaluated for newly generated nodes
- no back trace, ignore visited (C, C')
- (A,B,D,F): NOT the shortest path

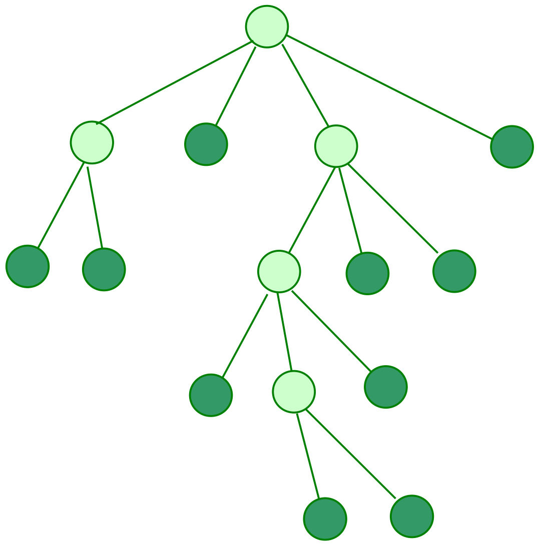
Discussions on Hill Climbing

- Do not consider all frontier nodes
- No back tracking
- Local optimal(successors)
- Optimal path not guaranteed



Hill Climbing

Best-First Search



Front-end nodes in a **priority queue** sorted by some function f(n)

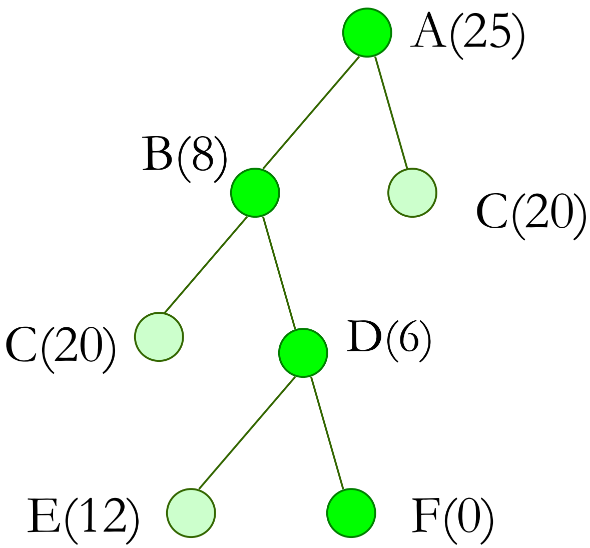
- Open list
  - Containing the frontier nodes (e.g. those in dark green)
  - as a priority queue
  - Choose the most promising states for expansion according to evaluation function $f(n)$.
- Whether optimality may be achieved depends on the choice of $f(n)$.

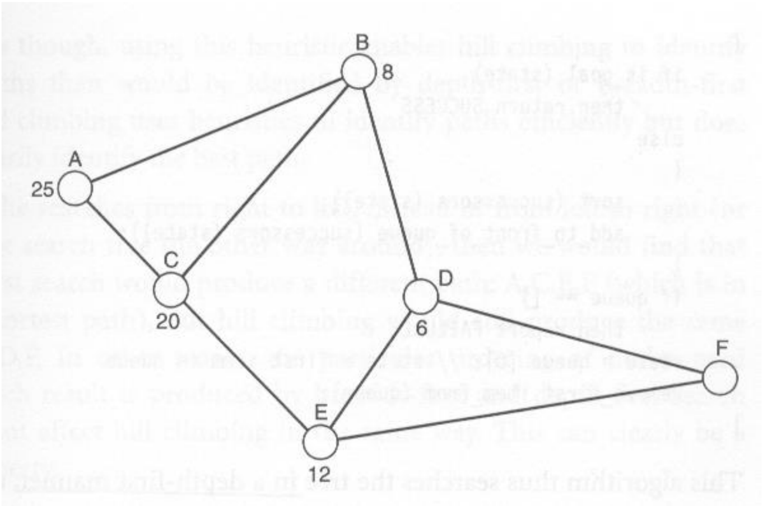
Pseudo codes for Best First Search

```
Function best_first(f)
{
    open_list = [ root_node ];
    while(true) {
        state = pop(open_list );
        if(is_goal(state)) then return(state); // goal found
        else add_to_priority_queue(successors(state), f);
        if open_list == [ ] then  return(null); // goal not existing
    }
}
```

- f(·) : evaluation function
- priority queue is sorted by f(n)

Example of Best First Search





heuristic h(n) for state n

[A(25)]	pop A
[B(8) C(20)]	pop B
[D(6) C'(20) **C(20)**]	pop D
[F(0) E(12) **C'(20) C(20)**]	pop F

- (A,B,D,F) is not shortest path
- h(n) consider the estimated cost (distance) from the current state (n) to the goal state; it does not consider the distance from the initial state to the current state.

Discussions on Best-first Search

- Neither hill-climbing nor best-first search achieve best path (shortest path) in the TSP example.
- If the evaluation function f(n) is chosen arbitrarily for informed search, no optimality about the search result could be guaranteed!
- Optimality of solution path is guaranteed only if f(n) can satisfy certain conditions.
- That is why algorithm A/A* is defined.

ALGORITHM A

- Algorithm A: Based on the best-first strategy
- Consider an evaluation function $f(n) = g(n) + h(n)$ for each state n, where
  - g(n) is the cost of n from the start state (have known)
  - h(n) is the heuristic estimate of the cost of going from n to a goal node
- If such evaluation function is used with the best-first search strategy, the algorithm is called Algorithm A.
  - g(n)/h(n) could be defined as depth, distance, or others, based on what is to be optimized.
  - Breadth-first search: g(n) is the depth of state n, $h(n)=0$.

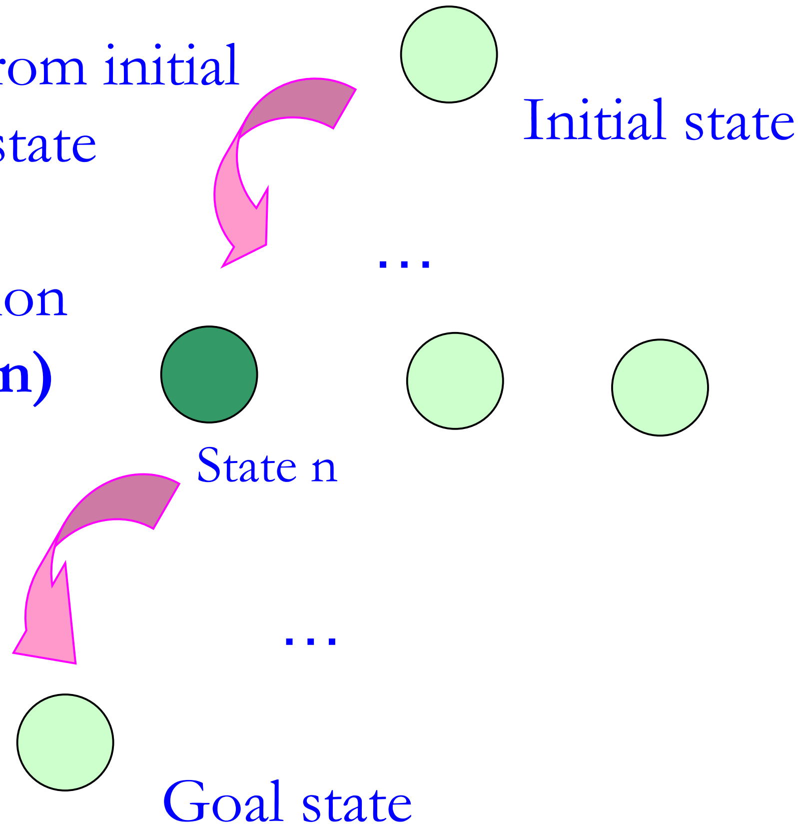
Heuristic of Algorithm A

g(n) : the cost from initial state to current state

Evaluation function

$$\mathbf{f(n) = g(n) + h(n)}$$

h(n) : estimated cost from current state to goal state



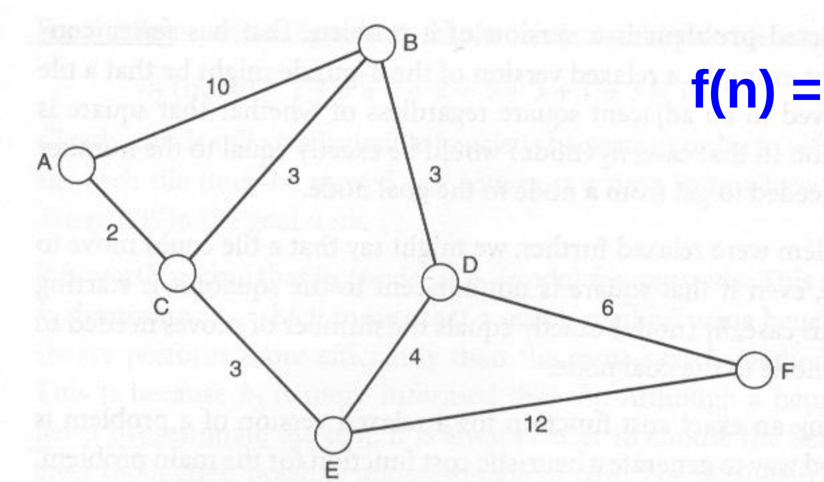
Algorithm A*: Optimality

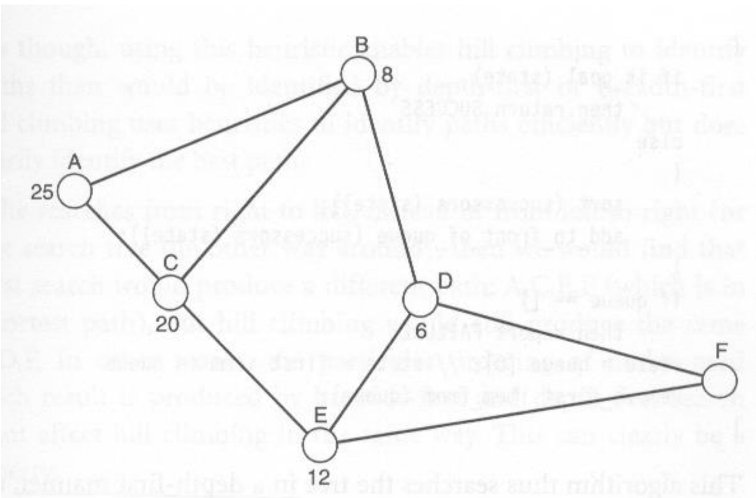
- Algorithm A*
  - If *algorithm A* is used with an evaluation function in which *h(n) is less or equal to the cost of the minimum path from n to the goal, h*(n)*, the resulting search algorithm is called *Algorithm A*.* (Admissible)
  - $h(n) \leq h^*(n)$ *for all n* (under-estimate)
- Informed-ness
  - For two A* heuristics h1 and h2, if $h_1(n) \leqq h_2(n)$ *fir all states n* in the search space, heuristic h_2 is said to be more informed than h_1.
  - Good heuristic: estimated cost (h(n)) approaches to the real minimum cost (h*(n))
  - Good heuristic: find the answer more quickly (using less nodes in search tree, or higher efficiency)

Special cases of Algorithm A*

- Breadth-first search
  - $f(n) = g(n)$ ($h(n) = 0$), where $g(n)$ is defined as the depth of n
  - Guaranteed to find minimum-depth goal
  - $h(n) = 0$ is not close to $h^*(n)$, so it takes longer steps to reach the goal
- Uniform-cost search (Dijkstra)
  - $f(n) = g(n)$ ($h(n) = 0$), where $g(n)$ is defined as the accumulated distance till current state n
  - Guaranteed to find minimum-distance path

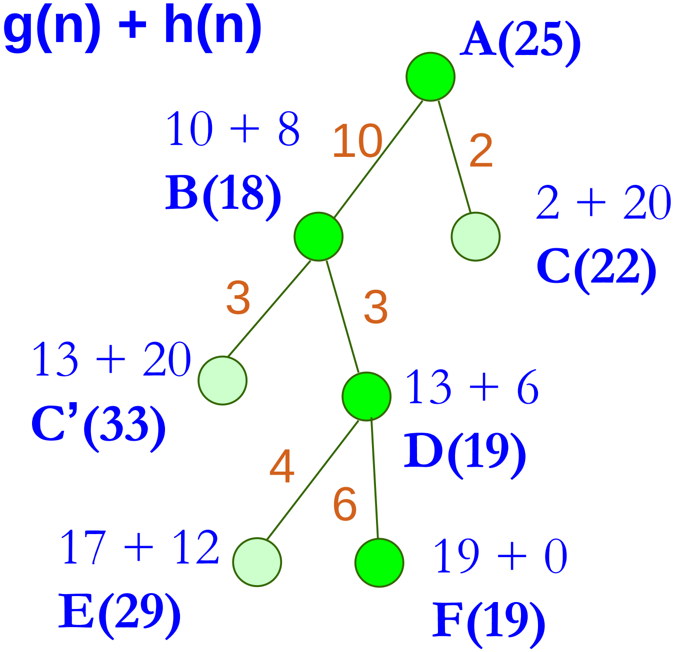
Algorithm A : Example 1





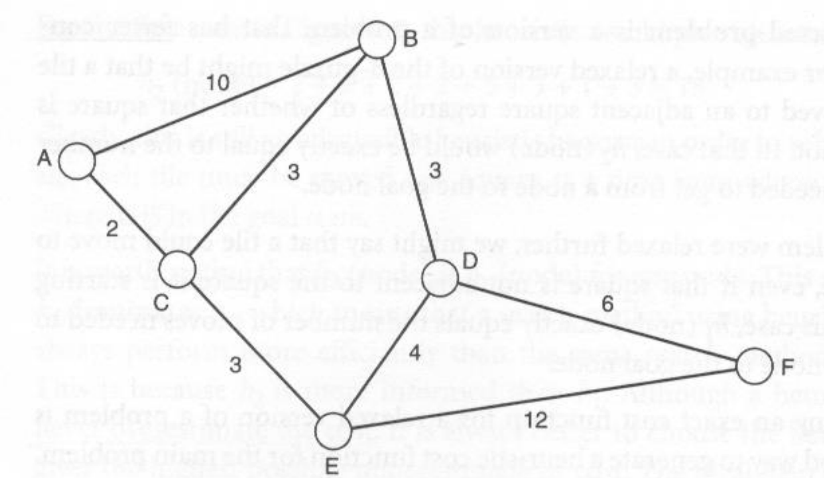
heuristic **h(n)**

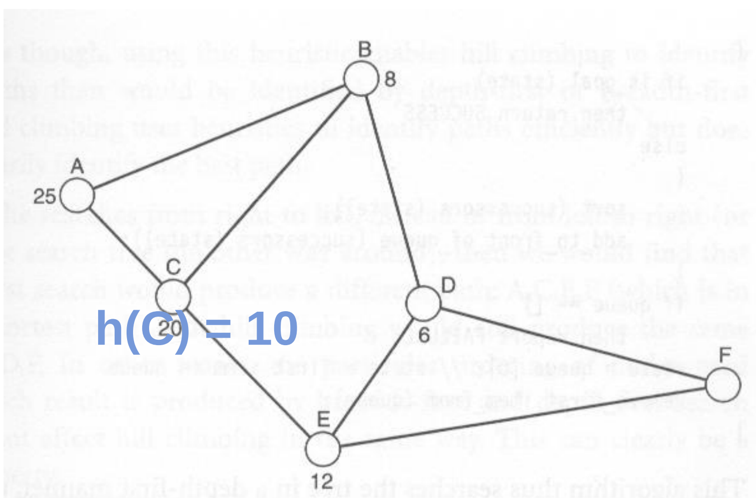
f(n) = g(n) + h(n)



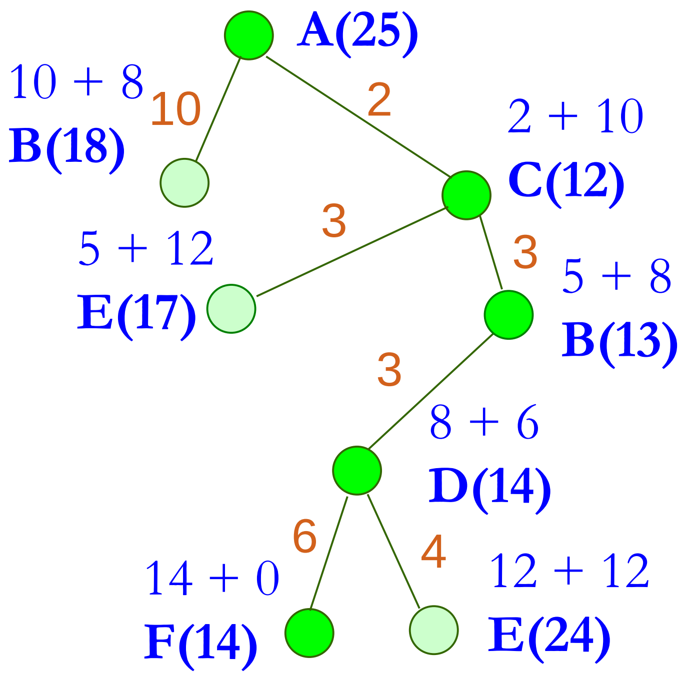
[A25]	pop A
[B18, C22]	pop B
[D19, C22, C'33]	pop D
[F19, C22, E29, C'33]	pop F

ALGORITHM A* : EXAMPLE 2





heuristic h(n) (state n)



[A25]	pop A25
[C12, B18]	pop C12
[B13, E17, B18]	pop B13
[D14, E17, B18]	pop D14
[F14, E17, B20, E24]	pop F14

Discussions on Algorithm A

- Best-first strategy + evaluation function f(n)

  → It is not guaranteed that the first solution path is the optimal path.

  - heuristic h(n) is too arbitrary.

- In the example, h(n) is not good

  e.g. the f(n) for node C is 20, but the real shortest distance from C to F is 12 (C-E-D-F). So, the first obtained solution path A-B-D-F is not optimal.

- Constraint on h(n) → Algorithm A*

Discussions on Algorithm A

- Path A-C-B-D-F is the shortest path
- Why the shortest path can be obtained after h(C)is changed as 10?
  - $h^*(C)$: real minimum cost from the current state C to the goal state F.

    C-E-F: distance=15

    C-E-D-F: distance=13

    C-B-D-F: distance = 12

    therefore $h^*(C) = 12$ (true minimum distance)
  - $h(C)=10$ now satisfies the criteria of optimality!

    $h(C) \leqq h^*(C)$

HEURISTICS FOR 8-PUZZLE

All heuristics satisfy the constraint of A* algorithm, h(n) ≦ h*(n) (i.e. optimal steps could be found for every heuristic), but with **different efficiency**

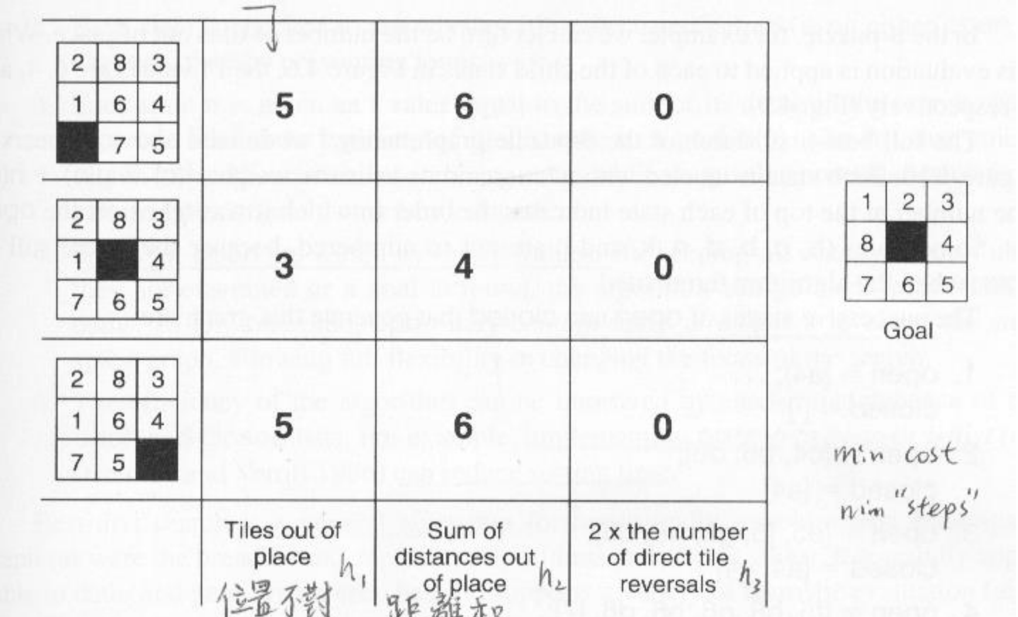
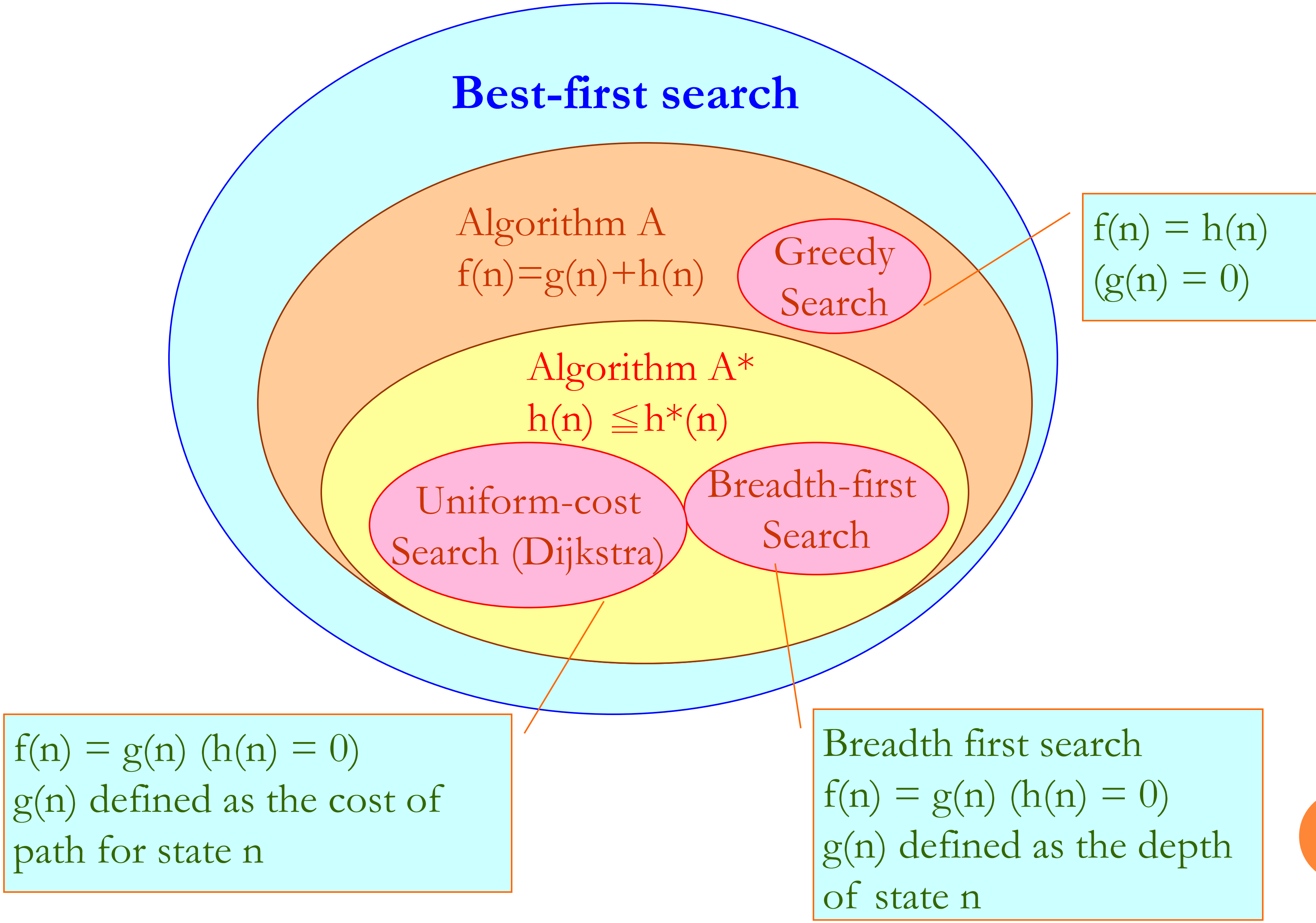


Figure 4.8 Three heuristics applied to states in the 8-puzzle.

- $h_1(n)$: # of tiles out of their target places
- $h_2(n)$: sum of Manhattan distances
- $h_3(n)$: $h_2(n)$ + 2 * (number of direct swapping pairs)
- Informedness : $h^*(n) \geqq h_3(n) \geqq h_2(n) \geqq h_1(n) \geqq 0$


Best-first search
Algorithm A
f(n)=g(n)+h(n)
Greedy Search
Algorithm A*
h(n) ≦h*(n)
Uniform-cost Search (Dijkstra)
Breadth-first Search
f(n) = h(n)
(g(n) = 0)
f(n) = g(n) (h(n) = 0)
g(n) defined as the cost of path for state n
Breadth first search
f(n) = g(n) (h(n) = 0)
g(n) defined as the depth of state n

Discussions

- Optimism
  - Under-estimate the cost(h(n) <= h*(n))
  - Try more possibilities
    - search tree might be larger (more nodes)
  - Will not miss optimal path
    - Path with minimum cost
  - Good heuristic: estimated cost (h(n)) approaches the real minimum cost (h*(n)).

Discussions

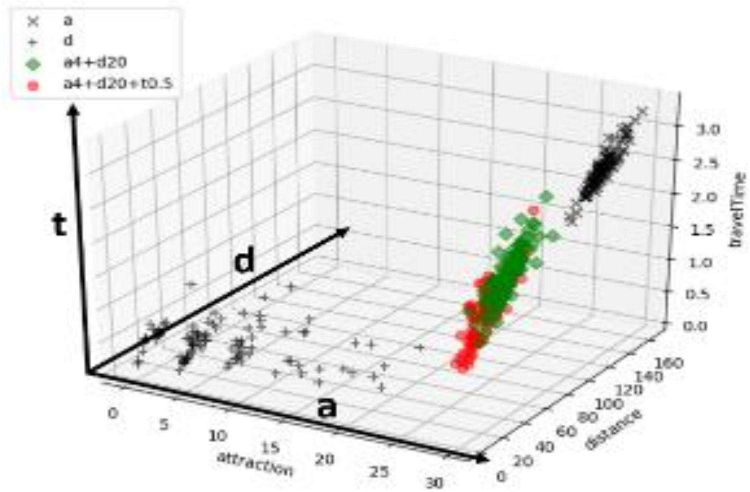
- Pessimism (non-A* algorithm)
  - Over-estimate the cost ($h(n) > h^*(n)$)
  - Do not try some possibilities
  - Probable to miss the optimal path
- Greedy Search
  - Consider local information
  - Find local optimal quickly
  - Seldom achieve global optimum

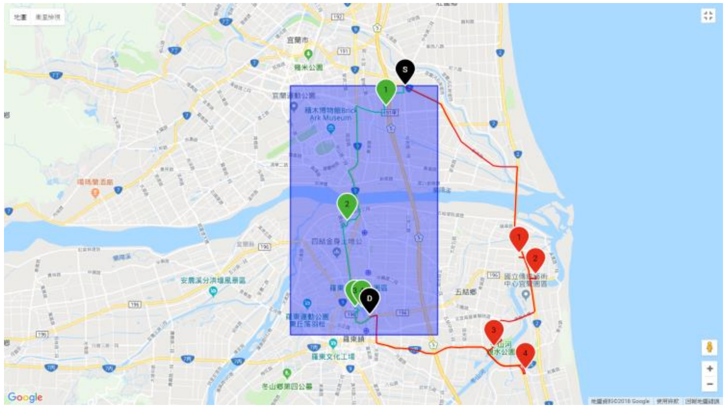
Discussions on A* Search

- A* could be generalized to find the path with minimum cost or maximum profit.
  - g(n): accumulated profits.
  - h(n): estimated profits
  - Admissible: h(n) >= h*(n) (true maximum profit)
- What if there exist multiple goals?
  - Shortest time, inexpensive, comfortable, safe, quite, etc.
- If the state space is huge, search efficiency might not be good enough even with heuristic

  → Use local search to find sub-optimal solution.

Joint Optimization





- Multiple objectives for trip planning problem
  - Time (旅程時間)
  - Distance (旅行距離)
  - Attraction (景點熱門度)
- Exceptions
  - traffic jams (塞車)

Iterative Deepening A*

- The concept of "iterative deepening" can be applied to depth-first search
  - DFS is performed with depth constraint increased in every iteration.
- A* is applied iteratively, with incrementally increasing limits on the f(n).
- The method is complete, and has a low memory requirement, like depth-first search.

Discussions

- State: could possibly be any data structure for describing the problem domains
  - e.g. multiple dimensional vector, 2-D array (puzzle), 2-D coordinate (shortest path), graphs, images, objects, …
- Goal state: could probably not be described precisely as that in the puzzle problem
  - Knapsack problem: “full”
  - Scheduling problem: no contradiction
  - Furniture setting: put the furniture to satisfactory positions
  - Space adjustment: good looking
  - Traveling: low cost, less time, within budget, have fun

SEARCH STATE: ROBOT ARM

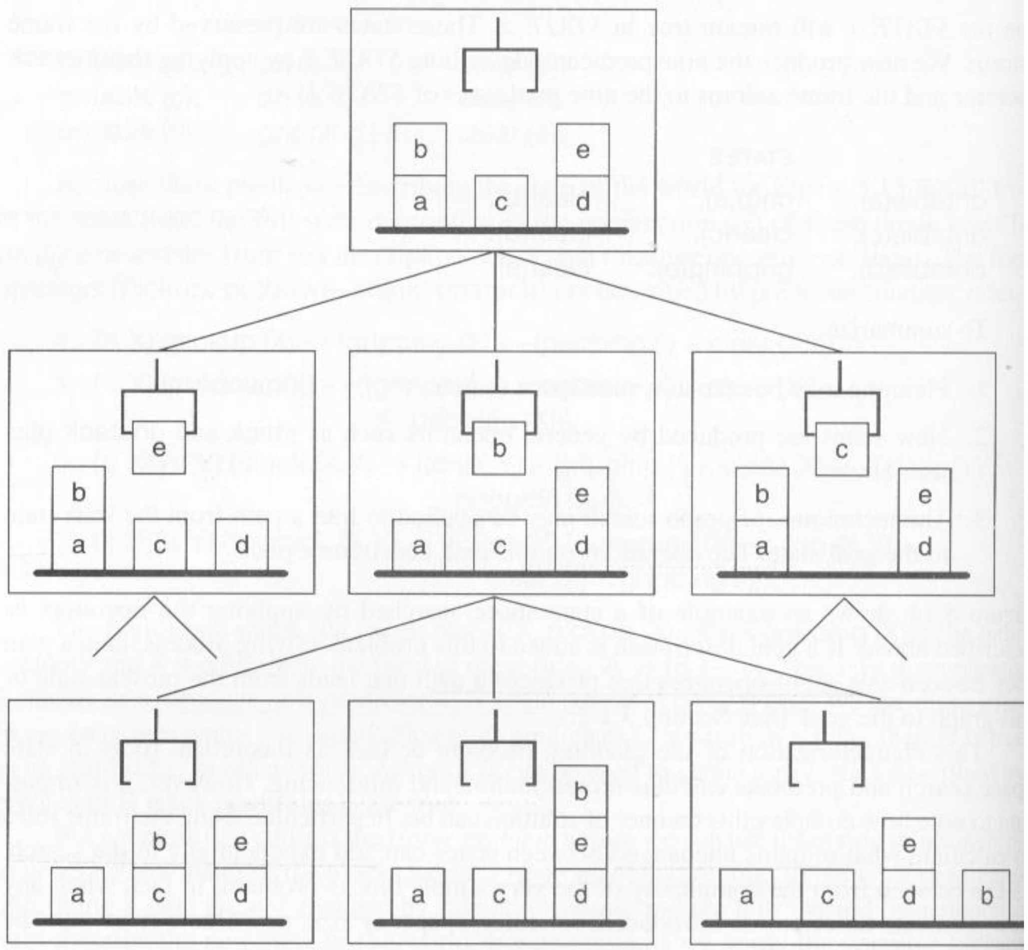


Figure 5.14 Portion of the state space for a portion of the blocks world.

SAFE-CROSSING-RIVER PROBLEM

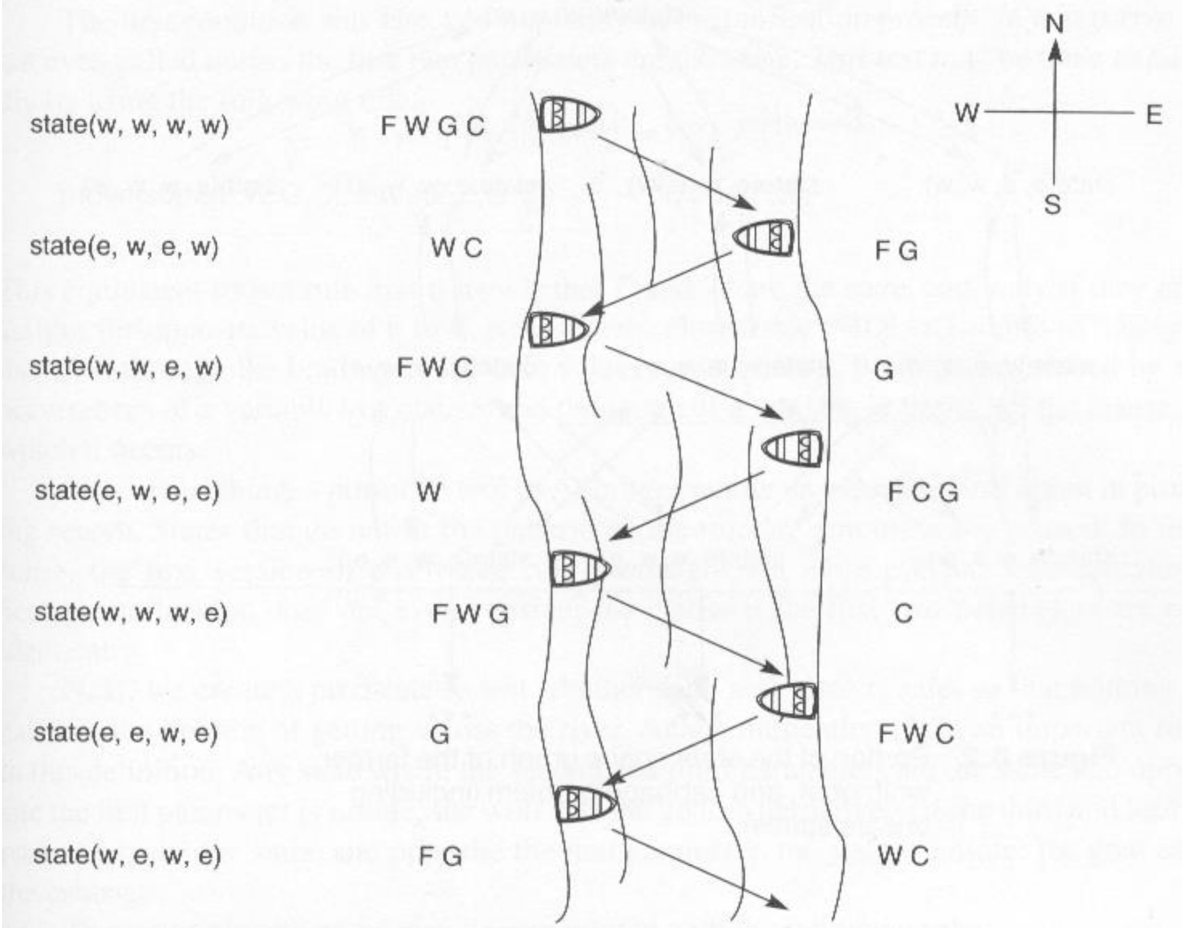
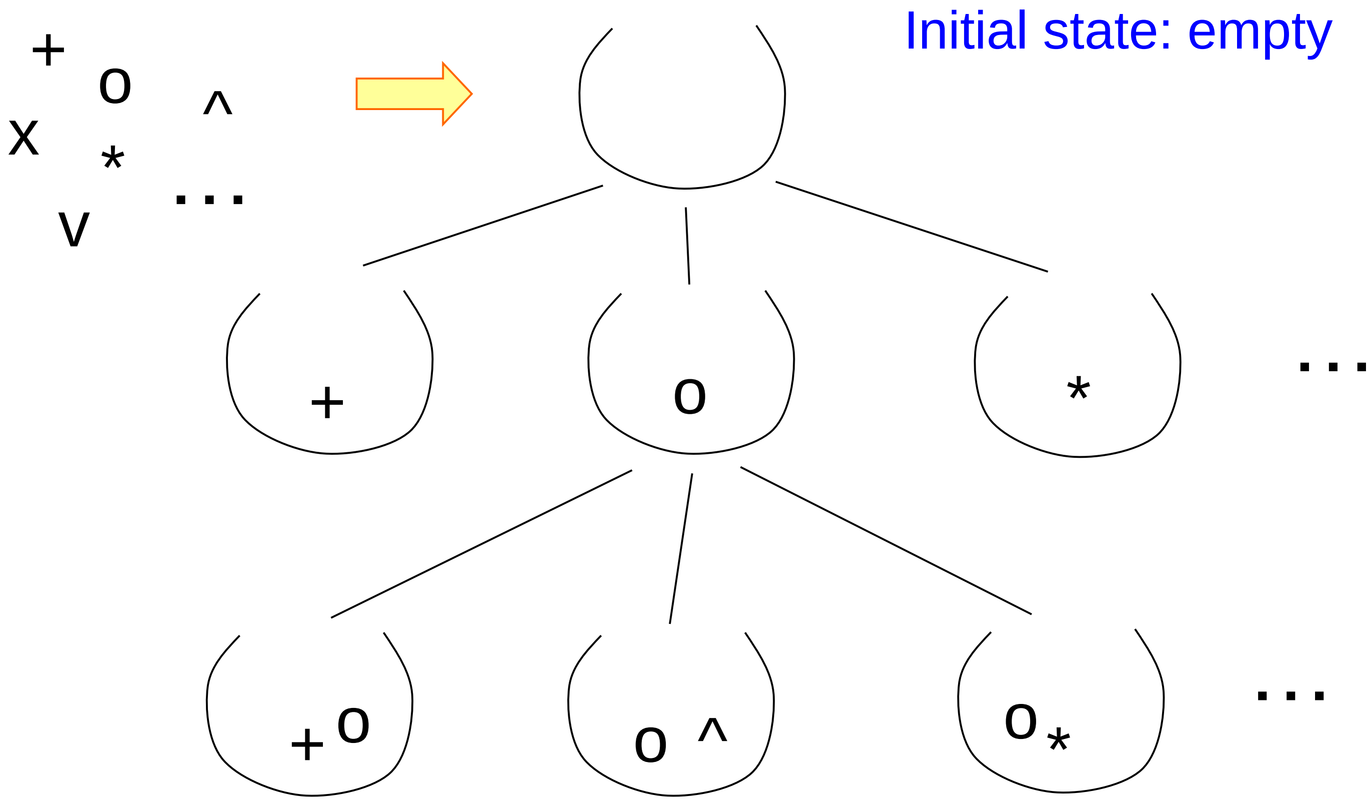


Figure 6.1 Sample crossings for the farmer, wolf, goat, and cabbage problem.

Knapsack problem

Object (value, weight)



Limit of weight W → how to add the objects with the highest values?

APPLICATIONS

- State space search
  - Can find the optimal trajectory in object space
    - the order for playing a set of images
  - Web crawlers, community discovery

    web page → linked to other pages (successors)
  - Exploring social networks
  - Trip planning
  - Sentence synthesis
- Business modeling
  - State: cash, technology, money, environment,…
  - Goal : to achieve business success
  - Rules: what might influence the state transition

Agenda

- Fundamentals of Search
- Uninformed Search
- Informed Search
- Appendix

Characteristics of Search Algorithms

- Completeness
  - A search method is complete if it is guaranteed to find a goal state provided there exists goal state.

  e.g. DFS for puzzle problem is not complete.
- Optimality
  - A search method is optimal if it is *guaranteed to find the best* solution that exists.
  - "Best" is relevant to the definition of "Goodness" .
- Irrevocability (no back tracing)
  - Methods that do not use backtracking are irrevocable.
  - Irrevocable methods, such as hill-climbing, tend to be fooled by local optima.

Characteristics of Search Algorithms

- Admissibility
  - A search algorithm is admissible if it is guaranteed to find a optimal path to a solution whenever such a path exists.
- Any algorithm A* is admissible.

Recursive Implementation of DFS

```
Function depth_recursive(state)
{
   if(is_goal(state) then return state;
   children = successors(state);
   for(each child in children) {
      result = depth_recursive(child);
      if(result != null) return result;
   }
   return null;
}
```