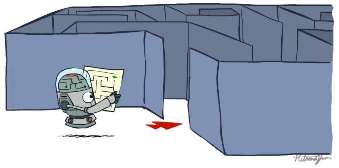
Artificial Intelligence

Module 3: Search Strategies

PART 3.1: Search
PART 3.2: Uninformed Search



Dr. Chandra Prakash
Assistant Professor
Department of Computer Science and Engineering

(Slides adapted from StuartJ. Russell, B Ravindran, Mausam, Prof. Pallab Dasgupta, Prof. Partha Pratim Chakrabarti, Saikishor Jangiti

Module 3: Search Strategies

- PART 3.1: Search
- PART 3.2: Uninformed Search
  - Depth First Search
  - Breadth First Search
  - More in Uninformed Search
- PART 3.3: Informed/Heuristic Search
- PART 3.4: Beyond Classical Search
  - Local Search
  - Problem Reduction
- PART 3.5: Constraint Satisfaction Problems
- PART 3.6: Adversarial Search

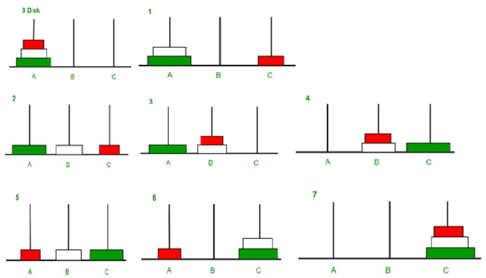
AUTOMATED PROBLEM SOLVING in AI

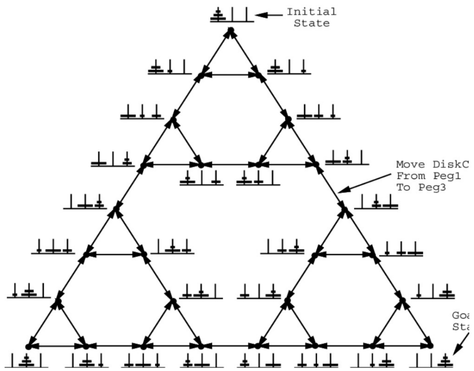
- Generalized Techniques for Solving Large Classes of Complex Problems
- Problem Statement is the Input and solution is the Output, sometimes even the problem specific algorithm or method could be the Output
- **Problem Formulation by AI Search Methods** consists of the following key concepts
  - Configuration or State
  - Constraints or Definitions of Valid Configurations
  - Rules for Change of State and their Outcomes
  - Initial or Start Configurations
  - Goal Satisfying Configurations
  - An Implicit State or Configuration Space
  - Valid Solutions from Start to Goal in the State Space
  - General Algorithms which SEARCH for Solutions in this State Space
- ISSUES
  - Size of the Implicit Space, Capturing Domain Knowledge, Intelligent Algorithms that work in reasonable time and Memory, Handling Incompleteness and Uncertainty

BASICS OF STATE SPACE MODELLING

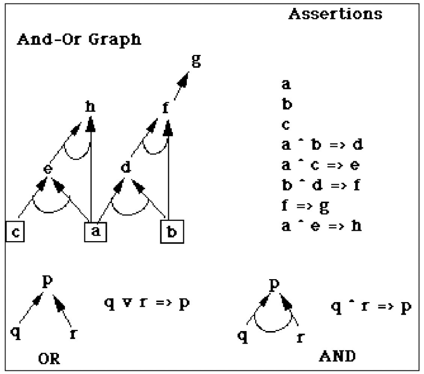
- STATE or CONFIGURATION:
  - A set of variables which define a state or configuration
  - Domains for every variable and constraints among variables to define a valid configuration
- STATE TRANSFORMATION RULES or MOVES:
  - A set of RULES which define which are the valid set of NEXT STATE of a given State
  - It also indicates who can make these Moves (OR Nodes, AND nodes, etc)
- STATE SPACE or IMPLICIT GRAPH
  - The Complete Graph produced out of the State Transformation Rules.
  - Typically too large to store. Could be Infinite.
- INITIAL or START STATE(s), GOAL STATE(s)
- SOLUTION(s), COSTS
  - Depending on the problem formulation, it can be a PATH from Start to Goal or a Sub-graph of And-ed Nodes
- SEARCH ALGORITHMS
  - Intelligently explore the Implicit Graph or State Space by examining only a small sub-set to find the solution
  - To use Domain Knowledge or HEURISTICS to try and reach Goals faster

3 DISK, 3 PEG TOWER of HANOI STATE SPACE

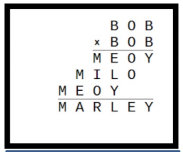




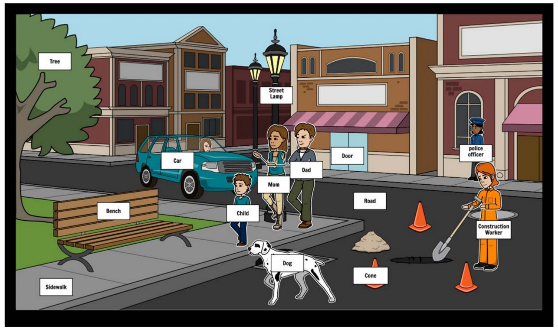
AND / OR STATE SPACES



CONSISTENT LABELLING BY CONSTRAINT SATISFACTION

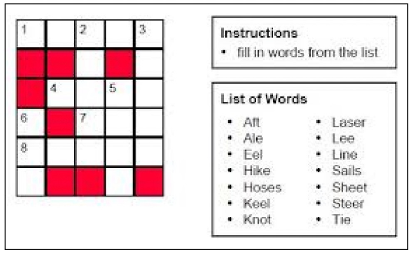


Cryptarithmetic



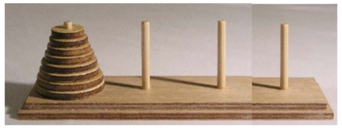
Scene Analysis

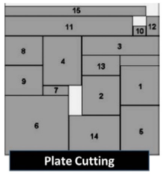
CONSISTENT LABELLING BY CONSTRAINT SATISFACTION

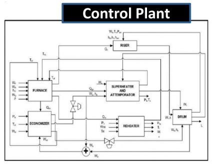


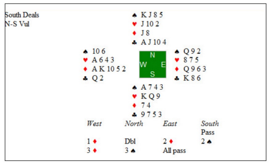
STATES, SPACES, SOLUTIONS, SEARCH

- States
  - Full / Perfect Information and Partial Information States
- State Transformation Rules
  - Deterministic Outcomes
  - Non-Deterministic / Probabilistic Outcomes
- State Spaces As Generalized Games
  - Single Player: OR Graphs
  - Multi-Player: And / Or, Adversarial, Probabilistic Graphs
- Solutions
  - Paths
  - Sub-graphs
  - Expected Outcomes
- Costs
- Sizes
- Domain Knowledge
- Algorithms for Heuristic Search









MODELLING AND/OR GRAPHS:

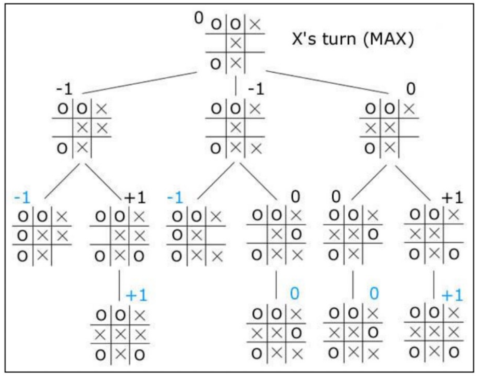
- OR Nodes are ones for which one has a choice.
- The AND nodes could be compositional (sum, product, min, max, etc., depending on the way the sub-problems are composed),

Adversarial (game where the other parties have a choice)

or

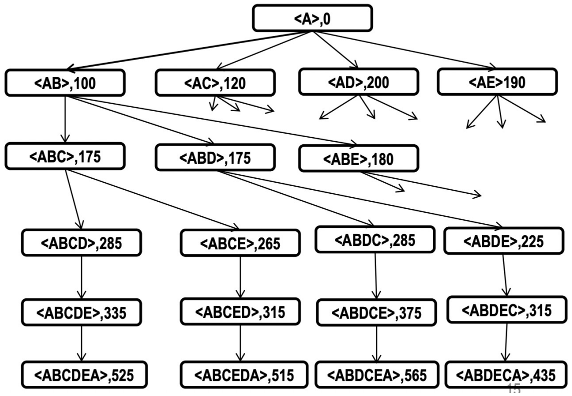
Probabilistic (Environmental Actions)

AND/OR GRAPHS: COMPOSITIONAL / ADVERSARIAL / PROBABILISTIC



SEARCHING IMPLICIT GRAPHS

- Given the start state the SEARCH Algorithm will create successors based on the State Transformation Rules and make part of the Graph EXPLICIT.
- It will EXPAND the Explicit graph INTELLGENTLY to rapidly search for a solution without exploring the entire Implicit Graph or State Space
- For OR Graphs, the solution is a PATH from start to Goal.
- Cost is usually sum of the edge costs on the path, though it could be something based on the problem



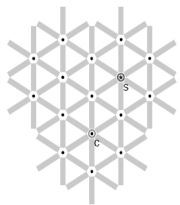
Measuring Search Algorithm Performance

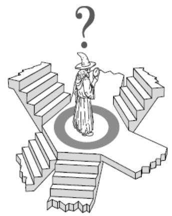
- Completeness :
  - Is the algorithm guaranteed to find a solution when there is one?
- Optimality :
  - Does the algorithm find the optimal solution?
- Time Complexity :
  - How long does it take to find the solution?
- Space Complexity :
  - How much memory is needed to perform the search?
- Possibility to backtrack
- Informedness

- **Time and space complexity**
  - Search in AI is represented by initial state, actions and transitions which usually result in infinite Nodes and Edges in a graph.Hence, the complexity is rather measured by
    - Branching Factor (b): Maximum number of successors of any node
    - Depth (d) of the shallowest goal, i.e., number of steps from initial node
    - Maximum length (m) of any path in state space (may be ∞)

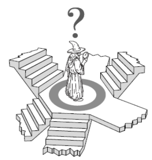
Search

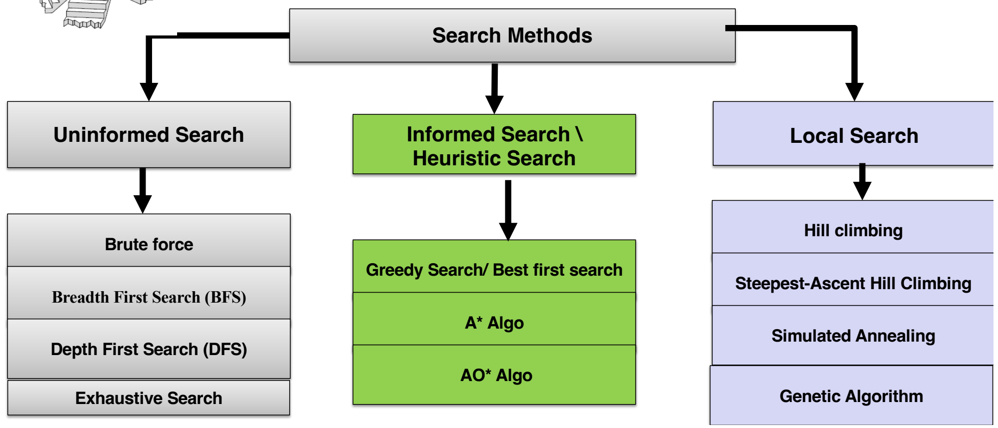
- Process of locating a solution to a problem by systematically looking at nodes in a search tree or a search space until a goal node is found.
- a class of techniques for systematically finding or constructing solutions to problems.
- Many (all?) AI problems can be formulated assearch problems!
- Examples:
  - Path planning
  - Games
  - Natural Language Processing
  - Machine learning





Type of Search Methods





Uninformed search strategies

- Uninformed: While searching you have no clue whether **one non-goal state is better than any other.** Your search is blind.
- Also known as blind search
- Covered problems that considered the whole search space and produced a sequence of actions leading to a goal.
- Various blind strategies:
  - Brute force
  - Depth-first search
  - Breadth-first search
  - Uniform-cost search
  - Iterative deepening search

1. Brute Force /Generate-and-Test

- Try all possibilities
- Acceptable for simple problems.
  - Eg : finding key of a 3 digit lock.

- Inefficient for problems with large space.
- Use DFS as all possible solution generated, before they can be tested.

Algorithm

1. Generate a possible solution.
2. Test to see if this is actually a solution.
3. Quit if a solution has been found. Else, return to step 1.

Generate-and-Test: 8-puzzle

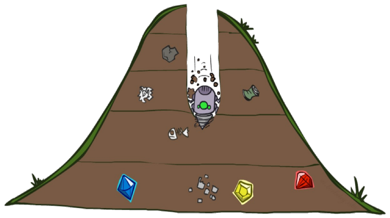
1	4	3
7		6
5	8	2

1	2	3
8		4
7	6	5

2. Depth First Search (DFS)

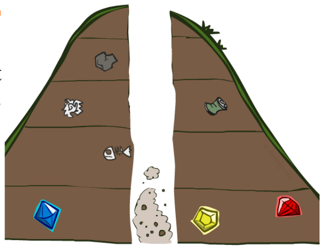
Algorithm:

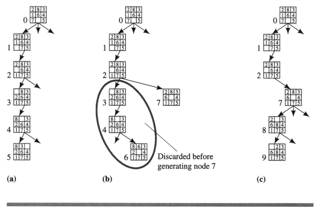
1. [Initialize] Initially the OPEN List contains the Start Node s. CLOSED List is Empty.
2. [Select] Select the first Node n on the OPEN List. If OPEN is empty, Terminate
3. [Goal Test] If n is Goal, then decide on Termination or Continuation / Cost Updation
4. [Expand]
   a) Generate the successors n_1, n_2, …. n_k, of node n, based on the State Transformation Rules
   b) Put n in LIST CLOSED
   c) For each n_i, not already in OPEN or CLOSED List, put n_i in the FRONT of OPEN List
   d) For each n_i already in OPEN or CLOSED decide based on cost of the paths
5. [Continue] Go to Step 2



Backtracking

- A variant of depth-first search
- In this search, we pursue a single branch of the tree until it yields a solution or until a decision to terminate the path is made.
- It makes sense to terminate a path if it reaches dead-end, produces a previous state. In such a state backtracking occurs
- Chronological Backtracking:
  - Order in which steps are undone depends only on the temporal sequence in which steps were initially made.
  - Specifically most recent step is always the first to be undone.
  - This is also simple backtracking.





Generation of the First Few Nodes in a Depth-First Search

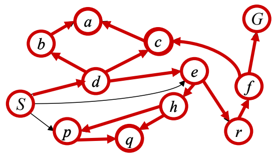
Depth-First Search

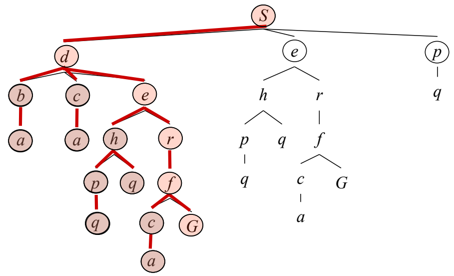
Strategy:

- expand a deepest node first

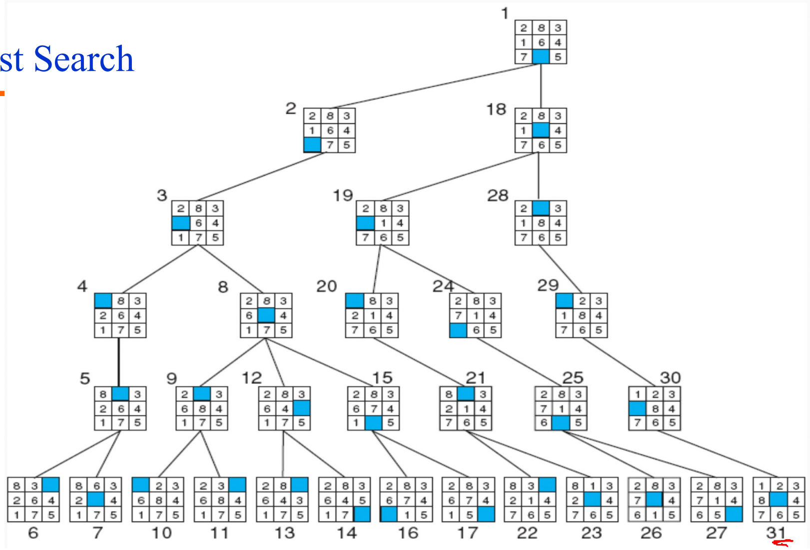
Implementation:

- Fringe is a LIFO stack





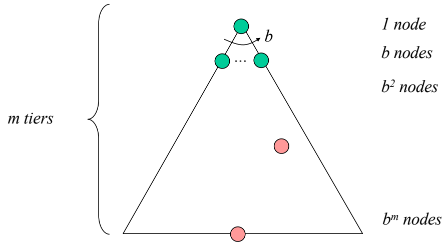
Depth-First Search



Recall : Search Algorithm Properties

- Complete: Guaranteed to find a solution if one exists?
- Optimal: Guaranteed to find the least cost path?
- Time complexity?
- Space complexity?

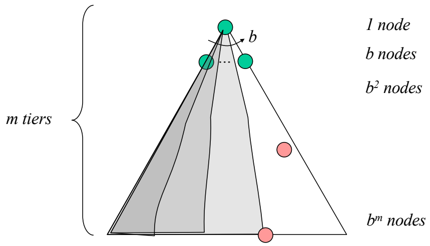
- Cartoon of search tree:
  - b is the branching factor
  - m is the maximum depth
  - solutions at various depths



- Number of nodes in entire tree?
  - $1 + b + b^2 + \ldots.\ b^m = O(b^m)$

Depth-First Search (DFS) Properties

- **What nodes DFS expand?**
  - Some left prefix of the tree.
  - Could process the whole tree!
- How much time does the fringe take?
  - If m is finite, takes time $O(b^m)$,
  - m=maximum depth
- How much space does the fringe take?
  - Only has siblings on path to root, so O(bm)
- Is it complete?
  - No: fails in infinite-depth spaces
  - Can modify to avoid repeated states along path
  - m could be infinite, so only if we prevent cycles (more later)
- Is it optimal?
  - No, it finds the "leftmost" solution, regardless of depth or cost
  - It may find a non-optimal goal first



Comments on Depth-First Search

Advantages

- DFS requires less memory since only the nodes on the current path are stored.
- By chance, DFS may find a solution without examining much of the search space at all.
- if solutions are dense, may be much faster than breadth-first

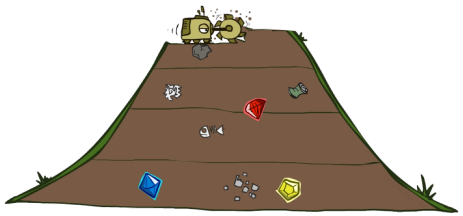
Drawback

- The major drawback of DFS is the determination of the depth until which the search has to proceed. This depth is called cut-off depth. The value of cut-off depth is essential because otherwise the search will go on and on.
- If cut-off depth is small, solution may not found and if cut-off depth is large, time-complexity will be more.

3. Breadth First Search

Algorithm:

1. [Initialize] Initially the OPEN List contains the Start Node s. CLOSED List is Empty.
2. [Select] Select the first Node n on the OPEN List. If OPEN is empty, Terminate
3. [Goal Test] If n is Goal, then decide on Termination or Continuation / Cost Updation
4. [Expand]
   - a) Generate the successors n_1, n_2, …. n_k, of node n, based on the State Transformation Rules
   - b) Put n in LIST CLOSED
   - c) For each n_i, not already in OPEN or CLOSED List, put n_i in the END of OPEN List
   - d) For each n_i already in OPEN or CLOSE decide based on cost of the paths
5. [Continue] Go to Step 2



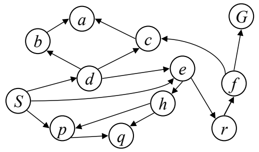
Breadth-First Search

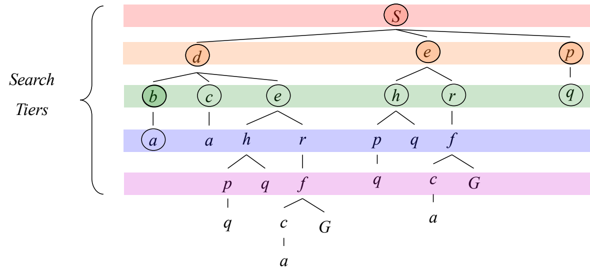
Strategy:

- expand a shallowest node first

Implementation:

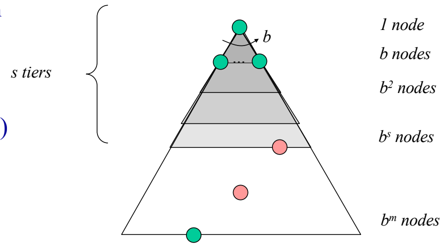
- Fringe is a FIFO queue



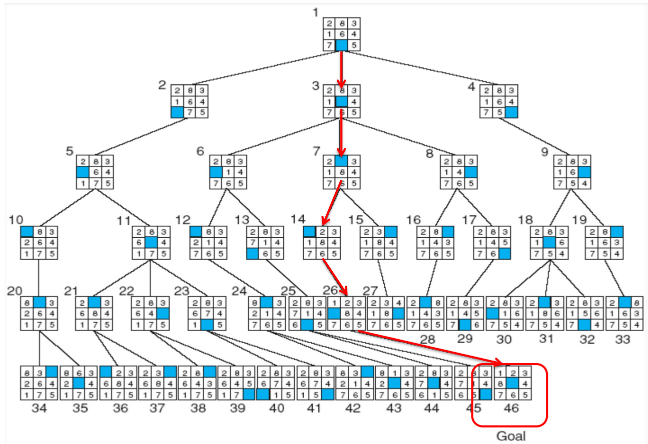


Breadth-First Search (BFS) Properties

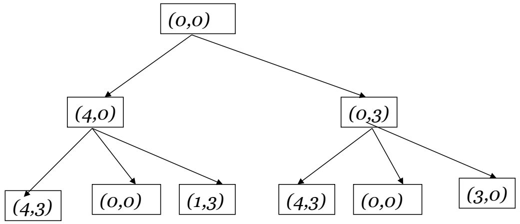
- What nodes does BFS expand?
  - Processes all nodes above shallowest solution
  - Let depth of shallowest solution be s
- How much time does the fringe take?
  - $1+b+b^2+b^3+\ldots+b^s+(b^{s+1}-b)) = O(b^s)$
- How much space does the fringe take?
  - Has roughly the last tier, so $O(b^s)$
- Is it complete?
  - Yes it always reaches goal (if *b* is finite)
  - s must be finite if a solution exists
- Is it optimal?
  - Only if costs are all 1 (more on costs later)



Breadth First Search



BFS Tree for Water Jug problem



Comments on BFS

Advantages

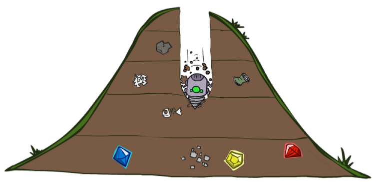
- BFS will not get trapped exploring a blind alley.
- If there is a solution, BFS is guaranteed to find it.
- If there are multiple solutions, then a minimal solution will be found.

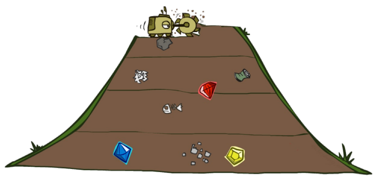
Limitation

- Amount of time needed to generate all the nodes is considerable because of the time complexity.
- Memory constraint is also a major hurdle because of the space-complexity.
- The searching process remembers all unwanted nodes which is of no practical use for the search.

DFS vs BFS

- When will DFS outperform BFS?





- When will BFS outperform DFS?

Video of Demo Maze Water DFS/BFS

States light up first time explored. Which one?
1st- Breadth
2nd - Depth

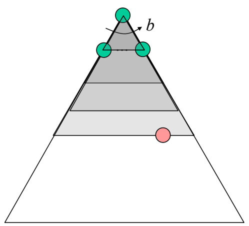
More in Uninformed Search

Infinite depth Problem

- To avoid the infinite depth problem of DFS, we can decide to only search until depth L, i.e. we don't expand beyond depth L.
- → Depth-Limited Search
  - What of solution is deeper than L? → Increase L iteratively.
- → Iterative Deepening Search
- As we shall see: this inherits the memory advantage of Depth-First search.

Iterative Deepening

- Idea: get DFS's space advantage with BFS's time / shallow-solution advantages
  - Run a DFS with depth limit 1. If no solution…
  - Run a DFS with depth limit 2. If no solution…
  - Run a DFS with depth limit 3. …..



- Isn't that wastefully redundant?
  - Generally most work happens in the lowest level searched, so not so bad!

Iterative deepening search

- Number of nodes generated in a depth-limited search to depth d with branching factor b:

$$N_{DLS} = b^0 + b^1 + b^2 + \ldots + b^{d-2} + b^{d-1} + b^d$$

- Number of nodes generated in an iterative deepening search to depth d with branching factor b:

$$N_{IDS} = (d+1)b^0 + d\,b^1 + (d-1)b^2 + \ldots + 3b^{d-2} + 2b^{d-1} + 1b^d =$$

$$O(b^d) \neq O(b^{d+1})$$

BFS

- For $b = 10$, $d = 5$,
  - $N_{DLS} = 1 + 10 + 100 + 1{,}000 + 10{,}000 + 100{,}000 = 111{,}111$
  - $N_{IDS} = 6 + 50 + 400 + 3{,}000 + 20{,}000 + 100{,}000 = 123{,}450$
  - N_{BFS} = ... = 1,111,100

Properties of iterative deepening search

- Complete? Yes
- Time? $(d+1)b^0 + d\,b^1 + (d-1)b^2 + \ldots + b^d = O(b^d)$
- Space? $O(bd)$
- Optimal? Yes, if step cost = 1 or increasing function of depth.

Bidirectional Search

- Idea
  - simultaneously search forward from S and backwards from G
  - stop when both "meet in the middle"
  - need to keep track of the intersection of 2 open sets of nodes
- What does searching backwards from G mean
  - need a way to specify the predecessors of G
    - this can be difficult,
    - e.g., predecessors of checkmate in chess?
  - what if there are multiple goal states?
  - what if there is only a goal test, no explicit list?

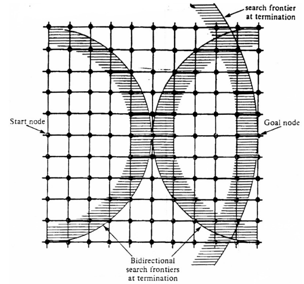
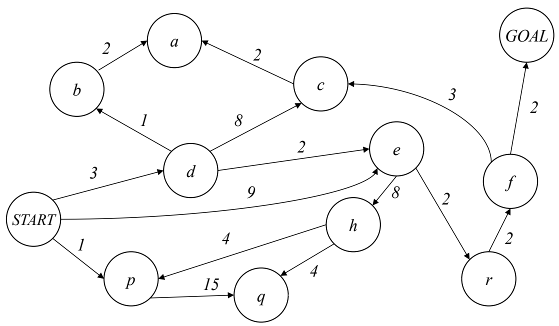


Fig. 2.10 Bidirectional and unidirectional breadth-first searches.

Complexity: time and space complexity are: $O(b^{d/2})$

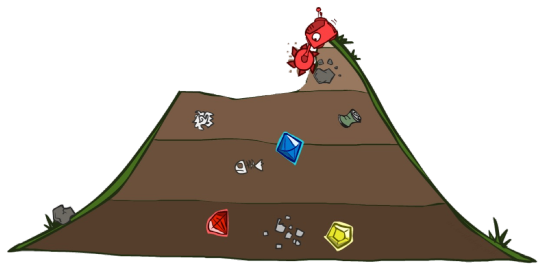
Cost-Sensitive Search



BFS finds the shortest path in terms of number of actions.
It does not find the least-cost path. We will now cover
a similar algorithm which does find the least-cost path.

How?

Uniform Cost Search (UCS) / Dijkstra's Algorithm



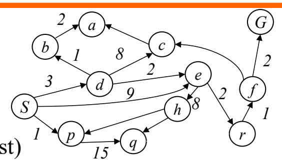
Uniform Cost Search (UCS)

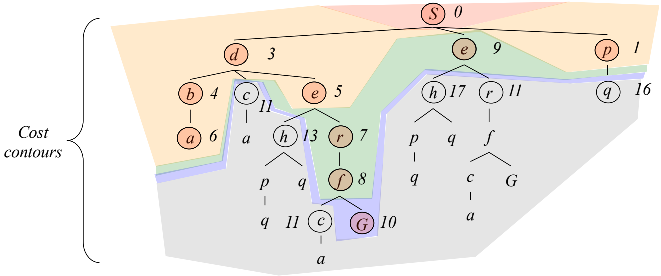
Strategy:

- expand a cheapest node first:

Implementation:

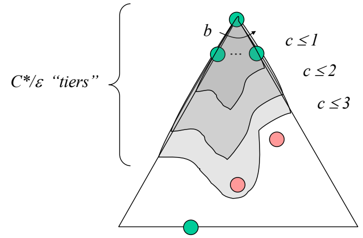
- Fringe is a priority queue (priority: cumulative cost)





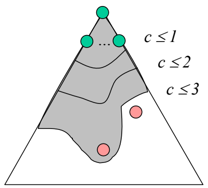
Uniform Cost Search (UCS) Properties

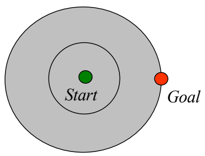
- What nodes does UCS expand?
  - Processes all nodes with cost less than cheapest solution!
  - If that solution costs C^* and arcs cost at least ε, then the "effective depth" is roughly C^*/ε
  - Takes time $O(b^{C^*/\varepsilon})$ (exponential in effective depth)
- How much space does the fringe take?
  - Has roughly the last tier, so $O(b^{C^*/\varepsilon})$
- Is it complete?
  - Assuming best solution has a finite cost and minimum arc cost is positive, yes!
- Is it optimal?
  - Yes! (Proof next lecture via A*)



Uniform Cost Issues

- Remember: UCS explores increasing cost contours
- The good: UCS is complete and optimal!
- The bad:
  - Explores options in every "direction"
  - No information about goal location
- We'll fix that soon!





[Demo: empty grid UCS (L2D5)]
[Demo: maze with deep/shallow water DFS/BFS/UCS (L2D7)]

Video of Demo Empty UCS

Video of Demo Maze with Deep/Shallow Water --- DFS, BFS, or UCS?

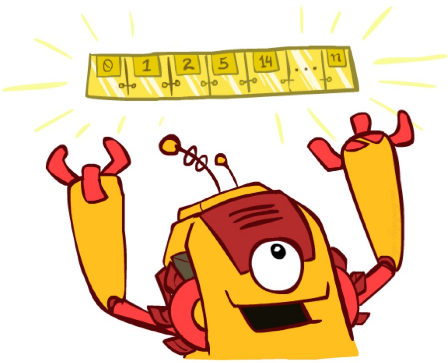
1. is BFS L2D7 B U D
2. is UCS : expansion slows down when you hit deep water
3. This is DFS

Summary of algorithms

Criterion	Breadth-First	Uniform-Cost	Depth-First	Depth-Limited	Iterative Deepening
Complete?	Yes	Yes	No	No	Yes
Time	$O(b^{d+1})$	$O(b^{\lceil C^*/\epsilon \rceil})$	$O(b^m)$	$O(b^l)$	$O(b^d)$
Space	$O(b^{d+1})$	$O(b^{\lceil C^*/\epsilon \rceil})$	$O(bm)$	$O(bl)$	$O(bd)$
Optimal?	Yes	Yes	No	No	Yes

The One Queue

- All these search algorithms are the same except for fringe strategies
  - Conceptually, all fringes are priority queues (i.e. collections of nodes with attached priorities)
  - Practically, for DFS and BFS, you can avoid the log(n) overhead from an actual priority queue, by using stacks and queues
  - Can even code one implementation that takes a variable queuing object



Branch and Bound

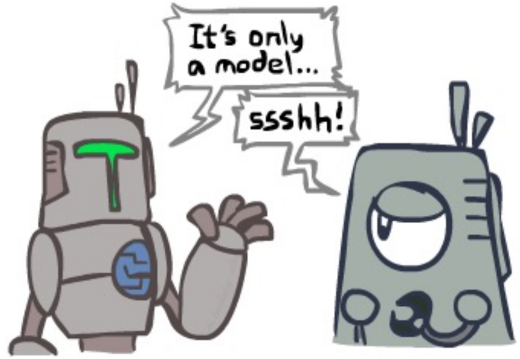
- Begin generating complete paths, keeping track of the shortest path found so far.
- Give up exploring any path as soon as its partial length becomes greater than the shortest path found so far.
- Using this algorithm, we are guaranteed to find the shortest path.
- It still requires exponential time.
- The time it saves depends on the order in which paths are explored.

Take-away

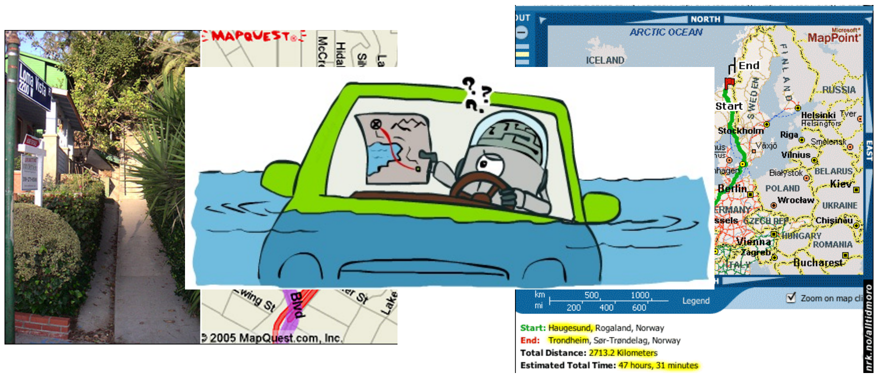
- Problem formulation usually requires abstracting away real-world details to define a state space that can feasibly be explored
- Variety of uninformed search strategies
  - Breadth-First Search: Queue is FIFO.
  - Uniform-Cost Search: Queue explores cheapest descendant first.
  - Depth-First Search: Queue picks deepest remaining. (Can be implemented differently.)
  - Depth-Limited: DFS with a bound.
  - Iterative Deepening: Keep increasing the bound.
  - Bidirectional Search: Run two searches, one backward from "the goal."
- Iterative deepening search uses only linear space and not much more time than other uninformed algorithms

Search and Models

- Search operates over models of the world
  - The agent doesn't actually try all the plans out in the real world!
  - Planning is all "in simulation"
  - Your search is only as good as your models…



Search Gone Wrong?

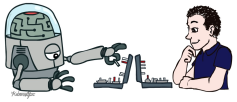


Next :

- Module 3: Search Strategies
  - PART 3.1: Search
  - PART 3.2: Uninformed Search
    - Depth First Search
    - Breadth First Search
    - More in Uninformed Search
  - PART 3.3: Informed/Heuristic Search
  - PART 3.4: Beyond Classical Search
    - Local Search
    - Problem Reduction
  - PART 3.5: Constraint Satisfaction Problems
  - PART 3.6: Adversarial Search

References

- Stuart Russell, Peter Norvig, Artificial intelligence : A Modern Approach, Prentice Hall
- Artificial Intelligence by Elaine Rich & Kevin Knight, Third Ed, Tata McGraw Hill
- Artificial Intelligence and Expert System by Patterson
- http://www.cs.rmit.edu.au/AI-Search/Product/
- http://aima.cs.berkeley.edu/demos.html (for more demos)
- Artificial Intelligence and Expert System by Patterson
- Slides adapted from CS188 Instructor: Anca Dragan, University of California, Berkeley
- Slides adapted from CS60045 ARTIFICIAL INTELLIGENCE



(some slides adapted from http://aima.cs.berkeley.edu/)