VICTORIA UNIVERSITY OF WELLINGTON
TE HERENGA WAKA

School of Engineering and Computer Science
Te Kura Mātai Pūkaha, Pūrorohiko

COMP 307/AIML 420 — Lectures 02 and 03

Problem Solving and Search Techniques

Dr Andrew Lensen

Andrew.Lensen@ecs.vuw.ac.nz

(with thanks to Prof. Mengjie Zhang)

Housekeeping

- I'm lecturing the first six weeks of the course
- This half of the course is assessed by A1 and A2, the first test and part of the second test
- No strict office hours. To meet with me, please book via https://tinyurl.com/bookLensen — but consider helpdesks first!
- If you need to email me, please include [COMP307] in the subject, so it gets prioritised accordingly
- Questions? Use the forum! Others will have the same ones
- All extension requests go to Dr Yi Mei via the Extension system
- *Akoranga*: constructive feedback is always welcome!
- Anything else upfront?

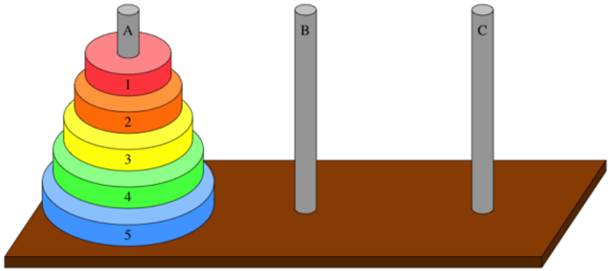
Outline

- Why Search?
- Search strategies
- Uninformed/Blind search
- Informed search/Heuristic search
- Local search: Hill Climbing
- Local search in continuous space
- Genetic beam search
- Advanced discussions
- Game Playing

Why Search (1)

Many puzzle and game playing problems need search:

- The Monkey and Bananas Problem
- The Missionaries and Cannibals Problem
- The 8-puzzle
- The Tower of Hanoi
- Wolf, Goat and Cabbage
- Water jug
- Route finding



- Chess, Bridge, Go (AlphaGo, AlphaZero,...all using search!)

Why Search (2)

Many real-world complex and engineering problems need search:

- Touring problems
- Travelling Salesperson Problem (TSP)
- Robot navigation
- University timetabling
- Job shop scheduling

Search is used in almost all AI techniques such as machine learning (ML) and evolutionary computation (EC)

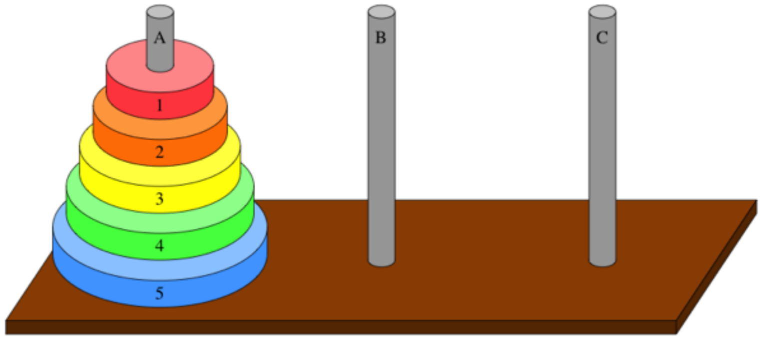
State Space Search

Problem solving as State Space Search

- State: a state of the world
  - State space: Collection of all possible states
  - Initial state: where the search starts
  - Goal state: where the search stops
- Operators: links between pairs of states, i.e. what actions can be taken in any state to get to a new state
- A path in the state space is a sequence of operators leading from one state to another
- Solve problem by searching for path from initial state to goal state through legal states
- Each operator has an associated cost
  - Path cost: the sum of the costs of operators along the path

An Example: Towers of Hanoi

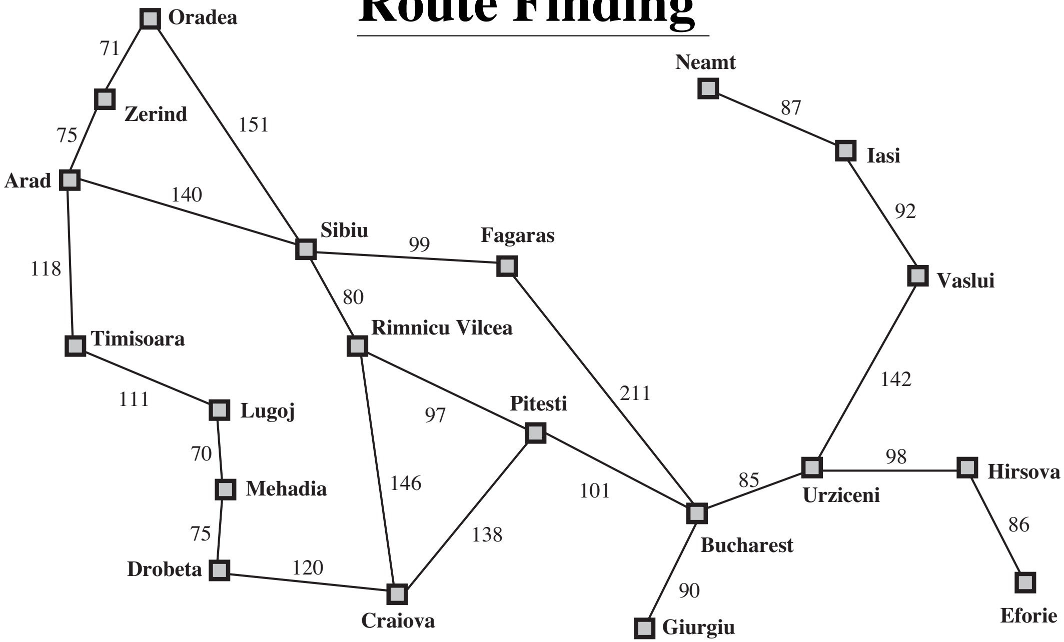
(...or "Jacinda's Beehives")



Formulating problems:

- Initial State: All disks on A. Goal State: All disks on C.
- States: location of the 5 discs
- Operators: moving a disc to another pole (given constraints)
- Path cost: number of moves
- Represent as a tree/graph?

Route Finding



- State: current location on map (part of Romania)
  - Initial state: city A
  - Goal state: city B
- Operators: move along a road to another location
- Path cost: sum on lengths of road

General Search Algorithm

For general tree-search:

```
Initialise the frontier using the initial state of the  problem
 loop
   If the frontier/fringe is empty
     then return failure
   choose a leaf node and remove it from the frontier
   If the node represents the goal state
     then return the corresponding solution (as success)
   else expand the node and put the children notes on the frontier
              (and ordering)
 end loop
```

For general graph-search, similar (see text book)

Search Variants and Performance Measure

Three search variants:

- FIFO (queue): pop the oldest element
- FILO (stack): pop the blue newest element
- Priority queue: pop the element with the highest priority based on some ordering function

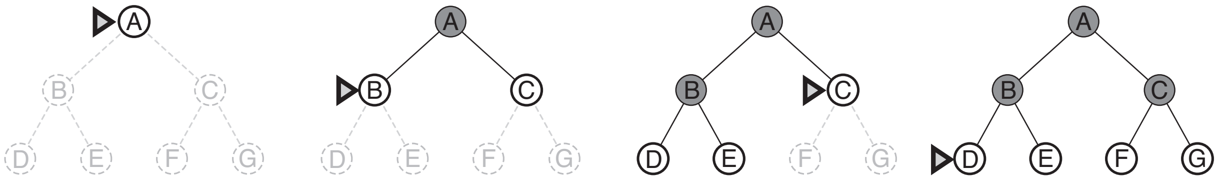
Four measures:

- Completeness: is the strategy **guaranteed** to find a solution when one exists?
- Optimality: does the strategy find the **highest-quality** solution (lowest cost) when there are several solutions?
- Time complexity: **how long** does it take to find a solution?
- Space complexity: **how much memory** does it need to perform the search?

Search Strategies

- Uninformed (blind) search
  - Breadth first
  - Uniform cost
  - Depth first
  - Depth limited
  - Iterative deepening
  - Bidirectional
- Informed (Heuristic) search
  - Greedy best-first search
  - A* search
- Beyond classic search (*Beyond COMP261 — core to AI/ML!*)
  - Hill climbing
  - Gradient descent
  - Simulated Annealing
  - Beam search
  - Bound and bound (x)
  - dynamic programming (x)

Breadth First Search



- Start at the initial state;
- Find all states reachable from the initial state;
- Find all states reachable from the states at the previous step;
- Repeat the previous step until the final state is reached

Issues for Breadth-first Search

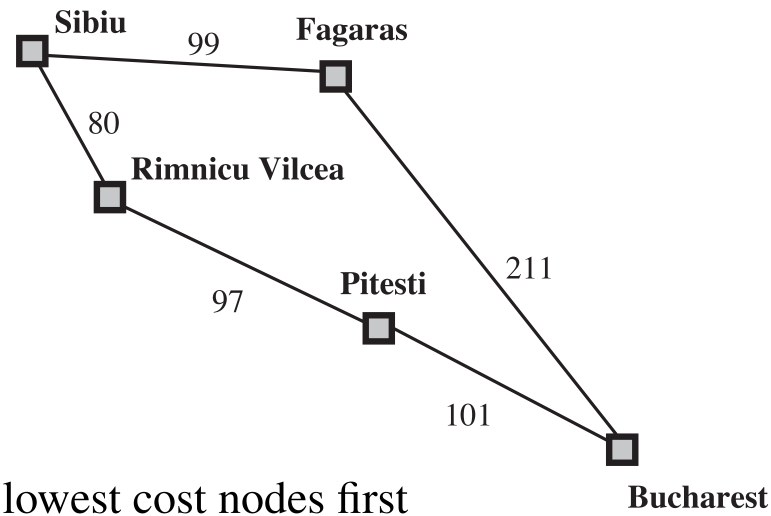
- Breadth-first search is guaranteed to find the shortest path
  – Complete
  – Optimal if all operators have same cost
  (so shallowest solution is cheapest solution)

- In practice, breadth-first search is very expensive
  – Time complexity $O(b^d)$ *Remember big-Oh?*
  – Space complexity $O(b^d)$

- b: the branching factor of nodes (i.e., average number of children a node has)

- d: the depth of the desired solution

Uniform Cost Search



- Expand lowest cost nodes first
- Same as breadth-first search if all operators have the same cost
- Complete
- Optimal if all operators have positive cost *(Why?)*
- Time Complexity $O(b^d)$
- Space Complexity $O(b^d)$

Depth-first Search

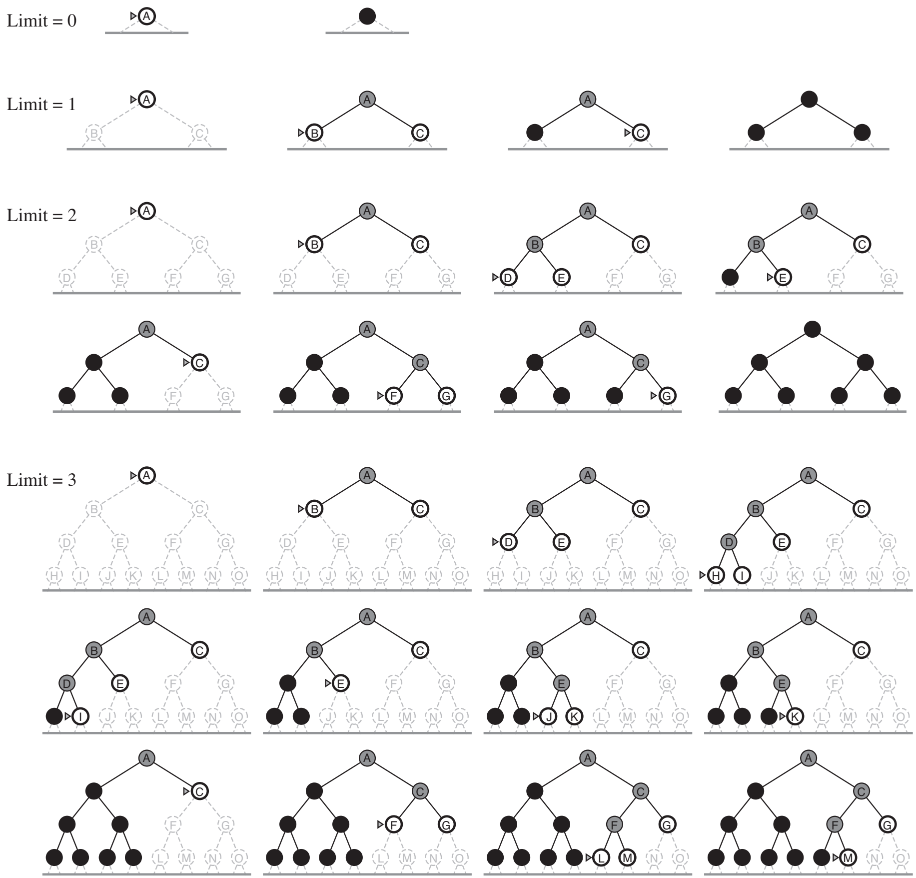
- Make the initial state the current state;
- If current state is a final state then succeed
  – otherwise, update the current state to be another state reachable from the current state;
- Repeat previous step until success, backtracking if necessary

Issues for DFS and Depth Limited Search

- Depth-first search is prone to being lost
  - Complete only if search tree is *finite*
  - Not optimal
- In practice, depth-first search is (more) efficient
  - Time Complexity $O(b^m)$
  - Space Complexity $O(bm)$ (m=max depth of search tree)
- Good when many solutions in deep (but finite) trees

- **Depth Limited Search:** Like depth-first, but with depth cut off
- Complete iff solution is at depth $<=$ c where c is the depth limit
- Can be used to search infinite search trees (e.g. cyclic graphs)
- Not optimal; Time complexity $O(b^c)$; Space complexity $O(bc)$

Iterative Deepening (Depth-first) Search



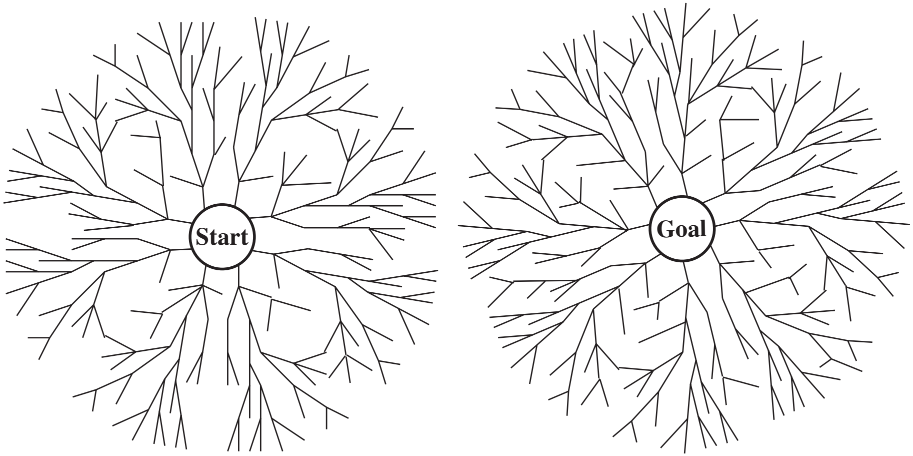
Iterative Deepening Search

- Optimal and Complete (if operators all have same cost)
- Time complexity $O(b^d)$
- Space complexity $O(bd)$

- Expands some nodes multiple times
  – But "wasted" effort is actually quite small and goes down as the branching factor goes up

- In general, iterative deepening is the preferred search method on a large search space when the depth of the solution is unknown.
- **Homework:** What are the main differences between iterative deepening search and breadth-first search?

Bidirectional Search



- Two breadth-first search: one forward from the initial state and the other backward from the goal state
- Complete and optimal
- Time complexity $O(b^{d/2})$
- Space complexity $O(b^{d/2})$
- Not applicable for some problems (*Such as?*)

Heuristic Search

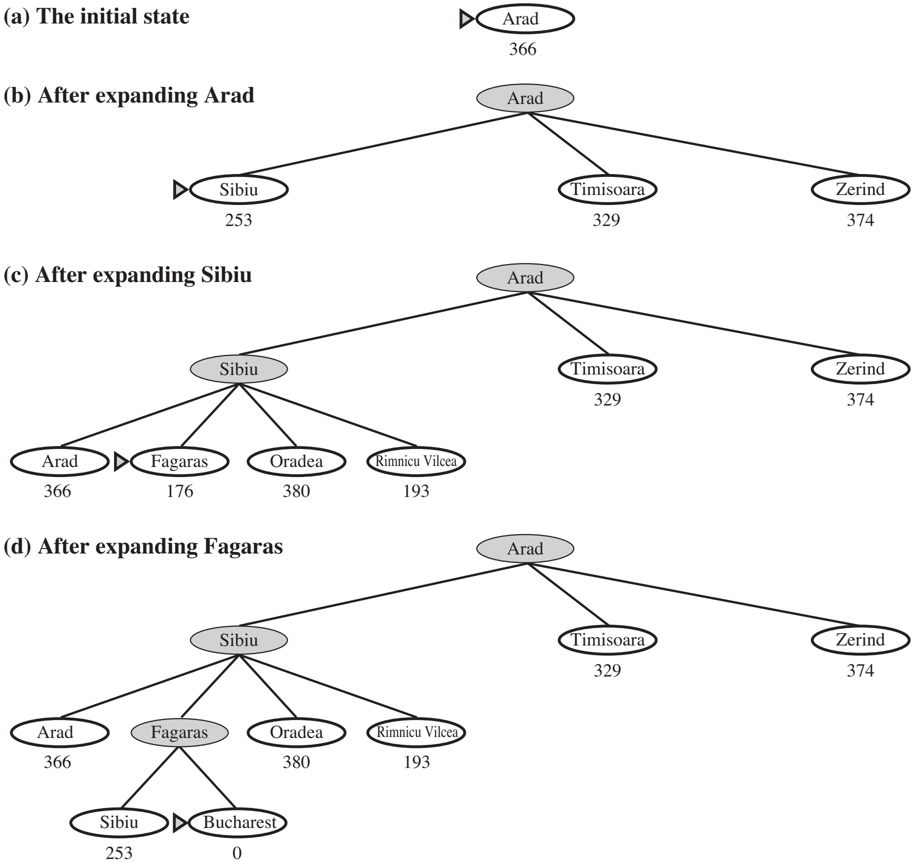
- Looking ahead and estimating cost to goal
  - no information: blind search
  - perfect information: search is easy
  - partial information: look ahead gives *fuzzy* picture
- A heuristic function estimates the cost from the current state to the goal state
- Why Heuristics?
  - Chess: b=35 d=100 (*average*)
  - Go: b=250, d=?
  - Magic: The Gathering: b=∞...
  - (b: branching factor; d: the depth of the desired solution)
- h(n) = estim. cost of the cheapest path from node n to goal state
- Admissible: h(n) **never** overestimates the cost to reach the goal
  - Route Finding: straight line distance
  - Find heuristics by *relaxing* the problem

Greedy (Best First) Search

- Minimize estimated cost to reach the goal, i.e. always expand node whose state looks closest to the goal state (Bucharest)
- f(n) = h(n)
- Route Finding — straight line distance:

Arad	366	**Mehadia**	241
Bucharest	0	**Neamt**	234
Craiova	160	**Oradea**	380
Drobeta	242	**Pitesti**	100
Eforie	161	**Rimnicu Vilcea**	193
Fagaras	176	**Sibiu**	253
Giurgiu	77	**Timisoara**	329
Hirsova	151	**Urziceni**	80
Iasi	226	**Vaslui**	199
Lugoj	244	**Zerind**	374

Greedy (Best First) Search



Issues for Greedy search

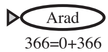
- Not optimal, not complete
- Can go down false paths
- Worst case time & space complexity is $O(b^m)$ (m: maximum depth of the search tree)
- In practise, nonetheless, can work well (*e.g. big problems!*)

- Does not consider path cost g(n): cost from the start node to node n

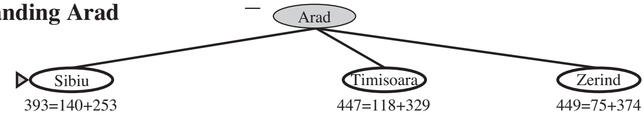
A* search

- A* attempts to minimize total estimated path cost:
- $f(n) = g(n) + h(n)$
- g(n): cost from the start node to node n
- h(n): estimated cost of the cheapest path from node n to the goal
- f(n): estimated cost of the cheapest solution through n

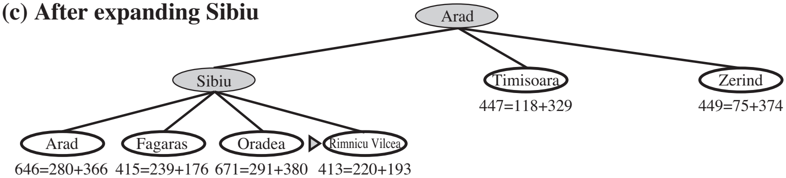
(a) The initial state



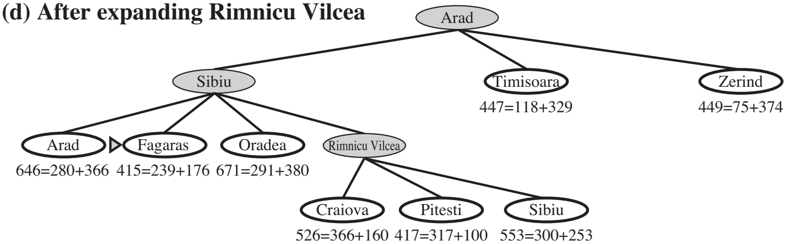
(b) After expanding Arad



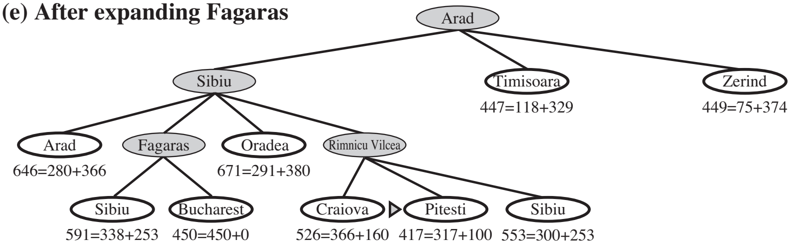
(c) After expanding Sibiu



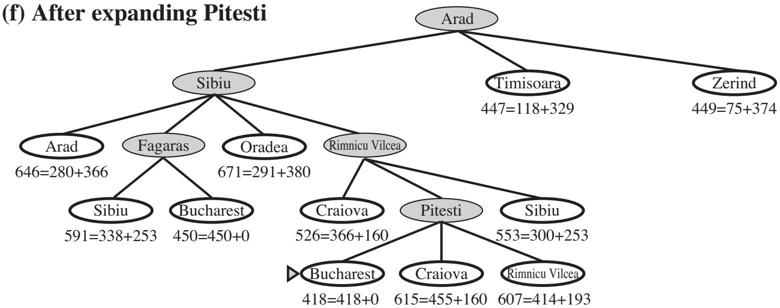
(d) After expanding Rimnicu Vilcea



(e) After expanding Fagaras



(f) After expanding Pitesti



Issues for A* search

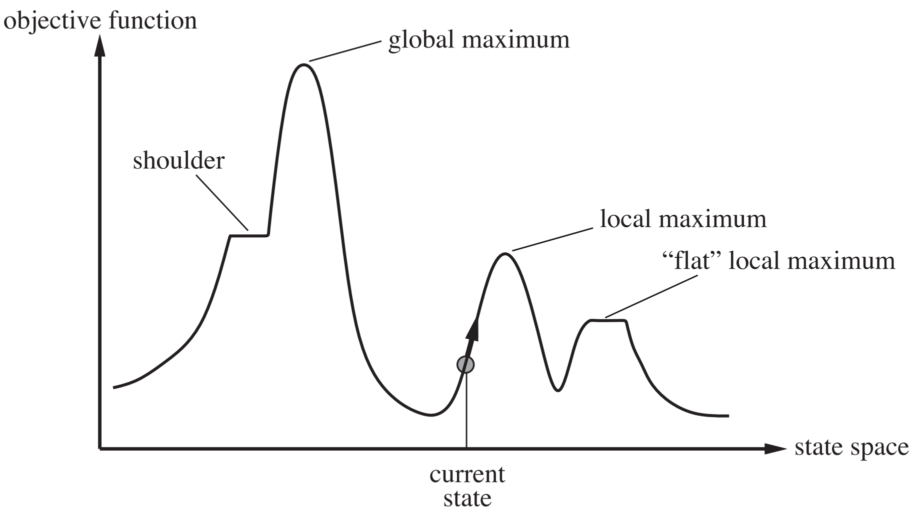
- If f^* is (true) cost of solution path, then
  – A* expands all nodes with $f(n) \leq f^*$
- Optimal and complete
  – Condition: h(n) must be admissible, that is, h(n) **never** overestimates the cost to goal
- Both time and space complexity are $O(b^d)$, because A* stores all the nodes it visits.

Local Search

- So far, the search techniques we have discussed assume observable, deterministic, known environments where the solution is a sequence of actions
- In those algorithms, the *path* is a solution to the problem
- In many other problems, however, the path is *irrelevant*
- e.g. integrated circuit design (EEEN), job shop scheduling (COMP), automatic programming (SWEN/COMP), telecommunication network optimisation (NWEN/COMP), many data mining tasks (AIML/DATA)
- Local search algorithms operate using a single current node (in general) to move to neighbours of that node
- Local search algorithms are *not* systematic:
  – use very little memory
  – can often find **reasonable solutions** in large or even infinite state space

Local Search — "Hill Climbing"

- Local search is useful for solving optimisation problems
- Aim to find the best state according to an **objective function**
- Only keep *one* state (node) and its evaluation h(n): the *quality*
- Continually move in the direction of increasing h(n)
- Choose the best successor; if more than one, choose at random



Simulated Annealing (1)

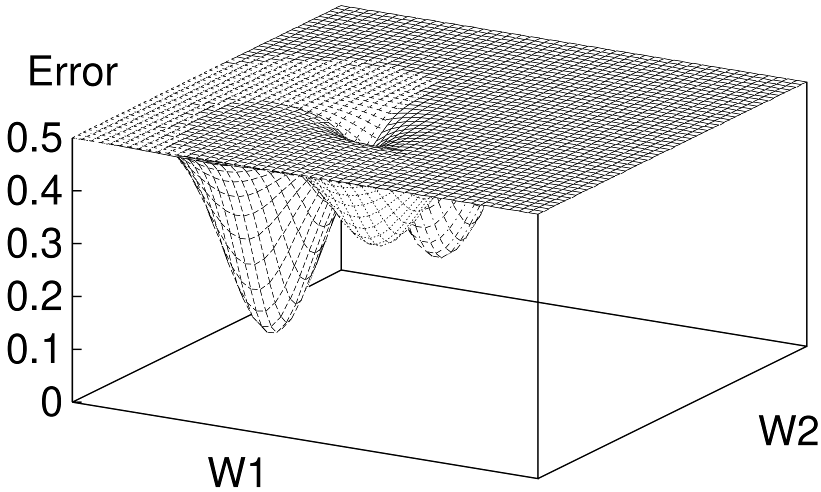
- Hill climbing (HC) never makes “worse” moves toward states with a higher cost. (*It is greedy*)
- HC can easily get stuck in a local optimum (maximum) and is (nearly always) incomplete.
- Purely random walk is complete in general, but can be extremely inefficient (*Halting problem?*)
- Can we learn anything from the “real-world” to improve HC?
- This is the idea of simulated annealing (SA)!

Simulated Annealing (2)

- SA borrows an idea from metal annealing in Physics which is used to temper or harden metals
  – heat them to a high temperature then *gradually* cool them
  – allows them to reach a low energy crystalline state
- Instead of using the best move, SA uses a **random** move
- But still uses the HC idea — if the move improves the situation, accept it; otherwise, accept it with some probability less than 1.
  – It uses the probability and "temperature" to control the loop

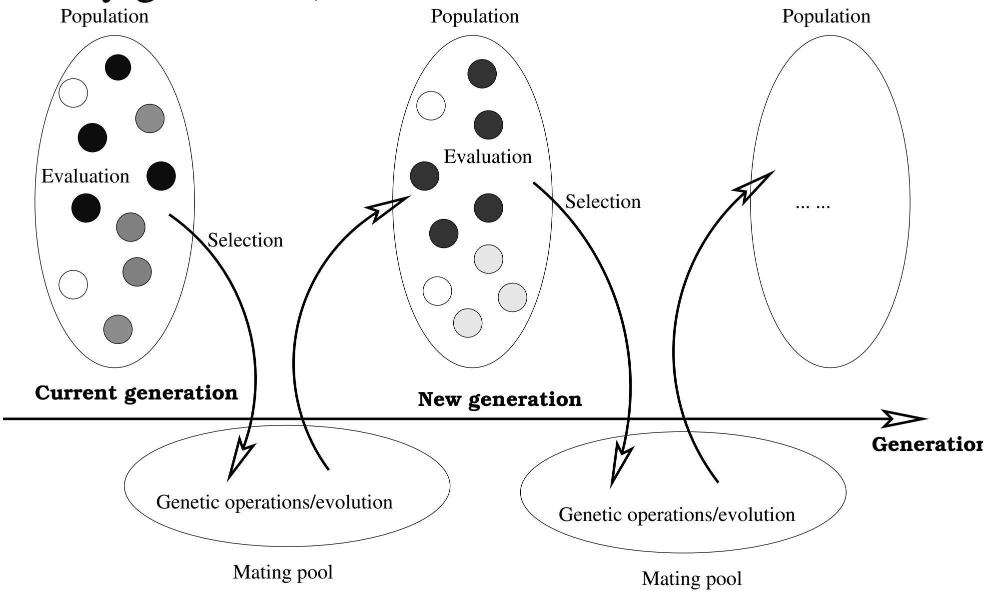
Gradient Descent Search

- HC only considers *discrete* steps
- If the problem (objective function) is continuous, then the idea of HC can still work but the mechanism will need to change
- Gradient Descent (or Ascent) search
- e.g. neural network training using back propagation



(Genetic) Local Beam Search

- Like HC, beam search (BS) considers only the solution space.
- Instead of considering only one neighbour, BS considers one or more neighbours
- Maintains a beam of multiple best *candidate solutions* (initially, randomly generated) and modifies the candidate solutions



Building Solutions vs Searching Solutions?

- Building solutions step by step
  - search path/space is partial solutions
  - Classic search
  - all uninformed/blind search: BFS, UC, DFS, ...
  - all classic/heuristic search: greedy BFS, A*, ...
- Searching solution space
  - path does NOT matter or irrelevant
  - modifying solutions step by step
  - HC, GDS, SA, BS
- # Search solutions
  - A single solution per run: classic search, HC, GD, SA
  - Multiple solutions per run: Beam search
- Partial solutions vs candidate solutions
  - Partial: HC, GD, SA
  - Candidate: (genetic) beam search

More Discussions

- How many states would be checked (as the current) for neighbours during search?
  - 1 states/nodes: HC
  - 1 or more states: BS — mutation (1), crossover (>=2), ...
  - in gradient direction: GD
  - random? SA, ...
- Fringe/frontier
  - pruning and ordering
  - genetic operators?
- When to stop?
  - goal? classical search
  - local optima? HC, GD
  - random temperature? SA
  - convergence? BS
  - after some amount of time?

Further Discussions

- Paths and solutions: Explicit graphs (including trees)?
- Paths and solutions: Implicit graphs (construct as you go)?
- Local search vs Global search
- Online search vs off-line search
- Satisficing vs. Optimising?
- Dynamic environments?

(*Lots of things...we will discuss **some** of them in this course!*)

Game Playing

- States are 'board' positions
- Operators are moves of the game
- Initial state is initial board position
- Final state is winning (or drawn) position

- COMP313: Computer Game Development

Summary

- Applications of Search
- Search strategies and techniques
  - Uninformed
  - Informed
  - Local search
- Other search techniques
  - Bound and Bound
  - Dynamic programming
  - ...
- Characteristics

- Suggested readings: Chapters 3 and 4
- Next Topic: Machine Learning