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

问题解决和搜索技术

Problem Solving and Search Techniques

Dr Andrew Lensen

Andrew.Lensen@ecs.vuw.ac.nz

This lecture **will** be recorded.

Housekeeping

- I'm lecturing the first half of the course, including Assignments 1 & 2, and the first test
- My office hours: **1–2pm, Wednesday** until Week 7 in **CO344**
- 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 to Dr Yi Mei (yi.mei@ecs.vuw.ac.nz)
- *Akoranga*: constructive feedback is always welcome!
- Anything else upfront?

Outline

- Why Search? 为什么要search？
- Search strategies 搜索策略
- Uninformed/Blind search 不知情/盲目搜索
- Informed search/Heuristic search •信息搜索/启发式搜索
- Local search: Hill Climbing •本地搜索：爬坡
- Local search in continuous space •连续空间中的本地搜索
- Genetic beam search •遗传束搜索
- Advanced discussions •高级讨论
- Game Playing •游戏

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
- Traveling Salesperson Problem (TSP)
- VLSI layout (Cell Layout and Channel routing)
- Robot nevigation
- Automatic Assembly sequencing
- University timetabling
- Job shop scheduling

许多现实世界中复杂的工程问题都需要搜索：
•巡回问题
•旅行商问题（TSP）
•VLSI布局（单元布局和通道路由）
•机器人导航
•自动装配排序
•大学时间表
•车间调度

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

进化计算（EC）

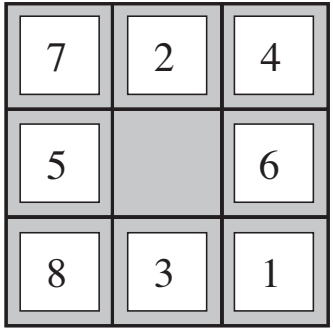
状态空间搜索

State Space Search

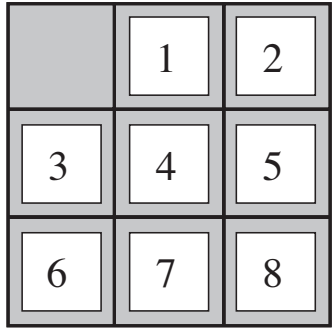
Problem solving as State Space Search 以状态空间搜索解决问题

- 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, that is, 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 状态空间中的路径是从一个状态到另一个状态的一系列operators
- Solve problem by searching for path from initial state to goal state through legal states
- Each operator has an associated cost 相关费用，(权重
  - Path cost: the sum of the costs of operators along the path

An Example: 8 Puzzle



Start State

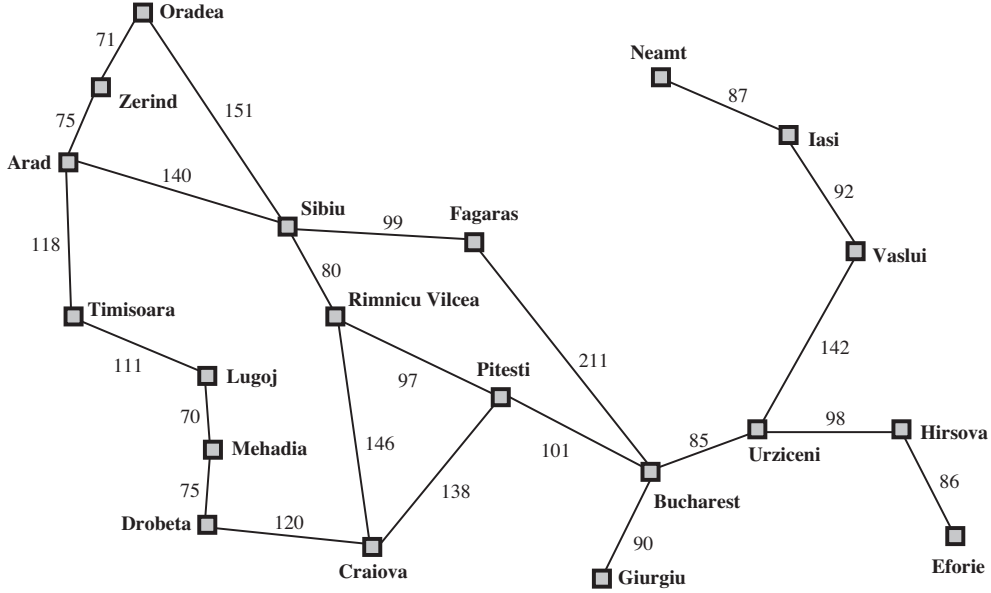


Goal State

Formulating problems: 提出问题：

- States: location of the 8 tiles
- Operators: blank moves left, right, up, or down
- Path cost: length of path
- 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

通用搜索算法

General Search Algorithm

用于一般树搜索

For general tree-search:

使用问题的初始状态初始化边界

```
Initialise the frontier using the initial state of the  problem
 loop          边界/边缘为空，意味着到了尽头，自然不可能是这个node
   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)  展开节点并将子注释放在边界上（和排序）
 end loop
```

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

搜索变体和效果指标

Search Variants and Performance Measure

Three search variants:

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

4个指标:

Four measures:

- Completeness: is the strategy guaranteed to find a solution when one exists? 完整性：是否可以保证找到一个解决方案的策略？
- Optimality: does the strategy find the highest-quality solution (lowest cost) when there are several solutions?

最优性：当存在多个解决方案时，该策略是否找到质量最高的解决方案（最低成本）？

- 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

•不知情（盲）搜索
-广度优先
-统一成本
-深度优先
-深度有限
-迭代加深
-双向

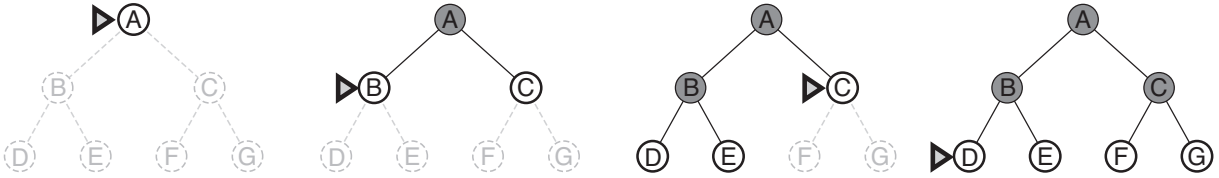
- Informed (Heuristic) search
  - Greedy best-first search
  - A* search

知情（启发式）搜索
-贪婪的最佳优先搜索
- A *搜索

- Beyond classic search
  - Hill climbing
  - Gradient descent
  - Simulated Annealing
  - Beam search
  - ~~Bound and bound (x)~~
  - ~~dynamic programming (x)~~

•超越经典搜索
-爬坡
-梯度下降
-模拟退火
-波束搜索
-边界（x）
-动态编程（x）

广度优先 Breadth First Search



- Start at the initial state;
- Find all states reachable from the initial state;
- Find all states reachable from the states available at the previous step; 从上一步中可用的状态中找到所有可到达的状态；
- Repeat the previous step until the final state is reached

重复上一步，直到达到最终状态

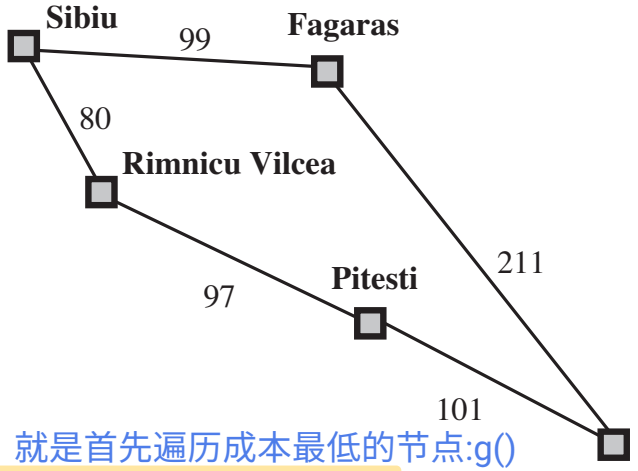
Issues for Breadth-first Search

- Breadth-first search is guaranteed to find the shortest path
  – Complete
  – Optimal if all operators have same cost (so shallowest solution is cheapest solution) 如果所有operators的成本都相同，则为最佳选择（因此，最浅的解决方案是最便宜的解决方案）
- In practice, breadth-first search is very expensive
  – Time complexity $O(b^d)$
  – Space complexity $O(b^d)$
- b: the branching factor of nodes (ie, average number of children a node has)
- d: the depth of the desired solution

•b：节点的分支因子（即，节点具有的平均子节点的数量）
•d：所需解的深度

统一成本搜索

Uniform Cost Search



和BFS的唯一区别，就是首先遍历成本最低的节点:g()

- 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

如果所有operators都具有正成本，则为最佳

- 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
  – else make the current state a state reachable from the current state;
- Repeat previous step until success, backtracking if necessary

DFS和深度限制搜索的问题

Issues for DFS and Depth Limited Search

深度优先搜索容易丢失–仅在搜索树有限时才完成
–并非最佳，因为dfs一旦找到了goal state就不会再搜了，这可能会遗漏更好的cost的path

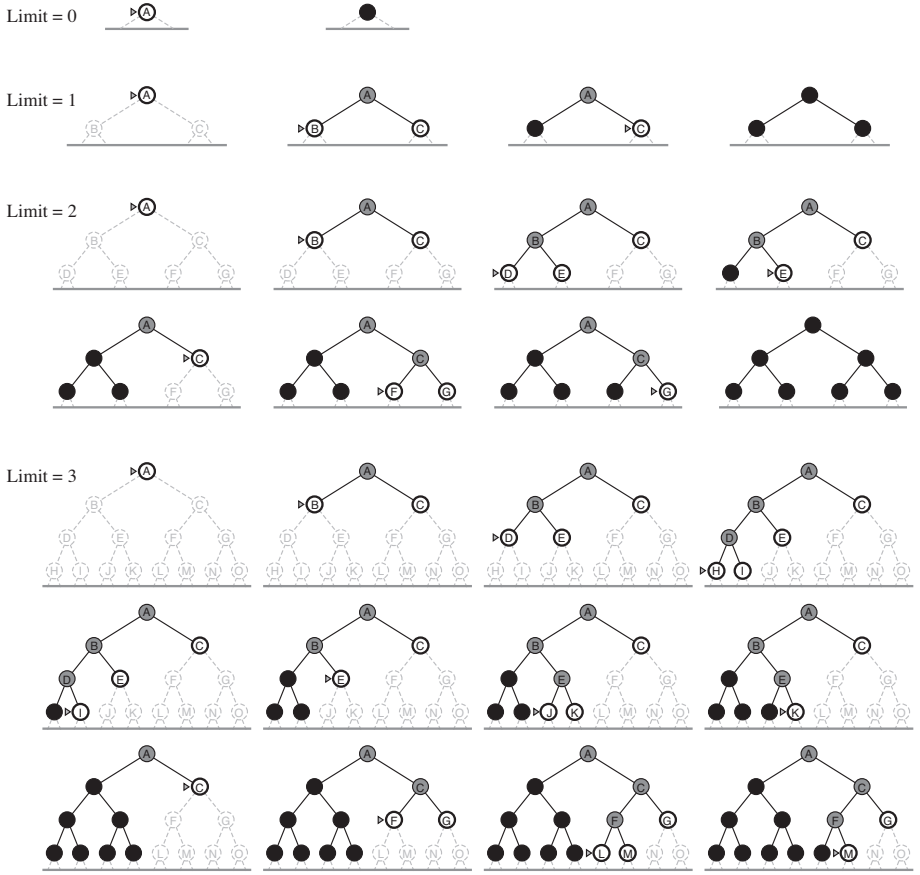
- Depth-first search is prone to being lost
  - Complete only if search tree is finite
  - Not optimal
- In practice, depth-first search is (more) efficient
  - Time Complexity $O(b^m)$
  - Space Complexity $O(bm)$ (m=max depth of search tree)
- Good when many solutions in deep (but finite) trees

像深度优先一样，但是深度被截断

- **Depth Limited Search:** Like depth-first, but with depth cut off
- Complete only if solution is at depth $<=$ c where c is the depth limit
- Not optimal; Time complexity $O(b^c)$; Space complexity $O(bc)$

迭代加深（深度优先）搜索

Iterative Deepening (Depth-first) Search



Iterative Deepening Search

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

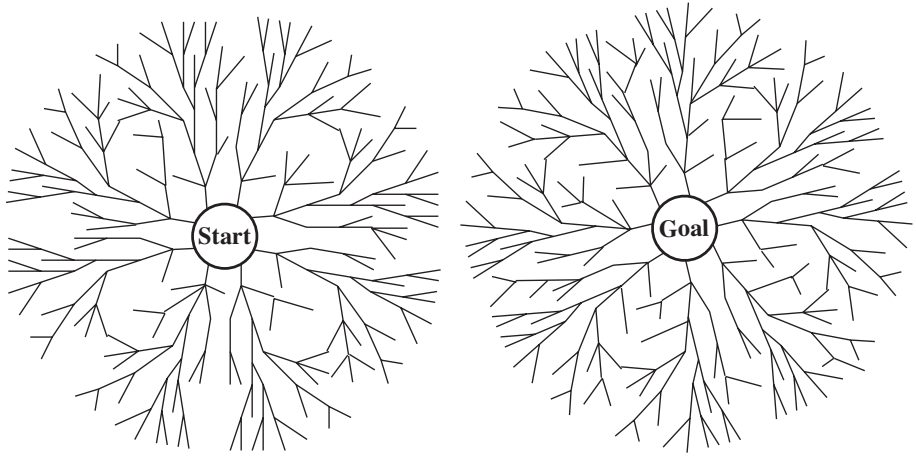
it's a depth-limited version of dfs and it's run repeately with increase the depth limit until the goal node is reached.

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

  多次扩展某些节点-但是"浪费"的工作实际上很小，
  并且随着分支因子的增加而减少
- In general, iterative deepening is the preferred search method on a large search space when the depth of the solution is unknown.

  通常，当解决方案的深度未知时，
  迭代加深是在较大搜索空间上的首选搜索方法

Bidirectional Search



双向搜索是2个广度优先搜索

- 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

启发式搜索

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)

所需的解决方案）

- 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

可允许的：h（n）永远不会高估达到目标的成本
-寻路：直线距离
-通过缓解问题来寻找启发式方法

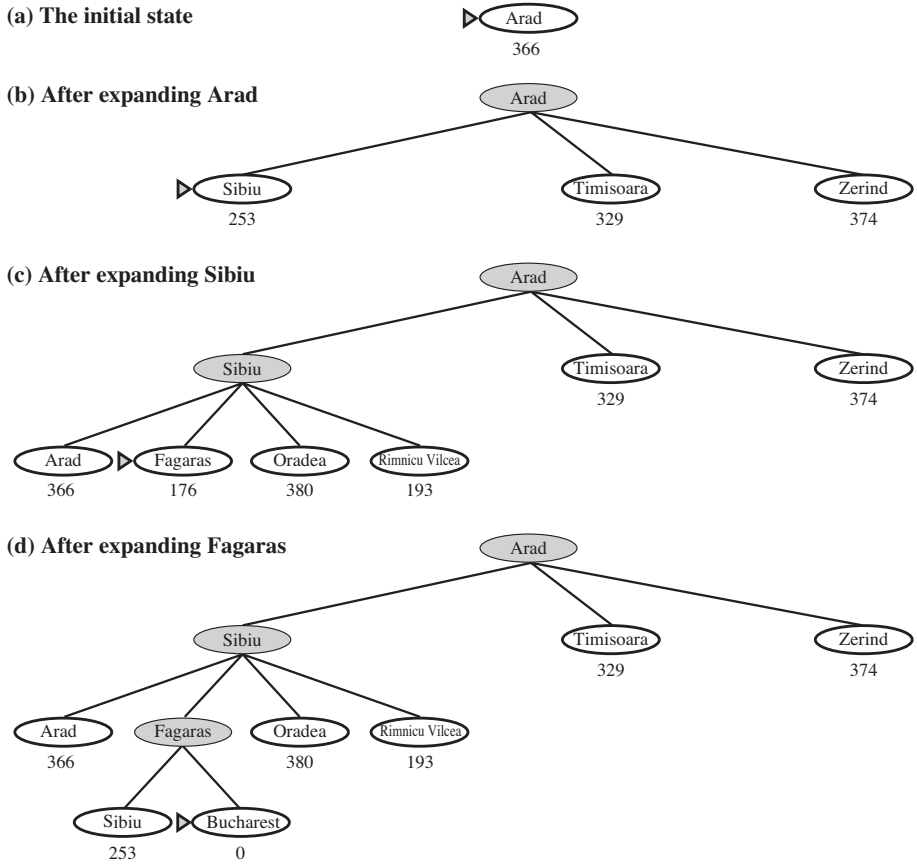
下面是属于:informed (heuristic search)

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

不理想，不完整

- 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

但是，实际上可以很好地工作

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

不考虑路径成本g（n）：从起始节点到节点n的成本

A* search

•A *尝试使总的估计路径成本最小化：

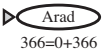
- A* attempts to minimize total estimated path cost:
- $f(n) = g(n) + h(n)$
- g(n): cost from the start node to node n
- h(n): estimated cost of the cheapest path from node n to the goal
- f(n): estimated cost of the cheapest solution through n

g（n）：从起始节点到节点n的成本

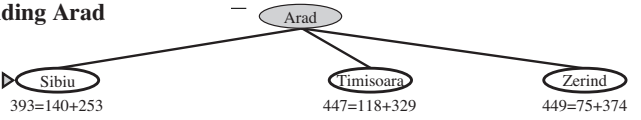
•h（n）：从节点n到目标的最便宜路径的估计成本

•f（n）：到n为止最便宜的解决方案的估计成本

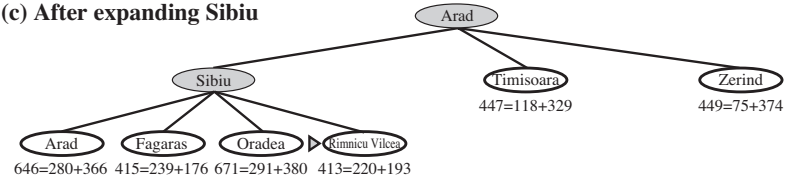
(a) The initial state



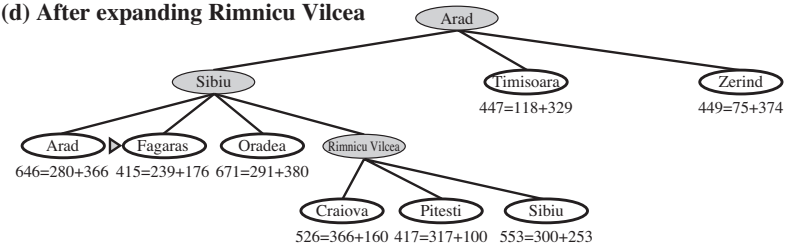
(b) After expanding Arad



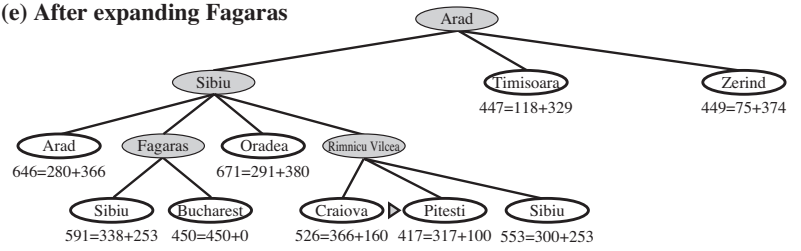
(c) After expanding Sibiu



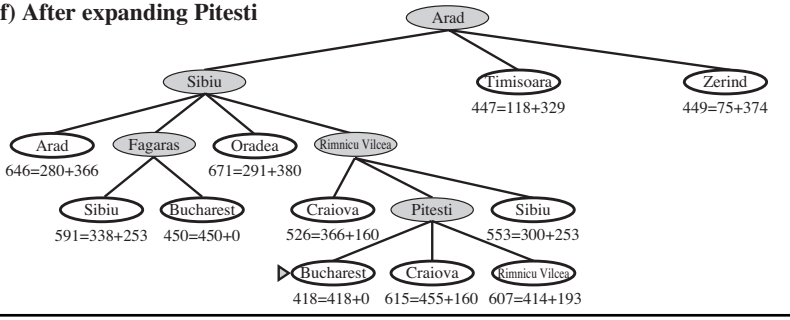
(d) After expanding Rimnicu Vilcea



(e) After expanding Fagaras



(f) After expanding Pitesti



Issues for A* search

g（n）：从起始节点到节点n的成本

- If f^* is (true) cost of solution path, then
  – A* expands all nodes with $f(n) \leq f^*$

•h（n）：从节点n到目标的最便宜路径的估计成本

•f（n）：到n为止最便宜的解决方案的估计成本

- Optimal and complete
  – Condition: h(n) must be admissible, that is, h(n) never over-estimates 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), factor-floor layout, job shop scheduling (COMP), automatic programming (SWEN/COMP), telecommunication network optimisation (NWEN/COMP), Vehicle routing, or portfolio management (other), classification, 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

确定性的

本地搜索算法通常使用单个当前节点进行操作，以移动到该节点的邻居

•本地搜索算法不是系统性的：
–使用很少的内存
–通常可以在大甚至无限状态空间中找到合理的解决方案

Local Search — "Hill Climbing"

优化问题

- Local search is useful for solving optimisation problems
- Aim to find the best state according to an *objective function*

目标函数

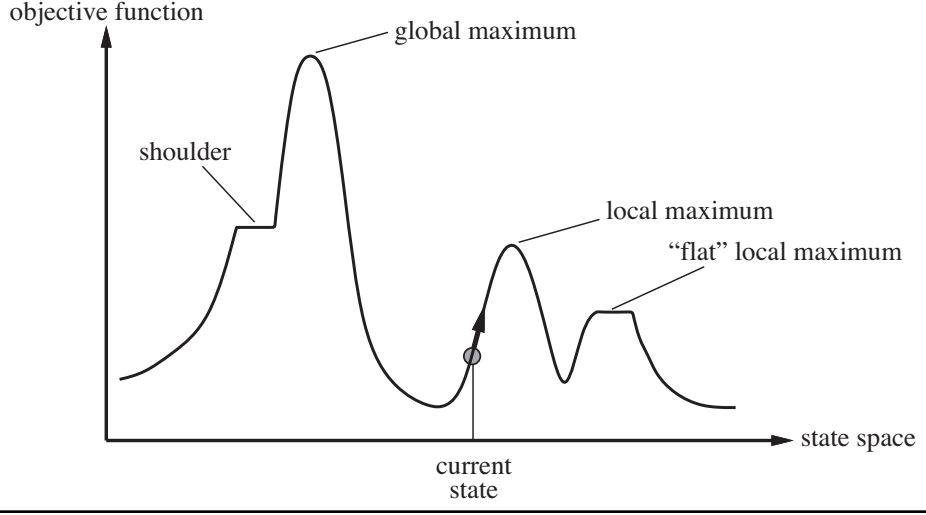
- Only keep *one* state (node) and its evaluation h(n): the quality

评估值h（n）：质量

- A loop that continually moves in the direction of increasing h(n). (decreasing h(n): if h(n) is the cost)

一个不断向increasing h（n）方向移动的循环。 （decreasing h（n）：如果h（n）是成本）

- Choose the best successor; if more than one, choose at random.



模拟退火（1）

Simulated Annealing (1)

- Hill climbing (HC) never makes “worse” moves toward states with a higher cost.
- HC can easily get stuck in a local optimum (maximum) and is almost guaranteed to be 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)

•爬山（HC）绝不会使“更糟糕”的事情朝成本更高的state发展。
•HC很容易卡在局部最佳值（最大值）中，几乎可以保证不完整。
•一般而言，纯粹的随机游走是完整的，但效率极低（停顿问题？）
•我们可以从“现实世界”中学到任何东西来改善HC吗？ •这是模拟退火（SA）的想法

Simulated Annealing (2)

- SA borrows an idea from metal annealing in Physics or Mechanical Engineering 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

逐渐冷却

但是仍然使用HC想法
-如果此举改善了情况，接受它；
否则，以小于1的概率接受它。

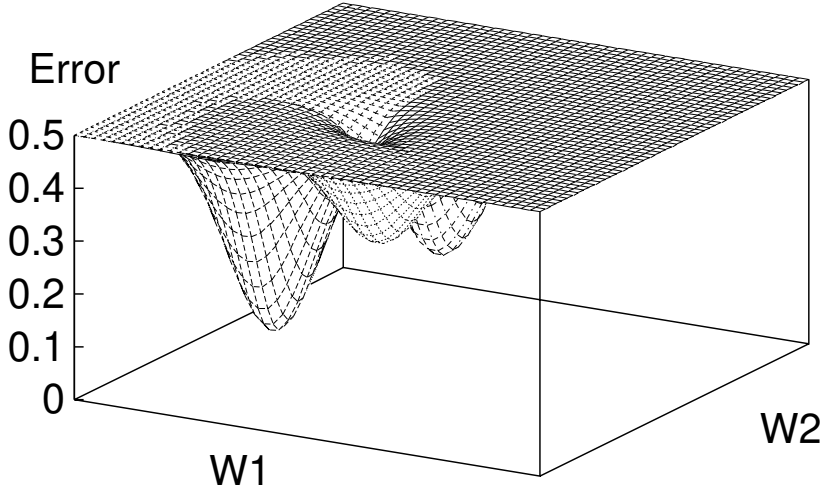
它使用概率和"温度"来控制回路

梯度下降搜索

Gradient Descent Search

- HC only considers discrete space
- 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 梯度下降(或上升)搜索
- e.g. neural network training using back propagation

•例如 反向传播的神经网络训练

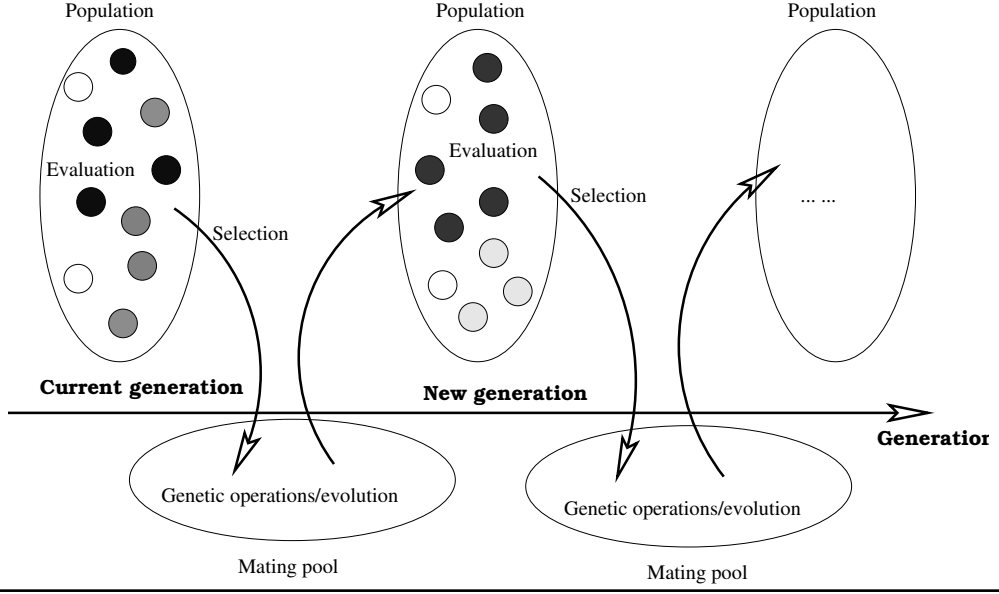


（遗传）局部波束搜索

(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

保持一束多个最佳候选解决方案（最初是随机生成）并修改候选解决方案



Building Solutions vs Searching Solutions

- Building solutions step by step
  - search path/space is partial solutions 部分解决方案
  - Classic search
  - all uninformed/blind search: BFS, UC, DFS, DLS, IDS, BiDS 不知情/盲目搜索：
  - all classic/heuristic search: greedy BFS, A*, etc. 经典/启发式搜索
- 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 offline search

- Satisficing vs. Optimising? 满意与优化？

- Dynamic environments?

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

•状态是“棋盘”位置
•operators是游戏的动作
•初始状态是棋盘的初始位置
•最终状态是获胜（或平局）的位置
•COMP313：计算机游戏开发

Summary

- Applications of Search 搜索的应用
- Search strategies and techniques
  - Uninformed 不知情
  - Informed -知情
  - Local search -本地搜索
- Other search techniques
  - Bound and Bound
  - Dynamic programming
  - ...
- Characteristics 特征

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