Optimisation Problems

- Problem setting so far: path finding
  - Goal: find a path from S to G
  - Solution: the path, optimal solution: least cost path
  - Search algorithms:
    - Uninformed: BFS, UCS, DFS, IDS
    - Informed: greedy, A*
- Now a new problem setting: optimisation problem
  - Each state has a score $v$
  - Goal: find the optimal state
    - $v$ = the state with the highest or lowest $v$ score (depending on what is desirable, maximum or minimum)
    - Solution: the state; the path is not important
  - A large number of states $\Rightarrow$ can’t be enumerated
  - $\Rightarrow$ We can’t apply the previous algorithms – too expensive

Optimisation Problems - Example

- n-queens problem
  - The solution is the goal configuration, not the path to it
  - Non-incremental formulation
    - Initial state: n-queens on the board (given or randomly chosen)
    - States: any configuration with n-queens on the board
    - Goal: no queen is attacking each other
    - Operators: move a queen or move a queen to reduce the number of attacks
**V-score Landscape**

- Each state has a score $v$ that we can compute; defined by a heuristic evaluation function (also called objective function)
- Goal: find the global maximum (the highest value) or the global minimum (the lowest value) – depends on the task
- Complete local search – finds a goal if one exists
- Optimal local search – finds the global maximum/minimum

**Hill-Climbing Algorithm**

- Also called iterative improvement algorithm
- Idea: Keep only a single state in memory, try to improve it

**Hill-climbing Search**

- Let’s assume that we are looking for a maximum
- Idea: move around trying to find the highest peak
  - Store only of the current state
  - Do not look ahead beyond the immediate neighbors of the state
  - “Like climbing Everest in thick fog with amnesia”

**Hill-climbing (or Gradient Ascent/Descent)**

- Ascent – the goal is the highest peak (maximum)
- Descent – the goal is the lowest point (minimum)

**Gradient descent:**

1) Select initial state $s$. Set current node $n$ to $s$.
   (The initial state can be given or can be a randomly selected node)
2) Generate the successors of $n$. Select the best successor $n_{best}$; it is the successor with the best $v$ score, $v(best)$ (i.e. the lowest score)
3) If $v(best) > v(n)$, return $n$ //stop - local or global maximum found
   Else set $n$ to $n_{best}$. Go to step 2 //keep searching
- Summary: Always expands the successor with the lowest value, no backtracking
Hill-climbing – Example 1

- $v$ - value is in brackets; the lower, the better (i.e. gradient descent)
- Expanded nodes: SAF

- S (7)
- A(4) B(5) C(8)
- D(3) E(7) F(2) H(1)

Hill-climbing – Example 2

- Aim: find coloring with minimum number of pairs of adjacent cells in 3x3 grid with the same color (R-red or B- blue)
- Ascending or descending?

- $v$ – # pairs of adjacent cells with the same color

Picture from N. Nielsen, AI, 1998

Hill-climbing Search

**Weaknesses:**
- Not a very clever algorithm – can easily get stuck in a local optimum (maximum/minimum)
- However, not all local maxima/minima are bad – some may be reasonably good even though not optimal

**Advantages:** good choice for hard, practical problems
- Uses very little memory
- Finds reasonable solutions in large spaces where systematic algorithms are not useful

Not complete, not optimal but keeps just one node in memory!

Hill-climbing – Escaping Bad Local Optima

- Hill climbing finds the closest local minimum or maximum
- Which may or may not be the global minimum/maximum
- The solution found depends on the initial state
  - For tasks when we start from a random state not a fixed given state, and when the solution found is not good enough:
    - run the algorithm several times starting from different points;
    - select the best solution found (i.e. the best local optimum)
- Plateaus: random walk - no change in $v$, wander endlessly, revisiting previously visited nodes
  - Solution: keep track of the number of times $v$ is the same and do not allow revisiting of nodes with the same $v$
  - Note: our version does not allow visiting states with the same $v$
**Hill-climbing – Escaping Bad Local Optima**

- Ridges – the current local maximum is not good enough; we would like to move up but all moves go down
  - but a combination of 2 or more moves can increase the height
  - Solution: combine 2 or more moves in a macro move or allow a limited number of look-ahead search

- Dark circles = states
- A sequence of local maxima that are not connected with each other. From each of them all available actions point downhill.

**Beam Search**

- It keeps track of $k$ states rather than just 1
- 2 versions: starts with 1 given state or $k$ randomly generated states
- At each iteration (level): generate all successors of all $k$ states
- If any one is a goal state, stop; else select the $k$ best successors from the list and go to the next level
- In nutshell: keeps only $k$ best states

**Beam Search - Example**

- Starting from S, run beam search with $k=2$ using the values in brackets as evaluation function (the smaller, the better)
- Expanded nodes: SABFH

**Beam Search and Hill-Climbing Search**

- Compare beam search with 1 initial state and hill climbing
  - Beam – 1 start node, at each step keeps $k$ best nodes
  - Hill climbing – 1 start node, at each step keeps 1 best node
- Compare beam search with 1 initial state and hill-climbing with $k$ random initial states
  - Beam – 1 start node, $k$ search threads are run in parallel and useful information is passed among them
  - Hill climbing – $k$ starting positions, $k$ threads run individually, no passing of information among them
Beam Search with A*

- Recall that memory was a big problem for A*
- Idea: keep only the top $k$ nodes in the fringe, i.e. use a priority queue of size $k$.

- Advantage: memory efficient
- Disadvantage: neither complete, nor optimal

- Variations: Keep only the nodes that are at most $\varepsilon$ worse than the best node in the fringe
  - $\varepsilon$ - beam width

Simulated Annealing

- What is annealing in metallurgy?
  - material’s temperature is gradually decreased (very slowly) allowing its crystal structure to reach a minimum energy state
- Similar to hill-climbing but selects a random successor instead of the best (step 2) below

1) Select initial state $s$. Set current node $n$ to $s$.
2) Randomly select $m$, one of $n$’s successors
3) If $v(m)$ is better than $v(n)$, $n=m$ //accept $m$
   Else $n=m$ with a small probability //accept $m$ with a small prob.
1) Go to step 2 until a predefined number of iterations reached or the state reached (solution found) is good enough

The Small Probability

- The probability is defined as
  $$P = e^{\frac{v(m) - v(n)}{T}}$$

  - Nominator - how good the successor $m$ is
    - Bad move: $v(n)>v(m)$ (assuming that we are looking for minimum)
    - $\Rightarrow$ $P$ decreases exponentially with the badness of the move

  - Denominator: parameter $T$ that decreases (anneals) over time based on a schedule, e.g. $T=T*0.8$
    - high $T$ – bad moves are more likely to be allowed
    - low $T$ – more unlikely; becomes more like hill-climbing
    - $T$ decreases with time and depends on the number of cycles completed, i.e. until “bored”

Simulated Annealing - Theorem

- What is the correspondence?
  - $v$ – total energy of the atoms in the material
  - $T$ - temperature
  - schedule – the rate at which $T$ is lowered

- Theorem: If the schedule lowers $T$ slowly enough, the algorithm will find global optimum
  - i.e. escapes local maxima and is complete and optimal given a long enough cooling schedule
  - $\Rightarrow$ annealing schedule is very important

  - Easy to implement but “slowly enough” is difficult to set
Genetic Algorithms

- Inspired by mechanisms used in evolutionary biology, e.g. selection, crossover, mutation
- Similar to beam search, in fact a variant of stochastic beam search
- Each state is called an individual. It is coded as a string.
- Each state \( n \) has a fitness score \( f(n) \) (evaluation function). The higher the value, the better the state.
- Goal: starting with \( k \) randomly generated individuals, find the optimal state
- Successors are produced by selection, crossover and mutation
- At any time keep a fixed number of states (the population)

Example – 8-queens Problem

One possible encoding is \((3 \ 2 \ 7 \ 5 \ 2 \ 4 \ 1 \ 1)\)

- column 1: a queen at position 3
- column 2: a queen at position ...

Example – 8-queens Problem (2)

- Suppose that we are given 4 individuals (initial population) with their fitness values
  - Fitness values = number of non-attacking pairs of queens (28 for a solution)
  - Let the probability for reproduction is proportional to the fitness (expressed in %)
- Let the probability for reproduction is proportional to the fitness (expressed in %)
  - Initial Population
    - 24748552 24 31%
    - 32752411 23 29%
    - 24415124 20 26%
    - 32543213 11 14%

Example – 8-queens Problem (3)

- Select 4 individuals for reproduction based on the fitness function
  - Individuals 2, 1, 2 and 3 are selected, i.e. individual 2 is selected twice while 4 is not selected
- Crossover – random selection of crossover point; crossing over the parents strings
- Mutation – random change of bits (in this case 1 bit was changed in each individual)
A Closer Look at the Crossover

(3 2 7 5 2 4 1 1) + (2 4 7 4 8 5 5 2) = (3 2 7 4 8 5 5 2)

- When the 2 states are different, crossover produces a state which is long way from either parents
- Given that the population is diverse at the beginning of the search, crossover takes big steps in the state space early in the process and smaller later, when more individuals are similar

Genetic Algorithm – Pseudo Code (1 variant)


1. Let $s_1, \ldots, s_n$ be the current population
2. Let $p_i = f(s_i) / \Sigma_i f(s_i)$ be the reproduction probs
3. FOR $k = 1; k < N; k++$
   - parent1 = randomly pick $s$ with probs $p$
   - parent2 = randomly pick another $s$ with probs $p$
   - randomly select a crossover point, swap strings of parents 1, 2 to generate children $f[k], f[k+1]$
4. FOR $k = 1; k < N; k++$
   - Randomly mutate each position in $f[k]$ with a small probability
5. The new generation replaces the old: $\{s\} \leftarrow \{f\}$
   Repeat until some individual is fit enough or max number of iterations have been reached

Genetic Algorithms - Discussion

- Combine uphill tendency with random exploration and exchange information among parallel threads; the main advantage comes from crossover
- Success depends on the representation (encoding)
- Easy to implement
- Not complete, not optimal

Acknowledgements

http://pages.cs.wisc.edu/~jerryzhu/cs540.html