**Artificial Intelligence**

**Session 4**

1. **Local search algorithms**: Good for optimisation problems:
   1. no goal test
   2. the path to the goal is irrelevant
   3. the goal state itself is the solution:
      1. integrated circuit design
      2. office assignment
      3. factory-floor layout
      4. telecommunications network optimisation
   4. Find configurations satisfying constraints.
   5. *State*
      1. keep a single "current" state, try to improve it
      2. f. searching a tree of possible paths for the optimal sequence
   6. *Advantages*
      1. Use little memory (usually constant amount)
      2. Can find reasonable solutions in large or infinite (continuous) problem spaces
   7. *State space landscape*:   
      A graph with state space and objective function on the x and y axis, respectively. The graph has several peaks and troughs with some constant value which form a straight line (called plateau). The maximum/minimum peak is called global maximum/minimum. The peak/trough in the immediate vicinity of our datapoint is called the local maximum/minimum.  
      1. A state has
         1. Location: given by the values of the state variables.
         2. Elevation: its heuristic cost or objective value. Note: we can’t see the whole function; we can calculate it for individual points
      2. Solutions (two ways of thinking about it, but equivalent)
         1. Heuristic cost (find minimum cost): global minimum
         2. Objective function (find best state or maximal utility): global maximum
      3. “Local” because we don’t care about the path through the search space
      4. A local search algorithm is
         1. complete: if it always finds a goal if one exists
         2. optimal: if it always finds the global minimum/maximum
      5. Problem: depending on the initial state, search can get stuck in local maxima or minima
2. **Hill-climbing search**:
   1. A loop that continuously moves in the direction of increasing value (uphill)
      1. Terminates when a peak is reached → greedy local search
   2. Like agenda-based search, but the agenda contains only one item
      1. Take the best option at each stage, and throw away all the others
3. **Hill-climbing: the 8-queens problem**:  
     
   The eight queens puzzle is based on the classic stategy games problem which is in this case putting eight chess queens on an 8×8 chessboard such that none of them is able to capture any other using the standard chess queen's moves. The color of the queens is meaningless in this puzzle, and any queen is assumed to be able to attack any other. Thus, a solution requires that no two queens share the same row, column, or diagonal. The eight queens puzzle is an example of the more general *n* queens puzzle of placing *n*8 queens on an *n*×*n* chessboard.  
   1. Successors
      1. set of states generated by moving one queen vertically
   2. Heuristic
      1. h = number of pairs of queens attacking each other, directly or indirectly
      2. h = 17 for the example
      3. h = 0 is a solution state (global minimum)
   3. Greedy local search
      1. grabs best neighbour without “thinking ahead”
   4. Rapid improvement in heuristic value
      1. But at a fatal cost
      2. Local Maxima/Minima will stop search altogether
         1. Ridges: a sequence of local maxima that make it difficult to escape.
         2. Plateau: heuristic values uninformative
         3. Foothills: local maxima that are not global maxima
   5. Algorithm reaches a point where no progress is made
   6. To test on the 8-queens example:
      1. run many searches starting from random configurations of 8-queens
         1. too many to do it exhaustively
      2. result
         1. 86% of starting configurations get stuck
         2. 14% of starting configurations succeed
      3. the algorithm works quickly, usually only evaluating 3-4 steps before either getting stuck or finding a solution
   7. Allow different moves?
      1. e.g., sideways
      2. must account for local maxima (foothills) otherwise could get stuck in an infinite loop
         1. e.g., cap number of sideways moves
         2. but we risk losing generality with such ad hoc solutions
      3. Result:
         1. 6% stuck
         2. 94% solution
      4. Works much slower
         1. evaluating 21-64 steps before either getting stuck or finding a solution
   8. Stochastic hill-climbing
      1. choose randomly between available uphill moves
         1. choose from uniform or non-uniform distribution
      2. converges more slowly but sometimes finds better solutions
         1. optimality still not guaranteed in all cases
   9. First-choice hill-climbing
      1. Uses Stochastic hill climbing but randomly generates successors and picks first larger one
         1. Good for big problems
   10. Random-restart hill-climbing
       1. If at first you don't succeed, try again, starting from a different place
          1. run a fixed number of restarts or run indefinitely
4. **Simulated annealing search**:
   1. Hill-climbing
      1. only improves on the current solution
      2. not complete
         1. may get stuck in local minima/maxima
      3. Random walk
         1. moves from state to state randomly
         2. complete (given infinite time) but very inefficient
      4. Simulated Annealing combines these two to give a compromise between search complexity and completeness
      5. Main ideas:
         1. Escape local maxima by allowing some bad moves
         2. Gradually decrease size and frequency of allowed bad moves as search proceeds
      6. By analogy with a process of hardening in steel production
         1. want crystal structure of metal to be all lined up
         2. heat the metal to make the crystals vibrate and “shake out” irregularities
         3. let it cool slowly so that the more ordered form stays ordered
      7. So in the algorithm, we have an imaginary notion of “temperature”, which allows random movement outside the hill-climb
         1. as the “temperature” drops, less random movement is allowed
5. **Example**:
   1. Lets say there are 3 moves available, with changes in the objective function of delta E\_1 = −0.1, delta E\_2 = 0.5, delta E\_3 = −5.
   2. Let T = 1, pick a move randomly
      1. If delta E\_2 (good move) is picked, move there
      2. If delta E\_1, move there with prob. exp^{(delta E)/T} = exp^{-0.1} = 0.9
      3. If delta E\_2 , move there with prob. exp^{(delta E)/T} = exp^{-5} = 0.05
   3. T = “temperature” parameter
      1. High T: probability of “locally bad” move is higher
      2. Low T: probability of “locally bad” move is lower
      3. Typically, T is decreased as the algorithm runs longer
         1. I.e., there is a “temperature schedule”
   4. A graph with delta E and exp^{(delta E)/T} n the x and y axis, respectively. Three temp concave curves enclosed on the y axis limits of [0, 1] representing the probability distribution.   
        
      If the “temperature” decreases slowly enough, simulated annealing search finds a global optimum with probability approaching 1.0.
   5. Like getting a ping-pong ball into the deepest crevice of a bumpy surface
      1. Left alone by itself, ball will roll into a local minimum
      2. If we shake the surface, we can bounce the ball out of a local minimum
      3. The trick is to shake hard enough to get it out of local minimum, but not hard enough to dislodge it from global one
      4. We start by shaking hard and then gradually reduce the intensity of shaking
   6. Widely used for optimisation problems such as VLSI layout, airline scheduling, etc.
6. **Local beam search**:
   1. Main idea: Keep track of k states rather than just one
   2. Start with k randomly selected states
   3. At each iteration
      1. generate all successors of all k states
         1. NB: not the same as an agenda k long!
      2. if any one is a goal state, stop
      3. else select the k best successors from the complete list and repeat
   4. So this is like a kind of selective breadth first search
   5. Local beam search looks like running k hill-climbing algorithms in parallel, but it is not
      1. the results of all k states influence each other
      2. if one state generates several good successors, they all end up in the next iteration
      3. states generating bad successors are weeded out
   6. This is b oth a strength and a weakness:
      1. unfruitful searches are quickly abandoned and searches making the most progress are intensified
      2. can lead to a lack of diversity: concentration in a small region of the search space
      3. remedy: choose k successors randomly, biasing choice towards good ones; or explicitly avoid keeping multiple similar solutions
7. **Genetic algorithms**:
   1. Genetic Algorithms (GAs) are a variant of local beam search
   2. A successor state is generated by combining two parent states
   3. Start with many randomly generated states (a population)
   4. An evaluation function or fitness function assesses the quality of a state
      1. higher values for better states
   5. Produce the next generation of states by: selection, crossover, mutation
   6. A state is represented as a string over a finite alphabet (e.g., binary)
   7. The way that you encode the problem influences the search
   8. Example: 8 queens, 8x8 board
      1. 64 bits, one for each square? (= 8x8 matrix)
      2. 8 octal digits? (One 3-bit row number for each column)
         1. 21641300 (octal)
         2. 010 001 110 100 001 011 000 000 (binary)
   9. Fitness function: number of non-attacking pairs of queens (min = 0, max = 28)
      1. Fitness here = 6 + 5 + 4 + 3 + 3 + 2 + 0 = 23
8. **Genetic algorithms diagram**:  
     
   A flowchart with the following cycle:
   1. Start
   2. Initial Population
   3. Fitness calculation
   4. Stop?
      1. If Yes, then End.
      2. If No, then Produce the next generation of states by Selection and Crossover to make New Population which is passed through Mutation and back to Step C.
9. **Genetic Selection**:
   1. After encoding states and calculating fitness, select pairs for reproduction
      1. many methods can be used
   2. Simplest: randomly choose pairs of states with non-uniform probability
10. **Cross-over**:
    1. Pairs are selected by one of a range of methods:
       1. Roulette-wheel (as in our example)
       2. Tournament: rank-influenced selection from a random subset of the population
    2. Cross-over point for each pair is randomly selected
       1. Resulting new chromosomes represent new states
11. **Mutation**:
    1. Cross-over is not enough
       1. if the population does not contain examples that have each bit of the chromosome at both possible values, parts of the search space are inaccessible
    2. So introduce mutation
       1. low probability of flipping a random bit at each cross-over step

**Past Exam Example**:

Consider the following 8-bit chromosomes, to be used as the initial state of a very small genetic algorithm.

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Bit Number | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| Chromosome C0 | 0 | 1 | 1 | 0 | 0 | 0 | 1 | 1 |
| Chromosome C1 | 1 | 1 | 0 | 0 | 1 | 0 | 1 | 1 |
| Chromosome C2 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 1 |
| Chromosome C3 | 0 | 1 | 1 | 1 | 0 | 0 | 1 | 1 |

(c) What would be the result of applying crossover to C0 and C2 between bits 4 and 5? [2 marks]

(d) Given the initial state above, explain why crossover is not enough to explore the entire search space described by this representation. Give a detailed explanation of how this problem is solved in genetic algorithms. [6 marks]

**SOLUTION A**:

(c) C0: 01100 | 011 C2: 00011 | 001

#1 C0 + #2 C2 = 01100001

#1 C2 + #2 C0 = 00011011

(d) Crossover is not enough to explore the entire search space because in some cases there are positions in the chromosome which, through the whole search space, don’t have all possible values. For example, in this search space, the position 7 only appears with value 1, so no matter how many times crossover is applied, position 7 will always be 1. That doesn’t help with the variance of the space and the solutions. In genetic algorithms, an operation is applied to try and solve this issue, it is called mutation. Mutation is a low-probability bit flip at a random position at the chromosome. It happens after the crossover operation, always, if a low probability threshold is achieved the algorithm changes a random bit in the chromosome, i.e., a bit in a random position.

**SOLUTION B**:

(c) C0 = 01100111   
C2 = 00010001   
  
The other chromosomes do not change.

(d) Genetic algorithms use mutation to randomly change bits to form compositions not achievable with crossover.