Local Search

Often in search problem the path to a goal state is of no consequence, it is the goal state itself that is the concern. For example in the 8-Queens problem it is the final configuration that is required

AI search version:

- Given a configuration of queens
  - One in each column
- Move each queen in its column
- So that a configuration of non-attacking queens is obtained

Such problems are abundant in real life

- in the scheduling of computational jobs
- in facility floor layout
- in automatic programming

If the path to a solution is of no consequence then local search algorithms

- Using only the current state and moving only to successors of that state
- Might be better employed than "global path-based" search methods

Local searches often have two advantages

- They use very little memory - having only to remember the current state
- They often give reasonable solutions in very large state spaced - where more systematic search algorithms are unsuitable

Definition: "Local state-based" search problem

Consists of:

- State space
  - States; state-to-state transitions; initial state
- Objective function f

Local search algorithms are useful for solving optimisation problems

- Aim is to find an optimal state according to some objective function defined on the states

A local search algorithm is optimal if:

- Whenever it finds a solution then it is a global minimum/maximum

1 Hill-climbing search

Hill climbing:

- Iteratively move to a better successor state
  - i.e., a successor state whose (objective function) f-value is higher until no such successor state exists when the algorithm terminates

Listing 1: Hill-climbing

```
current=initial state
loop do
    successor=sucessor of current with highest f-value
    if fsuccessor ⩽ f(current) then return current
    else current = successor
```

Notes:

- Hill-climbing doesn't look beyond the immediate neighbourhood
- Can get stuck in maxima/minima

2 Hill-climbing with the 8-Queens Problem

Local search algorithms include global information within the state

For the 8-Queens Problem this amounts to each state being a description of exactly where each queen is currently located

- The transition function details all states obtained by moving a single queen to another square within the same column
  - so each state has $8 \times 7 = 56$ successors
- The objective function f(to be minimized) is the number of pairs of queens that are mutually attacking each other

3 Exiting local maxima/minima - simulated annealing

Essential to this algorithm is a schedule which dictates the "rate of cooling"

- the schedule is a list which details the temperature at any given time

The algorithm iteratively performs a local search from the current state X until the temperature drops to 0 at which point the algorithm halts

A potential successor state Y in an interaction is chosen at random

- if $\Delta E = f(Y) - f(X) \geqslant 0$ then Y becomes the current state (f is the objective function that we are trying to maximise)

However

- If $\Delta E < 0$ then there is a chance that the successor state Y might still be chosen to become the current state (enables us to get out of local maxima)

4 Simulated Annealing

The probability that Y becomes the current state depends on

- T - the current temperature
- ΔE - the difference between the two objective values

In essence

- The "cooler" the temperature T, the less likely it is that Y is chosen
- The smaller the different ΔE, the less likely it is that Y is chosen
  - Note that ΔE is negative, so what we are saying is the worse the value f(Y) in comparison to f(X) the less likely that Y is chosen

These intuitive directions are encapsulated in the probability function

$$e^{\Delta E/T} = 1/e^{|\Delta E|/T}$$

A commonly used "cooling schedule"

- schedule[1] is user defined and schedule[t]=schedule[t-1]/β where β is user-defined

Listing 2: Simulated-annealing

```
current=initial state
for t=1 to ∞ do
    T[schedule[t]
    if T⩽ ϵ then
        return current
    else
        choose a successor state succ of current at random
        ΔE=f(succ)-f(current)
        if ΔR ⩾ 0 then
            current=succ
        else
            current=succ only with probability e^(ΔE/T)
```

5 Genetic algorithms

A genetic algorithm to solve a ("state-based search") problem

- starts with a randomly generated population of individuals, each with individual
  - ordinarily represented as a string of symbols
  - encoding a particular solution to the underlying problem

Each (potential) individual in the population is rated according to a fitness function

- measures how "good" an individual is (as a solution to the underlying problem)

In the 8-Queens Problem, for example

- an individual might represent a configuration of queens and be a string of digits from {1,2,...,8} of length 8
  - the first digit details the location of the first queen in the first queen in the first column, the second the location of the second queen in the second column, etc.
- The fitness of an individual is the number of pairs of non-attacking queens in the configuration

Having generated an initial population P at random

- We iteratively generate a new population newP from the current one P until some appropriate terminating condition is met
  - e.g., an individual in the population is "fit enough" or the number of iterations has hit some bound

Each iteration consists of the following process repeated —P— times

- randomly select two individuals X and Y from the current population P
  - so that "fitter" individuals are more likely to be selected
- from X and Y, reproduce a child Z
- with a small probability mutate the child Z
  - by e.g. replacing one randomly-chosen symbol in the string with some randomly-chosen symbol
- Add the child Z to the new population newP
- So, after this iteration, we have that —newP—=—P—
- The current population P is now replaced with the new population newP and the next iteration begins