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 (to be maximised/minimised)

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

Important: Disjoint graphs

If a graph is disjoint then your abstraction might not be suitable and so would never get over to the other section of the graph. Allowing empty sets in whatever you’re looking for will solve this.

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
- If there is an infinite number of successors then the first line of the loop will never finish
- In the TSP a complete tour is a state

1.1 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

2 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)

3 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
- The problem here is that the schedule can never reach zero as you are always dividing so you just get closer and closer. We solve this in the algorithm by using a small ϵ instead of zero so you can know when you are close enough to zero

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)
```

4 Genetic algorithms

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

- starts with a randomly generated population of individuals (set of states), 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

5 Reproduction and crossover

Reproduction of the child Z from two individuals X and Y is achieved through crossover:

- A random bit position in the strings X and Y is chosen so that both strings are partitioned into a prefix and a suffix (each of the same length)
- The suffix of Y is concatenated onto the prefix of X
- The suffix of X is concatenated onto the prefix of Y, so we get two children
- The fittest of the two children so obtained is taken to be the child Z

6 Proportional to fitness and minimising

- Choosing population members according to fitness is an implementation detail
- One way is via a roulette wheel approach
  - The total sum F of the fitness of all population individuals is calculated
  - Each population individual x whose fitness if f(x) is allocated a sector of the roulette wheel given by an angle of $(f(x) \times 360)/F$ deg at the centre. The wheel is ”spun” so as to choose a population member
- Genetic algorithms can also solve minimisation problems:
  - e.g. choose threshold value τ s.t. $\tau > f(x)$, for any population member x
  - New fitness function $f_{min} = \tau - f(x)$
  - The larger the τ, the closer the population’s fitness values are ”pushed together”

7 Genetic algorithm for the TSP

- In practice, we often have to adapt crossover and mutation to suit the problem in hand
- Consider the TSP - individuals consists of complete tours with fitness the tour length
- When we adapt a crossover, more often than not we find that the two resulting potential children do not actually encode tours
  - e.g. 1,4,5,3,2 and 3,5,4,2,1 result in 1,4,4,2,1 and 3,5,5,3,2
- We need to ”fix” the children so that they do correspond to tours
  - One way is to work through the suffixes replacing duplicates with missing cities, the order of occurrence in the other string
    - e.g. 1,4,4,2,1 and 3,5,5,3,2 become 1,4,3,2,5
- Also, we need to adapt the process of mutation for the same reason
  - e.g. swap the locations of two randomly chosen cities in the tour

Listing 3 Genetic-algorithm

```
1  P= randomly generated initial population
2  repeat
3      newP = ∅
4      for i = 1 to |P| do
5          X = randomly chosen individual of P with probability proportional to fitness
6          Y = randomly chosen individual of P with probability proportional to fitness
7          Z = child reproduced from X and Y
8          with small fixed probability mutate Z
9          newP = newP + Z
10     P=newP
11 until some individual is fit enough or a certain number of iterations have been done
12 return the fittest individual
```