Local Search

05 November 2020 13:11

Objectives:

1. Hill climbing
2. Greedy descent
3. Randomized algorithms
4. Simulated annealing
5. Genetic algorithms

Iterative Improvement Algorithms

Cases where we don't care about the path – we just want a goal state

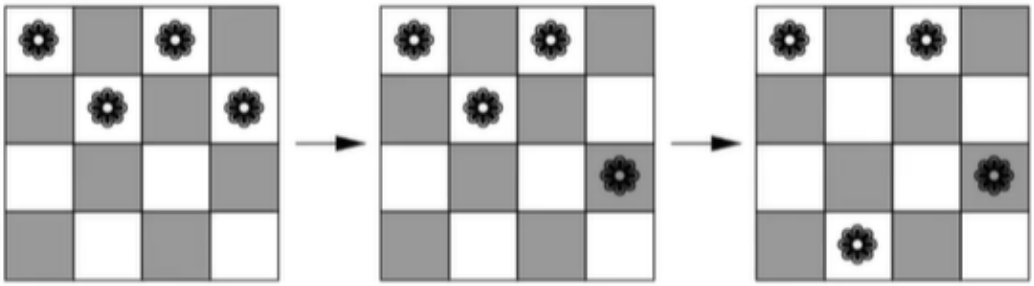
Systematically searching large spaces is not practical in many cases. We can use these iterative improvement algorithms to deal with these large search spaces.

The methods find solutions quickly on average, but do not guarantee that a solution will be found, even if one exists – they are therefore not able to prove a solution exists. They are useful when we already know a solution exists or is very likely to exist.

Local search methods start with a total assignment of a value to each variable and try to improve this assignment iteratively by taking improving steps, by taking random steps, or by restarting with another total assignment.

- Uses a single current state (not multiple paths) and typically moves to neighbours of state
- Not systematic, but low memory usage and can find solutions in continuous spaces
- Useful for optimisation problems including CSP's

The state space is the set of all configurations. The goal is a particular configuration
e.g. the travelling salesman or n-queens

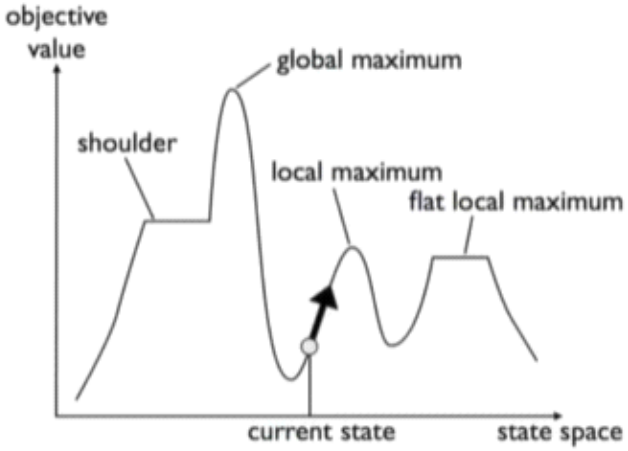


we can use iterative improvement algorithms on these problems, by keeping the current state and trying to improve it.

These algorithms often have constant space

Hill-Climbing

- Greedy local search – always tries to improve the current state, or reduce the cost if evaluation function is cost
- Each iteration moves in the direction of increasing value, or decreasing if evaluation is cost
- No search tree – just keep current state and its cost
- If > 1 alternative with equal cost, choose at random



Problems include local maxima where we have a local peak that is lower than the highest peak, but the algorithm will halt with a suboptimal solution

Also ridges, plateaux – flat area will conduct a random walk

Greedy Descent

Maintain an assignment of values for each variable
Repeatedly select a variable to change and a value for that variable

- Aim is to find an assignment with zero unsatisfied constraints
- Given an assignment of a value to each variable, a conflict is a violated constraint
- The goal is an assignment with zero conflicts
- Heuristic function to be minimized the number of conflicts

To choose a variable to change and add a new value:

- Find a variable-value pair that minimizes the number of conflicts
- Select a variable that participates in the most conflicts
- Select a value that minimizes the number of conflicts
- Select a variable that appeaars in any conflict
- Select a value that minimizes the number of conflicts
- Select a variable at random
- Select a value that minimizes the number of conflicts
- Select a variable and value at random, accept this change if it does not increase the number of conflicts

Complex Domains

- When the domains are small or unordered, the neighbours of an assignment can correspond to choosing another value for one of the variables
- When the domains are large and ordered, the neighbours of an assignment are the adjacent values for one of the variables
- If the domains are continuous, gradient descent changes each variable proportionally to the gradient of the heuristic function in that direction

The value of variable X_i goes from v_i to $v_i - \eta \frac{\partial h}{\partial X_i}$ where η is the step size.

Randomized Greedy Descent

As well as downward steps, we can allow for:

- Random steps: move to a random neighbour
- Random restart: reassign random values to all variables

Stochastic Local Search

Combination of:
Greedy descent – move to a lowest neighbour
Random walk
Random restart

Random Walk

Randomly sometimes choose a random variable value pair
When selecting a variable then a value:

- Sometimes choose any variable that participates in the most conflicts
- Sometimes choose any variable that participates in any conflict
- Sometimes choose any variable

Sometimes choose the best value and sometimes choose a random value

Simulated Annealing

1. Pick a variable at random and new value at random
2. If it is an improvement, adopt it
3. If it is not an improvement, adopt it probabilistically depending on a temperature parameter, T which is reduced over time

e.g. if T = 10 then there is a big chance we'll accept the proposed random change, but if T = 0.1 it is very small

- To prevent cycling we can maintain a tabu list of the last k assignments
- Do not allow an assignment that is already on the tabu list
- If k = 1, we do not allow an assignment of the same value to the variable chosen
- Can we very expensive if k is large

Simulated annealing requires an annealing schedule which specifies how T is reduced as the search progresss. Geometric cooling one of the most widely used schedules.

Parallel Search

A total assignment is called an individual
Idea: maintain a population of individuals instead of one
At every stage, update each individual in the population
Whenever an individual is a solution, it can be reported
Like k restarts but uses k times the minimum number of steps

Beam Search

Like parallel search, with k individuals but choose the k best out of all the neighbours
When k = 1, it is a greedy descent
Whe k = infinity it is a BFS
The value of k lets us limit space and parallelism

This can be made into a stochastic beam search is we probabilistically choose the k individuals at the next generation

The probability that a neighbour is chosen is proportional to its heuristic value
This maintains diversity among individuals

The heuristic value reflects the fitness of the individual

Like **asexual reproduction**: each individual mutates and the fittest ones survive

Genetic Algorithms

- Related to stochastic beam search
- Successor states obtained from two parents
- Start with a population of k individuals
- Each individual represented as a string over a finite alphabet, e.g. string of 0's and 1's. N-

queens would be [1 5 6 3 4 2 3 8] for one potential layout, for example.

Fitness Function

- Each individual in the population is evaluated by a fitness function
- Fitness function should return higher vlaues for better states
- Fitness function determines probability of being chosen for reproduction
- Pairs of individuals chosen according to these probabilities – those below a threshold can be culled

Crossover

For each chosen pair, a random crossover point is chosen from the string representation
Offspring are generated by crossing the parents strings at a chosen point
First child gets first part of string from 1 and second from 2, and second gets the opposite

We then subject the new individuals to mutation