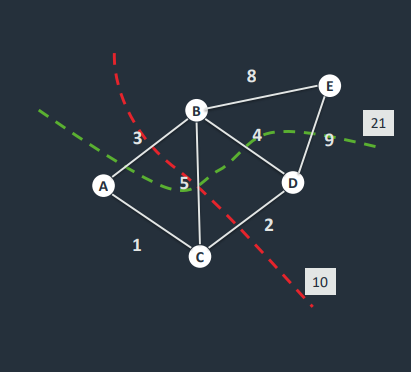
# Software 2 Theory Lecture 9 Optimisation

## Solving by Optimisation

## Maximum Cut Problem



Graph cut – The partition of the vertices of a graph into two sets – **the weight of the cut is the sum of the weights of the cut edges.**

**Maximum cut problem** – finding the partition with the max sum of weights.

Since each vertex is in either partition 1 or 2, we represent a possible solution by a set of binary variables:

* b(i) = 0 if vertex is in partition 1
* b(i) = 1 if vertex is in partition 2

There are 2n-1 possible solutions (too many to search for, for a large graph, so this would be slow with a brute force approach)

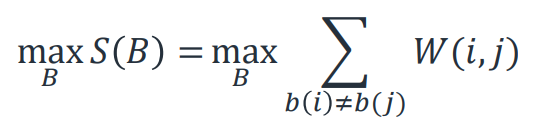
[00000] ~ [11111}; [11100] ~ [00011] -> duplicates (same values generated)

## Optimisation

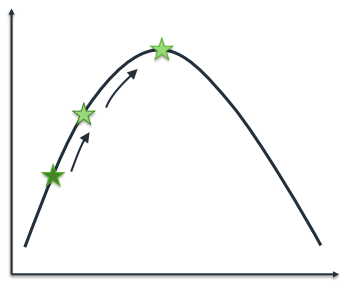
Lots of problems can be described in terms of an optimisation problem.

We define a **score function** – either a maximum or minimum at the solution to the problem, we then try to **find that maximum or minimum**.

Example: in the max-cut, the score function is the weight of the cut and the problem is:



Example 2: In the n-queens problem, we could score the number of attacks on a queen. The solution is the minimum of the score (ie: 0).

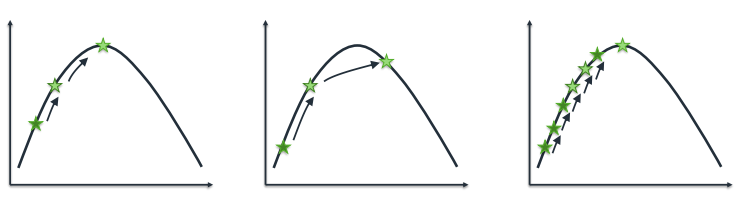
**Finding the minimum is as hard as solving the original problem**. However, the score function tells us something about the quality of all PSs and we can use this to develop **optimisation heuristics.**

The score function on the right –  
Start at some random point  
Move in the direction of increasing gradient  
We will reach the maximum – the solution

This is **gradient ascent**.

Gradient ascent is an example of a **local heuristic** in optimisation. We look at the local neighbourhood of the current PS to find a better PS (according to the score function).

A **move** is the operation which takes us from one configuration to a nearby one. IN the previous diagram, a move was one step to the left or right.



In the max-cut problem, a move could be swapping a vertex from one partition to another.

We explore all allowed moves, and select the one with the best score – repeating this takes us to the maximum (or minimum).

## Formal definition

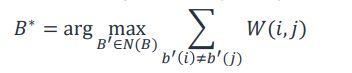
Let 𝑁 𝑚 = {𝑚′ |𝑚 → 𝑚′ } be the set of all moves we can take from m. These are called the **neighbours of m**.

Then gradient ascent chooses the move

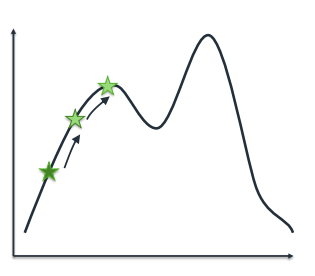
𝑚\* = arg max𝑚′∈𝑁(𝑚) 𝑆(𝑚′)

ie: the move from the set of allowed moves with the best score

Example: max-cut

 𝑁(𝐵 )= 𝐵 ∪ { (𝑏0, 𝑏1, … , ¬𝑏𝑖 , … , 𝑏𝑛−1) ∀𝑖}

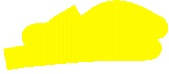
We explore n new configurations at each step.

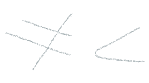
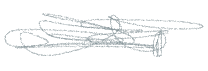
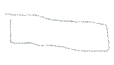


Gradient ascent is a **heuristic** – it wont **guarantee to find the best solution**. There are a number of potential solutions.

**Local maximum**- the method will find a maximum but it may not be the highest maximum in the score function.

We do not know about the existence of a better solution elsewhere.

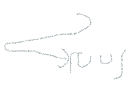


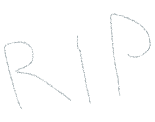
**Overshoot** – If moves are too big, we may miss the maximum (overshooting it), too small and it’ll take many steps to find the solution.

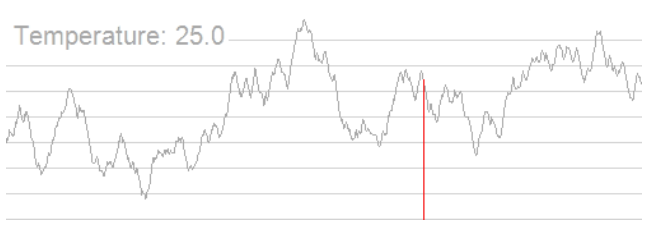
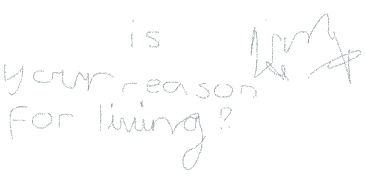
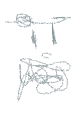


## Simulated annealing

**Simulated annealing** is a heuristic for optimisation which adds a random element to moves, to help overcome the problem of local maxima.

Moves are allowed that **decrease the score** – since a sequence of bad moves can get out of the local maximum and find a better one elsewhere.

The process is similar to the physical cooling of metal, hence the name HAHAHAHAHAHAHAHAHAHbugsislife





Dream gap ?manji=ro ‘opening’ sukima->gap2kki 2 -> ‘ni’(a pun)