What is optimisation?

Given a problem P, find the ***best*** solution to P.

Implications

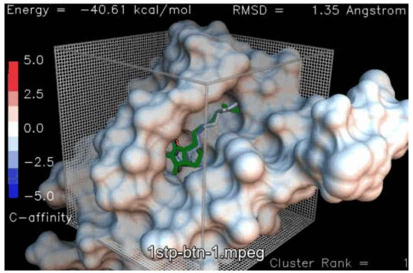
- There are multiple solutions
- We can ***measure*** the 'best' one.

Optimisation == iterative refinement?

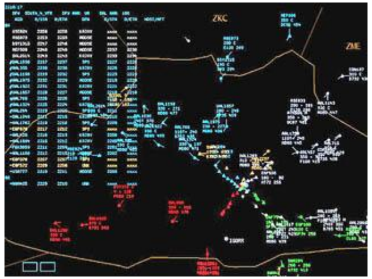
- Not necessarily.
- Some techniques 'move' closer to the best solution, while others take a random approach.

Machine Learning

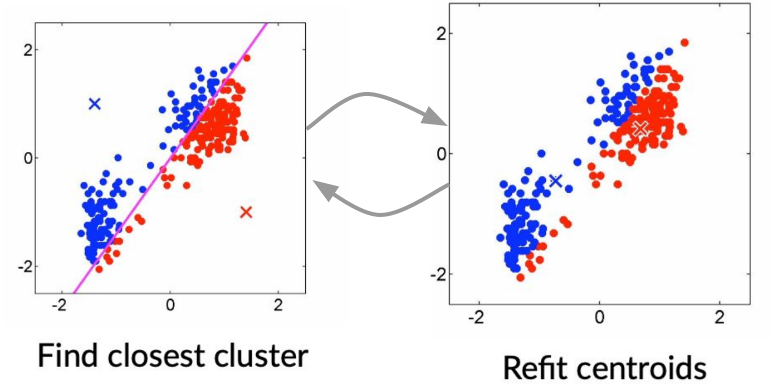
- Optimisation lies at the heart of machine learning.
- Learning with 'experience' (gradient descent)
- This said, not all ML tasks are strictly optimisation.



Lowest free-energy state molecular docking



Flight scheduling (Image courtesy of NASA.)

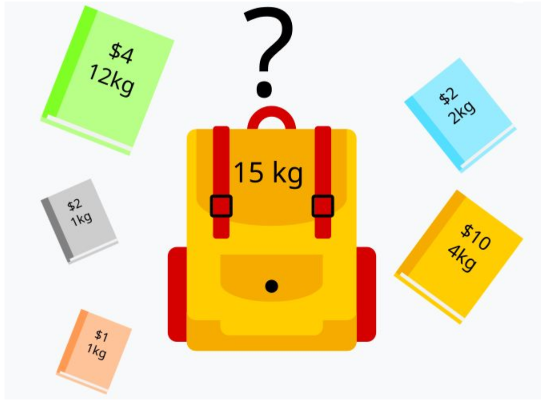


K-means iterative refinement of clusters

When do we use it?

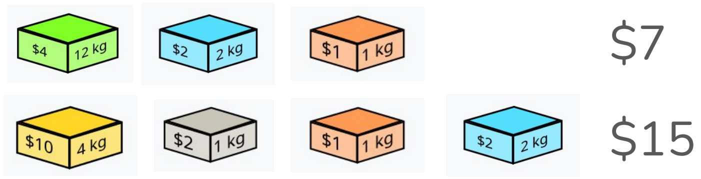
A problem is very difficult to understand, model, or solve using rule-based logic.

- Knapsack Problem (NP-complete complexity)
- Finding best evolutionary tree
- Minimize travel time between two locations



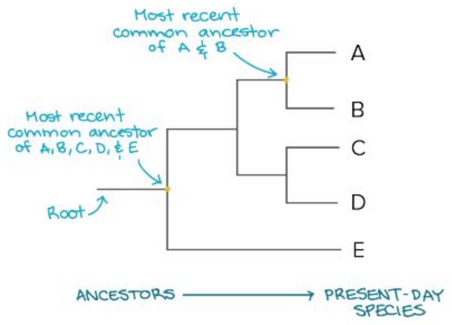
What can we do?

- Rather than trying to solve the problem, write a metric or function to assess how ***good*** a solution is.
- Generate many solutions, then pick the best.
- Generating solutions which ultimately reach (or approximate) the correct solution is optimisation.



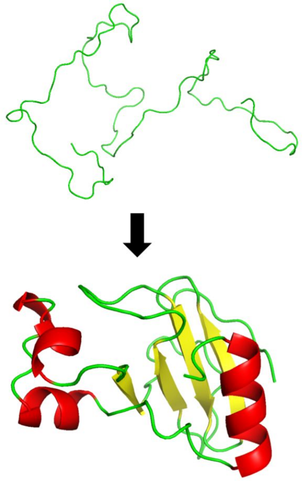
Applications in Bioinformatics

Phylogenetic Tree Building


Most recent common ancestor of A & B
A
B
C
D
E
Most recent common ancestor of A, B, C, D, & E
Root
ANCESTORS
PRESENT-DAY SPECIES

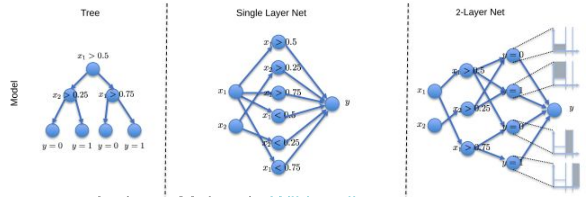
(Robert Bear via Khan Academy)

Protein folding



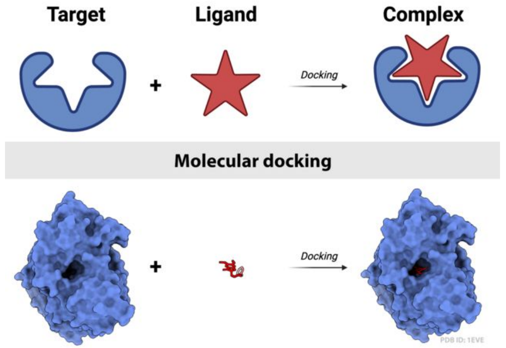
Christopher Rowley via Wikimedia commons

Deep Learning



Andreas Maier via Wikimedia commons

Molecular Docking



Diego Mariano via Wikimedia commons

Ingredients for Optimisation

Optimisation: Find the best solution to a task.

1. Generate solution(s)
2. Score fitness (objective function)
3. Return solution with best fitness.

Generating solutions

- Generate all solutions, a random sample, or…
- Use previous solutions to generate new solutions which may have better fitness.

Scoring fitness (objective function)

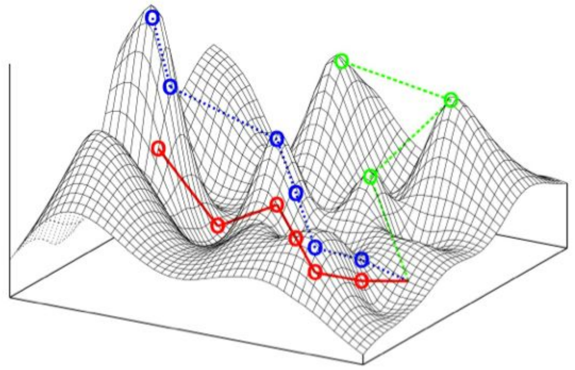
- Cost/energy function (want to minimize), or
- Score/reward function (want to maximise)

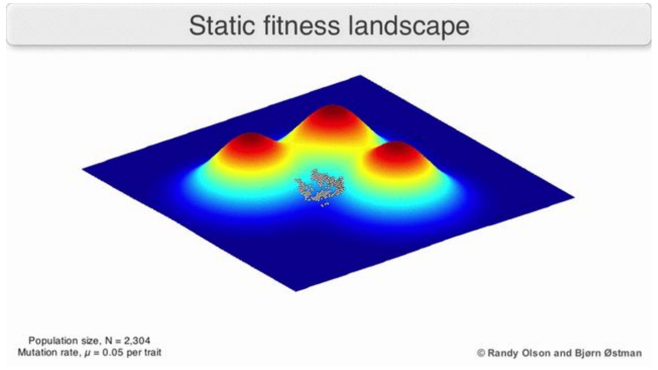
Search Heuristics

Some solutions are pretty good, while others are bad
Similar solutions (ie neighbours) often have similar fitness.

Rather than searching all viable solutions

- If we've found a good solution, can we refine it?
- If we've found a bad solution, can we re-roll?
  (ie move away from this solution)




Static fitness landscape
Population size, N = 2,304
Mutation rate, μ = 0.05 per trait
© Randy Olson and Bjørn Østman

Simulated Annealing

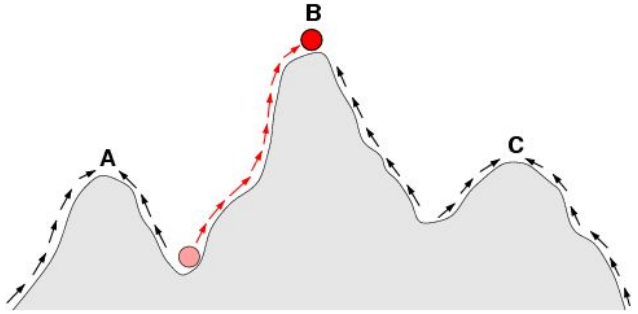
Use

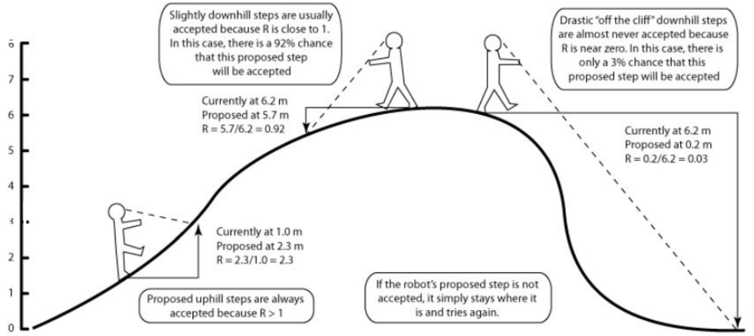
- Sometimes can’t use gradient descent. Eg discrete problems.
- Derivative of eqn. may not be known or too complex to compute.
- Example: The fitness landscape is not smooth.

Main ideas

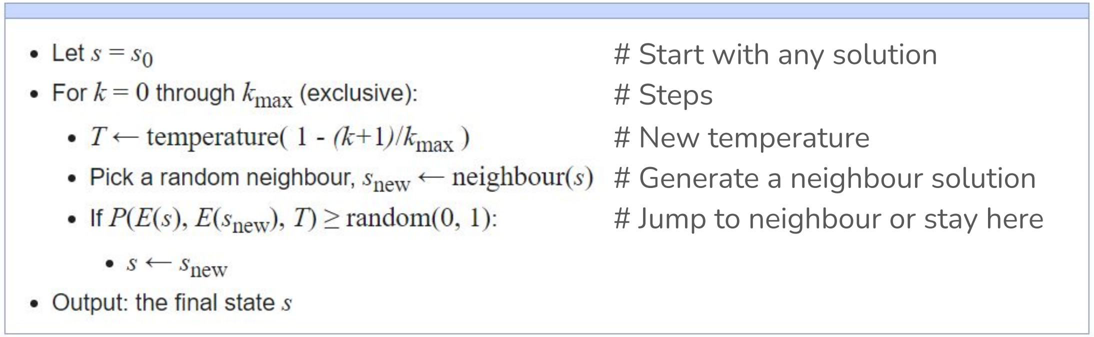
- Probabilistic technique
- If we find a better solution, we should jump to that solution.
- We can then generate ‘similar’ solutions, and see if any of those are better.

Naively, this would be prone to getting ‘stuck’ in local optima.
To remedy, let’s sometimes jump to worse solutions, hoping they eventually lead to better solutions.

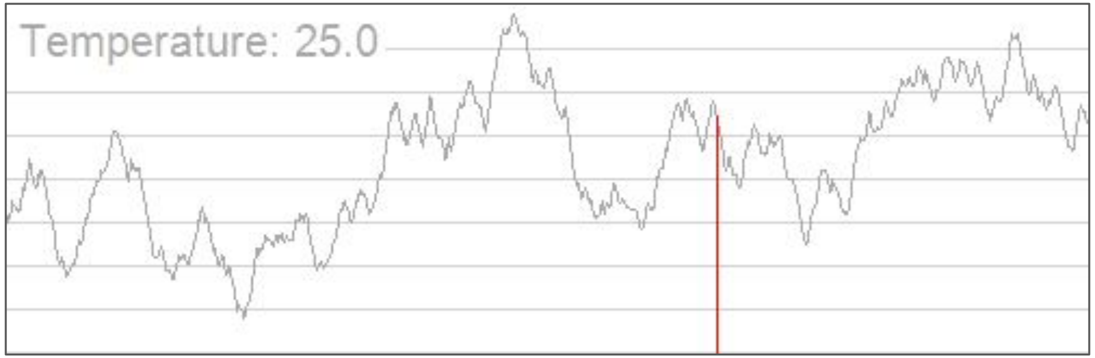



Slightly downhill steps are usually accepted because R is close to 1. In this case, there is a 92% chance that this proposed step will be accepted
Drastic "off the cliff" downhill steps are almost never accepted because R is near zero. In this case, there is only a 3% chance that this proposed step will be accepted
Currently at 6.2 m
Proposed at 5.7 m
R = 5.7/6.2 = 0.92
Currently at 6.2 m
Proposed at 0.2 m
R = 0.2/6.2 = 0.03
Currently at 1.0 m
Proposed at 2.3 m
R = 2.3/1.0 = 2.3
Proposed uphill steps are always accepted because R > 1
If the robot's proposed step is not accepted, it simply stays where it is and tries again.
0
1
2
4
5
6
7

Simulated Annealing



```
• Let $s = s_0$                                          # Start with any solution
• For $k = 0$ through $k_{max}$ (exclusive):            # Steps
    • $T \leftarrow$ temperature( $1 - (k+1)/k_{max}$ )  # New temperature
    • Pick a random neighbour, $s_{new} \leftarrow$ neighbour($s$)   # Generate a neighbour solution
    • If $P(E(s), E(s_{new}), T) \geq$ random(0, 1):     # Jump to neighbour or stay here
        • $s \leftarrow s_{new}$
• Output: the final state $s$
```



Neighbour

Generate new solution by making small change to current solution.
Eg change single edge in evolutionary tree.

Randomness

Allows us to jump to a ‘worse’ solution occasionally.
Enables us to break out of local maxima!

Temperature

Reduces over time.
Near the end of our run, we’re less likely to jump to ‘worse’ solutions.
Ensures convergence at (hopefully) global maxima.