Simulated Annealing

Rui Mendes

Numerical Optimization

- Set of numerical variables ($X \in \Re^n$)
- Objective function $f : \Re^n \to \Re$
- Our aim is to find a value X that minimizes or maximizes the objective function f
- $\{X \in \Re^n \,|\, \forall Y \in \Re^n \; f(X) \leq f(Y)\}$ if we're talking about minimization
- We are interested in non-linear optimization with no gradient information

Local Optimization

- We start with an initial value $X \in \Re^n$
- Choose a value Y in the vicinity of X
- If $f(Y)$ is better than $f(X)$, our current value will be Y
- Repeat this process while there are improvements

Local Optimization

Advantages

- Easy to implement

Disadvantages

- It often stagnates
- This happens because all points in the vicinity of the current solution X aren't better

Simulated Annealing (SA)

- Like local optimization, always accept better solutions
- Accept worst solutions with a probability that depends on the temperature
- Hot temperatures imply more permissive acceptance of worst solutions
- Cold temperatures imply it is much harder to accept worst solutions (i.e., they cannot be a lot worse)
- Temperature decreases with time
- There are several iterations of the algorithm with the same temperature, to reach the equilibrium
- The objective function is usually called energy and a lower energy means a better solution

Simulated Annealing

Variables

E_t	Current energy
E_{t-1}	Previous energy
T	Temperature
ΔE	$E_t - E_{t-1}$

Acceptance Probability

$$\begin{aligned} \Delta E <= 0 &\implies 1 \\ \Delta E > 0 &\implies e^{\frac{-\Delta E}{T}} \end{aligned}$$

Simulated Annealing

Algorithm

1. Generate initial solution;
2. Calculate its energy;
3. If the final temperature has not been reached:
4. For each of the N experiments:
    1. Locally perturb the current solution (may depend on the temperature);
    2. Calculate the energy of the new solution;
    3. If the energy is lower, accept the new solution;
    4. If the energy is higher, accept it with probability $e^{\frac{-\Delta E}{T}}$;
5. Lower the temperature and return to step 3.

Simulated Annealing

Cooling Schedule

1. $T_k = a \cdot T_{k-1}$ (or, alternatively $T_k = a^k \cdot T_0$) where $0 < a < 1$
2. $T_k = \frac{c}{\log k + d}$ where d is usually 1

- Scheme number 1 is the most common, but does not guarantee the optimal solution
- Scheme number 2 with a sufficiently high c (the highest possible energy found by the system) guarantees convergence; but it can be quite slow

Simulated Annealing

Parameters

- Initial temperature T_0
- Final temperature (or number of times the temperature is decreased)
- Number of iteratioons for each temperature
- Cooling schedule
- Perturbation type for the current solution (e.g., Gaussian mutation)
- Possible parameters for the perturbation
  - Standard deviation in the case of Gaussian mutation
  - It can use a monotonically increasing function of the current temperature (e.g., could simply be proportional to the temperature)

Pairing Cooling Schedules with Perturbation

Logarithmic

$T_k = \frac{T_0}{\log k}$
Gaussian perturbation

Cauchy

$T_k = \frac{T_0}{k}$
Cauchy perturbation

Simulated Annealing

Discussion

- Similar to local search
- But can accept worse solutions
- Is capable of leaving local optima
- When the temperature decreases, the probability of accepting worse solutions decreases
- If the cooling schedule is too fast, the algorithm will *stagnate*
- If the cooling schedule is too slow, the algorithm will not be *efficient*
- If the cooling schedule is slow enough, it is theoretically possible to find the optimum (*ergodic* property)
- Can be used with any kind of representation, as long as it is possible to mutate a solution

Other Encodings

- SA can be used for other encodings besides numerical optimization
- It has also been successfully used in combinatorial optimization
- Need to provide other perturbation types since Gaussian Mutation only works in real optimization
- Perturbation should be local
- Perturbation can be parametrized by the current temperature

Simulated Annealing applied to the Knapsack Problem

The Problem

- You have a knapsack with a carrying capacity C
- You have N objects
- Each object i has a value V_i and a weight W_i
- Your goal is to maximize the value of the objects while not exceeding the Carrying capacity

Encoding

- A solution will be a binary vector $X \in \mathcal{B}^N$
- The constraint is $\sum_{i=1}^{N} X_i \cdot W_i \leq C$
- The objective function is $f(X) = \sum_{i=1}^{N} X_i \cdot V_i$
- Our goal is to find $X^* = \max_{X \in \mathcal{B}^N} f(x)$

Suggestions

- Consider using Optuna to help you find the best parameters for this problem