Introduction to Optimization

Sergio Peignier

1 What is Optimization?

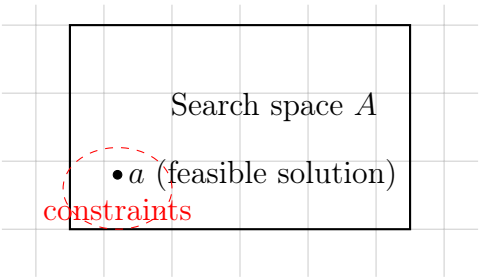
Definition 1.1. Let $f : A \to \mathbb{R}$ be a function defined on a set A (the *search space*). An optimization problem in minimization form with explicit constraints is often written:

$$\begin{aligned} \min_{a \in \mathbb{R}^n} \quad & f(a) \\ \text{subject to} \quad & g_i(a) \leq 0, \quad i = 1, \ldots, m, \\ & h_j(a) = 0, \quad j = 1, \ldots, p. \end{aligned}$$

Definition 1.2. The set of minimizers is denoted as:

$$\text{ArgMin}_{a \in A} f(a)$$

If the minimizer is unique we write $a^* = \text{ArgMin}_{a \in A} f(a)$. A Minimizer $a^* \in A$ satisfies $f(a^*) \leq f(a)$ for all $a \in A$. A is the *search space* — It is a set with all candidates we can consider. Object $a \in A$ is a *candidate solution* or *feasible solution* (if it respects the contraints). $f(a)$ is the *objective function* (also called loss, cost, utility depending on context). Maximization is analogous, We often prefer minimization notation and to maximize a function F we minimize $-F$.



1.1 Applications

- Statistics: minimize the residual norm or maximize the likelihood.
- Machine Learning: minimize prediction error on a dataset.
- Logistics: minimize total delivery distance.
- Engineering: minimize fuel consumption subject to performance constraints.

1.2 Discrete vs continuous search spaces

Continuous: $A \subseteq \mathbb{R}^n$, for example:

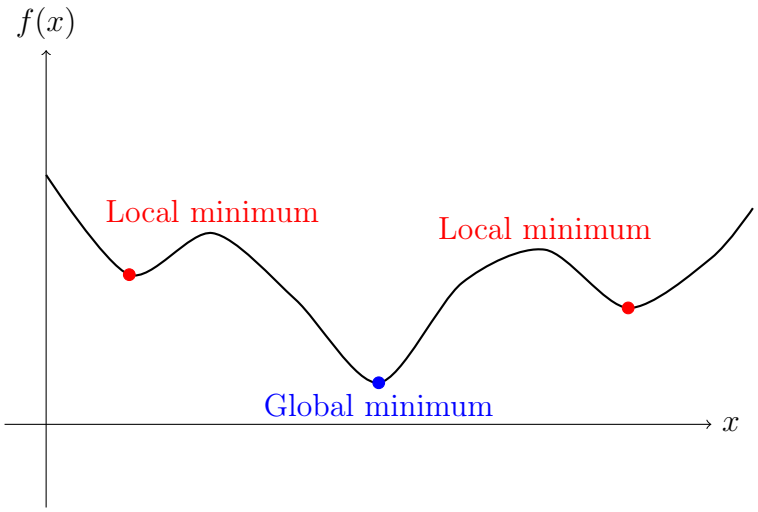
- Coefficients of a linear regression task.
- Find the barycenter of a group of points.

Discrete: A finite or combinatorial problem, for example:

- **[Traveling salesman problem]** we must visit five cities: Lyon, Madrid, Mexico DF, Buenos Aires, La Paz, which order would minimize the total distance?
- **[knapsack problem]** Let us consider a set of objects with different weights and values. Which elements should I include in my knapsack such that the full weight is less than or equal to a given limit (constraint) and the total value is as large as possible (maximization).

2 Local and Global Optima and Convexity

Definition 2.1 (Local and global minima)**.** A point $a' \in A$ is a *local minimizer* if there exists $\delta > 0$ such that for all $a \in A$ with $\|a - a'\| \leq \delta$ we have $f(a') \leq f(a)$. If the inequality holds for all $a \in A$ then a' is a *global minimizer*.



2.1 Convexity

Definition 2.2. Convex function and set A set $C \subseteq \mathbb{R}^n$ is convex if for any $x, y \in C$ and $\lambda \in [0, 1]$ we have $\lambda x + (1 - \lambda)y \in C$. A function f is convex on C if

$$f(\lambda x + (1 - \lambda)y) \leq \lambda f(x) + (1 - \lambda) f(y)$$

for all $x, y \in C$, $\lambda \in [0, 1]$.

Theorem 2.1. *If f is convex and differentiable on a convex domain C, any local minimizer is a global minimizer.*

Proof. **[sketch] Convexity implies a global lower bound via the gradient.**
For differentiable convex functions, the first-order inequality holds:

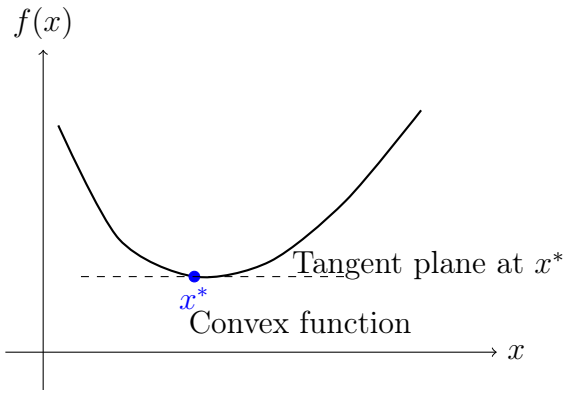
$$f(y) \geq f(x) + \nabla f(x)^\top (y - x) \quad \text{for all } x, y \in C.$$

Apply this with $x = x^*$. Using $\nabla f(x^*) = 0$ from Step 1 gives

$$f(y) \geq f(x^*) + 0^\top (y - x^*) = f(x^*).$$

Thus $f(y) \geq f(x^*)$ for all $y \in C$, proving that x^* is a global minimizer. □

Geometric Intuition Convexity ensures that at any point x, the tangent hyperplane lies *below* the graph of the function. If the gradient at x^* is zero, this hyperplane is horizontal. Because the function never dips below its own tangent plane, $f(x^*)$ must be the lowest possible value.



3 Multiobjective Optimization and Pareto Optimality

3.1 Problem statement

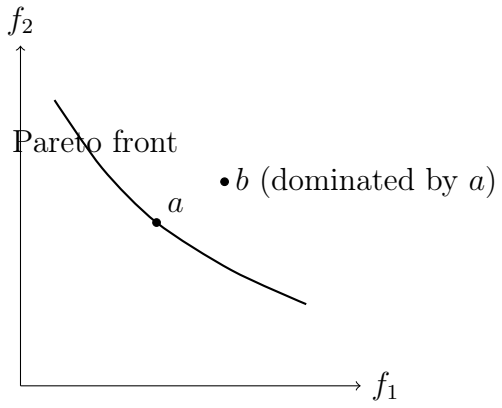
Multiobjective optimization involves several objectives:

$$\min_{a \in A} (f_1(a), f_2(a), \ldots, f_k(a)).$$

3.2 Pareto dominance and Pareto set

Definition 3.1 (Pareto dominance)**.** A solution a *dominates* b if $f_i(a) \leq f_i(b)$ for all i and $f_j(a) < f_j(b)$ for at least one j.

Definition 3.2 (Pareto optimal)**.** A solution is Pareto optimal if no other solution in A dominates it. The set of all Pareto optimal solutions is the Pareto set (or Pareto front when plotted in objective space).



3.3 Examples

- Designing an Efficient Yet Fast Car: minimize fuel consumption (use less gasoline) and maximize speed.
- Allocating Limited Budget in Public Health: maximize disease prevention (e.g., vaccination coverage), and minimize total cost.
- Choosing a Smartphone Design: maximize battery life and minimize device weight and size.
- Design a "Green Machine Learning Model": minimize the error of the model, and minimize the computation costs.

4 Optimization and Machine Learning

4.1 Short intuitive definition of Machine Learning

Machine Learning (ML) is the study of algorithms that learn patterns or predictive functions from data. Training most ML models is formulated as an optimization problem: find parameters that minimize a loss computed on some dataset.

4.2 Supervised example (linear regression)

The linear regression task, that you have aleady studied has been formulated as an optimization task.

4.2.1 Problem

Given data matrix $X \in \mathbb{R}^{n \times d}$ and responses $y \in \mathbb{R}^n$, minimize the square of the norm of the residual $\mathcal{L}(w)$:

$$\mathcal{L}(w) = \|Xw - y\|^2 = (Xw - y)^\top (Xw - y).$$

4.2.2 Formal solution

Set gradient to zero:

$$\nabla_w \mathcal{L}(w) = 2X^\top (Xw - y) = 0 \quad \Rightarrow \quad X^\top X w = X^\top y.$$

If $X^\top X$ is invertible,

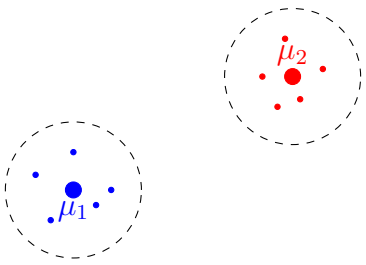
$$w^* = (X^\top X)^{-1} X^\top y.$$

4.2.3 Properties

In this case, we have a quadratic objective and a convex problem.

4.3 Unsupervised K-Means Clustering as an Optimization Problem

K-means is a classical unsupervised learning method used to partition a dataset into K clusters. The goal is to group data points so that points in the same cluster are "close" to each other, while points in different clusters are well separated. K-means can be seen as an optimization problem.



Intuition

Given n data points $x_1, \ldots, x_n \in \mathbb{R}^d$, the idea is to build K representative points, called **centroids**, such that each data point is assigned to the nearest centroid. The centroids act as "summaries" of clusters. K-means seeks centroids that minimize the total intra-cluster squared distance (as well as the corresponding assignments).

Optimization Formulation

Let

$$C = \{1, \ldots, K\}$$

be the set of cluster indices, and let

$$\mu_1, \ldots, \mu_K \in \mathbb{R}^d$$

be the cluster centroids. For each data point x_i, define its cluster assignment:

$$c(i) \in C.$$

The k-means objective, also called the *intra-cluster sum of squares* (SS), is

$$\min_{\mu_1,\ldots,\mu_K,\, c(1),\ldots,c(n)} \sum_{i=1}^{n} \|x_i - \mu_{c(i)}\|^2.$$

Thus k-means simultaneously searches over:

- the discrete assignments $c(i)$ (a *discrete search space*), and
- the continuous centroids $\mu_k \in \mathbb{R}^d$ (a *continuous search space*).

This makes k-means a discrete–continuous optimization problem.

Optimal Centroids for Fixed Assignments

If the assignments $c(i)$ are fixed, minimizing over the centroids yields a closed-form solution. Let $\mathcal{C}_k = \{i \mid c(i) = k\}$ be the set of points assigned to centroid k. For each $k \in \{1, \ldots, K\}$, the optimal centroid is the mean of all assigned points:

$$\mu_k = \frac{1}{|\mathcal{C}_k|} \sum_{i \in \mathcal{C}_k} x_i.$$

Optimal Assignments for Fixed Centroids

If the centroids are fixed, the optimal assignment for each point is:

$$c(i) = \arg\min_{k \in C} \|x_i - \mu_k\|^2,$$

i.e. assign each point to the nearest centroid.

Alternating Minimization

This leads an alternating minimization method:

1. Fix assignments, update centroids using the mean.
2. Fix centroids, update assignments using nearest-centroid rules.

Each step decreases aims at decreasing the objectives.

5 Optimization Algorithms: Overview and Examples

5.1 Gradient Descent

5.1.1 Assumptions and intuition

Let $f : \mathbb{R}^n \to \mathbb{R}$ be differentiable. The gradient $\nabla f(a)$ points in the direction of steepest increase. To decrease f, move opposite to the gradient.

5.1.2 Update rule

$$a^{(k+1)} = a^{(k)} - \gamma_k \nabla f(a^{(k)}),$$

where $\gamma_k > 0$ is the step size (learning rate). For small enough constant γ, gradient descent converges to a local minimum for smooth functions; for convex f it converges to the global minimum under mild conditions.

5.1.3 Pseudocode

```
a <- random_initial_guess()
for k = 1..K:
    g <- grad_f(a)
    a <- a - gamma * g
```

5.1.4 Choices and practicalities

- **Learning rate** γ: When it is too large it may lead to divergence, while if it is too small it may lead to slow convergence.
- **Stopping criteria**: gradient norm lower than a given threshold, change in objective lower than a given threshold, or maximal number of iterations reached.
- **Variants** used in deep-learning: momentum, adaptive rates (Adam, RMSprop), stochastic gradient descent (SGD).

5.2 Stochastic Optimization: Simulated Annealing

5.2.1 Intuition

Simulated Annealing is a probabilistic method to approximate a global optimum. It allows occasional uphill moves to escape local minima. The probability of accepting worse solutions decreases over time via a temperature decrease.

5.2.2 Components

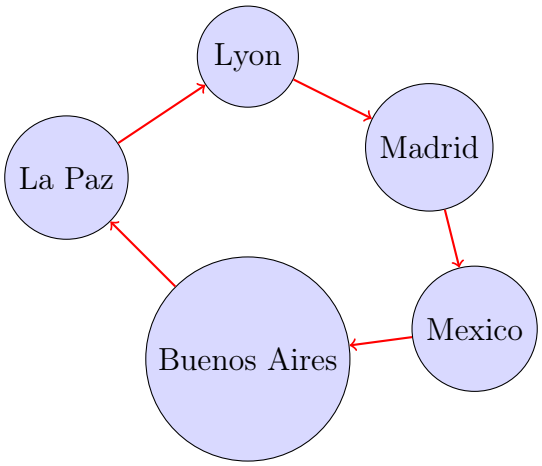
- State space S (search space).
- Energy $E(s)$ (objective to minimize).

- Neighbor function: propose s' near s.
- Acceptance probability $p(E(s'), E(s), T_k) = \exp(-(E(s') - E(s))/T_k)$ for $E(s') > E(s)$.
- Temperature schedule T_k decreasing with iteration k.

5.2.3 Pseudocode

```
s <- random_initial_state()
for k = 1 to K:
    T <- Compute Temperature(k)
    s' <- Compute neighbor(s)
    if E(s') < E(s):
        s <- s'
    else:
        with probability exp(-(E(s')-E(s))/T):
            s <- s'
```

5.2.4 Example: Traveling Salesman



In the Traveling Salesman Problem, the goal is to find the shortest possible route that visits each city exactly once and returns to the starting point (example: Lyon, Madrid, Buenos Aires, La Paz, Mexico). This is a discrete combinatorial optimization problem, with a search space that becomes extremely large as the number of cities increases. A state represents a candidate tour. A natural encoding is a sequence of cities for example: $s = (Lyon, Madrid, BuenosAires, LaPaz, Mexico)$ Each list order reflects the travel order.

Neighbor Function: To explore nearby solutions, a simple neighbor function could simplu swap two cities, for example: For:

$s = (Lyon, Madrid, BuenosAires, LaPaz, Mexico)$

swapping Madrid and La Paz leads to:

$s' = (Lyon, LaPaz, BuenosAires, Madrid, Mexico)$.

This modification SA to explore the search space gradually.

Energy function: The energy $E(s)$ corresponds to the objective function: the total length of the tour s, this is the sum of Euclidean distances between consecutive cities: A lower energy leads to a shorter tour and thus to a better solution.

5.3 Evolutionary Algorithms

5.3.1 Intuition

Evolutionary Algorithms are inspired on natural selection, the algorithm aims at evolving a population of candidates: some individuals are select according to fitness to reproduce and then variation is applied (mutation/crossover) to produce offspring, and the process is iterated for a given number of generations.

5.3.2 Components

- Population P of genomes.
- Fitness function F (to maximize).
- Selection mechanism (tournament, roulette, rank).
- Variation operators (mutation, crossover).

5.3.3 Pseudocode

```
P <- initialize_population()
while not termination:
    evaluate_fitness(P)
    parents <- select(P)
    offspring <- reproduce(parents)  # crossover + mutation
    P <- offspring
```

5.4 Example: Evolutionary Algorithm for the Traveling Salesman Problem

In the Traveling Salesman Problem, we are given a set of cities and must find the shortest possible tour that visits each city exactly once and returns to the starting point. This is a classical NP-hard combinatorial optimization problem, making it an excellent candidate for evolutionary algorithms (EAs), which excel on large, rugged, discrete search spaces.

Representation (Genome Encoding)

A common encoding for this problem could be a sequence of cities: For a set of cities

$$C = \{\text{Lyon}, \text{Madrid}, \text{Buenos Aires}, \text{La Paz}, \text{Mexico DF}\},$$

a genome (individual) may look like:

$$g = (\text{Lyon}, \text{Buenos Aires}, \text{Madrid}, \text{La Paz}, \text{Mexico DF}).$$

Each permutation represents a unique candidate.

Fitness Function

The fitness is defined as the **inverse of the tour length**:

$$\text{fitness}(g) = \frac{1}{\text{TourLength}(g)}.$$

The tour length is computed as the sum of Euclidean distances between consecutive cities (including the return to the starting city).

Initialization

A population is created by sampling random permutations. For example, with population size 4:

$$g_1 = (\text{Lyon}, \text{Madrid}, \text{Mexico}, \text{La Paz}, \text{Buenos Aires})$$
$$g_2 = (\text{Madrid}, \text{Mexico}, \text{La Paz}, \text{Lyon}, \text{Buenos Aires})$$
$$g_3 = (\text{Buenos Aires}, \text{La Paz}, \text{Lyon}, \text{Madrid}, \text{Mexico})$$
$$g_4 = (\text{Mexico}, \text{Buenos Aires}, \text{Madrid}, \text{Lyon}, \text{La Paz}).$$

Selection

Several selection operators exist, for example:

- **Tournament selection**: pick k individuals and select the best.
- **Fitness-proportional**: pick parents with a probability proportional to their fitness.
- **Rank-based** selection: In this case the probability for picking an individual as parent depends on its ranking (e.g., Each individual has 0.5 chances to be the offspring of the best individual, 0.25 to be the offspring of the second best and so on)

Suppose we apply tournament selection with $k = 2$ and pick g_1 and g_3 as parents.

Crossover Operator

In this case, the crossover must preserve permutation structure of the candidate solutions. For example, we could:

1. Choose a substring from parent 1.
2. Copy it into the child.
3. Fill remaining positions in the order they appear in parent 2.

Example:

$$p_1 = (\text{Lyon},\ \text{Madrid},\ \text{Mexico},\ \text{La Paz},\ \text{Buenos Aires})$$
$$p_2 = (\text{Buenos Aires},\ \text{La Paz},\ \text{Lyon},\ \text{Madrid},\ \text{Mexico})$$

Choose cutpoints 2 and 3 (using 1-based indexing). The substring from p_1 is:

$$(\text{Madrid},\ \text{Mexico}).$$

Fill remaining positions using p_2:

$$(\text{La Paz},\ \text{Lyon},\ \text{Buenos Aires}).$$

Final child:

$$c = (\text{La Paz},\ \text{Madrid},\ \text{Mexico},\ \text{Lyon},\ \text{Buenos Aires}).$$

Mutation Operator

A good example of point mutation could imply picking two cities and swaping them, for example:

(La Paz, **Madrid**, Mexico, Lyon, **Buenos Aires**) → (La Paz, **Buenos Aires**, Mexico, Lyon, **Madrid**)