I. Introduction

Depending on the context of the optimization problem, we can evaluate a solution, using:

- simulations
- physical experiments
- user study

The black-box (function) returns a single number indicating how good is the proposed solution (for now). We submit different solutions until we are happy enough (in theory. In practice we are constrained by our budget.

How do I decide the next solution?

> **Definition (Key performance measure)**
>
> The key performance measure is the number of function (black-box) evaluations.

> **Example (Applications)**
>
> Wherever **simulations** or **experiments** are needed (e.g. if we don't have an explicit model for the problem, in biology, engineering, machine learning, ...), or there is **no problem-specific algorithm available**.

Interactive exercise: finding the max value of a function defined over $\{1, \ldots, 10\} \times \{1, \ldots, 20\}$. Notice how we naturally alternate between **exploration** and **exploitation**.

II. Common black-box optimization algorithms

In practice, some real-world powerful algorithms are surprisingly simple.

How to overcome local optima?

1. Restart (either cause we're stuck or at certain intervals)
    1. at random
    2. with diversity in mind
2. Consider a larger neighborhood
3. Exploring taboo regions
4. Accept inferior solutions

Name	**Remarks**
Random sampling	Simple yet (surprisingly) efficient
Local search	Beware of local optimum
Simulated annealing	Tradeoff between exploration and exploitation (as a function of time)
Evolutionary algo.	Different generations
Genetic algo.	Evolutionary + mutations, crossovers
Population-based	See above with multiple individuals
Estimation of distribution	...
Swarm intelligence	...
Surrogate-based opt. (Bayesian opt.)	...
Partition-based methods	...

Name	Remarks
Gradient descent	…

Part of real-life research is to stick different methods together, and know *when* to switch between them. Different algorithms have different results on different problems.

We study the maximization of a function $f : \{0,1\}^n \to \mathbb{R}$.

III. Randomized Local Search (RLS)

- Sample $x \in \{0,1\}^n$ uniformly at random (*uar*) and evaluate $f(x)$ (**initialization**)
- For $i = 1, 2, \ldots$ do
  - sample $j \in [n]$ uar
  - create y by setting $y_k = \begin{cases} x_k \text{ for } k \neq j \\ 1 - x_k \text{ for } k = j \end{cases}$ (**mutation** or **variation**)
  - evaluate $f(y)$
  - if $f(y) \geq f(x)$ then $x \leftarrow y$ (**greedy selection**)

This is simple but it might be inefficient to find quickly big values and it could get trapped in local optimum.

Consider the function $\texttt{OneMax} : \{0,1\}^n \to \mathbb{R}, x \mapsto \sum_{i=1}^{n} x_i$.

Remark.
This is relevant because it is equivalent to the following problem for any $z \in \{0,1\}^n$:

$$f_z : x \mapsto |\{i \in [n] \mid x_i = z_i\}|$$

The lower bound for this problem is $\Theta\left(\frac{n}{\log} n\right)$.
Note that this is a game of **Mastermind**, which is literally a black-box optimization problem.

How long does RLS needs to solve `OneMax` in expectation?
See coupon collector problem (clickable self-promotion): $\mathcal{O}(n \log n)$.

Homework 1. Think of the lower bound $\mathcal{O}(n \log n)$.
Homework 2. Coupon collector problem.