# Optimization problems



Among all possible objects $x \in \mathcal{F} \subset \mathcal{S}$, we want to determine such an object $x_{\mathrm{OPT}}$ that optimizes (minimizes) the function $f$:

$$x_{\mathrm{OPT}} = \arg \min_{x \in \mathcal{F} \subset \mathcal{S}} f(x) \tag{1}$$

The space of candidate solutions $\mathcal{S}$ and the objective function $f$:

1. Representation of the solution
   - ✔ syntactical structure that holds the 'solution'
   - ✔ induces the search space $\mathcal{S}$ and its feasible part $\mathcal{F}$
2. Optimization criterion, objective function, evaluation function $f$
   - ✔ function that is optimized
   - ✔ 'understands' the solution representation
   - ✔ adds meaning (semantics) to the solution representation
   - ✔ gives us the measure of the solution quality (or, at least, allows us to say that one solution is better than the other)

**Black-box optimization**: the inner structure of function $f$ is unknown—it is black box for us (and for the algorithm); the function can be

- ✔ continuous × discrete
- ✔ differentiable
- ✔ decomposable
- ✔ noisy, time dependent

# Neighborhood, local optimum



The **neighborhood** of a point $x \in \mathcal{S}$:

$$N(x,d) = \{y \in \mathcal{S} | dist(x,y) \leq d\} \tag{2}$$

Measure of the **distance between points** $x$ **and** $y$: $\mathcal{S} \times \mathcal{S} \to \mathcal{R}$

- ✔ binary space: Hamming distance
- ✔ real space: Euclidean, Manhattan (City-block), Mahalanobis, ...
- ✔ matrices: Amari
- ✔ in general: number of applications of some operator that would transform $x$ into $y$ in $dist(x,y)$ steps

**Local optimum:**

- ✔ Point $x$ is local optimum, if $f(x) \leq f(y)$ for all points $y \in N(x,d)$ for some positive $d$.
- ✔ Small finite neighborhood (or the knowledge of derivatives) allows for validation of local optimality of $x$.

# Conventional optimization methods



Why are there so many of them?

- ✔ they are not robust, small change in the problem definition usually requires a completely different method

Classification of optimization algorithms (one of many possible):

1. Constructive algorithms
   - ✔ work with partial solutions, construct full solutions incrementally
   - ✔ not suitable for black-box optimization, must be able to evaluate partial solutions
   - ✔ require discrete search space
   - ✔ based on decomposition, arranging the search space in the form of a tree, ...
2. Generative algorithms
   - ✔ work with complete candidate solutions, generate them as a whole
   - ✔ suitable for black-box optimization, only complete solutions need to be evaluated
   - ✔ can be interrupted anytime—always have a solution to provide; this solution is usually not optimal (needn't be even locally optimal)

# The Problem of Local Optimum



Goal of these algorithms:

- ✔ to lower the risk of getting stuck in local optimum.

How can we avoid local optimum?

1. Run the optimization algorithm from a different initial point.
   - ✔ e.g. iterated hill-climber
2. Introduce memory and do not stop the search in LO
   - ✔ e.g. taboo search
3. Introduce a probabilistic aspect
   - ✔ stochastic hill-climber
   - ✔ simulated annealing
   - ✔ evolutionary algorithms, swarm intelligence
4. Perform the search in several places in parallel
   - ✔ evolutionary algorithms, swarm intelligence (population-based optimization algorithms)

# Local Search, Hill-Climbing

**Algorithm 1:** LS with First-improving Strategy

```
1 begin
2     x ← Initialize()
3     while not TerminationCondition() do
4         y ← Perturb(x)
5         if BetterThan(y, x) then
6             x ← y
```

Features:

- ✔ usually stochastic, possibly deterministic, applicable in discrete and continuous spaces

**Algorithm 2:** LS with Best-improving Strategy

```
1 begin
2     x ← Initialize()
3     while not TerminationCondition() do
4         y ← BestOfNeighborhood(N(x, D))
5         if BetterThan(y, x) then
6             x ← y
```

Features:

- ✔ deterministic, applicable only in discrete spaces, or in descretized real-valued spaces, where $N(x, d)$ is finite and small

The influence of the neighborhood size:

- ✔ Small neighborhood: fast search, huge risk of getting stuck in local optimum (in zero neghborhood, the same point is generated over and over)
- ✔ Large neighborhood: lower risk of getting stuck in LO, but the efficiency drops. If $N(x, d) = \mathcal{S}$, the search degrades to
  - ✘ random search in case of first-improving strategy, or to
  - ✘ exhaustive search in case of best-improving strategy.

# Taboo Search



**Algorithm 5:** Taboo Search

```
1 begin
2     x ← Initialize()
3     y ← x
4     M ← ∅
5     while not TerminationCondition() do
6         y ← BestOfNeighborhood(N(y, D) − M)
7         M ← UpdateMemory(M, y)
8         if BetterThan(y, x) then
9             x ← y
```

Meaning of symbols:

- ✔ $M$ — memory holding already visited points that become taboo
- ✔ $N(\mathbf{x}, d) \cap M$ — set of states which would arise by taking back some of the previous decisions

Features:

- ✔ canonical version is based on the local search with best-improving strategy
- ✔ first-improving can be used as well
- ✔ difficult use in real domain, usable mainly in discrete spaces

# Stochastic Hill-Climber

Assuming minimization:

**Algorithm 6:** Stochastic Hill-Climber

```
1 begin
2     x ← Initialize()
3     while not TerminationCondition() do
4         y ← Perturb(x)
5         p = 1 / (1 + e^((f(y) − f(x)) / T))
6         if rand < p then
7             x ← y
```

Probability of accepting a new point $\mathbf{y}$ when $f(\mathbf{y}) - f(\mathbf{x}) = -13$:

| T | $e^{-\frac{13}{T}}$ | $p$ |
|---|---|---|
| 1 | 0.000 | 1.000 |
| 5 | 0.074 | 0.931 |
| 10 | 0.273 | 0.786 |
| 20 | 0.522 | 0.657 |
| 50 | 0.771 | 0.565 |
| $10^{10}$ | 1.000 | 0.500 |

Features:

- ✔ It is possible to move to a worse point *anytime*.
- ✔ $T$ is the algorithm parameter and stays constant during the whole run.
- ✔ When $T$ is low, we get local search with first-improving strategy
- ✔ When $T$ is high, we get random search

Probability of accepting a new point $\mathbf{y}$ when $T = 10$:

| $f(\mathbf{y}) - f(\mathbf{x})$ | $e^{\frac{f(\mathbf{y}) - f(\mathbf{x})}{10}}$ | $p$ |
|---|---|---|
| -27 | 0.067 | 0.937 |
| -7 | 0.497 | 0.668 |
| 0 | 1.000 | 0.500 |
| 13 | 3.669 | 0.214 |
| 43 | 73.700 | 0.013 |

# Simulated Annealing

**Algorithm 7:** Simulated Annealing

```
1  begin
2      x ← Initialize()
3      T ← Initialize()
4      while not TerminationCondition() do
5          y ← Perturb(x)
6          if BetterThan(y,x) then
7              x ← y
8          else
9              p = e^{-(f(y)-f(x))/T}
10             if rand < p then
11                 x ← y
12         if InterruptCondition() then
13             T ← Cool(T)
```

Very similar to stochastic hill-climber

Main differences:

- ✔ If the new point **y** is better, it is *always* accepted.
- ✔ Function `Cool`$(T)$ is the *cooling schedule*.
- ✔ SA changes the value of $T$ during the run:
  - ✘ $T$ is high at beginning: SA behaves like random search
  - ✘ $T$ is low at the end: SA behaves like deterministic hill-climber

Issues:

- ✔ How to set up the initial tempereature $T$ and the cooling schedule `Cool`$(T)$?
- ✔ How to set up the interrupt and termination condition?