AI505

Optimization

Beyond Local Optima

Marco Chiarandini

Department of Mathematics & Computer Science
University of Southern Denmark

Outline

Benchmarking in the COCO Platform

- Functions divided in suites.
- Functions, f_i, within suites are distinguished by their identifier $i = 1, 2, \ldots$.
- parametrized by the (input) dimension, n, and
- instance number, j. (j as an index to a continuous parameter vector setting, eg, search space translations and rotations).

$$f_i^j \equiv f[n, i, j] : \mathbb{R}^n \to \mathbb{R} \qquad \boldsymbol{x} \mapsto f_i^j(\boldsymbol{x}) = f[n, i, j](\boldsymbol{x}).$$

- Varying n or j leads to a variation of the same function i of a given suite.
- Fixing n and j of function f_i defines an **optimization problem instance** $(n, i, j) \equiv (f_i, n, j)$ that can be presented to the solver.

Why?

Varying the instance parameter j represents a natural randomization for experiments in order to:

- generate repetitions on a single function for deterministic solvers, making deterministic and non-deterministic solvers directly comparable (both are benchmarked with the same experimental setup)
- average away irrelevant aspects of the function definition
- alleviate the problem of overfitting, and
- prevent exploitation of artificial function properties

BBOB Functions

- All benchmark functions are scalable with the dimension.
- Most functions have no specific value of their optimal solution (they are randomly shifted in x-space).
- All functions have an artificially chosen optimal function value (they are randomly shifted in f-space).

Runtime and Target Values

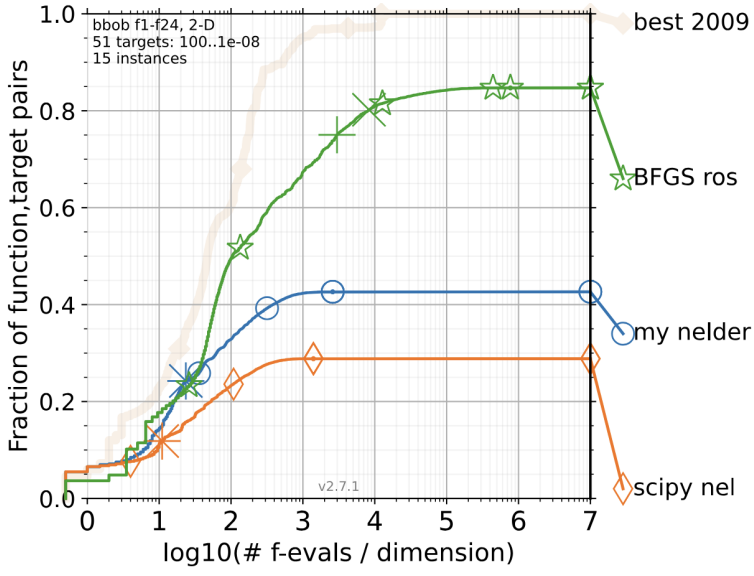
- Runtime of a solver on a problem is the hitting time condition.
- define a non-increasing quality indicator measure and prescribe a set of **target values**, t.
- target values are compared with the best so-far-seen f-value.
- For a single run, the solver run is successful on the problem instance (f_i, n, j) when the best-so-far f-value reaches the target value t.
- COCO collects hundreds of different target values from each single run.
- targets $t(i, j)$ depend on the problem instance in a way to make problems comparable
- typically, target values are set to known or estimated optimal solution plus an added **precision**
- **runtime** is the number of f-evaluations needed to solve the problem $(fi, n, j, t(i, j))$.
- only runtimes to comparable target values can be aggregated among problem instances.

Simulated Restarts

- If a solver does not hit the target t in a given single run, the run is considered to be unsuccessful.
- The runtime of this single run remains undefined but is bounded from below by the number of evaluations conducted during the run $\tau \in [T, \infty]$
- T depends on the termination condition encountered. It can be the budget of evaluations.
- For hard problem instances COCO uses **budget-based target values**:
  For any given budget, COCO selects from the finite set of recorded target values the easiest (i.e., largest) target for which the expected runtime of all solvers (ERT) exceeds the budget.
- With unsuccessful runs: draw further runs from the set of tried problem instances, uniformly at random with replacement, until find an instance, j, for which $(f_i, n, j, t(i,j))$ is solved.
  the runtime is then the sum of the overall time spend and associated to the initially unsolved problem instance.
  `print: '|' if problem.final_target_hit, ':' if restarted and '.' otherwise.`

Aggregation

- Aggregation is to compute a statistical summary over a set or subset of problem instances over which we assume a uniform distribution
- If we can distinguish between problems easily, for example, according to their input dimension, we can use the information to select the solver, hence not worth aggregating data
- Empirical cumulative distribution functions of runtimes (**runtime ECDFs**)
  - Absolute distributions vs Performance profiles (ECDFs of runtimes relative to the respective best solver)
  - aggregate runtimes from several targets per function (!?)
- arithmetic average, as an estimator of the expected runtime. The estimated expected runtime of the restarted solver, ERT, is often plotted against dimension to indicate scaling with dimension.
  alternatives: average of log-runtimes $\equiv$ geometric average or shifted geometric mean


bbob f1-f24, 2-D
51 targets: 100..1e-08
15 instances
best 2009
BFGS ros
my nelder
scipy nel
v2.7.1
Fraction of function,target pairs
log10(# f-evals / dimension)
0.0
0.2
0.4
0.6
0.8
1.0
0
1
2
3
4
5
6
7

Reference

Nikolaus Hansen, Anne Auger, Raymond Ros, Olaf Mersmann, Tea Tušar & Dimo Brockhoff (2021) COCO: a platform for comparing continuous optimizers in a black-box setting, Optimization Methods and Software, 36:1, 114-144, DOI: 10.1080/10556788.2020.1808977

Outline

- Employ randomness strategically to help explore design space
- Randomness can help escape local minima
- Increases chance of searching near the global minimum
- Typically rely on pseudo-random number generators to ensure repeatability
- Control over randomness and the exploration vs exploitation trade off.

Noisy Descent

- Saddle points, where the gradient is very close to zero, can cause descent methods to select step sizes that are too small to be useful
- add Gaussian noise at each descent step

$$\boldsymbol{x}_{k+1} \leftarrow x_k + \alpha \nabla f(\boldsymbol{x}_k) + \epsilon_k$$

$\epsilon_k \sim \mathcal{N}(0, \sigma_k^2)$

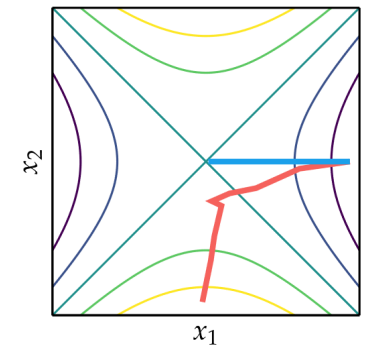
- $\sigma_k = \frac{1}{k}$

Stochastic Gradient Descent

- evaluates gradients using randomly chosen subsets of the training data (**batches**)
- significantly less expensive computationally than calculating the true gradient at every iteration and yields same effect as noisy gradient approximation
- helping traverse past saddle points Convergence guarantees for stochastic gradient descent require that the positive step sizes be chosen such that:

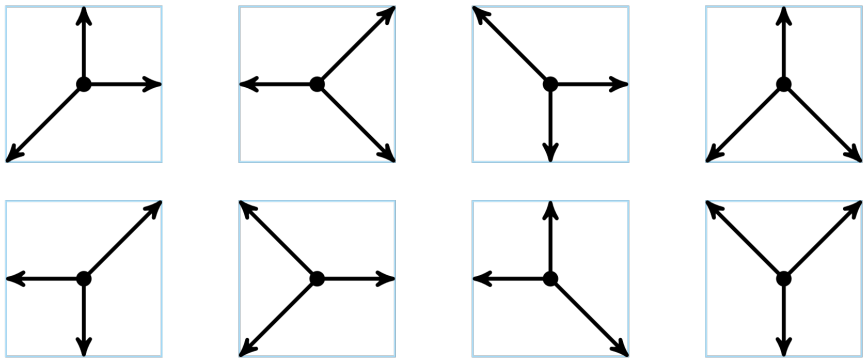
$$\sum_{k=1}^{\infty} \alpha_k = \infty \qquad \sum_{k=1}^{\infty} \alpha_k^2 < \infty$$

- ensure that the step sizes decrease and allow the method to converge, but not too quickly so as to become stuck away from a local minimum


x_2
x_1
stochastic gradient descent
steepest descent

Mesh Adaptive Direct Search

- Similar to generalized pattern search but uses random positive spanning directions
- Example: set of positive spanning sets constructed from nonzero directions $d_1, d_2 \in \{-1, 0, 1\}$.



- Construct lower triangular matrix L sampling from:

$$\{-1/\sqrt{\alpha_k}+1, -1/\sqrt{\alpha_k}+2, \ldots, 1/\sqrt{\alpha_k}-1\}$$

- permute rows and columns of L randomly to obtain a matrix D whose columns correspond to n directions that linearly span $\mathbb{R}^n$. The maximum magnitude among these directions is $1/\sqrt{\alpha_k}$
- add one additional direction $\boldsymbol{d}_{n+1} = -\sum_{i=1}^{n} \boldsymbol{d}_i$ or add n additional directions $d_{n+j} = -\boldsymbol{d}_j$
-

$$\alpha_{k+1} \leftarrow \begin{cases} \alpha_k/4 & \text{if no improvement was found in this iteration} \\ \min(1, 4\alpha_k) & \text{otherwise} \end{cases}$$

- If $f(\boldsymbol{x}_k = \boldsymbol{x}_{k-1} + \alpha\boldsymbol{d}) < f(x_{k-1})$, then the queried point is $\boldsymbol{x}_{k-1} + 4\alpha\boldsymbol{d} = x_k + 3\alpha\boldsymbol{d}$

Simulated Annealing

- often used on functions with many local minima due to its ability to escape local minima.
- a candidate transition from $\boldsymbol{x}$ to $\boldsymbol{x}'$ is sampled from a transition distribution p
- **Metropolis acceptance criterion:**

$$p(\boldsymbol{x}, \boldsymbol{x}') = \begin{cases} 1 & \text{if } \Delta y \leq 0 \\ e^{-\frac{\Delta y}{t}} & \text{if } \Delta y > 0 \end{cases}$$

$$\Delta y = f(\boldsymbol{x}') - f(\boldsymbol{x})$$

Annealing Plan

- a logarithmic annealing schedule

$$t_k = t_0 \frac{\ln(2)}{\ln(k+1)}$$

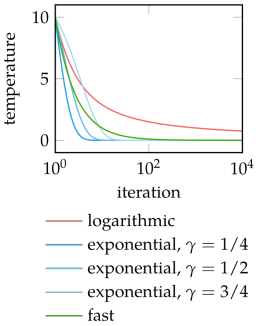
guaranteed to asymptotically reach the global optimum under certain conditions, but it can be slow in practice.

- exponential annealing schedule, more common, uses a simple decay factor:

$$t_{k+1} = \gamma t_k$$

- fast annealing

$$t_k = \frac{t_0}{k}$$



Simulated Annealing

- Corana et al 1987 introduced variable step-size $\boldsymbol{v}$ (separate directional components)
- Cycle of random moves, one in each direction

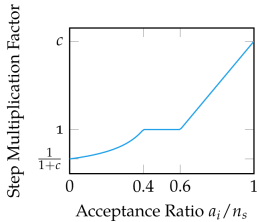
$$\boldsymbol{x}' = \boldsymbol{x} + r v_i e_i$$

where r is randomly sampled from $\{-1, 1\}$

- after n_s cycles, step size is adjusted according to

$$v_i = \begin{cases} v_i \left(1 + c_i \frac{a_i/n_s - 0.6}{0.4}\right) & \text{if } a_i > 0.6 n_s \\ v_i \left(1 + c_i \frac{0.4 - a_i/n_s}{0.4}\right)^{-1} & \text{if } a_i < 0.4 n_s \\ v_i & \text{otherwise} \end{cases}$$

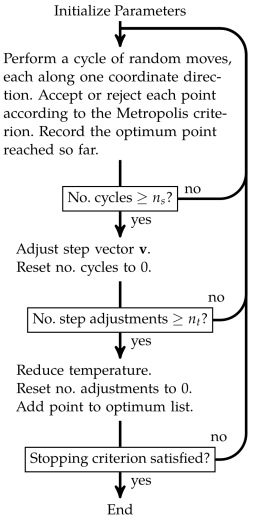
$\boldsymbol{a}$: accepted steps in each direction; $\boldsymbol{c}$: typically 2.



regulates the ratio of accepted-to-rejected points to about 50%.

Simulated Annealing

- Temperature reduction occurs every n_t step adjustments, which is every $n_s \cdot n_t$ cycles
- termination when the temperature sinked low and no improvement expected or when no movement more than ϵ in last n_ϵ iterations



Cross-Entropy Method

- Maintains explicit probability distribution over design space often called a **proposal distribution**
- Requires choosing a family of parameterized distributions
- At each iteration, a set of design points are **conditionally independently** sampled from the proposal distribution; these are evaluated and ranked
- The best-performing subset of samples, called **elite samples**, are retained
- The proposal distribution parameters are then updated based on the elite samples, and the next iteration begins

Cross-Entropy Method

- Cross-entropuy is a measure of divergence between two probability distributions p and q (related to Kullback-Leibler divergence)
- Here we measure cross-entropy in a case where the distribution of p (the one of optimal solutions) is unknown.
- A model is created and then its cross-entropy is measured on the elite set to assess how accurate the model is in predicting this set.
- Let q be the true distribution of the optimal solutions, and p the distribution of solutions as predicted by the model. Since the true distribution is unknown, cross-entropy cannot be directly calculated. Instead, an estimate of cross-entropy is:

$$H(T, p) = -\sum_{i=1}^{N} \frac{1}{N} \log_2 p(x_i)$$

where N is the size of the elite set, and $p(x)$ is the probability of solution x estimated from the training set.

Cross-Entropy Method

cross-entropy $\equiv$ Maximum likelihood estimation

- A widely used frequentist estimator is maximum likelihood, in which $\boldsymbol{\theta}$ is set to the value that maximizes the likelihood function $p(\boldsymbol{x} \mid \boldsymbol{\theta})$.
- This corresponds to choosing the value of $\boldsymbol{\theta}$ for which the probability of the observed data set is maximized.
- In the machine learning literature, the **negative log of the likelihood function** is called an error function. Because the negative logarithm is a monotonically decreasing function, maximizing the likelihood is equivalent to minimizing the error.
- Suppose our data set consists of N data points $\boldsymbol{x} = \{x_1, \ldots, x_N\}$:

$$\mathcal{L}(\boldsymbol{x} \mid \boldsymbol{\theta}) = p(\boldsymbol{x} \mid \boldsymbol{\theta}) = \prod_{i=1}^{N} p(x_i \mid \boldsymbol{\theta}) \qquad \text{likelihood}$$

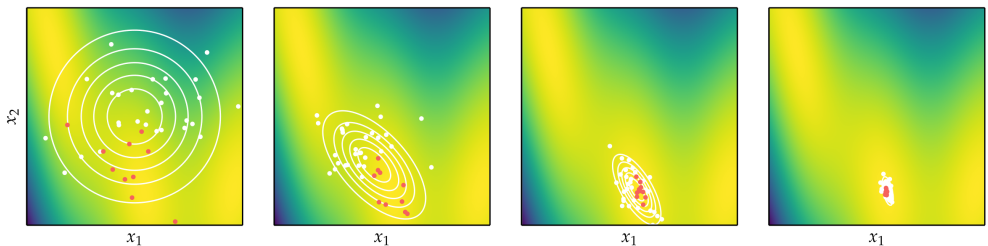
$$\mathcal{E}(\boldsymbol{x} \mid \theta) = -\log \mathcal{L}(\boldsymbol{x})$$

Cross-Entropy Method

- $$\min_{\boldsymbol{\theta}} \mathcal{E}(\boldsymbol{x} \mid \theta) = \min_{\boldsymbol{\theta}} \left(-\log \mathcal{L}(\boldsymbol{x})\right) = -\max_{\boldsymbol{\theta}} \log \mathcal{L}(\boldsymbol{x})$$ maximum log-likelihood

$$\min \left(-\sum_{i=1}^{N} \log p(x_i \mid \boldsymbol{\theta}) \right)$$

- hence minimizing the negative of the log-likelhoood is equivalent to minimizing the entropy



Natural Evolution Strategies

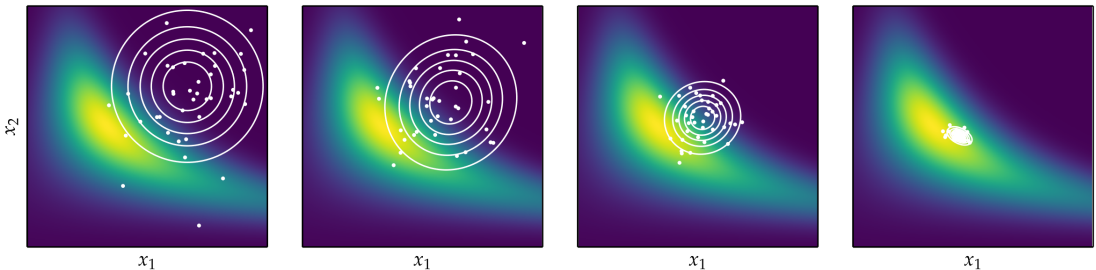
- Similar to cross-entropy method, except instead of parameterizing distribution based on elite samples, it is optimized using gradient descent
- The aim is to minimize the expectation

$$E_{\boldsymbol{x}\sim p(\cdot|\boldsymbol{\theta})}[f(\boldsymbol{x})].$$

- The distribution parameter gradient is estimated from the set of function evaluations

Input: $f, \boldsymbol{\theta}, k_{MAX}, N = 100, \alpha = 0.01$
Output: $\boldsymbol{\theta}$
for k in $1, \ldots, k_{MAX}$ **do**
  Let $X = \{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_N\}$ be a conditionally independent sample of size N from $p(\boldsymbol{\theta})$;
  $\boldsymbol{\theta}_{k+1} = \boldsymbol{\theta}_k - \alpha \frac{1}{N} \sum_{i=1}^{N} f(\boldsymbol{x}_i) \nabla_{\boldsymbol{\theta}} \log p(\boldsymbol{x}_i, \theta)$;

$$
\begin{aligned}
\nabla_\theta \, \mathbb{E}_{\mathbf{x} \sim p(\cdot \mid \theta)}[f(\mathbf{x})] &= \int \nabla_\theta \, p(\mathbf{x} \mid \theta) f(\mathbf{x}) \, d\mathbf{x} \\
&= \int \frac{p(\mathbf{x} \mid \theta)}{p(\mathbf{x} \mid \theta)} \nabla_\theta \, p(\mathbf{x} \mid \theta) f(\mathbf{x}) \, d\mathbf{x} \\
&= \int p(\mathbf{x} \mid \theta) \nabla_\theta \log p(\mathbf{x} \mid \theta) f(\mathbf{x}) \, d\mathbf{x} \\
&= \mathbb{E}_{\mathbf{x} \sim p(\cdot \mid \theta)}[f(\mathbf{x}) \nabla_\theta \log p(\mathbf{x} \mid \theta)] \\
&\approx \frac{1}{m} \sum_{i=1}^{m} f(\mathbf{x}^{(i)}) \nabla_\theta \log p(\mathbf{x}^{(i)} \mid \theta)
\end{aligned}
$$


x_2
x_1
x_1
x_1
x_1

Covariance Matrix Adaptation Evolutionary Strategy (CMA-ES)

Summary

- Stochastic methods employ random numbers during the optimization process
- Simulated annealing uses a temperature that controls random exploration and which is reduced over time to converge on a local minimum
- The cross-entropy method and evolution strategies maintain proposal distributions from which they sample in order to inform updates
- Natural evolution strategies uses gradient descent with respect to the log likelihood to update its proposal distribution
- Covariance matrix adaptation is a robust and sample-efficient optimizer that maintains a multivariate Gaussian proposal distribution with a full covariance matrix