Parameter Tuning.
Automatic Algorithm Configuration

Petr Pošík

Configurable algorithms

Many algorithms for computationally hard problems (not only optimization) have a number of tunable parameters affecting their performance:

- EAs: population size, recombination operators, crossover and mutation rate, ...
- CPLEX (MIP solevr): e.g., different branching strategies
- Machine-learning pipelines: model type, its parameters

Why?

- There is no single optimal setting for all possible applications.
- For iterative algorithms, the optimal setting also depends on the number of iterations already performed.

Approaches:

- **Offline parameter tuning (automatic algorithm configuration):** a configuration is found for certain class of problem instances *before* the algorithm is applied to new ones.
- **Online parameter control:** a configuration is adapted *during* the optimization run.

Contributions of automatic algorithm configuration

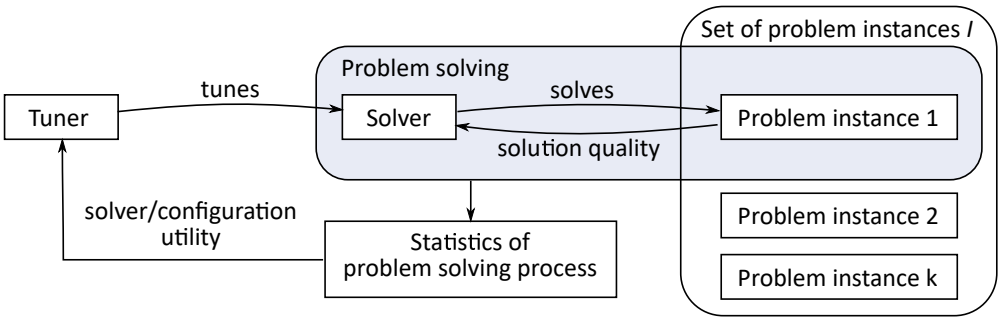
- **Development of complex algorithms**: Setting the parameters of a heuristic algorithm is a highly labour-intensive task, and indeed can consume a large fraction of overall development time. The use of automated algorithm configuration methods can lead to significant time savings and potentially achieve better results than manual, ad-hoc methods.
- **Empirical studies, evaluations, and comparisons of algorithms**: The majority of heuristic algorithm comparisons is qustionable, because the algorithms are used with their default settings. It is not clear whether the superiority of one algorithm is not caused just by more suitable configuration for a particular problem class. Automatic algorithm configuration methods can mitigate this problem of unfair comparisons and thus facilitate more meaningful comparative studies.
- **Practical use of algorithms**: Complex heuristic algorithms are often applied in contexts that were not envisioned by the algorithm designers. End users often have little or no knowledge about the impact of the algorithm parameter settings on its performance, and thus simply use default settings. Automatic algorithm configuration methods can be used to improve performance in a principled and convenient way.

Algorithm configuration problem

Find a good static configuration of solver before applying the solver on a problem at hand.

- I: the set of problem instances representing certain problem class (can be given by a distribution P_I over admissible instances, or by a problem generator).
- S: a solver (suitable for problem class I) with parameters $\theta = (p_1, \ldots, p_k) \in \Theta$ that affect its performance. $S(\theta)$ is the instance of the solver S configured with θ.
- Θ: a set of all possible configurations, i.e. all possible combinations of values of p_i.
- $C(\theta, i, t) = C(S(\theta), i, t)$: assigns a cost value to each configuration θ when running $S(\theta)$ on instance $i \in I$ for time t. It is often a random variable and we observe it realizations. $C \sim P_C(c|\theta, i, t)$.
- **The problem:** find configuration $\theta^* \in \Theta$ such that $S(\theta)$ yields the best utility u, i.e.

$$\theta^* = \arg\max_{\theta \in \Theta} u(\theta), \qquad \text{where } u(\theta) = f(\theta | I, P_I, P_C, t).$$



The process resembles ordinary ML process: fit algorithm (solver S) to the training data (instances I).

Characteristics of a configuration problem

Why is the tuning problem a complex optimization task?

- The cost function is often *stochastic*, either due to the stochasticity of the target problem class, or due the stochasticity of the solver itself.
- The measurements of the cost function are *expensive*: one has to execute the problem solving subtask many times, and such a process is time consuming.
- $\Longrightarrow$ The tuner usually has only a *limited budget* in terms of candidate solver configuration trials.
- The individual parameters p_i are of *different types* (nominal, ordinal, real-valued).
- The parameters are often *hierarchically structured*, i.e. some parameters are relevant only when other parameters are set to some particular value(s).
- Parameters are generally not independent!

The cost may represent

- the computational resources consumed by the given algorithm (runtime, memory, communication bandwith, ...),
- the approximation error,
- the improvement achieved over an instance-specific reference cost,
- the quality of the solution found.

The utility is a function of

- the (negative) expected cost,
- the (negative) median cost, ...

How to solve the configuration problem?

Manual methods:

- **Grid search:**
  - All parameters discretized, all combinations evaluated on all training instances, the best is selected.
  - Only limited number of configurations can be tried.
- **Manually executed local search:**
  - Researchers often tune parameters one by one, with a single small modification at a time.
  - More configurations can be tried, but many arbitrary choices are done.

Automated methods:

- **Classical black-box optimizers** when all parameters are real-valued: CMA-ES, BOBYQA, MADS, ...
- **Random search:** Methods from Design of Experiments, latin hypercubes, quasi-random numbers, ...
- **Evolutionary approches:** Meta-GA, REVAC, EVOCA
- **Racing methods:** F-Race, irace
- **Iterated local search:** ParamILS
- **Surrogate modeling:** SPOT, SMAC, Spearmint (Bayesian optimization)

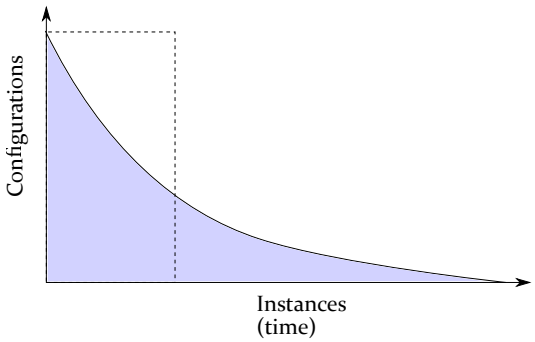
Random search vs Racing

Brute force approach

- estimate the quantities P_C and P_I by means of a sufficiently large number of runs of each candidate configuration on a sufficiently large set of training instances.
- The training set must be defined prior to the computation – how large?
- How many runs of each configuration on each instance should be performed?
- The same computational resources are allocated to each configuration – wasting time on poor configs!

Racing algorithm

- Provides a better allocation of computational resources among candidate configurations and allows for a clean solution to the problems with fixing the number of instances and the number of runs to be considered.
- Sequentially evaluates candidate configs and discards poor ones as soon as statistically sufficient evidence is gathered against them.
- Elimination of the inferior candidates speeds up the procedure and allows to evaluate the promising ones on more instances.
- As the evaluation proceeds, the race focuses more and more on the promising configurations.



Iterated racing

Generalization of F-race:

1. Sample new configurations according to a particular distribution.
2. Select the best configurations from the newly sampled ones by means of racing.
3. Update the sampling distribution (add bias towards the best configurations).

Similar to Estimation-of-distribution algorithms (EDAs).

See the attached slides from Jiří Kubalík.

ParamILS

Iterated Local Search in parameter space:

- Assumes discrete parameters (continuous params must be discretized).
- Perturbation in local search is done by exchanging value for a single parameter.
- "Iterated" is not the same as "restarted":
  - "Restarted" would mean starting the local search again from a new uniformly chosen starting point.
  - "Iterated" means applying a large(r) perturbation to the solution found in previous iteration to get the new starting point (i.e. kicking out the candidate solution out of a local optimum).

See the attached slides from Jiří Kubalík.

Algorithm configuration with surrogates

A **surrogate model** is a cheap (regression) model of the expensive function we want to optimize.

How can we use it for optimization?

- We can use the data points sampled so far to build a (much cheaper) regression model of that function.
- Then we find the optimum of the model:
  - either analytically, if the models allows, or
  - by applying numerical optimization to the model (evaluations of the model are cheap).
- The best point (configuration) according to the model is then evaluated by the true, expensive utility function.
- The evaluated point is added to the training set, the surrogate model is updated accordingly, and the process is repeated.

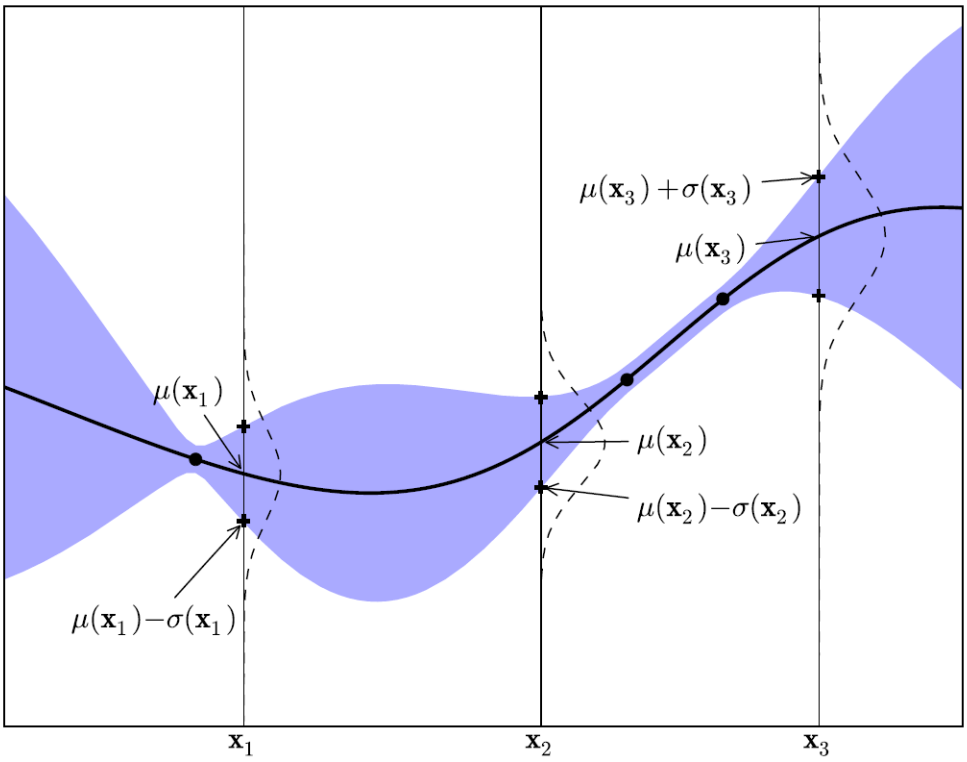
The above approach is too greedy:

- Almost no emphasis on exploration, i.e. on learning a good model.
- A good modeling method should allow to model not only the function itself, but also **our uncertainty about the model values!**

Gaussian Processes

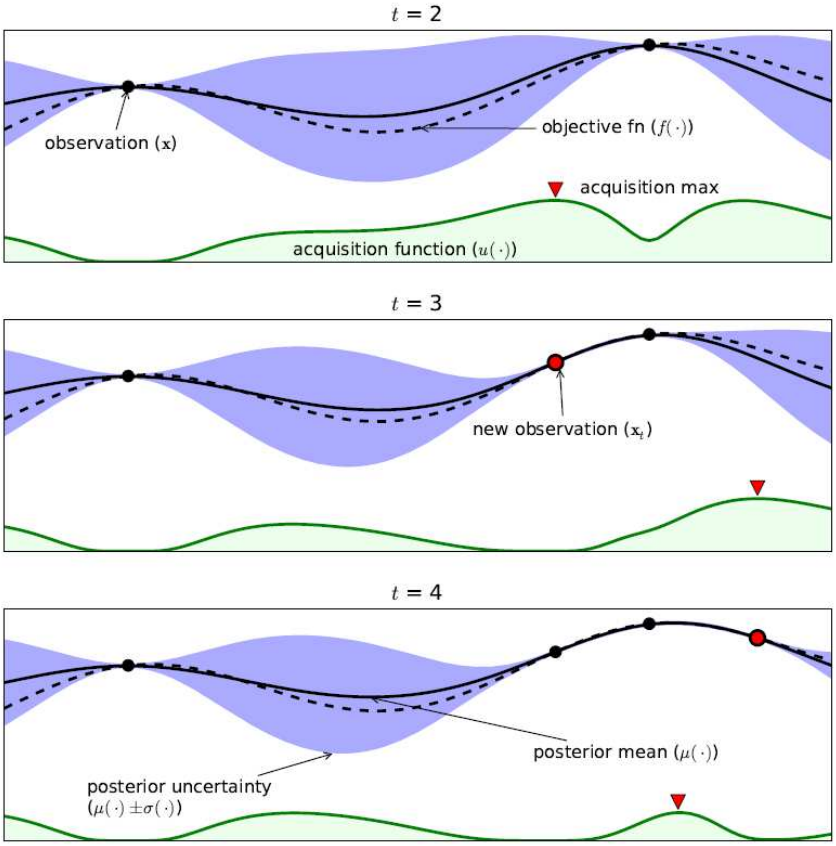
Gaussian Process is a distribution over a family of *functions*.

- For each point, it provides not only the estimate of the expected value at that point, but also an estimate of the prediction variance.



Optimization with GP as a surrogate

Several steps of function optimization with GP as a model (maximization):



GP: Acquisition function

To determine where to sample the next configuration for evaluation by the expensive cost function, the algorithm must optimize the (hopefully cheap) **acquisition function**:

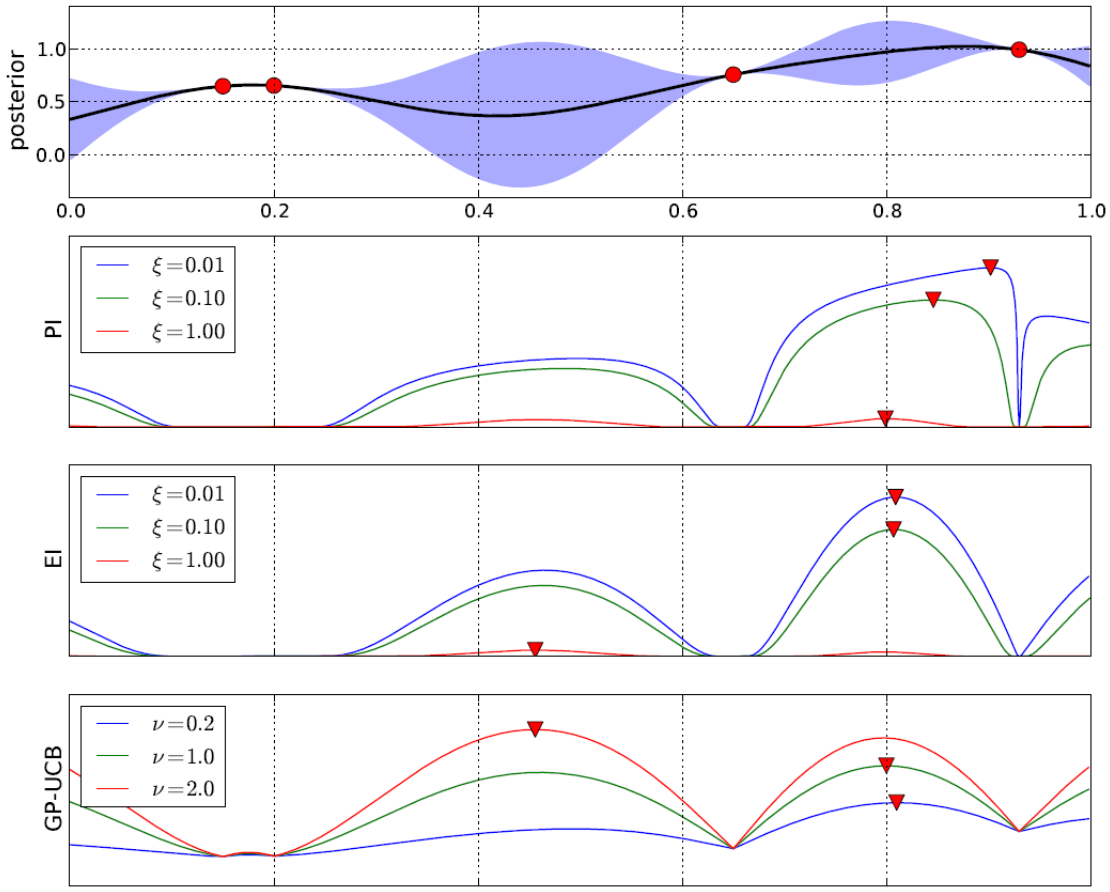
- Using another optimization solver (DIRECT, EA, CMA-ES, ...).
- The found "optimum" is just an approximation (which does not matter much since the model itself is only an approximation).

Types of acquisition functions:

- **Probability of improvement (PI)**: what is the probability that sampling at point θ will improve the cost function, given the current GP model?
- **Expected improvement (EI)**: what is the expected "size" of the improvement at point θ, given the current GP model?
- **Upper confidence bound (UCB)**: $UCB\theta = \mu(\theta) + \kappa\sigma(\theta)$

GP: Acquisition function (cont.)

The influence of various acquisition functions with different parameters:



Gaussian Processes: Summary

Optimization using GPs is based on sound principles, but:

- the method has an assupmption that the function must be Lipschitz-continuous;
- the complexity of GP model quickly increases with the number of evaluated data points.

Learning outcomes

After this lecture, a student shall be able to

- identify metaparameters (tunable parameters) in various optimization tasks and distinguish them from decision variables;
- explain the difference between *parameter tuning* and *parameter control*, and give examples of both;
- define the task of parameter tuning;
- explain the complex nature of the parameter tuning problem and describe characteristics that make it complex;
- list several contributions of parameter tuning;
- exemplify a few manual methods usable for parameter tuning, list their advantages and disadvantages;
- describe and explain racing techniques, F-race and iterated racing, and its advanatges/disadvantages;
- describe and explain ParamILS algorithm (how the local search is done, how the *iteration* of local search is done) + its advantages/disadvantages;
- explain the principle of using surrogate models in optimization and describe possible shortcommings;
- describe Gaussian process and explains its difference to the majority of regular regression models;
- explain the role of an *acquisition function* in Gaussian process-based optimization and list a few examples of acq. functions.