Parameter Tuning.
Automatic Algorithm Configuration

Petr Pošík

Czech Technical University in Prague
Faculty of Electrical Engineering
Department of Cybernetics

Using some slides from **Jiří Kubalík**, CIIRC CTU, with permission.

Configurable algorithms

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

- Local search: neighborhoods, perturbations, tabu length, annealing, . . .
- EAs: population size, mating scheme, recombination operators, crossover and mutation rates, local improvement stages, . . .
- Tree search (esp. for SAT): pre-processing, branching strategies, restarts, data structures,. . .
- Machine-learning pipelines: preprocessing method, model type, regularization, learning rate schedules, optimizers & their params, . . .
- Deep learning: #layers and their types, #units/layer, dropout settings, weight initialization and decay, pre-training, . . .

Which of the following statements about algorithm configuration is *mostly false*?

A There is no single optimal setting for all possible applications.

B For iterative algorithms, the optimal setting also depends on the number of iterations already performed.

C Metaparameters allow us to tune the algorithm to the problem class at hand.

D Practitioners heavily tune their algorithms before applying them to their problems.

Approaches to algorithm configuration

Online parameter control: the configuration is adapted *during* the optimization run.

- 1/5th rule in ES
- Adaptive penalty coefficients in constrained optimization
- Population sizing in Parameter-less GA
- . . .

Offline parameter tuning or **Automatic algorithm configuration, AAC**: a configuration is found for certain class of problem instances *before* the algorithm is applied to new ones.

- The topic of this lecture.

Contributions of automatic algorithm configuration

Development of complex algorithms:

- Algorithm configuration is a labour- and time-intensive task.
- AAC trades costly human time for cheap CPU time.
- AAC methods can lead to *significant time savings* and potentially achieve better results than manual, ad-hoc tuning methods.

Empirical studies, evaluations, and comparisons of algorithms:

- The majority of existing comparisons of heuristic algorithms are questionable: the algorithms are used with their default settings.
- Not clear whether the superiority of one algorithm is not caused just by a more suitable configuration for a particular problem class.
- AAC methods can provide *fair comparisons* and thus facilitate more meaningful comparative studies.

Practical use of algorithms:

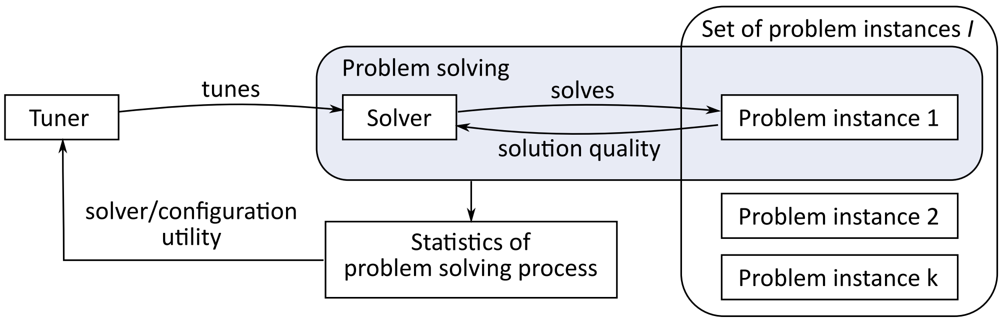
- Algorithms are often applied in contexts that were not envisioned by the algorithm designers.
- End users often use the default settings (no knowledge of the impact of the algorithm configuration on its performance).
- AAC methods can be used to *improve performance in a principled and convenient way*.
- AAC can ”modify” the algorithm for a different than original objective (speed, accuracy, memory, energy consumption, latency, ...)

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 its realizations. $c(\theta, i, t) \sim P_C(c|\theta, i, t)$.
- **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 to 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.
- 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 of a single run for certain configuration θ 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 of a configuration θ (cost aggregated across many runs) 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, ...
- **Systematic sampling:** 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
- **Model-based approaches:** SPOT, SMAC, Spearmint (Bayesian optimization)

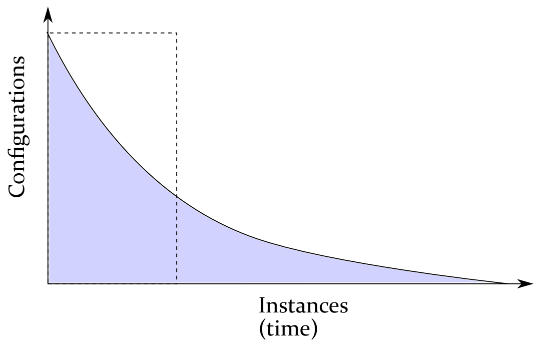
Brute force vs Racing

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.
- No need to fix the number of instances and/or the number of runs for each configuration.
- 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.



F-Race

F-Race [BSPV02] is a particular racing procedure:

- The process starts with a given *finite pool of candidate configurations* $\Theta_0 \subset \Theta$.
- Candidates are eliminated as soon as they become inferior to at least one other candidate. The decision is based on Friedman test.
- Applied to repetitive problems where many similar instances are available or appear over time.

Notation:

- k is the current step of the race process, $n = |\Theta_{k-1}|$ configurations are still in the race.
- $\boldsymbol{i} = (i_k)_{k=1}^K$ is a random sequence of training instances. Instance i_k is drawn from P_I independently for each k.
- $\boldsymbol{c}^k(\theta, \boldsymbol{i})$ is an array of k terms $c(\theta, i_l)$ for $l \in 1, \ldots, k$.
  - $c(\theta, i_l)$ is the cost of the best solution found by configuration θ on instance i_l).
  - For a given θ, the array $\boldsymbol{c}^k$ can be obtained from $\boldsymbol{c}^{k-1}$ by appending the cost $c(\theta, i_k)$ on the current instance i_k.
- A *block* is n-variate random variable $(c(\theta_1, i_l), c(\theta_2, i_l), \ldots, c(\theta_n, i_l))$ that corresponds to the results on instance i_l for each configuration still in the race at step k.

[BSPV02] Mauro Birattari, Thomas Stützle, Luis Paquete, and Klaus Varrentrapp. A racing algorithm for configuring metaheuristics. In *Proceedings of the 4th Annual Conference on Genetic and Evolutionary Computation*, GECCO'02, page 11–18, San Francisco, CA, USA, 2002. Morgan Kaufmann Publishers Inc.

F-Race: Algorithm

F-Race generates a sequence $\Theta_0 \supseteq \Theta_1 \supseteq \Theta_2 \supseteq \ldots$, and the step from Θ_{k-1} to Θ_k is done as follows:

1. At step k, sample a new instance $i_k \sim P_I$.
2. Take each candidate $\theta \in \Theta_{k-1}$ still in the race, execute it on i_k and append the observed cost $c(\theta, i_k)$ to already known costs on previous instances $\boldsymbol{c}^{k-1}(\theta, \boldsymbol{i})$.
3. Compare the results of all n configurations on all k instances using a statistical test.
   - Use non-parametric Friedman repeated measures analysis of variance by ranks.
   - The null hypothesis is that all configurations are equally good, i.e., all possible rankings of the candidates within each block are equally likely.
4. Are there significant differences? (Is the null hypothesis is rejected?)
   - Yes, there are differences: Perform pairwise comparisons between the best candidate and each other candidate using the t-test. All candidates significantly worse than the best one are discarded, $\Theta_k \subset \Theta_{k-1}$.
   - No, no significant differences: $\Theta_k = \Theta_{k-1}$.

Iterated racing

F-Race is not suitable for applications with large configuration spaces.

- Need to specify candidate configurations before the race.
- High number of configurations: F-Race takes a lot of time.
- Low number of configurations: low confidence in the result.
- Used mainly for configuration problems with a few parameters with small domains.

Iterated F-Race [BYBS10] is a generalization of F-race (a variant of Estimation-of-distribution algorithm):

1. Sample new configurations according to the current distribution P_C.
2. Select the best configurations from the newly sampled ones by means of F-Race.
3. Update the sampling distribution P_C (add bias towards the best configurations).
4. Repeat until budget not exhausted.

Remarks:

- The individual F-Races do not have to run until only a single configuration remains (can be stopped sooner).
- Individual F-Races may be relatively small (because more runs with different set of candidates are expected).

[BYBS10] Mauro Birattari, Zhi Yuan, Prasanna Balaprakash, and Thomas Stützle. *F-Race and Iterated F-Race: An Overview*, pages 311–336. Springer Berlin Heidelberg, Berlin, Heidelberg, 2010.

ParamILS

ParamILS [HHLBS09]: Iterated Local Search in parameter space

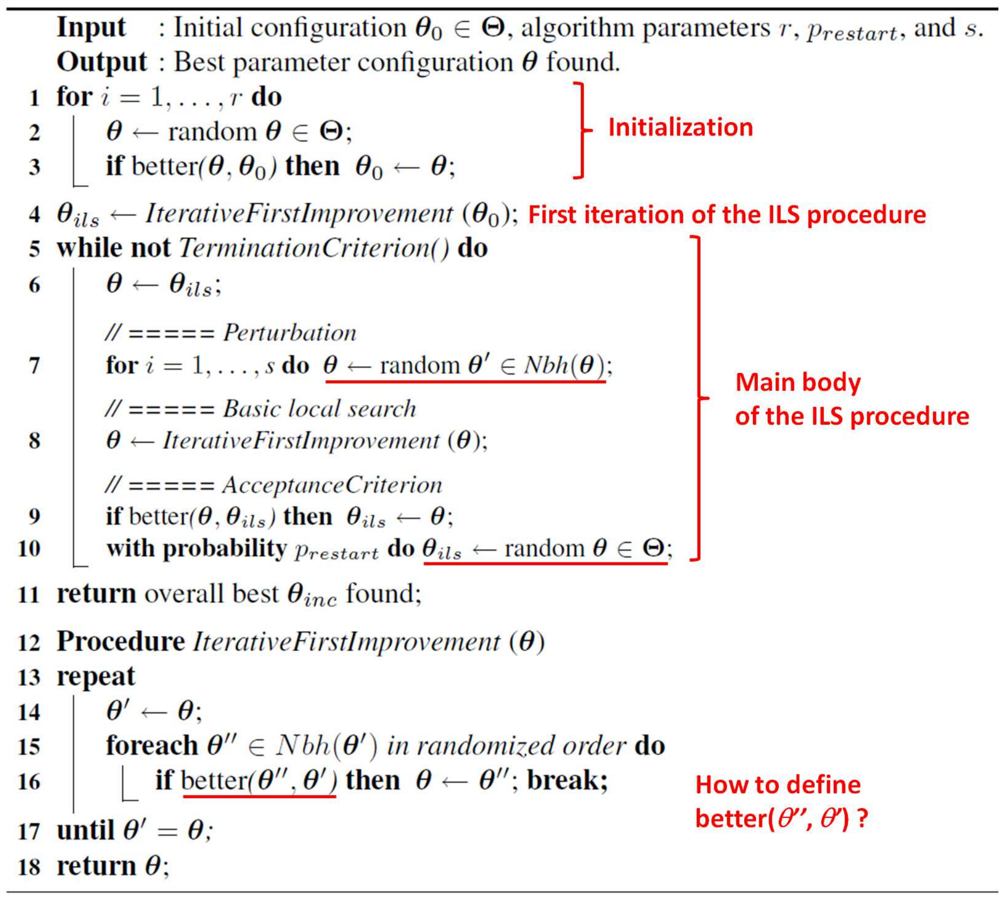
- Uses *iterated*, not *restarted*, local search:
  - Restarted: restart the local search from a new uniformly chosen starting point.
  - Iterated: apply a large(r) perturbation to the current solution to get a new starting point (i.e., kick the candidate solution out of a local optimum).
- ILS builds a chain of local optima by iterating the following steps:
  1. *Perturb a solution* to escape from local optimum.
  2. *Apply local search* to get to a (hopefully different) local optimum.
  3. *Decide whether to keep or reject* the new local optimum (acceptance criterion).

$ParamILS(\theta_0, r, p_{\text{restart}}, s)$

- Assumes *discrete parameters* (continuous params must be discretized).
- Initialization uses one given configuration θ_0 and r randomly chosen configurations.
- Uses *one-exchange neighborhood*, i.e., perturbation in local search is done by exchanging value for a single parameter.
- Employs iterative first-improvement LS.
- Uses s random moves for the "large" perturbation (kicking out from LO).
- Always accepts better or equally good configurations.
- Re-initializes the search at random with probability p_{restart}.

[HHLBS09] Frank Hutter, Holger H. Hoos, Kevin Leyton-Brown, and Thomas Stützle. Paramils: An automatic algorithm configuration framework. *J. Artif. Int. Res.*, 36(1):267–306, sep 2009.

ParamILS Algorithm



```
Input  : Initial configuration $\theta_0 \in \Theta$, algorithm parameters $r$, $p_{restart}$, and $s$.
Output : Best parameter configuration $\theta$ found.
for $i = 1, \ldots, r$ do
    $\theta \leftarrow$ random $\theta \in \Theta$;
    if better($\theta, \theta_0$) then $\theta_0 \leftarrow \theta$;
$\theta_{ils} \leftarrow$ IterativeFirstImprovement ($\theta_0$);
while not TerminationCriterion() do
    $\theta \leftarrow \theta_{ils}$;
    // ===== Perturbation
    for $i = 1, \ldots, s$ do $\theta \leftarrow$ random $\theta' \in Nbh(\theta)$;
    // ===== Basic local search
    $\theta \leftarrow$ IterativeFirstImprovement ($\theta$);
    // ===== AcceptanceCriterion
    if better($\theta, \theta_{ils}$) then $\theta_{ils} \leftarrow \theta$;
    with probability $p_{restart}$ do $\theta_{ils} \leftarrow$ random $\theta \in \Theta$;
return overall best $\theta_{inc}$ found;
Procedure IterativeFirstImprovement ($\theta$)
repeat
    $\theta' \leftarrow \theta$;
    foreach $\theta'' \in Nbh(\theta')$ in randomized order do
        if better($\theta'', \theta'$) then $\theta \leftarrow \theta''$; break;
until $\theta' = \theta$;
return $\theta$;
```

Initialization
First iteration of the ILS procedure
Main body of the ILS procedure
How to define better(θ'', θ') ?

Basic acceptance criterion

Basic variant of ParamILS, **BasicILS**, uses criterion $better_N(\theta_1, \theta_2)$:

- Configurations θ_1 and θ_2 are compared using their cost approximations $\widehat{c}_N(\theta_1)$ and $\widehat{c}_N(\theta_2)$.
- Cost approximation $\widehat{c}_N(\theta)$ is based on exactly N samples from the respective cost distribution $C(\theta, P_I)$. The same N instances are used for all configurations θ_i.
- It also updates the best-so-far solution, θ_{inc}.

```
Input       : Parameter configuration $\boldsymbol{\theta}_1$, parameter configuration $\boldsymbol{\theta}_2$
Output      : True if $\boldsymbol{\theta}_1$ does better than or equal to $\boldsymbol{\theta}_2$ on the first $N$ instances; false otherwise
Side Effect : Adds runs to the global caches of performed algorithm runs $\mathbf{R}_{\boldsymbol{\theta}_1}$ and $\mathbf{R}_{\boldsymbol{\theta}_2}$; potentially
              updates the incumbent $\boldsymbol{\theta}_{inc}$
1 $\hat{c}_N(\boldsymbol{\theta}_2) \leftarrow objective(\boldsymbol{\theta}_2, N)$
2 $\hat{c}_N(\boldsymbol{\theta}_1) \leftarrow objective(\boldsymbol{\theta}_1, N)$
3 return $\hat{c}_N(\boldsymbol{\theta}_1) \leq \hat{c}_N(\boldsymbol{\theta}_2)$
```

FocusedILS

BasicILS: How to choose the optimal number of training instances, N?

- Too small N leads to good training performance, but poor generalization to previously unseen test benchmarks.
- On the other hand, we cannot evaluate every configuration on a big training set (slow progress).

FocusedILS is a variant of ParamILS that *adaptively varies the number of training samples* used for individual configurations to **focus on promising configurations**.

- $N(\theta)$ denotes the number of runs available to estimate the cost statistic $c(\theta)$.

How to compare configurations θ_1 and θ_2 for which $N(\theta_1) \neq N(\theta_2)$?

- *What if we computed the empirical statistics based on the available number of runs for each configuration*?
- Can lead to systematic bias if, for example, the first instances are easier than the average ones.

Focused Acceptance Criterion

Domination: Configuration θ_1 dominates θ_2 when at least as many runs have been conducted on θ_1 as on θ_2, and the performance of $S(\theta_1)$ on the first $N(\theta_2)$ runs is at least as good as that of $S(\theta_2)$ on all of its runs.

$$\theta_1 \text{ dominates } \theta_2 \text{ if and only if } N(\theta_1) \geq N(\theta_2) \text{ and } \widehat{c}_{N(\theta_2)}(\theta_1) \leq \widehat{c}_{N(\theta_2)}(\theta_2).$$

FocusedILS uses procedure $better_{Foc}(\theta_1, \theta_2)$ which implements a domination-based comparison.

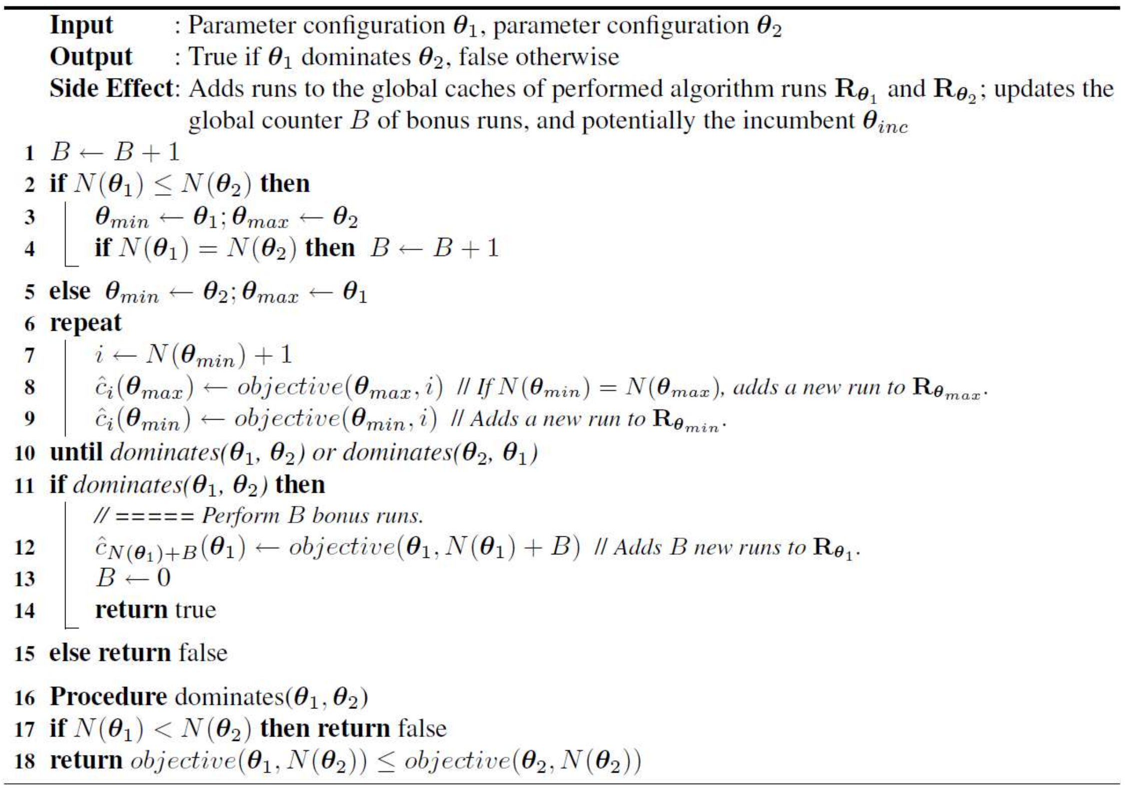
1. Acquire one additional run for the configuration θ_i with smaller $N(\theta_i)$, or one run for both configurations if $N(\theta_1) = N(\theta_2)$.
2. Continue performing runs in this way until one configuration dominates the other.
3. Returns true if θ_1 dominates θ_2; false otherwise.

It keeps track of the number, B, of configurations evaluated since the last improving step:

- Whenever $better_{Foc}(\theta_1, \theta_2)$ returns true, perform B extra runs for θ_1, reset B to 0.
- This ensures that many runs are performed with good configurations $\Longrightarrow$ the error made in every comparison of configurations θ_1 and θ_2 decreases in expectation.

Focused Acceptance Criterion (cont.)

Function $better_{\text{Foc}}(\theta_1, \theta_2)$



```
   Input      : Parameter configuration $\boldsymbol{\theta}_1$, parameter configuration $\boldsymbol{\theta}_2$
   Output     : True if $\boldsymbol{\theta}_1$ dominates $\boldsymbol{\theta}_2$, false otherwise
   Side Effect: Adds runs to the global caches of performed algorithm runs $\mathbf{R}_{\boldsymbol{\theta}_1}$ and $\mathbf{R}_{\boldsymbol{\theta}_2}$; updates the
                global counter $B$ of bonus runs, and potentially the incumbent $\boldsymbol{\theta}_{inc}$
1  $B \leftarrow B + 1$
2  if $N(\boldsymbol{\theta}_1) \leq N(\boldsymbol{\theta}_2)$ then
3      $\boldsymbol{\theta}_{min} \leftarrow \boldsymbol{\theta}_1; \boldsymbol{\theta}_{max} \leftarrow \boldsymbol{\theta}_2$
4      if $N(\boldsymbol{\theta}_1) = N(\boldsymbol{\theta}_2)$ then $B \leftarrow B + 1$
5  else $\boldsymbol{\theta}_{min} \leftarrow \boldsymbol{\theta}_2; \boldsymbol{\theta}_{max} \leftarrow \boldsymbol{\theta}_1$
6  repeat
7      $i \leftarrow N(\boldsymbol{\theta}_{min}) + 1$
8      $\hat{c}_i(\boldsymbol{\theta}_{max}) \leftarrow objective(\boldsymbol{\theta}_{max}, i)$ // If $N(\boldsymbol{\theta}_{min}) = N(\boldsymbol{\theta}_{max})$, adds a new run to $\mathbf{R}_{\boldsymbol{\theta}_{max}}$.
9      $\hat{c}_i(\boldsymbol{\theta}_{min}) \leftarrow objective(\boldsymbol{\theta}_{min}, i)$ // Adds a new run to $\mathbf{R}_{\boldsymbol{\theta}_{min}}$.
10 until dominates($\boldsymbol{\theta}_1$, $\boldsymbol{\theta}_2$) or dominates($\boldsymbol{\theta}_2$, $\boldsymbol{\theta}_1$)
11 if dominates($\boldsymbol{\theta}_1$, $\boldsymbol{\theta}_2$) then
       // ===== Perform B bonus runs.
12     $\hat{c}_{N(\boldsymbol{\theta}_1)+B}(\boldsymbol{\theta}_1) \leftarrow objective(\boldsymbol{\theta}_1, N(\boldsymbol{\theta}_1) + B)$ // Adds $B$ new runs to $\mathbf{R}_{\boldsymbol{\theta}_1}$.
13     $B \leftarrow 0$
14     return true
15 else return false
16 Procedure dominates($\boldsymbol{\theta}_1, \boldsymbol{\theta}_2$)
17 if $N(\boldsymbol{\theta}_1) < N(\boldsymbol{\theta}_2)$ then return false
18 return $objective(\boldsymbol{\theta}_1, N(\boldsymbol{\theta}_2)) \leq objective(\boldsymbol{\theta}_2, N(\boldsymbol{\theta}_2))$
```

Quiz

Assume the following situation:

- We configure an algorithm that either optimally solves a problem instance, or proves that the instance is unsolvable.
- Our goal is to minimize the algorithm runtime.
- Configuration θ_1 takes a total of 10 s to solve $N = 100$ instances (mean runtime is 0.1 s per instance).
- Now we are assessing configuration θ_2. It has been running for more than 10 s solving the first instance.

What is the right thing to do now?

A To make a fair comparison, we have to run $S(\theta_2)$ on all 100 instances too.

B We can stop $S(\theta_2)$ after it finishes solving the first instance.

C We can stop $S(\theta_2)$ immediately.

D We should not have run $S(\theta_2)$ at all; it was clear that it cannot be better than θ_1 even without running it.

Adaptive Capping

Capping is a group of methods used in ParamILS to prevent wasting resources by evaluating configurations which are much worse than other, previously seen configuration.

- Using a bound on configuration performance, it preemptively stops evaluations of those configurations that cannot improve current configuration.
- *Trajectory-preserving capping* provably preserves ParamILS search trajectory, yet can lead to large computational savings.
- *Aggressive capping* changes the behavior of ParamILS, but can provide even better performance.

ParamILS: Final Remarks

ParamILS is suitable for configuration problems with many parameters and huge configuration spaces.

Successful applications of ParamILS:

- **SPEAR**, a complete SAT solver
  - 26 parameters,
  - $8.34 \cdot 10^{17}$ possible configurations,
  - FocusedILS produced configurations that solved test problems about 100 times faster than previous state-of-the-art solvers.
- **CPLEX**, a prominent solver for mixed integer programming problems with carefully chosen default parameter settings
  - 76 parameters,
  - $1.9 \cdot 10^{47}$ possible configurations,
  - FocusedILS obtained substantial improvements in terms of both the time required to find optimal solutions (speedup factors from 1.98 to 52.3) and minimizing the optimality gap with factors from 1.26 to 8.65.

Model-based methods

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 or algorithm configuration?

- 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.

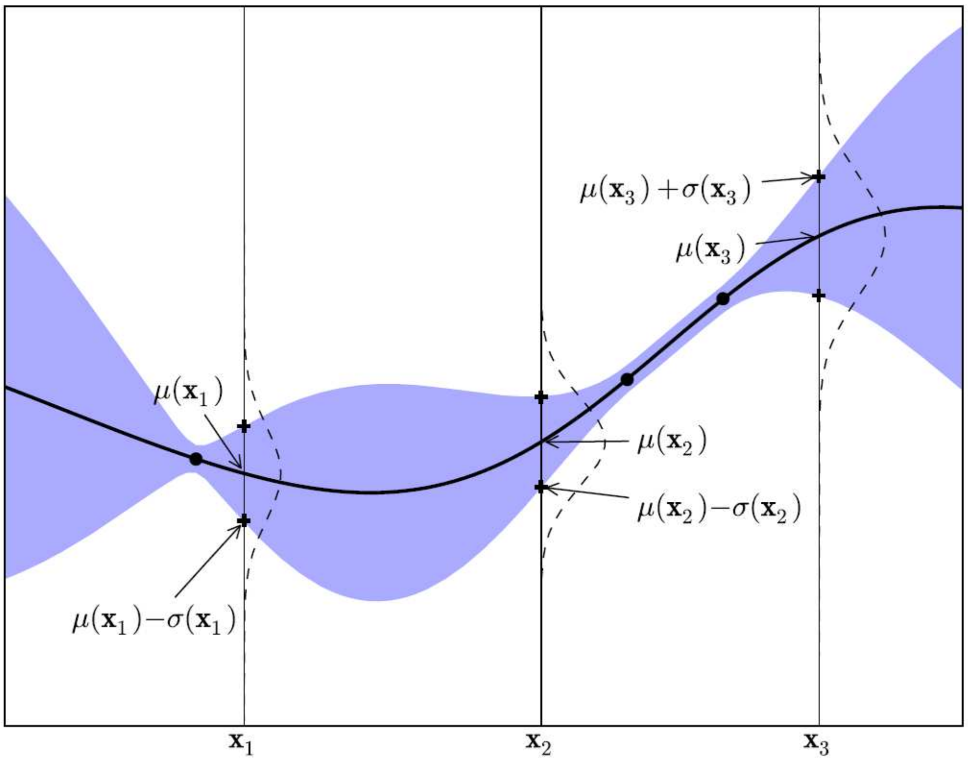
The above approach is too greedy:

- 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

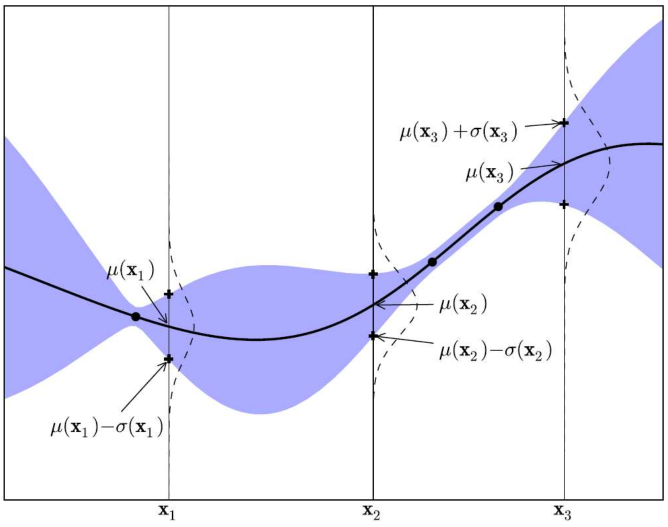
Gaussian Process is a distribution over a family of *functions*.

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



Quiz

Assume we want to *minimize* a function. We have the following model:



Which of the points is the best candidate for the next sample?

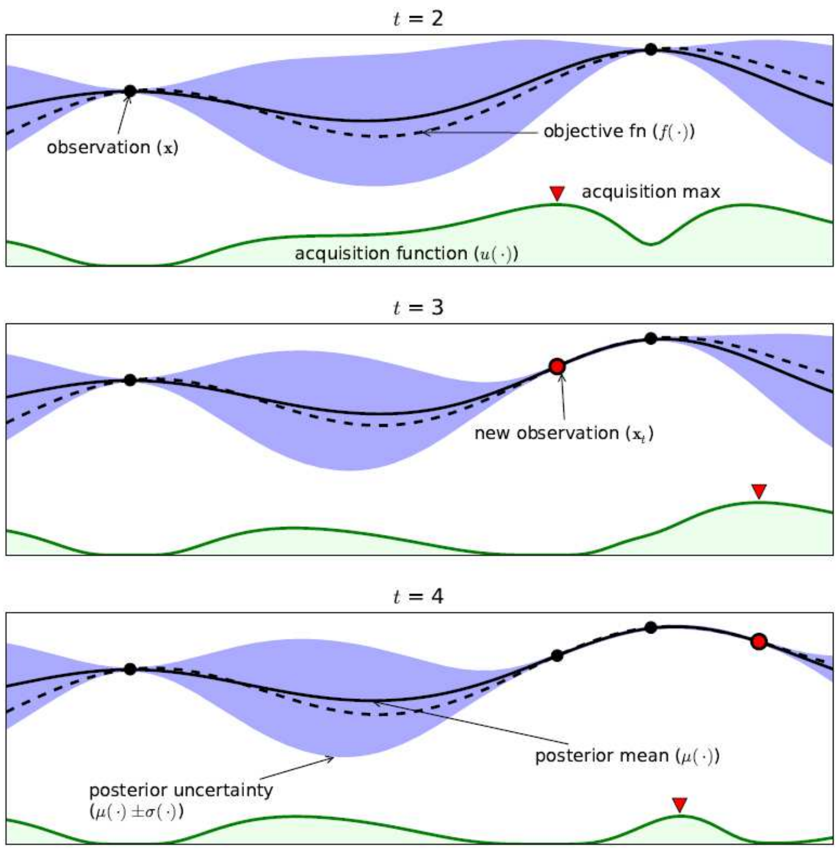
A x_1

B x_2

C x_3

Optimization with GP as a surrogate

Several steps of function *maximization* with GP as a model:



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**:

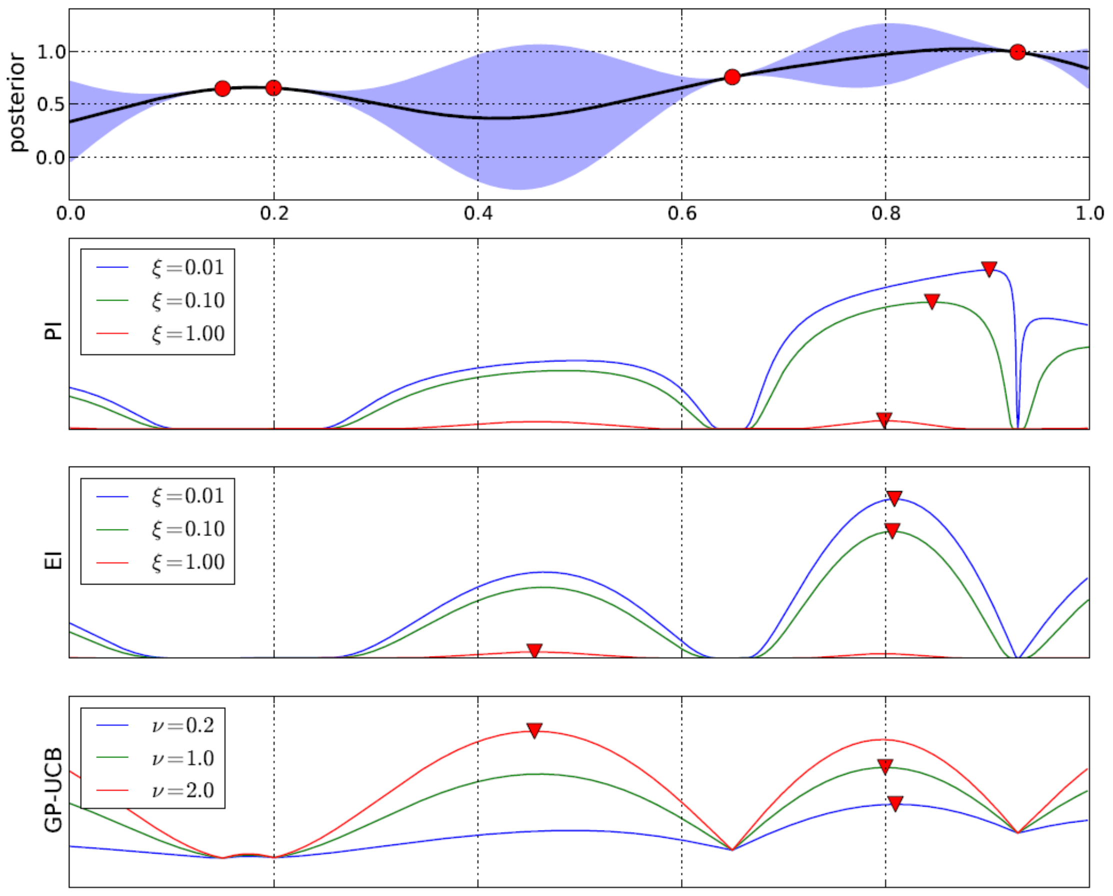
- Using another optimization solver (DIRECT, EA, CMA-ES, . . .).
- The found "optimum" is just an approximation (which does not matter much since the model itself is only an approximation).

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:



Bayesian Optimization

General approach:

- Fit a *probabilistic model* (e.g., Gaussian process, Random Forest, . . .) to the function samples collected so far.
- Use the model to guide the optimization (combine exploration and exploitation in a reasonable way).

Advantages:

- Based on sound principles.
- Efficient in the number of function evaluations.
- Works when objective isn non-convex, noisy, has known derivatives, etc.

Disadvatages:

- The complexity of the model (e.g., in case of GP) quickly increases with the number of evaluated data points.

SMAC: Basic info

Sequential Model-based Algorithm Configuration, SMAC [HHLB11]:

- The search is guided by a *predictive model of algorithm performance*.
- Uses aggressive racing & adaptive capping.

A single iteration of SMAC:

- Construct a model to predict performance.
- Use the model to select promising configurations.
- Compare each selected configuration against the best known (similar to FocusedILS).

[HHLB11] Frank Hutter, Holger H. Hoos, and Kevin Leyton-Brown. Sequential model-based optimization for general algorithm configuration. In Carlos A. Coello Coello, editor, *Learning and Intelligent Optimization*, pages 507–523, Berlin, Heidelberg, 2011. Springer Berlin Heidelberg.

Automated Machine Learning

Automated Machine Learning

ML is successful in many applications.

- But it still requires human ML experts to
  - preprocess the data,
  - select/create features,
  - select model family,
  - optimize hyperparameters,
  - construct ensembles,
  - . . .
- *AutoML takes the human expert out of the loop.*

AutoML:

- Introduced by Auto-WEKA [THHLB13]:
  - Expose the choices in ML framework (algorithms, hyperparameters, preprocessors, . . .)
  - Optimize crossvalidation performance using Bayesian optimization.
- $\Longrightarrow$ A true push-button solution for machine learning!

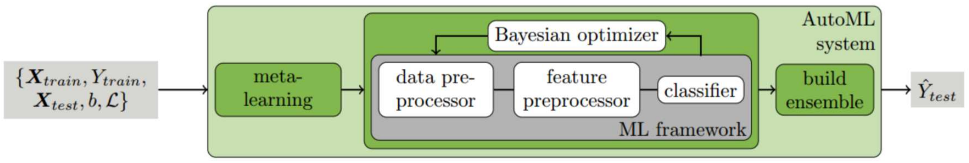
[THHLB13] Chris Thornton, Frank Hutter, Holger H. Hoos, and Kevin Leyton-Brown. Auto-weka: Combined selection and hyperparameter optimization of classification algorithms. In *Proceedings of the 19th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '13, page 847–855, New York, NY, USA, 2013. Association for Computing Machinery.

Auto-sklearn

Auto-sklearn [FKE+15]:

- Application of AutoWEKA principles to `scikit-learn` Python package.
- Adding new components to AutoWEKA approach:
  - Meta-learning to warmstart Bayesian optimization.
  - Automated post-hoc ensemble construction combining the models evaluated by Bayesian optimization.

  Source code: `https://github.com/automl/auto-sklearn`



Trivial usage:

```
>>> import autosklearn.classificaton as cls
>>> automl = cls.AutoSklearnClassifier()
>>> automl.fit(X_train, y_train)
>>> y_hat = automl.predict(X_test)
```

Similar solutions for other ML packages exist (Auto-Keras, Auto-PyTorch, . . .)

[FKE+15] Matthias Feurer, Aaron Klein, Katharina Eggensperger, Jost Tobias Springenberg, Manuel Blum, and Frank Hutter. Efficient and robust automated machine learning. In *Proceedings of the 28th International Conference on Neural Information Processing Systems - Volume 2*, NIPS'15, page 2755–2763, Cambridge, MA, USA, 2015. MIT Press.

Summary

Conclusions

Automatic algorithm configuration is very useful:

- Improves results, increases productivity.
- Enables automated machine learning.

To use an AAC method, you need:

- an algorithm with exposed parameters,
- a training set of instances (or a generator of instances from certain distribution),
- a performance metric you care about.

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.