Notes on Sequential Model-based Optimization

Jingyuan Hu

Reference: *Hutter, Frank, Holger H. Hoos, and Kevin Leyton-Brown. "Sequential model-based optimization for general algorithm configuration (extended version)." Technical Report TR-201010, University of British Columbia, Computer Science, Tech. Rep. (2010).*

Contents

1 Introduction

- Model-free algorithm configuration methods
  - Racing: F-RACE
  - Iterated local search: PARAMILS
  - Genetic: GGA
- Model-based approaches: Sequential model-based optimization (SMBO) iterates between fitting models and using them to make choices about which configurations to investigate
- Main Contribution: remove limitations on
  - only supports numerical parameters
  - only optimizes target algorithm performance for single instances
- Two novel SMBO instantiations capable of general algorithm configuration
  - simple model-free Random Online Adaptive Racing (ROAR)
  - Sequential Model-based Algorithm Configuration (SMAC)

2 Random Online Aggressive Racing(ROAR)

- random: set of candidates is selected at random.
- online: each candidate is accepted or rejected online.
- aggressive: this online decision is made aggressively, before enough data has been gathered to support a statistically significant conclusion.
- race: runs each candidate configuration only as long as necessary to establish whether it is competitive.

ROAR is completely specified by the four components:

1. *Initialize* performs a single run with the target algorithms default parameter configuration (or a random configuration if no default is available) on an instance selected uniformly at random.
2. *FitModel* procedure simply returns a constant model which is never used since ROAR is model-free.
3. *SelectConfigurations* returns a single configuration sampled uniformly at random from the parameter space.
4. *Intensify*
    - Takes as input a list of promising configurations, $\vec{\Theta}_{new}$, and compares them in turn to the current incumbent configuration until a time budget for this comparison stage is reached.
    - In each comparison of a new configuration, θ_{new}, to the incumbent, θ_{inc}, we first perform an additional run for the incumbent, using a randomly selected <instance, seed> combination.
    - Then, we iteratively perform runs with θ_{new} (using a doubling scheme) until either θ_{new}s empirical performance is worse than that of θ_{inc} (in which case we reject θ_{new}) or we performed as many runs for θ_{new} as for θ_{inc} and it is still at least as good as θ_{inc} (in which case we change the incumbent to θ_{new}).
    - The <instance, seed> combinations for θ_{new} are sampled uniformly at random from those on which the incumbent has already run.
    - Every comparison is based on a different randomly selected subset of instances and seeds.

Procedure 2: Intensify($\vec{\boldsymbol{\Theta}}_{new}$, $\boldsymbol{\theta}_{inc}$, $\mathcal{M}$, $\mathbf{R}$, $t_{intensify}$, Π, $\hat{c}$)
$\hat{c}(\theta, \Pi')$ denotes the empirical cost of θ on the subset of instances $\Pi' \subseteq \Pi$, based on the runs in $\mathbf{R}$; *maxR* is a parameter, set to 2 000 in all our experiments

```
Input  : Sequence of parameter settings to evaluate, $\vec{\Theta}_{new}$; incumbent parameter setting,
         $\theta_{inc}$; model, $\mathcal{M}$; sequence of target algorithm runs, $\mathbf{R}$; time bound, $t_{intensify}$;
         instance set, $\Pi$; cost metric, $\hat{c}$
Output : Updated sequence of target algorithm runs, $\mathbf{R}$; incumbent parameter setting, $\theta_{inc}$
1  for $i := 1, \ldots, length(\vec{\Theta}_{new})$ do
2      $\theta_{new} \leftarrow \vec{\Theta}_{new}[i]$;
3      if $\mathbf{R}$ contains less than $maxR$ runs with configuration $\theta_{inc}$ then
4          $\Pi' \leftarrow \{\pi' \in \Pi \mid \mathbf{R}$ contains less than or equal number of runs using $\theta_{inc}$ and $\pi'$
           than using $\theta_{inc}$ and any other $\pi'' \in \Pi\}$ ;
5          $\pi \leftarrow$ instance sampled uniformly at random from $\Pi'$;
6          $s \leftarrow$ seed, drawn uniformly at random;
7          $\mathbf{R} \leftarrow$ ExecuteRun($\mathbf{R}$, $\theta_{inc}$, $\pi$, $s$);
8      $N \leftarrow 1$;
9      while true do
10         $S_{missing} \leftarrow \langle$instance, seed$\rangle$ pairs for which $\theta_{inc}$ was run before, but not $\theta_{new}$;
11         $S_{torun} \leftarrow$random subset of $S_{missing}$ of size $\min(N, |S_{missing}|)$;
12         foreach $(\pi, s) \in S_{torun}$ do $\mathbf{R} \leftarrow$ ExecuteRun($\mathbf{R}$, $\theta_{new}$, $\pi$, $s$);
13         $S_{missing} \leftarrow S_{missing} \setminus S_{torun}$;
14         $\Pi_{common} \leftarrow$ instances for which we previously ran both $\theta_{inc}$ and $\theta_{new}$;
15         if $\hat{c}(\theta_{new}, \Pi_{common}) > \hat{c}(\theta_{inc}, \Pi_{common})$ then break;
16         else if $S_{missing} = \emptyset$ then $\theta_{inc} \leftarrow \theta_{new}$; break;
17         else $N \leftarrow 2 \cdot N$;
18     if time spent in this call to this procedure exceeds $t_{intensify}$ and $i \geq 2$ then break;
19 return $[\mathbf{R}, \theta_{inc}]$;
```

3 Sequential Model-based Algorithm Configuration (SMAC)

SMAC can be understood as an extension of ROAR that selects configurations based on a model rather than uniformly at random. It instantiates Initialize and Intensify in the same way as ROAR.

For *Select Configurations*:

- New model classes that supports
  - Categorical parameters:
    - Weighted Hamming Distance Kernel Function for GP Models

$$K_{mixed}(\theta_i, \theta_j) = exp\big[\sum_{l \in \mathcal{P}_{cont}} (-\lambda_l \cdot (\theta_{i,l} - \theta_{j,l})^2) + \sum_{l \in \mathcal{P}_{cat}} (-\lambda_l \cdot [1 - \delta(\theta_{i,l}, \theta_{j,l})])\big]$$

    - Random Forests (target algorithm performance values at their leaves): construct a random forest as a set of B regression trees, each of which is built on n data points randomly sampled with repetitions from the entire training data set. We compute the random forests predictive mean μ_θ and variance σ_θ for a new configuration θ as the empirical mean and variance of its individual trees predictions for θ.
    - Transformations of the Cost Metric.
  - Multiple instances: Instance Features, Predicting Performance Across Instances
- Uses the model to select a list of promising parameter configurations:
  - Use models predictive distribution for a new configurations to compute it's $EI(\theta)$ over the incumbent

$$EI(\theta) := E[I_{exp}(\theta)] = f_{min}\Phi(v) - e^{\frac{1}{2}\sigma_\theta^2 + \mu_\theta} \cdot \Phi(v - \sigma_\theta)$$

    where $v := \frac{ln(f_{min} - \mu_\theta)}{\sigma_\theta}$
  - Identify configurations with large $EI(\theta)$:
    - Previous methods: simply applied random sampling for this task (in particular, they evaluated EI for 10 000 random samples)
    - The author instead perform a simple **multi-start local search** and consider all resulting configurations with locally maximal EI
- Detail of the multi-start local search
  - Compute EI for all configuations used in previous target algorithm runs
  - Pick the **ten** configurations with maximal EI, initialize a local search at each of them
  - To handle mixed categorical/numerical parameter spaces, they use a randomized one-exchange neighbourhood, including
    - set of all configurations that differ in the value of exactly one **discrete** parameter
    - four random neighbours for each **numerical** parameter: normalize the range of each numerical parameter to $[0, 1]$ and then sample four neighbouring values for numerical parameters with current value v from a univariate Gaussian distribution with mean v and standard deviation 0.2
  - Evaluate EI for all neighbours at once (batch model predictions much cheaper)
  - Stop each local search once none of the neighbours has larger EI

- Also compute EI for an additional 10,000 randomly-sampled configurations, and sort all 10,010 configurations in descending order of EI. (Note: For the ten initial one if none configuation outperforms the initial one, then it remains to be the initial one)
- Alternate between configurations pairs from the list (10,010) and additional configurations pairs sampled uniformly at random to provide unbiased training data for future models (e.g. Random Forest) and perform intensification on it.

Note: For details of intensification, please refer to Section 2