BOAT: Building Auto-Tuners with Structured Bayesian Optimization

Valentin Dalibard, Michael Schaarschmidt, Eiko Yoneki

Presented by: Andreas Pletschko (ap2535)

12.11.2025

R244 – Large-Scale Data Processing and Optimization

The Problem of Tuning Modern Systems

The Problem – Configuring Modern Systems

Figure 1: Visualization of a modern system's configuration space

Figure 1: Visualization of a modern system's configuration space

Existing Approaches

- **Getting rid of configurations:**
  - Generic System should serve a variety of workloads, which are very different in nature
  - Allow users to achieve a maximum of performance, if necessary
- **Manual (”white-box”) tuning:**
  - Require a lot of experience and deep knowledge of the underlying system
- **Black-Box tuner** (such as OpenTuner [1], traditional Bayesian Optimization, ...):
  - Require too many evaluations, which are expensive
  - Fail in large configuration spaces

Idea

Try to find a middle ground between ”knowing the system deeply” and ”not knowing anything at all”.

Gaussian Processes and Bayesian Optimization

Primer in Gaussian Processes

Gaussian Processes are just generalization of normal distributions over functions $f : \mathcal{S} \to \mathbb{R}$:

- Mean function: $\mu : \mathcal{S} \to \mathbb{R}$, such that $\mu(x)$ is the mean of $f(x)$
- Covariance function: $k : \mathcal{S} \times \mathcal{S} \to \mathbb{R}$, such that $k(x_1, x_2)$ is the covariance of $f(x_1)$ and $f(x_2)$

$\to$ Update belief over the functions, given some some observation $(x, f(x))$ (using Bayes' rule)

Gaussian Processes – Example

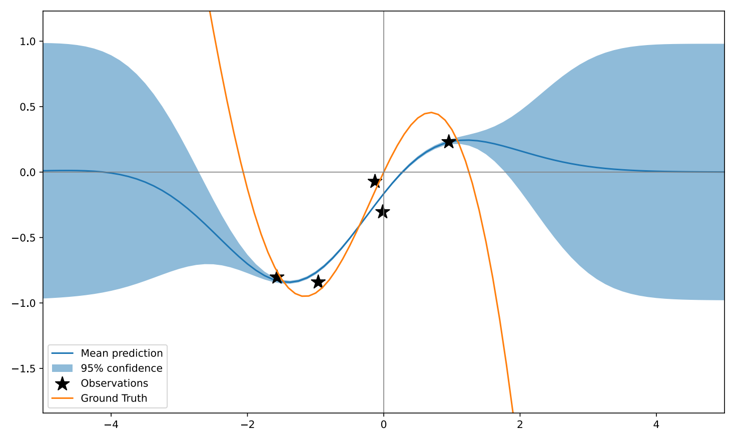


Figure 2: Example of function and it’s Gaussian Process approximation.

Bayesian Optimization (BO)

Idea: Just model the objective function using a GP:

- Use the GP model to choose the most promising configuration to evaluate next
- Update the GP using the newest evaluation

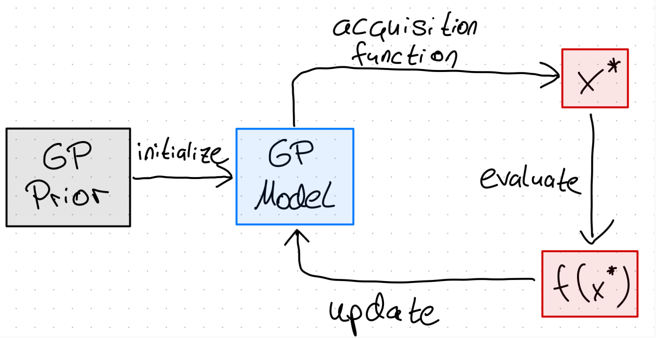


Figure 3: Flow chart of Bayesian optimization.

Bayesian Optimization in Practise

1. BO converges much quicker than other auto-tuners, such as OpenTuner [1]
2. BO fails to converge in configuration spaces with dimensions > 10 [3]

Hypothesis

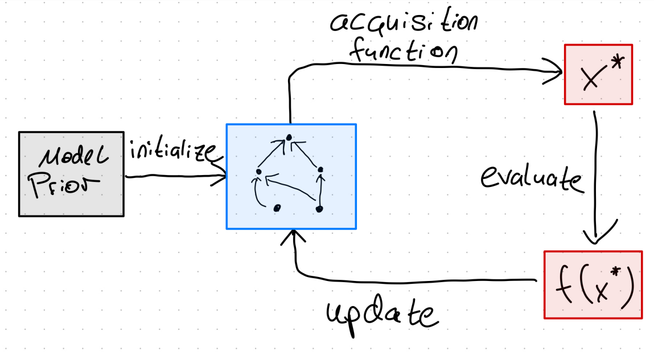
By making BO aware of the structure and interdependencies of the configuration space, one can

- drastically reduce the number of iterations
- find better configurations

Solution: Structured Bayesian Optimization

Structured Bayesian Optimization

Idea: Replace the GP model by a user-defined structured probabilistic model:



This allows to **induce knowledge** of the system's structure and dynamics into the optimization.

Example: Structured Probabilistic Model for a Web Server

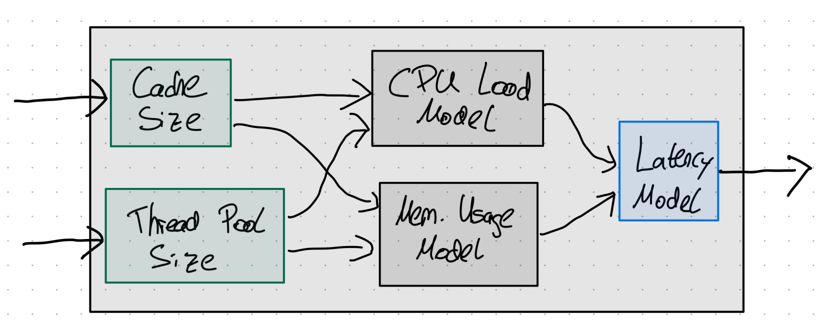


Figure 4: Example for a structured probabilistic model for a web server's latency

The BOAT Framework

The **BOAT** (**B**esp**O**ke **A**uto-**T**uner) framework for SBO-based auto tuning, by allowing for:

- Defining a configuration space
- Defining a objective function
- Defining structured probabilistic models

$\rightarrow$ runs optimization, given the optimizer definitions

The BOAT Framework

The **BOAT** (**B**esp**O**ke **A**uto-**T**uner) framework for SBO-based auto tuning, by allowing for:

- Defining a configuration space
- Defining a objective function
- **Defining structured probabilistic models**

$\rightarrow$ runs optimization, given the optimizer definitions

Defining Probabilistic Models in BOAT

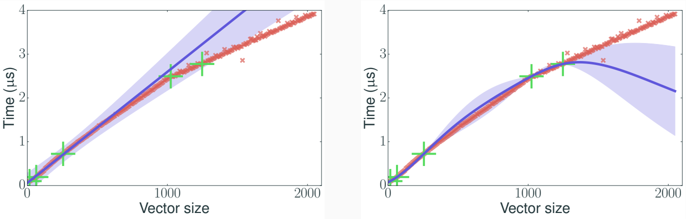
A probabilistic model in BOAT consists of **independent components** with the following properties:

- predict a single **observable value**
- **semi-parametric**

Those components are then implicitly combined in a **Directed Acyclic Graph**

Parametric vs Non-Parametric Models

Problem: Parametric models usually underfit, non-parametric models usually overfit to the data



(a) Parametric (Linear regression) (b) Non-parametric (Gaussian process)

Figure 5: Parametric vs Non-parametric models for vector insertion time, taken from [2]

Semi-Parametric Model

Solution: Semi-parametric models:

- Fit a **parametric model** to the **data**
- Fit a **non-parametric model** to the **residual** of the parametric model

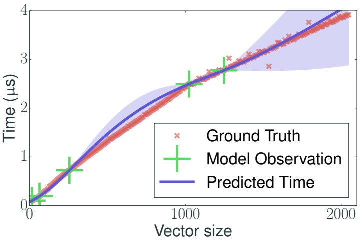


Figure 6: Semi-parametric model for vector insertion time, taken from [2]

Implementing Probabilistic Models in BOAT – 1/2

Declare a **model component** by extending the *SemiParametricModel* class:

```
struct LatencyModel : public SemiParametricModel<LatencyModel> {
    // Priors for the parametric part (the weights)
    double cpu_weight;
    double mem_weight;

    LatencyModel() {
        // We guess latency is more sensitive to CPU than Memory
        cpu_weight = std::uniform_real_distribution<>(100.0, 300.0)(generator);
        mem_weight = std::uniform_real_distribution<>(20.0, 80.0)(generator);
        // ... Omitted: also sample the GP parameters ...
    }

    // Parametric part (our "expert knowledge")
    double parameter(double cpu_load, double memory_usage) const {
        double base_network_latency = 10.0;
        return base_network_latency + (cpu_load * cpu_weight) + (memory_usage * mem_weight);
    }
};
```

Figure 7: Example of a semi-parametric model component in BOAT.

Implementing Probabilistic Models in BOAT – 2/2

Declare a **structured probabilistic model** by extending the *`DAGModel`* class:

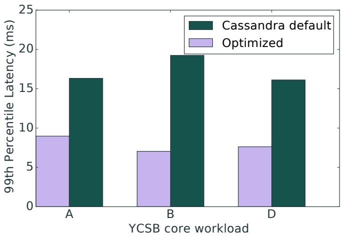
```
struct WebServerModel : public DAGModel<WebServerModel> {
    // Define ProbEngines for each node that has parents
    ProbEngine<CPULoadModel> cpu_load_model_;
    ProbEngine<MemoryUsageModel> memory_usage_model_;
    ProbEngine<LatencyModel> latency_model_;

    void model(int cache_size_mb, int thread_pool_count) {
        // 1. Calculate internal state: CPULoad
        double cpu_load = output("cpu_load", cpu_load_model_, cache_size_mb, thread_pool_count);
        // 2. Calculate internal state: MemoryUsage
        double memory_usage = output("memory_usage", memory_usage_model_, cache_size_mb,
                                     thread_pool_count);
        // 3. Calculate final metric: Latency
        double latency = output("latency", latency_model_, cpu_load, memory_usage);
    }
```

Figure 8: Example of a structured probabilistic model in BOAT.

Evaluation Tasks

1. JVM Garbage Collection
   - **Task:** Optimize the JVM garbage collector of a NoSQL distributed database (Cassandra)
   - **Configuration space:** 3 flags for the garbage collection algorithm
   - **Objective:** Minimize the 99th percentile latency
2. Distributed Neural Network Training
   - **Task:** Optimize the training process of a neural network, given an architecture, batch size and a cluster setup
   - **Configuration space:** 2 boolean and 1-2 integer parameters per machine
   - **Objective:** Minimize training duration of 20 SGD iterations

Garbe Collection – Results



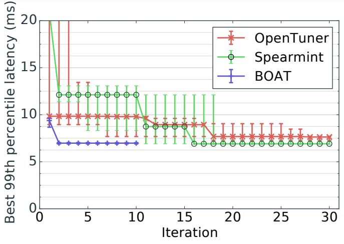


Figure 9: Left: Performance of BOAT-optimized configuration vs default configuration. Right: Performance of BOAT-optimization vs other auto-tuners. [2]

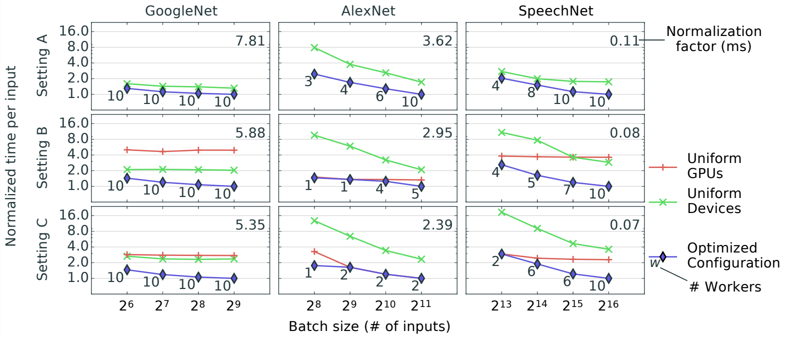


Figure 10: Comparison of BOAT-optimized configuration vs two default strategies. [2]

Distributed Neural Networks – Results 2/2

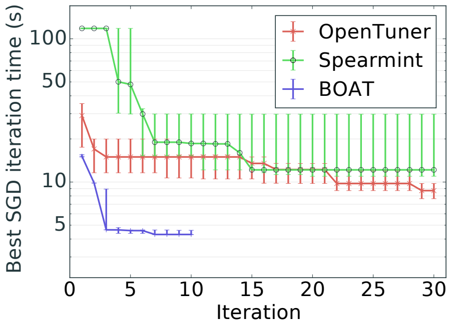


Figure 11: Performance of BOAT-optimization vs other auto-tuners. [2]

Conclusion

Opinions

Strengths:

- **Novel ”grey-box” model:** Smartly bootstraps expert knowledge of system structure
- **Superior Performance:** Finds better configurations up to 3x faster than SOTA tuners
- **Simple, Powerful Abstraction:** Hides complexities Bayesian Optimization from developer

Opinions

Strengths:

- **Novel ”grey-box” model:** Smartly bootstraps expert knowledge of system structure
- **Superior Performance:** Finds better configurations up to 3x faster than SOTA tuners
- **Simple, Powerful Abstraction:** Hides complexities Bayesian Optimization from developer

Limitations:

- **Requires Domain Knowledge:** Still needs an expert to manually and accurately define the system’s structure
- **Robustness unclear:** How does it perform if the expert’s structural ”beliefs” are wrong?
- **Tuner Fine-Tuning:** The model’s structure and hypertuning requires excessive fine-tuning
- **C++ Only API:** Restricts adoption in other ecosystems

TLDR;

The paper …

- introduced **Structured Bayesian Optimization** by replacing the GP model by a **user-defined structured model**
- presented **BOAT** as a framework for **implementing structured models** and running SBO
- demonstrated the **performance improvement** over default configurations & existing auto-tuners

Questions?

References i

J. Ansel, S. Kamil, K. Veeramachaneni, J. Ragan-Kelley, J. Bosboom, U.-M. O'Reilly, and S. Amarasinghe.
Opentuner: an extensible framework for program autotuning.
In *Proceedings of the 23rd International Conference on Parallel Architectures and Compilation*, PACT '14, page 303–316, New York, NY, USA, 2014. Association for Computing Machinery.

V. Dalibard, M. Schaarschmidt, and E. Yoneki.
Boat: Building auto-tuners with structured bayesian optimization.
In *Proceedings of the 26th International Conference on World Wide Web*, pages 479–488, 2017.

B. Shahriari, K. Swersky, Z. Wang, R. P. Adams, and N. de Freitas.
Taking the human out of the loop: A review of bayesian optimization.
Proceedings of the IEEE, 104(1):148–175, 2016.