BOAT: Building Auto-Tuners with Structured Bayesian Optimization

Paper by Valentin Dalibard, Michael Schaarschmidt, Eiko Yoneki

Presentation by Matthew Guest

Background

- Publication: WWW '17: Proceedings of the 26th International Conference on World Wide Web.
- April 2017
- Developed as part of Valentin Dalibard’s PhD:
  - https://www.cl.cam.ac.uk/techreports/UCAM-CL-TR-900.pdf
  - Very approachable intro to Gaussian Processes & Probabilistic Programming.

Problem

- Modern computer systems often have configuration parameters.
- Auto-tuning is the process of optimising the setting of these parameters to improve performance through automated search.
- Evaluating the success of a configuration often expensive.
- Space of configurations is often too large for tractable exploration.
- Common approaches to dealing with high dimensional search spaces require too many samples.
- Bayesian optimisation techniques effective in this area:
  - Efficiently utilise a small number of iterations
  - Suffer from the curse of dimensionality.

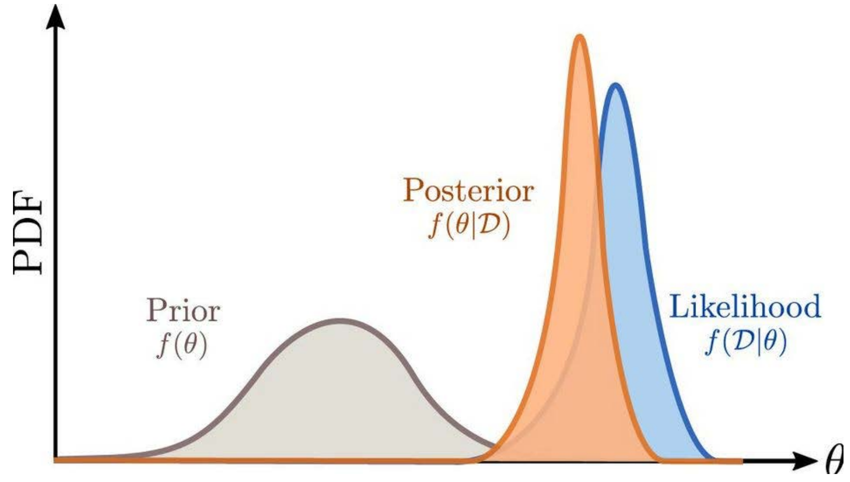
BOAT Overview

- This paper proposes Structured Bayesian Optimisation as a solution.
- SBO benefits from the efficient utilisation of samples from BO
- Reduces search space by leveraging structural information about the underlying system – bespoke approach.
- BOAT is evaluated against other optimisation approaches and demonstrates significant improvements in:
  - Effectiveness
  - Time to convergence

Intro

- Configuration Space: Compiler Flags, Workload Allocations.
- Models implemented using BOAT library probabilistic C++
- Runtime metrics used for inference:
    - Performance metric
    - Individual parameterisations: Communication time, Device performance

$$P(\theta|\mathcal{D}) = \frac{P(D|\theta)P(\theta)}{P(\mathcal{D})}$$



Motivation

- Expense of Evaluation – Minutes:
  - Evolution / Reinforcement approaches require thousands of iterations
  - For Garbage Collected case: SBO 2 iterations vs BO 16 iterations
- Curse of Dimensionality:
  - 10 machines, 3 parameters per machine, 30 parameters
  - BO (only in low dimensional spaces) due to complexity of underlying GP and numerical approximations.

Bayesian Optimisation

- For Objective Function f, $\min f(\overline{x})$ by constructing a probability distribution over the set of possible functions. Update with samples.
- Common to model distribution as a Gaussian Process (a multivariate normal distribution), which is unique charactered by a mean vector and a covariance kernel.
- Three Steps:
  - Sample - Numerical Optimisation
  - Evaluate - Expensive
  - Update - Bayesian Updating

Algorithm 1 The Bayesian optimization methodology

Input: Objective function $f()$
Input: Acquisition function $\alpha()$
Initialize the Gaussian process G
for $i = 1, 2, \ldots$ **do**
    Sample point: $\mathbf{x}_t \leftarrow \arg\max_{\mathbf{x}} \alpha(G(\mathbf{x}))$
    Evaluate new point: $y_t \leftarrow f(\mathbf{x}_t)$
    Update the Gaussian process: $G \leftarrow G \mid (\mathbf{x}_t, y_t)$
end for

Structured Bayesian Optimisation

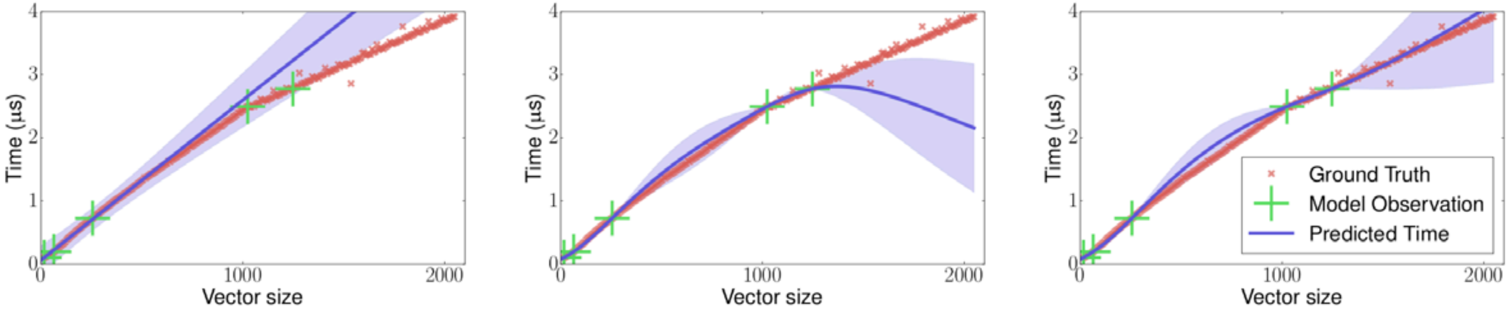
- Objective Function isn’t an arbitrary function, has known structure
- Leverage this known structure to reduce the configuration space
- Replace GP from BO with structured probabilistic model of a system.
- Begin with GP -> Incrementally add structure.
- Garbage Collection Example:
  - Configuration: Young generation size, Survivor ratio & Tenuring threshold
  - Goal: Minimise the 99% percentile latency
  - Attempt with GP, add notion of average duration of minor collection…
  - Small amounts of structure sufficient for large improvement in convergence.

BOAT Models

- Specify the configuration space (Compiler Flags, Device Scheduling…)
- Specify the target performance metric (Runtime, Latency)
- Runtime measurements (Metric, Parametric Model Targets)
- Specify probabilistic system behaviour model…

Semi Parametric Models

- Combination of parametric (fixed) & non-parametric (non-fixed) models
- Idea properties of model for optimization problems:
  - It should understand the general trend of the objective function to avoid exploring low performance regions.
  - It should have high precision in the region of the optimum, to find the point with highest performance.
- “The non-parametric model is used to learn the difference between the parametric model and the observed data.”



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

BOAT Design

- Models in BOAT are assembled from sub-models.
- Models should be Compartmentalised, each sub-model responsible for predicting a single observable value.
- Using a Directed Acyclic Graph for independence -> similar conditional independence properties to Bayesian Network!
- Each sub-model is semi-parametric.

BOAT Semi Parametric Model Implementation

- BOAT library Probabilistic C++
- Constructor: Instantiates parameters by sampling from initial prior:
  - Parametric model parameters
  - Gaussian Process parameters
- Parametric function returns prediction

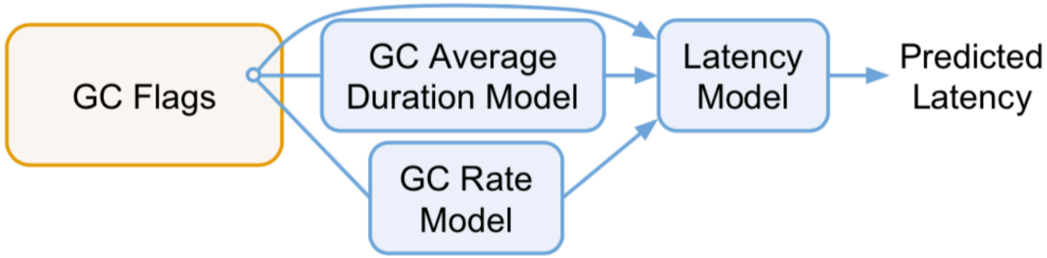
```cpp
struct GCRateModel : public SemiParametricModel<GCRateModel> {
  GCRateModel() {
    allocated_mbs_per_sec =
     std::uniform_real_distribution<>(0.0, 5000.0)(generator);
    // Omitted: also sample the GP parameters
  }
  double parametric(double eden_size) const {
    // Model the rate as inversly proportional to Eden's size
    return allocated_mbs_per_sec / eden_size;
  }
  double allocated_mbs_per_sec;
};

int main() {
  // Example: observe two measurements and make a prediction
  ProbEngine<GCRateModel> eng;
  eng.observe(0.40, 1024);  // Eden: 1024MB, GC rate: 0.40/sec
  eng.observe(0.25, 2048);  // Eden: 2048MB, GC rate: 0.25/sec
  // Print average prediction for Eden: 1536MB
  std::cout << eng.predict(1536) << std::endl;
}
```

Listing 1: The GCRate semi-parametric model.

Global Model

- Directed Acyclic Graph.
- DAGModel Class Defines Dataflow




```
struct CassandraModel : public DAGModel<CassandraModel> {
  void model(int ygs, int sr, int mtt){
    // Calculate the size of the heap regions
    double es = ygs * sr / (sr + 2.0);// Eden space's size
    double ss = ygs / (sr + 2.0);      // Survivor space's size
    // Define the dataflow between semi-parametric models
    double rate =     output("rate", rate_model, es);
    double duration = output("duration", duration_model,
                             es, ss, mtt);
    double latency =  output("latency", latency_model,
                             rate, duration, es, ss, mtt);
  }
  ProbEngine<GCRateModel> rate_model;
  ProbEngine<GCDurationModel> duration_model;
  ProbEngine<LatencyModel> latency_model;
};

int main() {
  CassandraModel model;
  // Observe a measurement
  std::unordered_map<std::string, double> m;
  m["rate"] = 0.40; m["duration"] = 0.15; m["latency"] = 15.1;
  int ygs = 5000, sr = 7, mtt = 2;
  model.observe(m, ygs, sr, mtt);
  /* Prints distributions (mean and stdev) of rate, duration
     and latency with a larger young generation size (ygs)*/
  std::cout << model.predict(6000, sr, mtt) << std::endl;
  // Print corresponding expected improvement of the latency
  std::cout << model.expected_improvement(
      "latency", 15.1, 6000, sr, mtt) << std::endl;
}
```

Listing 2: The full Cassandra latency model.

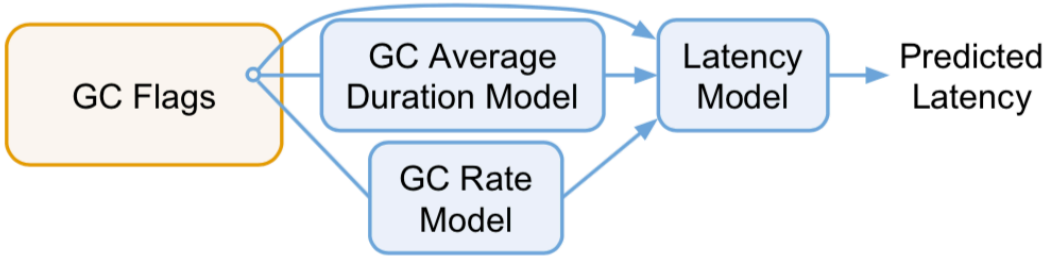
Evaluation Summary

- Demonstrates benefit of Auto-Tuning Approach
- Demonstrates benefit of SBO vs BO (OpenTuner & Spearmint)

- Garage Collection Demonstration
- Neural Network Optimisation

Garbage Collection Results

- Tuning: young generation size, survivor ratio and max tenuring threshold flags, of Cassandra JVM. Small Parameter Space.
- Objective Function: Minimise 99% latency
- Model:



- Results: Converged in 2 iterations vs 16

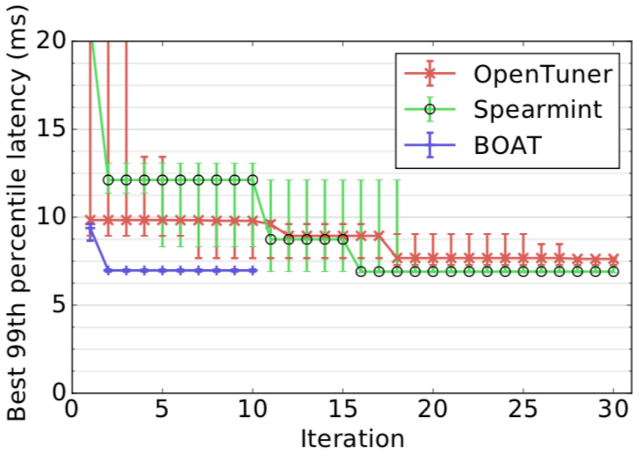


Figure 6: Convergence of the frameworks on workload B.

Neural Network

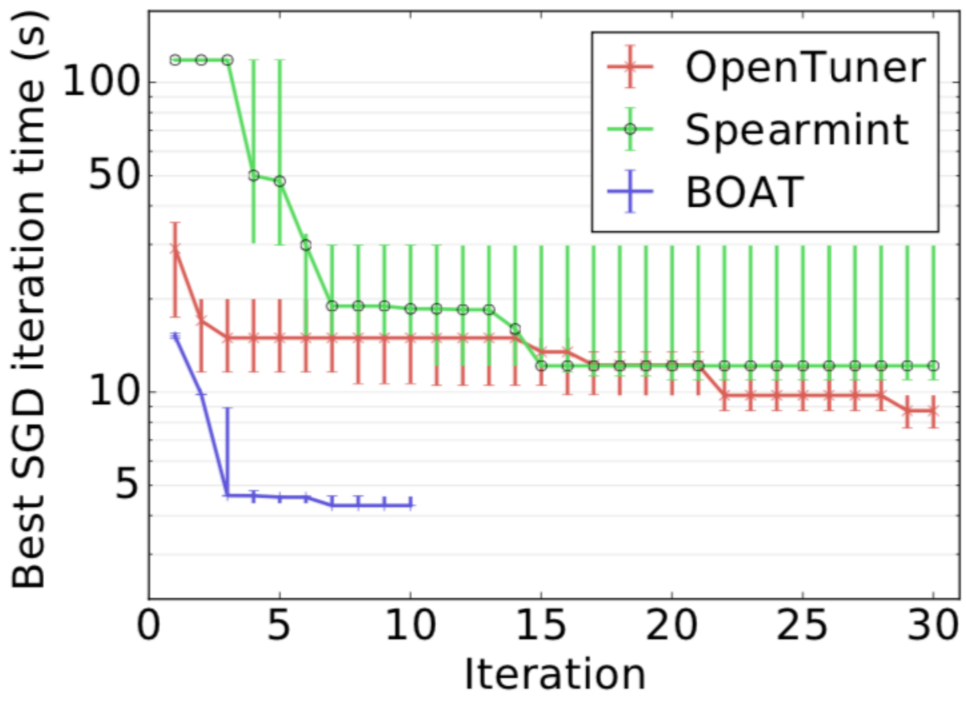
- Workload balancing using distributed TensorFlow.
- Configuration: choosing workers & parameter servers, partitioning workload among heterogenous machines, choosing the batch size.
- Workers & Parameter Servers:
  - Parameter Server tasks synchronize the gradients at every iteration and update the parameters.
  - Worker tasks compute the gradient estimates.
- Partitioning Workload:
  - Choosing workload for each machine including CPU vs GPU distribution.
- Batch Size for Stochastic gradient descent:
  - Higher Batch Size allows for more parallelism
  - Lower Batch Size tends to produce better accuracy
- Approximately 32 Parameters for 10 Machine setup

Neural Networks

- Objective Function: Average time of previous 10 SGD iterations
- Model:
  - Individual device (CPU / GPU) computation time: ∝ Assigned Workload
  - Individual machine computation time: Parametric component modelled sum of individual device computation time, non parametric component modelled gradient aggregation time.
  - Communication time for cluster: $\max_{m \in machines} \frac{transfer(m)}{connection\_speed_m}$
  - Total SGD iteration time is then a function of communication time and the maximum individual machine computation time.

Neural Networks

- Performance:



Previous Work

- None-Structured Bayesian Optimisation
- System Auto-Tuning – BOAT extends to new domains.

Discussion