BoGraph: Structured Bayesian Optimization From Logs for Expensive Systems with Many Parameters

Sami Alabed
University of Cambridge, UK
The Alan Turing Institute
sa894@cam.ac.uk

Eiko Yoneki
University of Cambridge, UK
eiko.yoneki@cl.cam.ac.uk

Abstract

Current auto-tuners struggle with computer systems due to their large complex parameter space and high evaluation cost. We propose BoGraph, an auto-tuning framework that builds a graph of the system components before optimizing it using causal structure learning. The graph contextualizes the system via decomposition of the parameter space for faster convergence and handling of many parameters. Furthermore, BoGraph exposes an API to encode experts' knowledge of the system via performance models and a known dependency structure of the components. We evaluated BoGraph via a hardware design case study achieving $5x - 7x$ improvement in energy and latency over the default in a variety of tasks.

ACM Reference Format:
Sami Alabed and Eiko Yoneki. 2022. BoGraph: Structured Bayesian Optimization From Logs for Expensive Systems with Many Parameters. In *2nd European Workshop on Machine Learning and Systems (EuroMLSys '22), April 5–8, 2022, RENNES, France.* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3517207.3526977

1 Introduction

Computer systems facilitate the execution of a wide range of diverse workloads by exposing tunable configurations to meet users' demands. Examples of these configurations are hardware designs [12] and database knobs [51]. Tuning systems configuration is tedious due to the many parameters involved and the long evaluation time. For example, in our case study on hardware design optimization, there are 2^{64} unique designs. This complexity necessitates for auto-tuners to aid practitioners. However, current research into auto-tuners focuses on optimizing neural network hyperparameters [8, 40, 48] and encounters significant challenges when applied to tuning computer systems specifically.

EuroMLSys '22, April 5–8, 2022, RENNES, France
© 2022 Copyright held by the owner/author(s).
ACM ISBN 978-1-4503-9254-9/22/04.
https://doi.org/10.1145/3517207.3526977

1.1 BoGraph

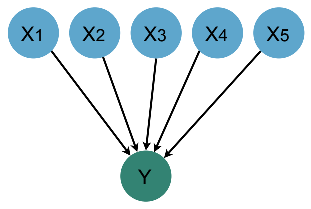
BoGraph builds on Structured Bayesian Optimization (SBO) [16] principles to contextualize the system and finds optimal system configurations in a few trials. SBO requires experts to encode their system knowledge using hand-crafted probabilistic models. BoGraph differentiates itself from previous works by simplifying the process of using SBO. It discovers probabilistic models from logs using its novel structure discovery pipeline. BoGraph's API allows experts to encode contextual knowledge of the system, such as models or dependencies between components, for a more robust and interpretable auto-tuner. Next, we summarize the challenges BoGraph targets in optimizing computer systems.

1.2 Challenges in systems optimization

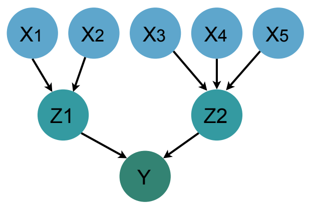
1.2.1 Evaluation is slow and expensive. Evaluating a single configuration on a system takes several minutes (see Figure 5a) to several hours [20, 36, 51]. This is costly especially when optimizing cloud instances [2] or databases [51]. Hence, the need to find optimized configurations with the fewest evaluations. As a result, tuners that rely on many iterations are too costly to use, e.g., reinforcement learning [49], random search [8], evolutionary search [3], hill-climbing [4], or population-search [29]. BoGraph leverages all available information from the system logs and captures it using Gaussian Process (GP) [42] a sample-efficient model to contextualize the system. Additionally, it allows incorporating existing experts' knowledge to speed up the process further.

1.2.2 Large number of parameters. Using GPs comes at the cost of increased computational complexity and difficulty to scale to $D > 10$ parameters without sacrificing their efficiency [34]. Standard Bayesian Optimization [48] methods often rely on GPs, which do not scale to many parameters. BoGraph enables the GPs to scale by using the system's dependency graph to naturally reduce each model dimension as illustrated in Figure 1.

1.2.3 Complex parameters dependency. Computer systems parameters often have a complex relationship; a parameter can depend on another parameter, e.g. the memory's structure and size. This relationship is non-linear and difficult to learn using traditional methods. BoGraph exposes an API for experts to encode a parametric probabilistic model that captures parameters' complex relationship.



(a) The model has to consider all the parameters jointly.



(b) Decomposition reduces the effective dimensions naturally.

Figure 1. The structure reduces the effective dimensions.

1.2.4 Designing bespoke models is complicated.

Handcrafting models that capture the system's complexity requires system and machine learning expertise. Furthermore, validating and evaluating the models is a trial-and-error process. Making tuners that rely on expert models such as BOAT [16] and Causal Bayesian Optimization [1] difficult to use. BoGraph allows encoding models if they exist, and it provides a pipeline that learns these models if necessary. It captures statistical dependency between parameters and system metrics then uses GPs to model their interactions.

1.3 Contributions

We utilized BoGraph to optimize the design space ($\approx 2^{64}$) of an accelerator simulator, gem5-Aladdin [24, 47]. BoGraph learned the system's components dependencies from logs, then modeled them using a graph of probabilistic models. BoGraph found configurations in twenty iterations that took the next best auto-tuner a hundred iterations. Moreover, while improving the energy-latency utilization by $5-7x$ over the default, no other method came close to its performance.

We summarize our contributions as follows:

- An auto-tuner for computer systems that handle many parameters and find optimal configurations quickly.
- A SBO framework automatically learns a graph of probabilistic models from the system's logs.
- An API for experts to express the system's dependencies or encode probabilistic models of its components.
- Optimized all the accelerator design choices in gem5-Aladdin and improved energy-latency by $5-7$x.

2 Background

2.1 Bayesian optimization

Bayesian Optimization (BO) [45] is an iterative sample-efficient method to optimize configurations. Formulated as $x^* = argmin(\alpha(\mathbb{M}(f))$, where x^* is the optimal configuration, α is the acquisition function, $\mathbb{M}$ is a probabilistic model of the system, and f is the system. Its efficiency comes from generating samples using $\mathbb{M}$ that α optimizes to find configurations that improve the system without direct interaction with it. The choice of $\mathbb{M}$ and α directly impact the required evaluations to find optimal configurations.

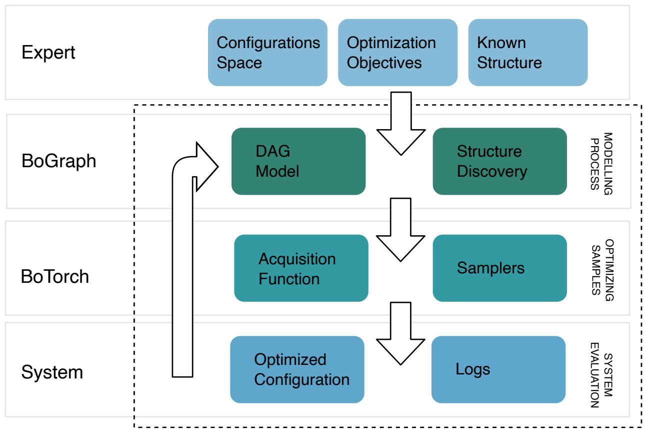


Figure 2. BoGraph's optimization loop. The expert provides specifications of the problem. Then, BoGraph learns a system structure from the logs and combines expert knowledge to build a probabilistic model. Finally, it generates samples from the model and optimizes them to propose optimized configurations to the system.

2.2 Gaussian Process

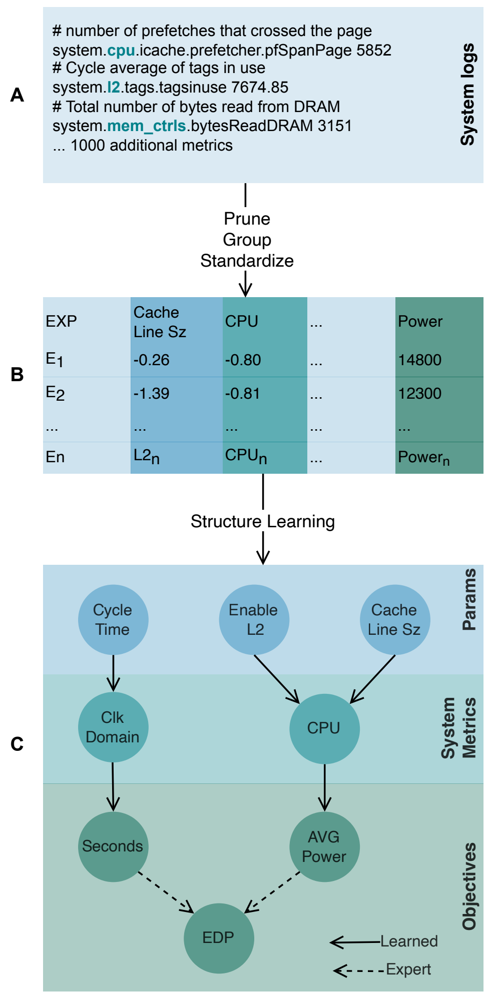
Gaussian Process (GP) [42] is a flexible sample-efficient model commonly used in BO. GP captures the impact of the configurations against the objective in the mean function and co-variance in its kernel. In practice, GP struggles with problems that have $D > 10$ [53] as it scales cubically to the data and dimensions. Its runtime requires an expensive matrix inversion operation for training and sampling, hence its scalability issues. The computational complexity of the GP renders it ineffective for high-dimensional problems.

2.3 Probabilistic DAG

BoGraph models the system using a directed-acyclic-graph (DAG), where the nodes are probabilistic models that capture a component of the system, and the edges are the conditional dependency between the models. This representation, a DAG of probabilistic models, is known as Bayesian Network [31]. Bayesian Network has the advantage of factorizing the joint distribution of a node into a local distribution that depends only on the parent: $P(X; G) = \prod_{j=i}^{p} P(X_i | X_{parent(i)})$ where X_i is the *ith* node in the graph and *parent(i)* is the direct parent of the node. Nodes factorization circumvents *the curse of dimensionality* and allows each node to scale independently.

3 BoGraph Framework

BoGraph is an auto-tuning framework for systems with many configurations. It builds on Structured Bayesian Optimization (SBO) [16] by utilizing contextual knowledge to converge quickly. BoGraph novelty is to allow a hybrid method for injecting expert knowledge into the system as well as learning system structure from logs. Using its pipeline, it identifies statistical links between the system components and uses them to build a system's probabilistic DAG.


```
# number of prefetches that crossed the page
system.cpu.icache.prefetcher.pfSpanPage 5852
# Cycle average of tags in use
system.l2.tags.tagsinuse 7674.85
# Total number of bytes read from DRAM
system.mem_ctrls.bytesReadDRAM 3151
... 1000 additional metrics
```

EXP	Cache Line Sz	CPU	...	Power
E_1	-0.26	-0.80	...	14800
E_2	-1.39	-0.81	...	12300
...	...	...	...	...
En	$L2_n$	CPU_n	...	$Power_n$

Figure 3. A) A snippet of gem5's (the environment) logs. B) The logs after summarizing and grouping. C) The structure resulted by calculating a statistical correlation between components (respecting parameters-component causality) and applying the expert-defined edges (the entire graph is too large for the paper).

Figure 2 shows BoGraph's main components. The expert encodes any knowledge of the system and the target objectives. Then, BoGraph scans the system logs to learn the statistical dependency of the system's components. Next, it combines the expert's knowledge with the learned dependency producing a probabilistic DAG. Finally, it suggests configurations to evaluate by generating many samples from the model and optimizing them using an acquisition function. Its modular design facilitates the use of any sampler and acquisition functions, with BoTorch [6] being the default.

3.1 Structure discovery from logs

BoGraph's first component is the structure discovery process shown in Figure 3 that learns a dependency DAG from the system logs. The pipeline first summarizes the logs, then discovers the statistical dependency using structure learning [31, 56], and finally connects any expert-defined edges to produce a dependency DAG of the system components.

Summarizing the logs. Computer systems generate logs used for monitoring and debugging by experts [9], BoGraph uses the logs to learn components dependencies. However, the logs need to be summarized first as there are often many log entries causing *the curse of dimensionality*. For example, in the case study described in section 4, gem5-Aladdin reports more than 1000 log entry (Figure 3A), BoGraph pipeline summarizes them to 7 entries (Figure 3B).

Preprocessing. First, BoGraph removes uninformative metrics by pruning ones with low variance between executions. Then, they are standardized to ensure that different measuring units do not skew the learning algorithm. Then, BoGraph exploits the structured property of logs to summarize them. The logging protocol encodes where the log was generated in the system [21, 55]. For example, in the case study on gem5, the structure is *Component.SubComponent.Measurement*, as shown in Figure 3A, BoGraph groups all measurements under *SubComponent* in one group as they influence the same component in the system, then they are compressed into a single statistic using Factor Analysis (FA) [30]. This compression works because logs generated at the *SubComponent* level are dependent on each other. The user can decide on the granularity of the grouping based on their need. A fine granularity is helpful to understand the system, but it is more expensive to evaluate. We used the highest level of granularity which mapped gem5 to six main categories: L2, Datapath, Mem Ctrls, Membus, Tol2Bus, and CPU. This process applies to all system logs, including previous system evaluations. At the end of this stage, BoGraph produces a summary of the logs as seen in Figure 3B for the structure learning algorithm.

3.2 Structure learning

Defining systems' structure is an error-prone process; hence the need to learn it from data, an active research area in the Bayesian community [44]. These methods propose several DAGs and score them based on a trade-off between data fit and the complexity of the graph. BoGraph uses a structure learning algorithm, NoTears [56]. NoTears approximates the DAG score using a differentiable function, then minimizes its error, which is easily parallelizable on GPUs [7]. It is simple to extend BoGraph with other structure learning algorithms by implementing the BoGraph's StructureLearning interface. We encode a set of restrictions in these algorithms that ensures the system's parameters are not allowed to have parents, and the objectives nodes are not allowed to have children to ensure the causal flow of the optimization task. Then apply any expert-defined edges resulting in Figure 3C. Optionally, BoGraph takes an update scheduler that decides when to perform structure learning. By default, it performs structure learning every quarter of the optimization budget.

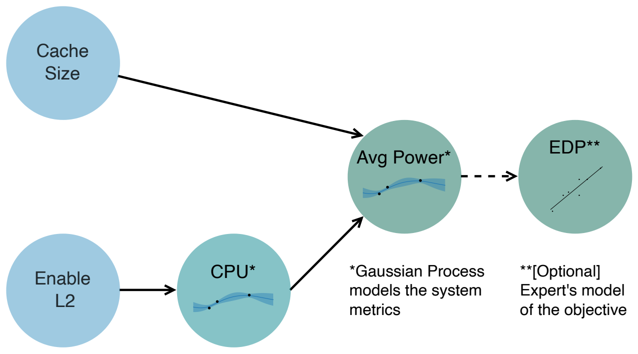

Cache Size
Enable L2
CPU*
Avg Power*
EDP**
*Gaussian Process models the system metrics
**[Optional] Expert's model of the objective

Figure 4. BoGraph maps the node in the structure to probabilistic models such as GPs or expert's models, creating probabilistic DAG.

Algorithm 1 Sampling the probabilistic DAG

```
Input: sampler $s$, DAG model $M$, system $SY$
initialize a samples cache: $Q := Dict()$
get nodes along the path to the objectives: $P = M.path(SY.obj)$
for node $n$ in $P.nodes$ do
    if $n \in SY.params$ then
        sample parameters: $p_{samples} = s.sample(n.space())$
        cache generated samples: $Q \cup (n.key, p_{samples})$
    else
        condition on parents: $C = n.predict(Q[n.parents])$
        cache node results conditioned on parents: $Q \cup (n.key, C)$
    end if
end for
Return: $Q[SY.objectives]$
```

3.3 Graph of a Probabilistic Models

BoGraph builds the probabilistic DAG by placing a GP node on intermediate nodes, and then it connects any expert-defined models as illustrated in Figure 4. The probabilistic DAG nodes' are *compartmentalized*, where each node predicts a single value conditioned on its parents directly and can be trained independently. As a result, any node can be an optimization target, providing free multi-objective modeling and enabling a parallelizable training. We will investigate both in a follow-up project.

Sampling process. Allowing general probabilistic models comes at the cost that closed-form acquisition function optimization is not possible; instead, BoGraph samples the model and then optimizes over the samples. This design gives it the freedom to use any probabilistic model inside the graph, such as Bayesian Neural Networks [35]. Algorithm 1 shows the process to generate a prediction and samples from a probabilistic DAG. First, BoGraph finds all the nodes with a direct path from the parameters to the objective. Then, it creates a cache to hold all the samples and avoid recomputing them for nodes with multiple children. Next, each node condition on the parents' prediction of the samples, and finally output the posterior of the objectives.

```
class ExpertModel(PyroNode): # User extends PyroNode to build models
  def model(x1: Tensor, obs: Tensor = None):
    i = pyro.sample("intercept", dist.Uniform(0., 5.))
    linear_mean = i + pyro.sample("bias_x1", dist.Normal(0.,1.))
    ... # a standard Bayesian model using Pyro
prior_dag =  BoGraphPrior() #  BoGraph uses `networkx` API
prior_dag.add_edges_from(['x1', 'f1(x1)']) # f1(x1) depends on x1
prior_dag.add_model("f1(x1)", ExpertModel()) # add expert's model
prior_dag.add_model("y", GPyTorchModel()) # standard GP model
prior_dag.add_tabu_edge("x1", "y") # Edges known to not be correlated
BoGraph.optimize(objective=Forrester2D, prior=prior_dag)
```

Listing 1. BoGraph API showing expert knowledge injection

3.4 Structured Bayesian Optimization

Algorithm 2 BoGraph Bayesian Optimization loop

```
Input: acquisition function $\alpha$, system $E$, budget
repeat
    build DAG model from logs: $DAG_M = BoGraph(E.logs())$
    select next to point to evaluate: $x_{n+1} = \alpha(s.sample(DAG_M))$
    submit $x_{n+1}$ to be evaluated: $E.update(x_{n+1})$
until $budget = 0$
```

Optimization loop. Algorithm 2 details the optimization loop. BoGraph's probabilistic DAG combines several models' posterior, producing a complicated posterior that cannot be optimized directly. Instead, we use quasi-acquisition functions that optimize the estimate from samples rather than the model directly. BoTorch [6] provides a suite of utility functions for BO frameworks, including quasi-acquisition functions and samplers. We use BoTorch's Sobol sequence sampler to generate diverse samples of the posterior, then use the quasi-Expected Improvement acquisition function to optimize the estimates and produce optimization candidates.

3.5 BoGraph API

BoGraph API streamlines the process of expressing system designers' knowledge by providing both high and low-level APIs. Listing 1 shows several of the functions BoGraph that enable experts to encode their knowledge in several ways: A high-level DAG API for expressing the dependencies between parameters and metrics. The API extends the popular library for building graphs, *networkX* [23] as such provides the ability to read all the popular methods for defining directed graphs. And a lower-level API that allows expressing experts' knowledge using *Pyro* [11], the probabilistic programming language. By default, BoGraph uses a GP as the metric model if no probabilistic model is defined. The GP implementation we use is GPyTorch [22] a performant GP implementation. The user can override the kernel of the GP to inject any prior information about the objective function distribution; this is only recommended for advanced users.

3.6 Summary

BoGraph simplifies the process of integrating SBO in systems for faster optimizer convergence. The main loop performs structure learning of the system's components by processing the logs and transforming it into a probabilistic model. The framework exposes an API that allows experts to encode dependency structure and probabilistic models.

4 Evaluation

System-on-a-chip (SoC) designers often experiment on a simulator [12, 15, 46] to reduce the energy and latency of integrated chips [10, 41]. Unfortunately, the design space is enormous ($\approx 2^{64}$ combinations) hence the need for BoGraph. We optimized Energy-Delay Product (EDP)[5] of gem5-Aladdin [47] where $EDP = energy * (\frac{1}{sim\_seconds})^2$ by tuning all the design parameters exposed by the simulator [25].

Setup. The experiment setup ran on RTX3060TI, Intel i7 8700k, and 32GB RAM. BoGraph uses GPs from GPyTorch v1.2 [22], samplers and acquistion functions from BoTorch v0.5 [6], and CausalNex v0.11 [7] for structure learning.

Benchmark. We benchmarked the MachSuite [43] that has a mix of data and compute workloads as is the standard in SoC design literature [41, 47]. The gem5-Aladdin simulator was compiled using MESI Two Level Aladdin protocol, and the power was estimated using MathExprPowerModel.

Baselines. We used these baselines in the experiments:

- **Default** [24]: the settings provided by the simulator.
- **Random** [8, 40]: useful when small subset of the dimensions significantly impact the objective.
- **PBTTuner** [28, 40]: a bag of model approach to finding a collection of optimal configurations.
- **BoTorch** [6]: shows the effectiveness of BoGraph with everything else being equal as BoGraph shares BoTorch's samplers and acquisition functions.
- **DeepGP** [17]: Provides unsupervised structure discovery in the latent space. We configured a two-layer GPs with 64 inducing points (Our GPU's max VRAM).

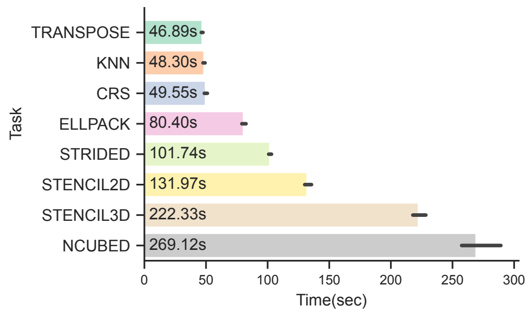
Expert knowledge. We provided BoGraph with the EDP formula: edges from power and latency to the EDP.

Evaluation goals. We evaluated the following claims:

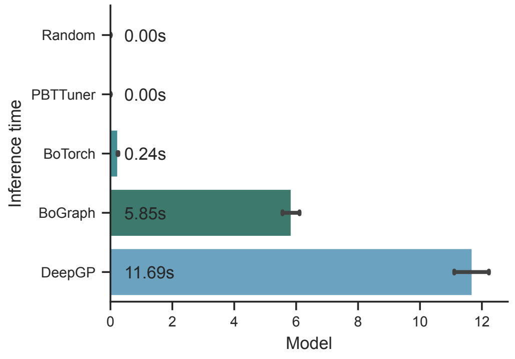
- The overhead of running BoGraph is lower than running a full system evaluation, 4.1.
- BoGraph converges towards optimal configurations faster than other state-of-the-art methods, 4.2.
- BoGraph produces interpretable system structures that reduces the maximum dimension, 4.3.

4.1 Execution time

We compared the execution time of the simulator for each task and compared it to the overhead of using an auto-tuner in Figure 5. The results show that the optimizers are cheaper than a full system evaluation. Hence the need for auto-tuners that perform better than simple parameter sweeps.



(a) Tasks' average execution time, with variance from 1800 runs.



(b) Auto-tuners' average time to train and propose a configuration, with variances from 300 runs.

Figure 5. Optimizers' execution time is dominated by the gem5-Aladdin task execution time.

Table 1. ELLPACK's performance for the median best configuration proposed by the models, lower values are better.

MODEL	LOG EDP	LATENCY	POWER (MWATTS)
BOGRAPH	**8.12 ± 0.28**	**11.06 ± 0.19**	**27.37 ± 11.59**
DEEPGP	10.14 ± 0.27	11.06 ± 3.92	203.15 ± 46.34
PBTTUNER	10.34 ± 0.60	10.81 ± 4.46	264.36 ± 147.79
BOTORCH	10.44 ± 0.62	11.06 ± 3.90	108.46 ± 102.97
RANDOM	10.74 ± 0.19	18.32 ± 4.81	143.94 ± 194.83
DEFAULT	14.50 ± 0	94.80 ± 0	220.34 ± 0

4.2 EDP Optimization

Figure 6 examines the improvement in the EDP from the default configurations across all tasks. BoGraph outperformed every other method and improved over the default by $5x - 7x$ factors. Examining that further, Table 1 shows that BoGraph found configurations with the least power consumption for the ELLPACK task. Other tuners are stuck in local minima that do not improve the energy as they are unaware the EDP is impacted by power. BoGraph is aware that the latency and power impact EDP thus can avoid the local minima. The same story applies to the other tasks in the benchmark.

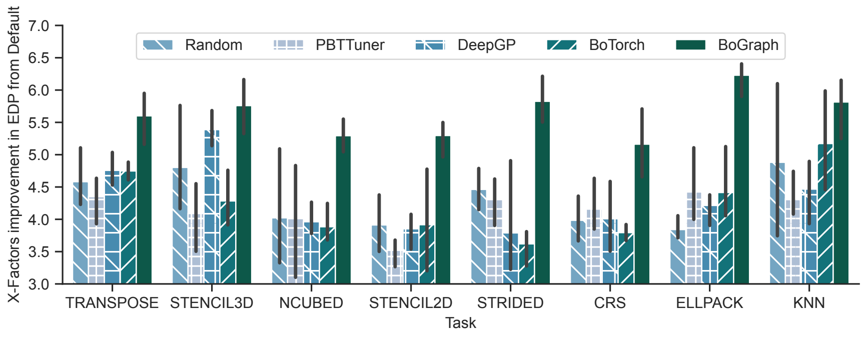


Figure 6. The best found EDP in 100 steps. The y-axis shows X-factors improvement over default settings and the variance of three runs.

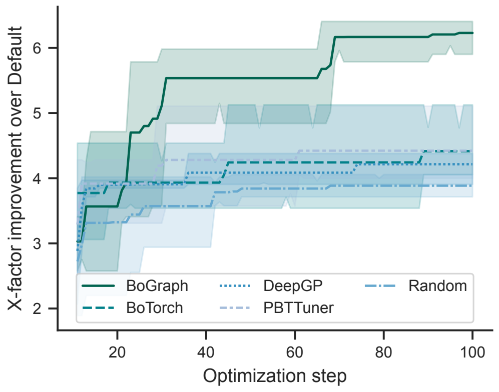


Figure 7. Auto-tuners' improvement over default, with a median best configuration and the error bars showing a minimum of 3 runs.

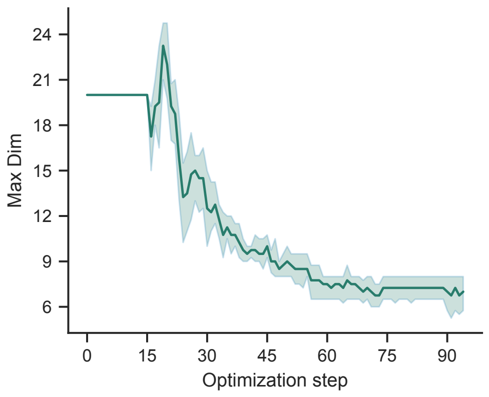


Figure 8. BoGraph DAG max-dimension (lower is better) of three repeated runs. The max dimension is defined as $MD = \forall_{n \in G} max(indegree(n))$, each node becomes a probabilistic model.

Convergence. Figure 7 examines the convergence rate of each optimizer as minimizing the evaluations reduces the cost significantly (Challenge 1.2.1). BoGraph reached the best configuration the quickest and continued to improve while other methods plateaued. Confirming that tuning computer systems is challenging for most auto-tuners. BoGraph improvement jumps at every 20th step in Figure 7 are caused by the structure learning algorithm, as its update frequency defaults to a quarter of the evaluation budget.

4.3 Structure discovery

There are over 1000 metrics reported in a gem5 statistics file. BoGraph shields the end-user from the difficult task of defining a probabilistic model while providing an insightful structure to the designer as shown in Figure 3C. Furthermore, Figure 8 shows that learning structure does reduce the maximum dimension for the models in the graph.

5 Oppurtinities and limitation.

5.1 Oppurtinities

BoGraph is still evolving, with aims to apply to different problem spaces, evaluate it on a multi-objective problem, and improve its computational engine.

Multi-objective optimization. Multi-objective modeling is an interesting opportunity for computer system objectives often compete (latency vs. throughput, energy vs. latency). Modeling multiple objectives is trivial in BoGraph as a result of using DAGs as an internal data structure as any node can be sampled. Then using off-the-shelf multi-objective utility functions such as expected hypervolume improvement function [18] to optimize the samples from multiple nodes.

Computational overhead. The overhead of using structured DAG is high compared to the standard approaches. During training and sampling, BoGraph performs sequential operations without exploiting the independence between nodes to batch and distribute the workload. Leveraging parallelization will reduce the computational time of BoGraph.

Application domains. We have shown BoGraph ability to find meaningful structure from logs and use that to optimize gem5-Aladdin. Other computer systems will benefit from BoGraph ability to make sense of logs in aiding the optimizer. We are working on a case study on optimizing the latency of PostgreSQL [38], BoGraph is using the reported 370 metrics to find links between the metrics and the latency objective and tuning of thirty parameters.

5.2 Limitations

BoGraph's pipeline overhead. BoGraph maintains the system trace (previous evaluation, configurations, and logs) on disk, reading and writing these traces have both a memory and execution time overhead. This is a design choice we took for the following reasons: First, the cost of the real system dominates the execution of the pipeline (subsection 4.1). Secondly, this enables BoGraph in the future to learn from distributed instances. Finally, it provides a fault-tolerance mechanism for when the instance has to restart.

Mitigation. Every stage and component of the BoGraph pipeline is configurable. For example, the user can decide to reduce the frequency of a full causal structure discovery or include preprocessing steps that reduce the computation further by extending the preprocessing interface. This flexibility offers some mitigation to the pipeline overhead. Furthermore, to mitigate this overhead, BoGraph can run a helper instance that runs the pipeline and communicate only the summarized information to the BoGraph main loop.

Log annotation. A key contribution of BoGraph is providing a pipeline to make use of system logs and metrics to provide additional context that aids the system model. Relying on the user to annotate their logs is one limitation of this work. The annotation process involves providing a parser of the metrics or system logs; in gem5 and PostgresSQL experiments, it is a one-line regex expression. Logs in computer systems follow a standard practice [55] as they are often produced to be ingested into metric monitoring applications such as Prometheus [13]. However, suppose the system is very different where the logs do not correlate with the system. In that case, it is possible to disable the structure learning pipeline in BoGraph and rely on a manual structure, resulting in a similar performance to the standard structure Bayesian Optimization method [16].

6 Related work

Search-based auto-tuners. Reinforcement learning [49], random search [8], evolutionary search (in PetaBricks) [3], hill-climbing (in OpenTuner) [4], or population-search [29] are simple auto-tuner to use and scale to many parameters. However, they require many system evaluations, waste valuable expert knowledge, and are highly dependent on initializing seed (unstable) [26]. These drawbacks render them unsuitable for our problem due to Challenge 1.2.1.

Dimensionality reduction. OtterTune [50] used Factor Analysis [30] to reduce the system dimensions. However, as the OtterTune experiment expanded to a large system [51], it made apparent that computer system parameters have complex interactions that standard dimensionality reduction methods fail to capture (Challenge 1.2.3). Principled Component Analysis [54] suffer from a similar problem.

BO surrogate models. RandomForest [33] used in SMAC [27] and HyperMapper [39] scales to many parameters. Unfortunately, non-Bayesian methods underestimate the models' variance [45] meaning they explore less frequently and miss on finding the optimal configurations. DeepGPs [17] use several layers of GPs that reduce the objective dimensionality by finding a latent structure of it. However, it requires many experiments to find a meaningful latent structure as it ignores existing expert knowledge.

Parallel based auto-tuners. AutoTVM [14], Hyperband [32] and BOHB [19] are techniques that depend on the environment being cheap to terminate and easily parallelizable. Computer systems are expensive and slow to evaluate; engineering them to launch many parallel instances is non-trivial without impacting the measurement sensitivity.

Experts-driven methods. Ernest [52], and RubberBand [37] use hand-crafted models to measure the proposed configurations' effectiveness quickly leading to quick optimization. While Causal Bayesian Optimization [1] assumes that a full causal graph of the system exists to speed up finding optimal system configurations. Hand-crafting models for systems is a complicated, error-prone task that requires expertise in systems and machine learning (Challenge 1.2.4).

7 Conclusion

BoGraph is a framework for optimizing systems with many parameters and slow execution. BoGraph contextualizes the system through learning association between its metrics and parameters. Furthermore, it provides an API to encode experts' knowledge as probabilistic models. The contextualization leads to tuning more parameters in fewer experiments. Using BoGraph we optimized the design of an accelerator to improve its energy-latency objective by $2x$ in $1/4$ of the evaluations of the next best method. For future work, we are applying BoGraph to PostgreSQL, showing applicability to a range of computer systems.

Acknowledgments

We would like to thank the reviewers for their valuable feedback. A special thanks to Thomas Vanderstichele, Mihai Bujanca, Yomna El-Serafy, Ibrahim Abdou, and Rana ElRashidy for their comments that improved the readability of the paper. This research was supported by the Alan Turing Institute.

References

[1] Virginia Aglietti, Xiaoyu Lu, Andrei Paleyes, and Javier González. Causal Bayesian Optimization. In *International Conference on Artificial Intelligence and Statistics*, pages 3155–3164. PMLR, 2020.

[2] Omid Alipourfard, Hongqiang Harry Liu, Jianshu Chen, Shivaram Venkataraman, Minlan Yu, and Ming Zhang. CherryPick: Adaptively Unearthing the Best Cloud Configurations for Big Data Analytics, 2017.

[3] Jason Ansel and Cy Chan. PetaBricks. *XRDS: Crossroads, The ACM Magazine for Students*, 17(1):32, 2010.

[4] Jason Ansel, Shoaib Kamil, Kalyan Veeramachaneni, Jonathan Ragan-Kelley, Jeffrey Bosboom, Una-May O'Reilly, and Saman Amarasinghe. OpenTuner. In *Proceedings of the 23rd International Conference on Parallel Architectures and Compilation - PACT '14*, pages 303–316, 2014.

[5] Omid Azizi, Aqeel Mahesri, Benjamin C Lee, Sanjay J Patel, and Mark Horowitz. Energy-performance tradeoffs in processor architecture and circuit design: A marginal cost analysis. *ACM SIGARCH Computer Architecture News*, 38(3):26–36, 2010.

[6] Maximilian Balandat, Brian Karrer, Daniel R. Jiang, Samuel Daulton, Benjamin Letham, Andrew Gordon Wilson, and Eytan Bakshy. BoTorch: A Framework for Efficient Monte-Carlo Bayesian Optimization. *arXiv:1910.06403 [cs, math, stat]*, December 2020.

[7] Paul Beaumont, Ben Horsburgh, Philip Pilgerstorfer, Angel Droth, Richard Oentaryo, Steven Ler, Hiep Nguyen, Gabriel Azevedo Ferreira, Zain Patel, and Wesley Leong. CausalNex, 2021.

[8] James Bergstra and Yoshua Bengio. Random search for hyper-parameter optimization. *Journal of machine learning research*, 13(Feb):281–305, 2012.

[9] Betsy Beyer, Chris Jones, Jennifer Petoff, and Niall Richard Murphy. *Site Reliability Engineering: How Google Runs Production Systems.* " O'Reilly Media, Inc.", 2016.

[10] Kshitij Bhardwaj, Marton Havasi, Yuan Yao, David M. Brooks, José Miguel Hernández Lobato, and Gu-Yeon Wei. Determining optimal coherency interface for many-accelerator socs using bayesian optimization. *IEEE Computer Architecture Letters*, 18(2):119–123, 2019.

[11] Eli Bingham, Jonathan P. Chen, Martin Jankowiak, Fritz Obermeyer, Neeraj Pradhan, Theofanis Karaletsos, Rohit Singh, Paul Szerlip, Paul Horsfall, and Noah D. Goodman. Pyro: Deep Universal Probabilistic Programming. October 2018.

[12] Nathan Binkert, Bradford Beckmann, Gabriel Black, Steven K Reinhardt, Ali Saidi, Arkaprava Basu, Joel Hestness, Derek R Hower, Tushar Krishna, and Somayeh Sardashti. The gem5 simulator. *ACM SIGARCH computer architecture news*, 39(2):1–7, 2011.

[13] Brian Brazil. *Prometheus: Up & Running: Infrastructure and Application Performance Monitoring.* " O'Reilly Media, Inc.", 2018.

[14] Tianqi Chen, Lianmin Zheng, Eddie Yan, Ziheng Jiang, Thierry Moreau, Luis Ceze, Carlos Guestrin, and Arvind Krishnamurthy. Learning to optimize tensor programs. *arXiv preprint arXiv:1805.08166*, 2018.

[15] Yu-Hsin Chen, Tien-Ju Yang, Joel Emer, and Vivienne Sze. Eyeriss v2: A Flexible Accelerator for Emerging Deep Neural Networks on Mobile Devices. *arXiv:1807.07928 [cs]*, May 2019.

[16] Valentin Dalibard, Michael Schaarschmidt, and Eiko Yoneki. BOAT: Building auto-tuners with structured Bayesian optimization. In *Proceedings of the 26th International Conference on World Wide Web - WWW '17*, pages 479–488, New York, New York, USA, 2017. ACM Press.

[17] Andreas Damianou and Neil D Lawrence. Deep gaussian processes. In *Artificial Intelligence and Statistics*, pages 207–215. PMLR, 2013.

[18] Samuel Daulton, Maximilian Balandat, and Eytan Bakshy. Differentiable expected hypervolume improvement for parallel multi-objective Bayesian optimization. *arXiv preprint arXiv:2006.05078*, 2020.

[19] Stefan Falkner, Aaron Klein, and Frank Hutter. BOHB: Robust and efficient hyperparameter optimization at scale. *arXiv preprint arXiv:1807.01774*, 2018.

[20] Avrilia Floratou, Ashvin Agrawal, Bill Graham, Sriram Rao, and Karthik Ramasamy. Dhalion: Self-regulating stream processing in heron. *Proceedings of the VLDB Endowment*, 10(12):1825–1836, 2017.

[21] Qiang Fu, Jieming Zhu, Wenlu Hu, Jian-Guang Lou, Rui Ding, Qingwei Lin, Dongmei Zhang, and Tao Xie. Where do developers log? an empirical study on logging practices in industry. In *Companion Proceedings of the 36th International Conference on Software Engineering*, pages 24–33, 2014.

[22] Jacob R. Gardner, Geoff Pleiss, David Bindel, Kilian Q. Weinberger, and Andrew Gordon Wilson. GPyTorch: Blackbox Matrix-Matrix Gaussian Process Inference with GPU Acceleration. *arXiv:1809.11165 [cs, stat]*, January 2019.

[23] Aric Hagberg, Pieter Swart, and Daniel S Chult. Exploring network structure, dynamics, and function using NetworkX. Technical report, Los Alamos National Lab.(LANL), Los Alamos, NM (United States), 2008.

[24] gem5-aladdin harvard-acc. Gem5-Aladdin SoC Simulator. Harvard Architecture, Circuits, and Compilers, October 2016.

[25] gem5-aladdin-param harvard-acc. Gem5-Aladdin SoC Simulator. Harvard Architecture, Circuits, and Compilers, October 2021.

[26] Peter Henderson, Riashat Islam, Philip Bachman, Joelle Pineau, Doina Precup, and David Meger. Deep reinforcement learning that matters. *arXiv preprint arXiv:1709.06560*, 2017.

[27] Frank Hutter, Holger Hoos, and Kevin Leyton-Brown. An evaluation of sequential model-based optimization for expensive blackbox functions. In *Proceedings of the 15th Annual Conference Companion on Genetic and Evolutionary Computation*, pages 1209–1216, 2013.

[28] Max Jaderberg, Valentin Dalibard, Simon Osindero, Wojciech M. Czarnecki, Jeff Donahue, Ali Razavi, Oriol Vinyals, Tim Green, Iain Dunning, and Karen Simonyan. Population based training of neural networks. *arXiv preprint arXiv:1711.09846*, 2017.

[29] Max Jaderberg, Valentin Dalibard, Simon Osindero, Wojciech M. Czarnecki, Jeff Donahue, Ali Razavi, Oriol Vinyals, Tim Green, Iain Dunning, Karen Simonyan, Chrisantha Fernando, and Koray Kavukcuoglu. Population Based Training of Neural Networks. November 2017.

[30] Paul Kline. *An Easy Guide to Factor Analysis.* Routledge, 2014.

[31] Daphne Koller and Nir Friedman. *Probabilistic Graphical Models: Principles and Techniques.* MIT press, 2009.

[32] Lisha Li, Kevin Jamieson, Giulia DeSalvo, Afshin Rostamizadeh, and Ameet Talwalkar. Hyperband: A novel bandit-based approach to hyperparameter optimization. *The Journal of Machine Learning Research*, 18(1):6765–6816, 2017.

[33] Andy Liaw and Matthew Wiener. Classification and regression by randomForest. *R news*, 2(3):18–22, 2002.

[34] Haitao Liu, Yew-Soon Ong, Xiaobo Shen, and Jianfei Cai. When Gaussian Process Meets Big Data: A Review of Scalable GPs. *arXiv:1807.01065 [cs, stat]*, April 2019.

[35] David JC MacKay. Bayesian neural networks and density networks. *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment*, 354(1):73–80, 1995.

[36] Azalia Mirhoseini, Hieu Pham, Quoc V. Le, Benoit Steiner, Rasmus Larsen, Yuefeng Zhou, Naveen Kumar, Mohammad Norouzi, Samy Bengio, and Jeff Dean. Device Placement Optimization with Reinforcement Learning. *CoRR*, abs/1706.0, June 2017.

[37] Ujval Misra, Richard Liaw, Lisa Dunlap, Romil Bhardwaj, Kirthevasan Kandasamy, Joseph E. Gonzalez, Ion Stoica, and Alexey Tumanov. RubberBand: Cloud-based hyperparameter tuning. In *Proceedings of the Sixteenth European Conference on Computer Systems*, pages 327–342, Online Event United Kingdom, April 2021. ACM.

[38] Bruce Momjian. *PostgreSQL: Introduction and Concepts*, volume 192. Addison-Wesley New York, 2001.

[39] Luigi Nardi, David Koeplinger, and Kunle Olukotun. Practical design space exploration. In *2019 IEEE 27th International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS)*, pages 347–358. IEEE, 2019.

[40] OpenSource. Microsoft/nni. Microsoft, December 2020.

[41] Andy D Pimentel. Exploring exploration: A tutorial introduction to embedded systems design space exploration. *IEEE Design & Test*, 34(1):77–90, 2016.

[42] Carl Edward Rasmussen and Christopher K. I. Williams. *Gaussian Processes for Machine Learning*. Adaptive Computation and Machine Learning. MIT Press, Cambridge, Mass., 3. print edition, 2008.

[43] Brandon Reagen, Robert Adolf, Yakun Sophia Shao, Gu-Yeon Wei, and David Brooks. Machsuite: Benchmarks for accelerator design and customized architectures. In *2014 IEEE International Symposium on Workload Characterization (IISWC)*, pages 110–119. IEEE, 2014.

[44] Mauro Scanagatta, Antonio Salmerón, and Fabio Stella. A survey on Bayesian network structure learning from data. *Progress in Artificial Intelligence*, pages 1–15, 2019.

[45] Bobak Shahriari, Kevin Swersky, Ziyu Wang, Ryan P Adams, and Nando De Freitas. Taking the human out of the loop: A review of Bayesian optimization. *Proceedings of the IEEE*, 104(1):148–175, 2015.

[46] Yakun Sophia Shao, Brandon Reagen, Gu-Yeon Wei, and David Brooks. Aladdin: A pre-rtl, power-performance accelerator simulator enabling large design space exploration of customized architectures. In *2014 ACM/IEEE 41st International Symposium on Computer Architecture (ISCA)*, pages 97–108. IEEE, 2014.

[47] Yakun Sophia Shao, Sam Likun Xi, Vijayalakshmi Srinivasan, Gu-Yeon Wei, and David Brooks. Co-designing accelerators and soc interfaces using gem5-aladdin. In *2016 49th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pages 1–12. IEEE, 2016.

[48] Jasper Snoek, Hugo Larochelle, and Ryan P Adams. Practical bayesian optimization of machine learning algorithms. In *Advances in Neural Information Processing Systems*, pages 2951–2959, 2012.

[49] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning : An Introduction*. MIT Press, Cambridge Mass., 1998.

[50] Dana Van Aken, Andrew Pavlo, Geoffrey J. Gordon, and Bohan Zhang. Automatic Database Management System Tuning Through Large-scale Machine Learning. In *Proceedings of the 2017 ACM International Conference on Management of Data - SIGMOD '17*, pages 1009–1024, New York, New York, USA, 2017. ACM Press.

[51] Dana Van Aken, Dongsheng Yang, Sebastien Brillard, Ari Fiorino, Bohan Zhang, Christian Bilien, and Andrew Pavlo. An inquiry into machine learning-based automatic configuration tuning services on real-world database management systems. *Proceedings of the VLDB Endowment*, 14(7):1241–1253, 2021.

[52] Shivaram Venkataraman, Zongheng Yang, Michael Franklin, Benjamin Recht, and Ion Stoica. Ernest: Efficient performance prediction for large-scale advanced analytics. In *13th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 16)*, pages 363–378, 2016.

[53] Ziyu Wang, Frank Hutter, Masrour Zoghi, David Matheson, and Nando de Feitas. Bayesian optimization in a billion dimensions via random embeddings. *Journal of Artificial Intelligence Research*, 55:361–387, 2016.

[54] Svante Wold, Kim Esbensen, and Paul Geladi. Principal component analysis. *Chemometrics and intelligent laboratory systems*, 2(1-3):37–52, 1987.

[55] Ding Yuan, Soyeon Park, and Yuanyuan Zhou. Characterizing logging practices in open-source software. In *2012 34th International Conference on Software Engineering (ICSE)*, pages 102–112. IEEE, 2012.

[56] Xun Zheng, Bryon Aragam, Pradeep Ravikumar, and Eric P Xing. Dags with no tears: Continuous optimization for structure learning. *arXiv preprint arXiv:1803.01422*, 2018.