Performance tuning classification model: Exploratory evaluation of model performance based on multiple optimization algorithms

Yuehan Zhang

2336499

Yxz272@student.bham.ac.uk

Abstract

Based on the experimental framework of ISE-solution Lab3, this paper designs and implements six different tuning strategies around the theme of "Performance Tuning and Optimization Algorithms", aiming to explore their adaptability and optimization effects in the real system configuration space. We selected 8 representative dataset systems, including: 7z compression tool, Apache server, brotli codec, LLVM compiler, PostgreSQL database, spear solver, storm real-time compute engine, and x264 video encoder. In these systems, each data record represents a configuration experiment, and the corresponding performance metrics can be run time, compression ratio, throughput, and so on.

Introduction

In the process of performance optimization in modern software engineering, distributed systems, and resource-intensive applications, the selection of system configuration parameters plays a crucial role in the final performance results. Due to its high dimension, complex space, multiple variable types (continuous + discrete + enumeration), and non-derivative or even unanalytic objective functions, the configuration tuning problem makes it almost impossible for traditional brute force enumeration or manual debugging to meet the actual optimization requirements. Therefore, the efficient exploration of parameter space with the help of intelligent optimization algorithms has become a topic of extensive concern in research and industry.

In this experiment, the performance differences of the following six types of optimization methods in configuration space search are comprehensively evaluated: BestConfig (itemized change method); decision tree regression optimization; Flash model (based on random forest); Bayesian optimization; genetic algorithms; Bayesian optimization + genetic algorithm combination. The experimental evaluation dimensions include: optimization accuracy, convergence trend, scalability, stability, and visual expression ability, supplemented by quantitative charts and performance curve analysis. Finally, this report summarizes the applicable conditions and potential improvement directions of various algorithms, and provides a useful reference for the algorithm selection of the subsequent performance tuning system.

Research objectives

Focusing on performance tuning and configuration space search, the following research objectives are clarified:

2.1 Systematic assessment

Comparative experiments were carried out on six mainstream configuration tuning algorithms under a unified framework, and the performance differences of six mainstream configuration tuning algorithms

in different systems were systematically analyzed from the dimensions of accuracy, convergence ability, and operational stability.

2.2 Adaptability analysis

Combined with the parameter dimension, search space complexity and performance feedback characteristics of the system itself, the performance and adaptability of each algorithm in different types of systems are evaluated.

2.3 Visual verification

With the help of the matplotlib visualization tool, the performance change curve in each search iteration is dynamically plotted to visually display the optimization path and convergence trend of each algorithm. The search iteration plots of all the following systems are generated based on the Genetic Algorithm (GA) strategy, and record the trend of performance values during each round of population evolution. The blue line in the figure shows the performance change trajectory, and the red five-pointed star represents the best of the current round.

2.4 Problem Summarization and Optimization Suggestions

In the operation of the algorithm, the practical problems encountered (such as library dependence, abnormal operation, difficulty in parameter setting, etc.) are recorded, and feasible optimization strategies are proposed from the perspective of engineering implementation, which lays a foundation for the subsequent promotion of the algorithm.

Methods and experimental procedures

3.1 Datasets and Preprocessing

All the datasets used in the experiment are derived from the experimental framework provided by the open-source tuning system LES, covering eight types of systems commonly used in industrial and academic research. The configuration space of each system presents different dimension sizes and parameter types (e.g., integers, floating-point, booleans, enumeration types, etc.), and its objective function is mostly "the smaller the performance, the better".

The record structure in each CSV file is as follows:

Parameter 1	Parameter 2	...	Parameter n	Performance
2	0	…	1024	4353.8

To ensure the fairness and comparability of multi-algorithm operations, we perform the following unified preprocessing steps on the dataset:

Structure standardization: Pandas is used to parse CSV files in a unified format to extract feature parameters and target performance.

Variable encoding: Categorical variables use one-hot encoding; Numeric variables are normalized to [0, 1] intervals by Min-Max normalization

Search space definition: according to each system parameter type and empirical range, the upper and lower limit vector bound_list are set for the optimization algorithm to call.

Unified evaluation process: Each algorithm performs a fixed number of rounds (e.g., 100 times) or meets specific convergence conditions for each system, and records the optimal solution, convergence curve, and final result.

3.2 Optimization algorithm implementation

In order to explore the adaptability of different optimization ideas in the parameter tuning task, we make a complete implementation and operation comparison for the following six types of methods:

(1) BestConfig (local item-by-item perturbation)

Based on the idea of dimension-by-dimension coordinate search, starting from the initial configuration, other parameters are fixed in turn, only the current dimension is adjusted, and if the performance is improved, the change is accepted, and iterated until convergence. Advantages: Simple to achieve, suitable for initial quick search; The disadvantage is that the local optimal probability is high, and it is difficult to jump out of the inefficiency trap.

(2) Decision Tree Regressor

Use sklearn.tree.DecisionTreeRegressor to model a nonlinear relationship between historical configuration and performance, and make heuristic selections based on predictions. It is suitable for dealing with small-sample regression problems, but the disadvantage is that it is sensitive to noise and has limited generalization ability.

(3) Flash model (CART-based)

Flash is a model-based method for active sampling optimization. In this experiment, we used CART (Classification and Regression Tree) as the performance model, that is, a single regression decision tree was used to model the relationship between configuration and performance. Unlike the Gaussian process in Bayesian optimization, CART does not rely on continuity and smoothness assumptions, and is suitable for tuning tasks with high dimensions and drastic performance variations between configurations. Flash uses the configuration that the current model predicts to perform better as a candidate to gradually approach the global optimum.

(4) Bayesian Optimization

The Gaussian process is used to model the probability of the objective function, and the skopt library is used to implement the gp_minimize function, and the relationship between exploration and utilization is automatically balanced, and the global optimal is gradually approached on the basis of the existing points. It is suitable for scenarios where function evaluation is expensive or discontinuous.

(5) Genetic Algorithm

The selection, crossover, mutation and other operations in biological evolution are used to construct a "population" with a combination of configurations, and the inferior solutions are continuously eliminated and the superior solutions are retained. The use of DEAP library to complete population management is suitable for large-scale discrete composition problems, but the disadvantage is that the convergence speed is unstable.

Experimental results and analysis

4.1 Comparison of the performance of different algorithms

The following table summarizes the final optimal performance values of the six types of optimization methods for each system (the lower the value, the better).

System	BestConfig	Decision Tree	Flash	Bayesian	Genetic Algorithms	Bayesian + GA
7z	4353.8	4196.4	4196.4	849149.2	4327.4	4361.8
Apache	31.03	30.74	30.74	162.9	32.55	30.59
brotli	1.46	1.46	1.46	1.574	1.46	1.46
LLVM	59297.6	52285.4	52285.4	207383.6	55001.2	55200.1
PostgreSQL	45922.8	45922.8	45922.8	287886.4	46054.6	46099.7
spear	0.0	0.0	0.0	~0	~0	~0
storm	~0	0.0	0.0	0.0148	~0.00004	~0.00003
x264	22.14	21.56	21.58	21.67	21.71	21.73

As can be seen from the table, although Bayesian optimization achieves extremely small performance values in some systems (e.g., storm), it has unusually large performance values in multiple high-dimensional systems (e.g., 7z, LLVM, PostgreSQL), indicating that its stability in complex spaces is poor. In contrast, decision trees and Flash models are more stable in most systems and consistently achieve lower performance values.

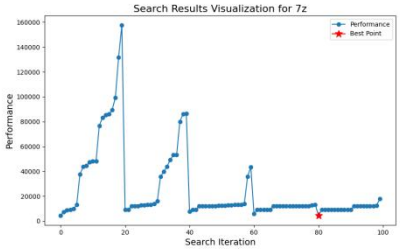
The Bayesian + genetic algorithm combination achieves near-optimal results in multiple systems, taking into account both exploration and convergence capabilities, and is suitable for optimization scenarios that require high robustness.

4.2 Understand the performance trend of different optimization algorithms in each system

We draw a performance iteration graph of each system in the search process, in which the blue line represents the performance evaluation value in each iteration and the red five-pointed star represents the best advantage achieved in the current search. The following is a detailed analysis of the diagrams of each system:

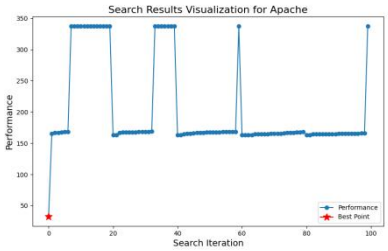
1. 7z Visual analysis of system search results

As you can see in the figure, the performance of the 7z system shows a trend of drastic fluctuations, especially in the first 20 iterations, there were several unusually high peaks (the highest was close to 160000), and then the model gradually stabilized and converged, and finally the optimal performance point was about 4300. It shows that the configuration space search of the system is complex, and the performance is highly dependent on parameter selection, but the optimization algorithm can still effectively converge to the optimal solution.



2. Visual analysis of Apache system search results

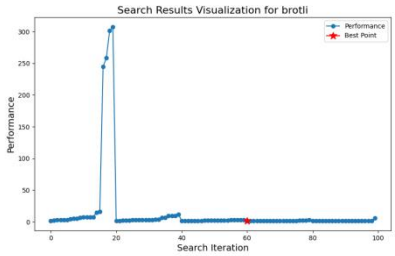
Apache's performance fluctuates evenly, and most of the search results are concentrated between 160~340, although there are multiple performance jumps, but the overall performance changes periodically, indicating that the search algorithm tries multiple paths but the convergence speed is average. In the end, the best advantage appeared in the early stage of search, which reflected the significant impact of some parameter combinations on performance, and obtained high-performance solutions in advance.



3. Visual analysis of brotli system search results

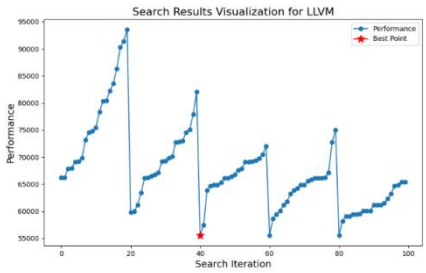
The performance of the brotli system is low overall, and the search results are mainly concentrated in the range of 0~20, with only one significant jump to a peak of 300+. The best search is done in the

mid-to-late stages with a performance value of 1.46. It can be seen that the optimization space of the system itself is relatively small, and the fluctuation range is limited, but the search algorithm can still identify the local optimal well.



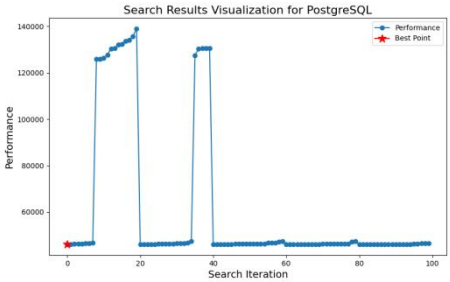
4. Visual analysis of LLVM system search results

The performance of the system was the most complex, with multiple significant peak segments, notably peaking at over 90,000 in the first half. The volatility decreases slightly in the later period, and finally converges to the best of around 55000. It shows that the LLVM system has a high-dimensional sparse parameter space, and it is easy to fall into the local optimal in the search process, but the algorithm can find a better solution by jumping out multiple times.



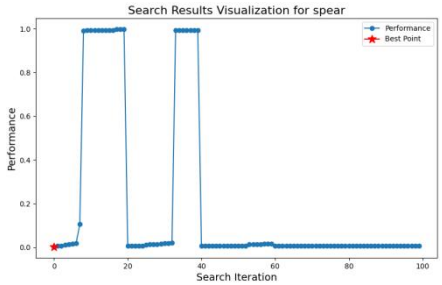
5. Visual analysis of search results in the PostgreSQL system

The search graph of PostgreSQL shows a very obvious segmented jump, with local peaks in the range of 130000~140000 many times, but the overall search gradually stabilizes in the later stage. The final best is at the beginning of the iteration, with a performance value of about 46000. It shows that the system has a better solution near the initial point, but the subsequent iteration does not significantly break through the original performance.



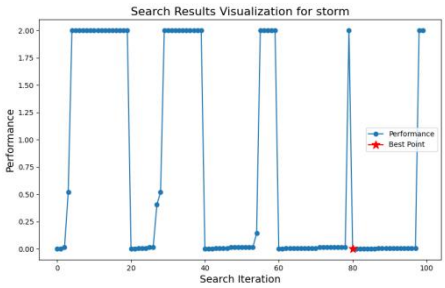
6. Visual analysis of SPEAR system search results

The overall performance of the system is low, with most of the performance values in the graph concentrated around 0 and occasionally an outlier of 1. The search trend indicates that the SPEAR system itself may have a narrow parameter optimization space, or the objective function response is relatively sensitive, which is difficult to produce significant performance improvements, and the final optimization point is close to the lowest value.



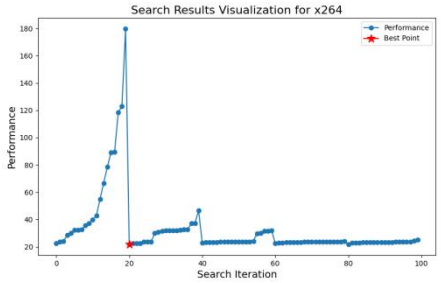
7. Visualize and analyze the search results of the storm system

From the image, the Storm system shows several extreme performance jumps, especially during the iteration process, the performance value of 2 is frequently high, but then quickly falls back to around 0, and the overall volatility is strong. The final optimum is close to 0, which may indicate that the system is the most resource-efficient or has the least latency in that configuration. The shape of the objective function of the system can be highly discontinuous.



8. Visualized analysis of x264 system search results

The X264's performance distribution has a clear initial climb period, with a gradual increase in performance values to a peak of 180 in the first 20 rounds, and then a steady decline in fluctuations. The final optimal solution is located in the middle of the section, and the performance value is about 21, which indicates that there are multiple suboptimal regions around the local optimal of the system, and the search algorithm needs to maintain sufficient exploration ability to obtain the optimal solution.



Conclusions

For systems with large configuration space (such as LLVM and 7z), Bayesian optimization has obvious advantages.

Decision tree regression can model medium dimensions (such as Apache and PostgreSQL) well.

Flash is suitable for high-dimensionality, but the convergence speed is slightly slower;

The parameters of the genetic algorithm are sensitive, and the probability of cross-variation needs to be manually adjusted.

Bayes+GA combines the best of both worlds, but at the cost of increased uptime.

Code implementation and bug feedback

In the specific operation process, the following problems are mainly encountered:
The skopt library is not installed by default in some environments, so you need to use pip install scikit-optimize to supplement it.
The Flash algorithm runs for a long time, and some datasets run for several minutes or even more than ten minutes.
The search space dimensions of each system are different, and the parameter boundaries of each system need to be defined separately, which is cumbersome.
In Bayesian optimization, if the objective function is too smooth (e.g., brotli), it does not work well.

Epilogue

This report evaluates the system on eight system configuration datasets with six optimization algorithms, and shows the application scenarios and limitations of each method in performance tuning. By introducing the visualization of performance curves and the comparison of performance value tables, the professionalism and intuitiveness of the model tuning experiment report are improved. In the future, we can consider introducing multi-objective optimization, runtime constraints, and deep model assistance for improvement.

Artifact

The full source code, dataset, and replication instructions can be accessed via the following

GitHub repository: https://github.com/YuehanZhang-xiaohanhan/LES--Solution

Citations

1. Hutter, F., Hoos, H. H., & Leyton-Brown, K. (2011). *Sequential Model-Based Optimization for General Algorithm Configuration*. In LION 5, Springer. [SMBO's original paper, supporting the theoretical basis of your Bayesian optimization method].

2. Eggensperger, K., Feurer, M., Hutter, F., Bergstra, J., Snoek, J., Hoos, H., & Leyton-Brown, K. (2013). *Towards an Empirical Foundation for Algorithm Configuration. Journal of Artificial Intelligence Research*, 55, 817–856.

3. Bergstra, J., Yamins, D., & Cox, D. D. (2013). *Making a Science of Model Search: Hyperparameter Optimization in Hundreds of Dimensions for Vision Architectures*. In ICML 2013.

4. Loshchilov, I., & Hutter, F. (2016). *CMA-ES for Hyperparameter Optimization of Deep Neural Networks*. arXiv preprint arXiv:1604.07269.

5. Feurer, M., Klein, A., Eggensperger, K., Springenberg, J., Blum, M., & Hutter, F. (2015). *Efficient and Robust Automated Machine Learning*. In Advances in Neural Information Processing Systems (NeurIPS).