School of Computer Science

UNIVERSITY OF BIRMINGHAM

Intelligent Software Engineering
Lab 3: Configuration Performance Tuning

Dr. Tao Chen

This is lab3 for the module, which can be chosen as the problem for the coursework. If you need specific support in terms of programming, please attend one of the lab sessions and ask the TAs for help.

The Content

For Lab3, you will build a **baseline tool** using Random Search to automatically search for better performance under a given budget, i.e., a certain number of measurements:

$$\begin{aligned} &\arg\min f(\mathbf{x}) \text{ or } \arg\max f(\mathbf{x}), \\ &\text{s.t. } r \leq R, \end{aligned} \tag{1}$$

where $\mathbf{x} = (x_1, x_2, \ldots, x_n)$ is a configuration with the values of n options (e.g., x_n) in search space $\mathcal{X}$. f represents the performance attribute of the target system. r and R respectively denote the cost consumption of configuration measurements and the budget of measurements allowed.

Random Search is fairly simple to be implemented: simply try to randomly sample n configurations, and return the best one found.

The Datasets of Systems

This lab covers the measured datasets for a range of widely studied configurable systems. The selection aligns with this study to ensure consistency and comprehensiveness. Specifically, these systems are carefully chosen to span a variety of application domains, performance objectives, and programming languages, including both Java and C/C++, thereby providing a solid foundation for our investigation into systems with diverse characteristics, as shown in Table **??**. Note that some performance attributes are to be minimised, e.g., latency, but some are to be maximised, e.g., throughput. But they can be converted using the additive inverse.

Table 1: Subject system characteristics. **#O:** No. of options; **#C:** No. of configurations.

System	Language	Domain	Perf.	#O	#C
7Z	C++	Compressor	Runtime & Energy	8	68640
BROTLI	C	Compressor	Runtime & Energy	2	180
LLVM	C	Compiler	Runtime & Energy	16	65536
POSTGRESQL	C	Database	Runtime & Energy	8	864
X264	C	Video Encoder	Runtime & Energy	10	4608
APACHE	C	Web Server	Runtime & Energy	8	640
SPEAR	C	SAT Solver	Runtime	14	16384
STORM	Clojure	Streaming Process	Latency	12	1557

Each system features a distinct configuration space, covering different option types (e.g., integers, boolean, and enumerates) and dimensions.

The link and details of the datasets can be accessed here: https://github.com/ideas-labo/ISE/tree/main/lab3.

How to use the Datasets?

The normal case for configuration performance tuning is that we need to measure how well a configuration performs by running it under the target system. However, since deploying a system can be highly complicated, in this module, we have simplified this by using datasets instead. That is, when you want to know how well a configuration performs, you do not need to feed it into a running system but simply query the dataset to find the corresponding performance under that said configuration. This should resemble the practical case as the datasets are measured from real systems.

Therefore, the procedure would be that you build a baseline tool, i.e., using Random Search; as part of the tuning/search when there is a need to measure a randomly generated configuration (you need to identify the range of possible values allowed for an option from the dataset), you simply query the dataset if check its performance. When the budget (e.g., 100 measurements) is exhausted, you return the configuration with the best performance. Note that the budget should not be greater than the total number of configurations, otherwise, we are doing a full traversal, please see the next point.

Note that you should only treat such a dataset as a system without having to deploy and run an actual one. **That is, all the information of correlations between the configuration and performance that you can get from the dataset needs to consume measurements of the budget.**

Can I Traverse the Whole Dataset to find the Optimal Configuration?

Ideally no. This is because although in our simplified problem, we run on the datasets, remember that in the real world, a single measurement under a real system can be costly. Therefore, even though it is cheap to traverse the dataset, it could be too expensive to do so in practice. Therefore, the most ideal is that the solution should be able to find the best possible configuration with the least number of measurements.

What if a Configuration I Want to Measure is not in the Dataset?

Since there might be dependencies in a system and the measured datasets might not be exhaustive, there is a chance that you wish to evaluate a configuration during your search, but it turns out to be no record in the dataset. However, there is no way for us to know whether this is due to the configuration being an invalid one or simply because the dataset has not captured its measurement. As such, in this case, we would simply consider that configuration as invalid. In your turning process, the most simple way would be to give the worst possible value for it, e.g., if you are minimising then for those invalid configurations you merely want to give it the maximum possible value of a Double. Bear in mind that invalid configurations are often easily known with our dataset, therefore, in this case, they should not consume the measurement budget.

The Metrics

The metric for testing the tools is very straightforward: simply compare and evaluate the better or worse of the performance found by the tool!

The Statistical Test

If there is nothing to compare, then you might not need the statistical test. The link to an existing statistical test library can be found here: `https://docs.scipy.org/doc/scipy/reference/stats.html`. You will find implementations of the tests mentioned in the module and those that we have not talked about.