Dear Reviewers,

We hereby submit the artifact documentation for the accepted paper entitled "Multi-Objectivizing Software Configuration Tuning" in ESEC/FSE 2021. The artifacts contain the data and code we used to run the experiments. We wish to apply for the badges **Artifacts Available**, **Artifacts Functional** and **Artifacts Reusable**.

1 Artifacts Available

The source code and data reported in the paper can be publicly accessed at https://github.com/taochen/mmo-fse-2021/. The `data` folder contains all the raw data as reported in the paper; most of the structures are self-explained but we wish to highlight the following:

- The data under the folder `1.0-0.0` and `0.0-1.0` are for the single-objective optimizers. The former uses O1 as the target performance objective while the latter uses O2 as the target. The data under other folders named by the subject systems are for all MMO instances (and the examined weights) and PMO.
- For those data of MMO instances and PMO, the folder `0` and `1` denote using uses O1 and O2 as the target performance objective, respectively.
- In the lowest-level folder where the data is stored (i.e., the `sas` folder), `SolutionSet.rtf` contains the results over all repeated runs; `SolutionSetWithMeasurement.rtf` records the results over different numbers of measurements.

The `code` folder contains all the information about the source code, as well as an executable jar file in the `executable` folder .

2 Artifacts Evaluated

To run the experiments, one can download the `mmo-experiments.jar` from the aforementioned repository (under the `executable` folder). Since the artifacts were written in Java, we assume that the JDK/JRE has already been installed. Next, one can run the code using ``java -jar mmo-experiments.jar [subject] [runs]'', where `[subject]` and `[runs]` denote the subject software system and the number of repeated run (this is an integer and 30 is the default if it is not specified), respectively. The keyword for the systems/environments used in the paper are: {`trimesh, x264, storm-wc, storm-rs, storm-sol, dnn-dsr, dnn-coffee, LSTM`}. For example, running ``java -jar mmo-experiments.jar trimesh'' would execute experiments on the `trimesh` software for 30 repeated runs.

For each software system, the experiment consists of the runs for MMO instances with all seven weight values, PMO and the four state-of-the-art single-objective optimizers. All the outputs would be stored in the `results` folder at the same directory as the executable jar file.

Note that, since wrapping the actual software system into a VM image resulted in a surprisingly large file (50GB+), we could not upload them. Further, their settings/installations are highly complicated and hence requires a lengthy documentation, which would not meet the required setup time of 30 minutes. Therefore, in this artifact, we use a simplified approach to emulate the experiments used in the paper by querying from the performance data we collected. This, as we have already discussed in the paper, would not change the conclusions drawn. However, to ensure realism, we have set them in a way that each run would take approximately 2 hours to complete.

We thank you for reviewing our artifacts.

Sincerely yours,

Tao Chen and Miqing Li