Artefact Evaluation for

“Advanced Consistency Restoration with Higher-Order Short-Cut Rules” by Lars Fritsche, Jens Kosiol, Alexander Lauer, Adrian Möller and Andy Schürr

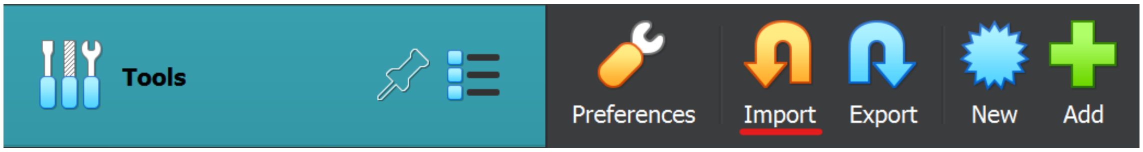
In the following, you will find all the steps to reproduce our results and run the evaluation on your own system. For this purpose, we created a virtual machine to ease the installation. Note that, compared with our previous contribution, a valid Gurobi license is needed.

VM Setup

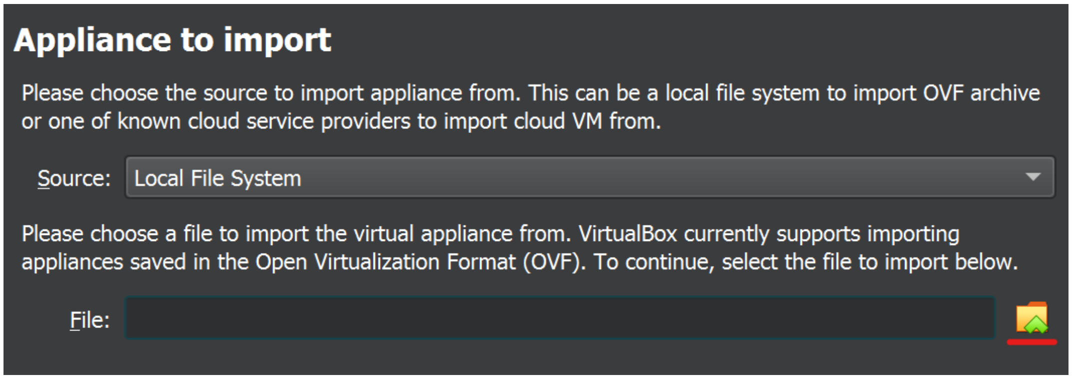
1. Please download the VM from Zenodo (10.5281/zenodo.7728966)

2. Install VirtualBox from here

3. Start VirtualBox and press **Import**



4. A new window opens. There, choose “Source: Local File System” and press the yellow file to the right to open the system’s file chooser



Appliance to import

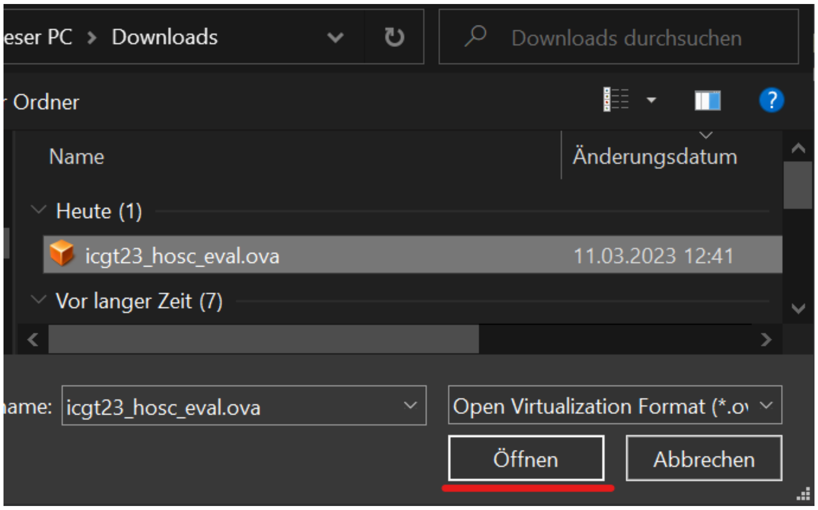
Please choose the source to import appliance from. This can be a local file system to import OVF archive or one of known cloud service providers to import cloud VM from.

Source: Local File System

Please choose a file to import the virtual appliance from. VirtualBox currently supports importing appliances saved in the Open Virtualization Format (OVF). To continue, select the file to import below.

File:

5. Navigate to the downloaded OVA-file and press **Open**



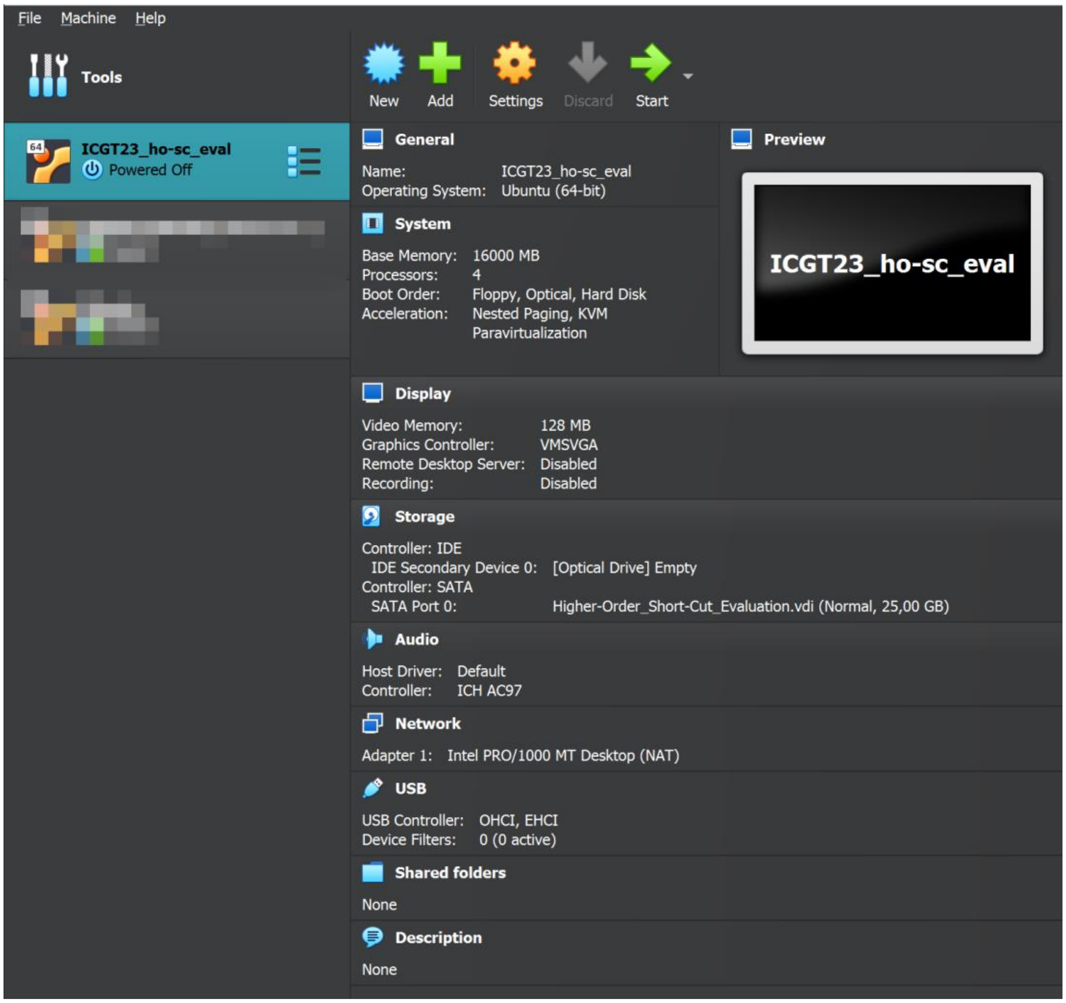
icgt23_hosc_eval.ova 11.03.2023 12:41

ame: icgt23_hosc_eval.ova

Open Virtualization Format (*.ov

6. Press **Next**. You should now see an overview of the virtual machine's parameters. Here, press **Finish**.

7. You should now see the VM in the upper left corner and a description of the configured system:



ICGT23_ho-sc_eval
Powered Off

General

Name: ICGT23_ho-sc_eval
Operating System: Ubuntu (64-bit)

System

Base Memory: 16000 MB
Processors: 4
Boot Order: Floppy, Optical, Hard Disk
Acceleration: Nested Paging, KVM
Paravirtualization

Display

Video Memory: 128 MB
Graphics Controller: VMSVGA
Remote Desktop Server: Disabled
Recording: Disabled

Storage

Controller: IDE
IDE Secondary Device 0: [Optical Drive] Empty
Controller: SATA
SATA Port 0: Higher-Order_Short-Cut_Evaluation.vdi (Normal, 25,00 GB)

Audio

Host Driver: Default
Controller: ICH AC97

Network

Adapter 1: Intel PRO/1000 MT Desktop (NAT)

USB

USB Controller: OHCI, EHCI
Device Filters: 0 (0 active)

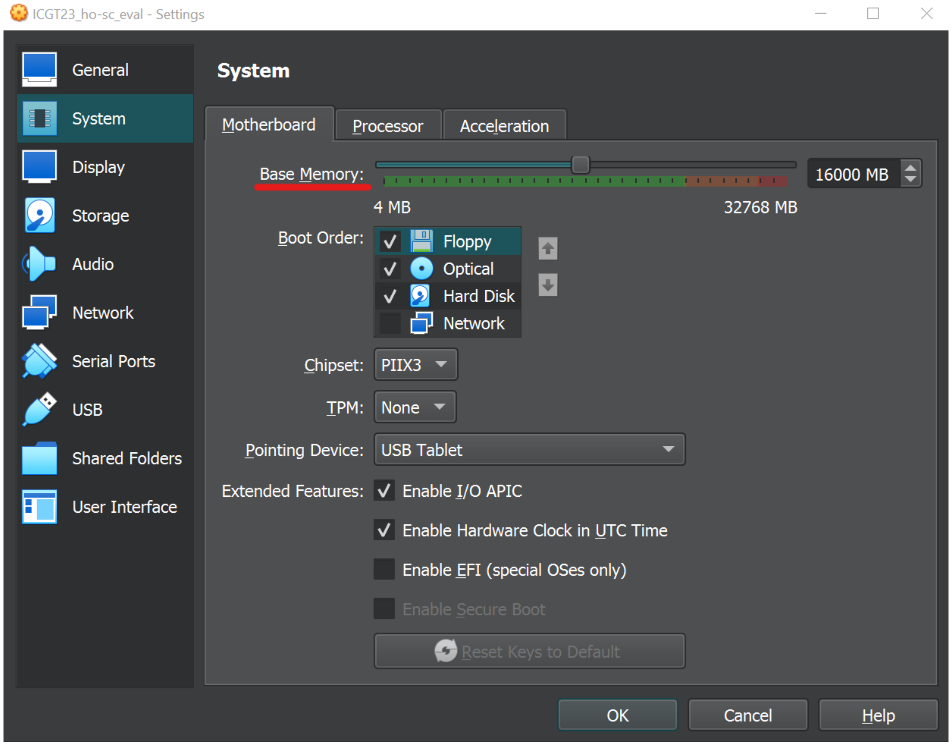
Shared folders

None

Description

None

8. Press **Settings** and go to **System** and then **Motherboard**. Here you can configure the base memory. For optimal performance, you should choose this value close to the red interval.



System

Base Memory: 16000 MB

Boot Order: ☑ Floppy ☑ Optical ☑ Hard Disk □ Network

Chipset: PIIX3

TPM: None

Pointing Device: USB Tablet

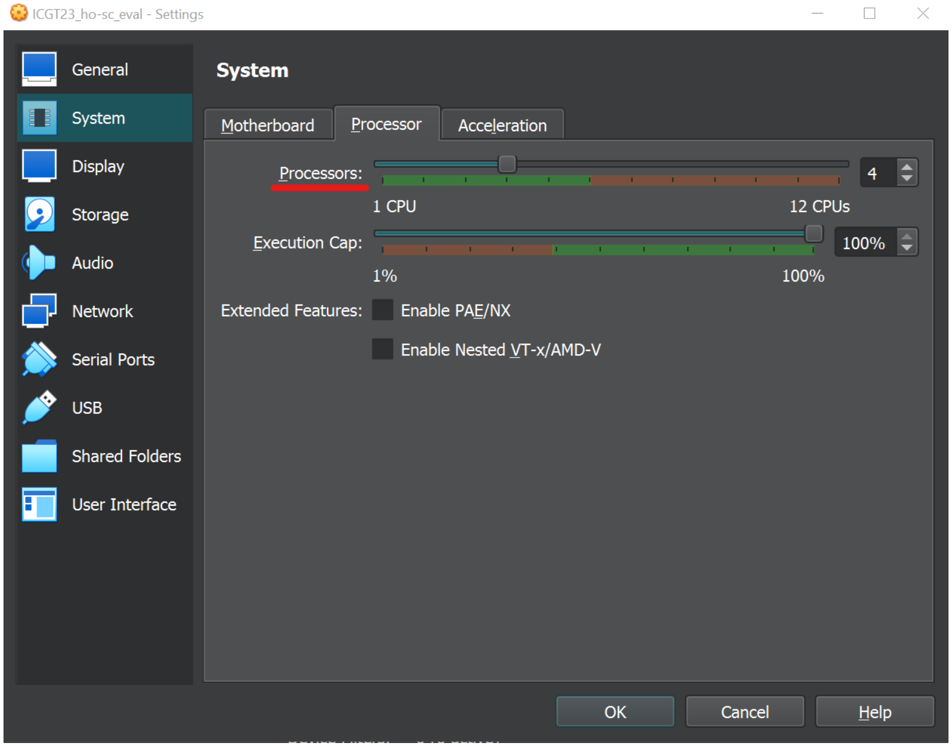
Extended Features: ☑ Enable I/O APIC

☑ Enable Hardware Clock in UTC Time

□ Enable EFI (special OSes only)

□ Enable Secure Boot

Then go to **Processor** and choose as many CPU cores as you can spare. eMoflon relies on a pattern matching engine that highly benefits from an increased number of cores.



System

Processors: 4

Execution Cap: 100%

Extended Features: □ Enable PAE/NX

□ Enable Nested VT-x/AMD-V

9. Select the VM and press Start. When asked for a username and password, type in **vagrant**

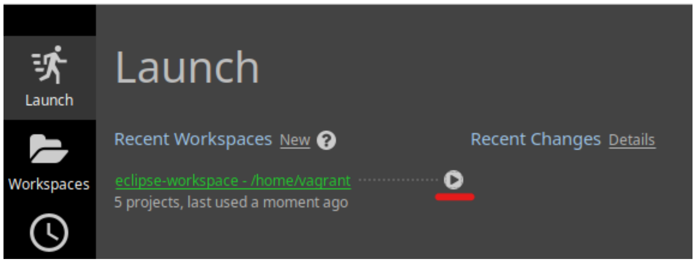
Eclipse Setup

In comparison to our previous submission, a valid Gurobi license is needed to execute the evaluation. You can get a license and download it from here. If possible, please pick and install 9.5.2. However, it should also work with newer versions.

On your desktop you will find a folder, a PDF file, and a link:

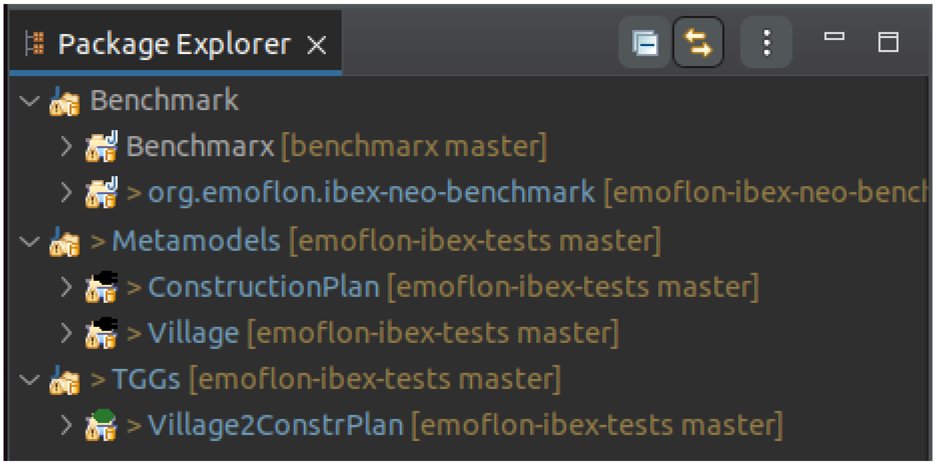
- The folder **paper-eval-results** contains the original results used in our paper.
- The **ICGT23-Setup.pdf** file is a copy of this document, but without the first chapter "VM Setup"
- The link **eMoflon::IBeX Eclipse** points to a pre-configured Eclipse version with eMoflon installed.

1. Double-click on the **eMoflon::IBeX Eclipse** link to run eMoflon
2. A quick workspace configuration follows, which you can click through
3. When prompted for the workspace location, make sure that it matches the following one and click the **launch-button**



eclipse-workspace - /home/vagrant
5 projects, last used a moment ago

Now, the Eclipse workspace should be up and running. On the left you should be able to see the three working sets **Benchmark**, **Metamodels,** and **TGGs**.



```
Benchmark
  Benchmarx [benchmarx master]
  > org.emoflon.ibex-neo-benchmark [emoflon-ibex-neo-bench
> Metamodels [emoflon-ibex-tests master]
  > ConstructionPlan [emoflon-ibex-tests master]
  > Village [emoflon-ibex-tests master]
> TGGs [emoflon-ibex-tests master]
  > Village2ConstrPlan [emoflon-ibex-tests master]
```

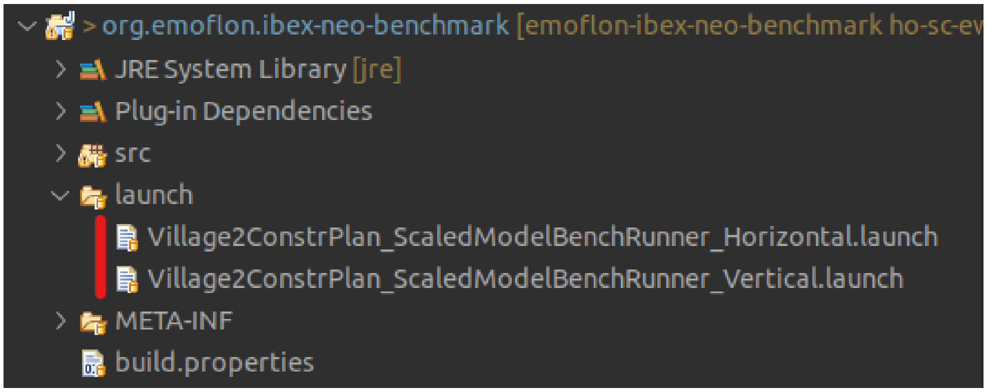
Metamodels contains ecore specifications of the used metamodels from our paper.
TGGs contains our example TGG project.
Benchmark contains the benchmark environment for performance evaluation.

Benchmarking

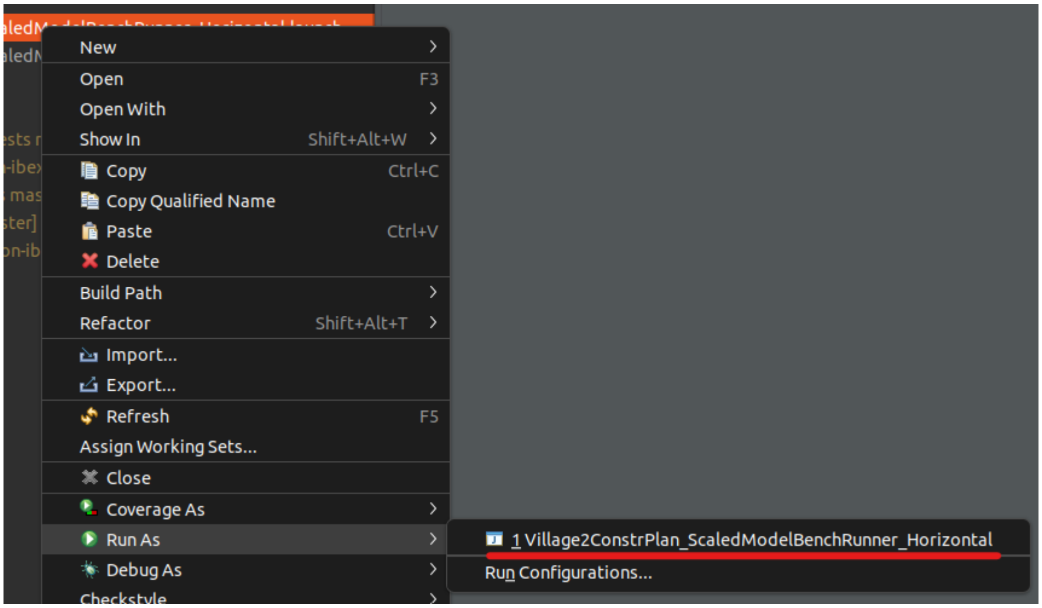
The benchmark can be run in two different modes:

- Horizontal: Increasing the model size by adding more house rows
- Vertical: Increasing the model size by adding more houses to one single row
- VerticalMultiChange: Increasing the number of changed houses.

You can find a **.launch**-file for each modes in the **launch** folder of the **org.emoflon.ibex-neo-benchmark** project, which starts a series of benchmarks in the respective mode.


> org.emoflon.ibex-neo-benchmark [emoflon-ibex-neo-benchmark ho-sc-ev
JRE System Library [jre]
Plug-in Dependencies
src
launch
Village2ConstrPlan_ScaledModelBenchRunner_Horizontal.launch
Village2ConstrPlan_ScaledModelBenchRunner_Vertical.launch
META-INF
build.properties

To run a **.launch**-file, right-click on it and choose **Run As**.

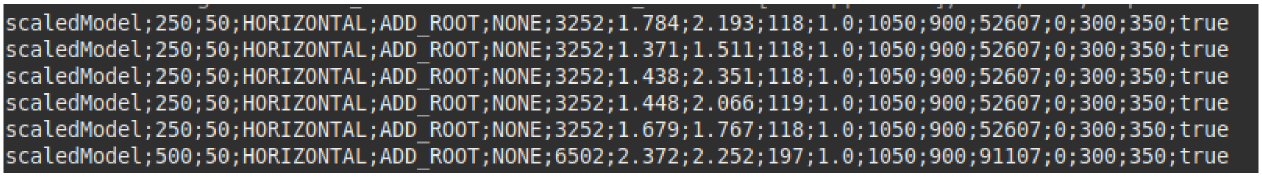


A benchmark series comprises three different scenarios (**ADD_ROOT**, **REMOVE_ROOT**, **MOVE_ROW**). Each one is run with **no repair** framework, with the **legacy repair** framework, and with the **higher-order short-cut repair** framework.

For each combination, the **model size** is varied.
In horizontal mode, this is combined with a variation of the number of **model changes**.

The **VerticalMultiChange** is special in that regard because it executes another scenario **CHANGE_TYPE**, where an increasing number of houses in a row is changed from Villa to Cube.

Executing the benchmarks as described above, after some time, you should see outputs like this:



```
scaledModel;250;50;HORIZONTAL;ADD_ROOT;NONE;3252;1.784;2.193;118;1.0;1050;900;52607;0;300;350;true
scaledModel;250;50;HORIZONTAL;ADD_ROOT;NONE;3252;1.371;1.511;118;1.0;1050;900;52607;0;300;350;true
scaledModel;250;50;HORIZONTAL;ADD_ROOT;NONE;3252;1.438;2.351;118;1.0;1050;900;52607;0;300;350;true
scaledModel;250;50;HORIZONTAL;ADD_ROOT;NONE;3252;1.448;2.066;119;1.0;1050;900;52607;0;300;350;true
scaledModel;250;50;HORIZONTAL;ADD_ROOT;NONE;3252;1.679;1.767;118;1.0;1050;900;52607;0;300;350;true
scaledModel;500;50;HORIZONTAL;ADD_ROOT;NONE;6502;2.372;2.252;197;1.0;1050;900;91107;0;300;350;true
```

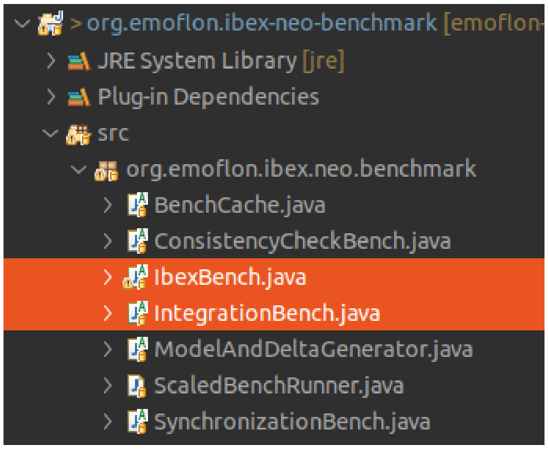
When the measurements are finished, the runner calculates average and median values for each run based on this output and appends them to the console.

model_scale: linear scaling factor for the generated model
num_of_changes: number of generated changes
elts: number of generated elements on source and target side
avg_init: average initialization time including initial pattern matching of generated model
median_init: median initialization time including initial pattern matching of generated model
avg_resolve: average time to propagate changes including repair operations
median_resolve: median time to propagate changes including repair operations
avg_ram: average memory used by the JVM
median_ram: median memory used by the JVM

In addition, we collected the number of created elements, deleted elements, found matches, repaired matches, revoked matches, and applied matches.

Implementation

The main benchmark implementation can be found in **IbexBench.java** and **IntegrationBench.java**.



> org.emoflon.ibex-neo-benchmark [emoflon-
JRE System Library [jre]
Plug-in Dependencies
src
org.emoflon.ibex.neo.benchmark
BenchCache.java
ConsistencyCheckBench.java
IbexBench.java
IntegrationBench.java
ModelAndDeltaGenerator.java
ScaledBenchRunner.java
SynchronizationBench.java

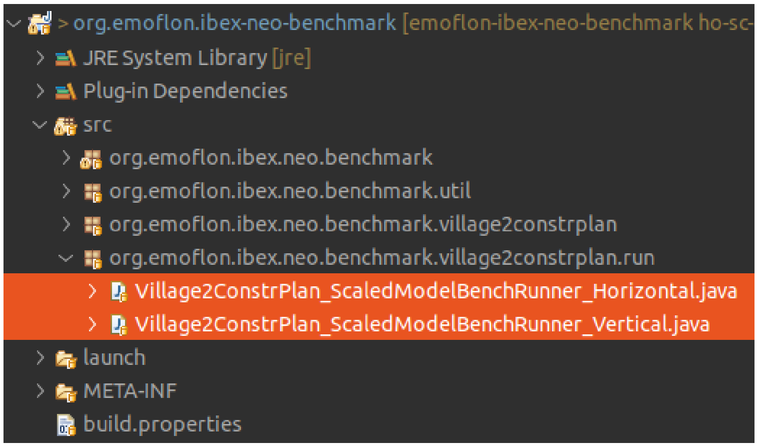
IbexBench::genAndBench is called, which first initializes an eMoflon stub and a model and delta generator, which generates the model and all deltas.

In **IntegrationBench::applyDeltaAndRun**, after initiating a dry run of eMoflon, the deltas are applied and eMoflon is run twice to propagate the changes. Finally, the benchmark results are collected and printed.

Troubleshooting

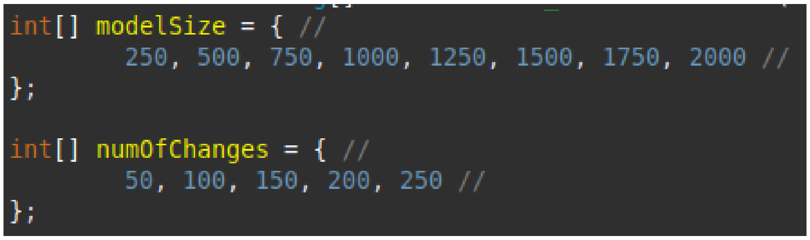
Note that our evaluation was performed on a workstation with 64GB RAM and an AMD Ryzen 9 3900x. eMoflon relies on incremental pattern matching, which at this moment still consumes a lot of memory. However, the currently employed technology highly benefits from multi core setups. However, this means that if your system has lower specs than this, you should alter the configurations of the benchmark.

For this, open a runner class:



```
> org.emoflon.ibex-neo-benchmark [emoflon-ibex-neo-benchmark ho-sc-
  JRE System Library [jre]
  Plug-in Dependencies
  src
    org.emoflon.ibex.neo.benchmark
    org.emoflon.ibex.neo.benchmark.util
    org.emoflon.ibex.neo.benchmark.village2constrplan
    org.emoflon.ibex.neo.benchmark.village2constrplan.run
      Village2ConstrPlan_ScaledModelBenchRunner_Horizontal.java
      Village2ConstrPlan_ScaledModelBenchRunner_Vertical.java
  launch
  META-INF
  build.properties
```

Adapt the **modelSize** and **numOfChanges** variables to fit your system:

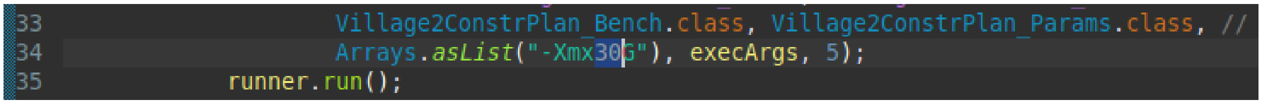


```java
int[] modelSize = { //
        250, 500, 750, 1000, 1250, 1500, 1750, 2000 //
};

int[] numOfChanges = { //
        50, 100, 150, 200, 250 //
};
```

If your system has less memory and also less cores, consider choosing lower values for **modelSize**.
Note that due to the implementation the values for **numOfChanges** should be equal or less than those of **modelSize**.

Problems with JVM memory:
If you run into errors while executing the benchmarks, try to increase the JVM memory limit. It can be found in the runners at line 34 (current value 30 GB):

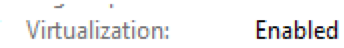


```java
33                      Village2ConstrPlan_Bench.class, Village2ConstrPlan_Params.class, //
34                      Arrays.asList("-Xmx30G"), execArgs, 5);
35              runner.run();
```

Problems with the VM:

If there are problems when starting the VM, please check if there are any warnings from VirtualBox. We recommend reading them thoroughly and solving them as recommended by the tool.

If you get an error stating that virtualization has been deactivated for your system, you may have to activate this feature in your BIOS.
If you are using Windows, you can find out if virtualization is enabled via accessing the **Task Manager -> Performance**.



Virtualization: Enabled

Errors during Execution:

If the log states that there were too many errors during measurement, please select all projects in your workspace and refresh them. This should make Eclipse find all missing files again.

If the log or an error signal that the solver Gurobi was not found. You should try restarting Eclipse. Else, please check that the Gurobi_Home variable is set properly. Also make sure that you license is registered according to the instructions from the Gurobi-Website.