

# Hydra Manual

Beta Lab

Department of Computer Science

University of British Columbia

Vancouver, BC V6T 1Z4, Canada

Author: Chris Cameron

Collaborators: Steve Ramage, Frank Hutter, Kevin Leyton-Brown, Holger Hoos

September 25, 2015

## Contents

<b>1</b>	<b>Introduction</b>	<b>2</b>
1.1	Version . . . . .	2
1.2	Contact . . . . .	2
1.3	System Requirements . . . . .	4
<b>2</b>	<b>Usage</b>	<b>4</b>
2.1	Unpacking . . . . .	4
2.2	Example Execution . . . . .	4
2.3	Getting Ready to Run . . . . .	6
2.4	Required Parameters . . . . .	6
2.4.1	SMAC options . . . . .	6
2.4.2	SMAC execution options . . . . .	7
2.4.3	SMAC . . . . .	7
2.4.4	MYSQLDB . . . . .	7
2.5	Important parameters . . . . .	8
2.6	Running Hydra . . . . .	9
2.7	Restarting Hydra . . . . .	9
2.8	Using MySQL database . . . . .	9
2.8.1	Contacting database . . . . .	9
2.8.2	Pool Parameter . . . . .	10
2.8.3	Running worker . . . . .	10
2.8.4	Useful option . . . . .	10
2.9	Zilla Portfolio Evaluation . . . . .	11
2.9.1	Log output . . . . .	12
2.10	Output . . . . .	13
2.10.1	Directory Structure . . . . .	14
2.11	Building . . . . .	14

# 1 Introduction

This document is the user manual for the Hydra software. See [5] and [7] to learn how the hydra algorithm works. In short, Hydra takes a configurable algorithm, instance set, and parameter set, and returns a set of algorithm configurations to use for portfolio-based algorithm selection. Hydra is a greedy algorithm and proceeds by first determining high-performance parameter configurations on the instance set and then iteratively finds and adds new configurations to the portfolio that improve portfolio-based selection performance.

Note the two significant changes between this implementation and the most recent Hydra publication [7].

1. We have changed our Portfolio Builder between hydra iterations to VBS (virtual best solver) from zilla (see [6]) i.e portfolio performance during configuration is now measured as the best performance of any configuration within the portfolio. We found circumstances where Hydra can prematurely stagnate when zilla is used as the portfolio builder. This is our recommended implementation for general use at this time.
2. We are using SMAC [2] as the algorithm configurator rather than ParamILS [1].

In slightly more detail, users of Hydra must provide:

- a parametric algorithm  $\mathcal{A}$  (an executable to be called from the command line),
- a description of  $\mathcal{A}$ 's parameters  $\theta_1, \dots, \theta_n$  and their domains  $\Theta_1, \dots, \Theta_n$ ,
- a set of benchmark instances,  $\Pi$ , and
- the objective function with which to measure and aggregate algorithm performance results.

Hydra starts with an initially empty portfolio  $P := \{\}$  and executes many SMAC runs with algorithm  $\mathcal{A}$ . SMAC executes algorithm  $\mathcal{A}$  with different *parameter configurations* (combinations of parameters  $\langle \theta_1, \dots, \theta_n \rangle \in \Theta_1 \times \dots \times \Theta_n$ , on instances  $\pi \in \Pi$ ), searching for the configuration  $C_i$  that yields overall best marginal improvement in performance on  $\mathcal{P}$  across the benchmark instances under the supplied objective. Hydra evaluates the incumbent configurations returned from SMAC  $\{C_1, C_2, \dots, C_n\}$  and adds the  $k$  best to the portfolio:  $P := P \cup \{C_1, C_2, \dots, C_k\}$ . Hydra then iteratively follows the same process and SMAC finds new configurations to add to the portfolio at each iteration. Hydra terminates after a predefined set of iterations or after performance stagnates. Figure 1 demonstrated this basic pipeline graphically.

## 1.1 Version

This is Hydra version 1.0-development.

## 1.2 Contact

Please direct any questions or issues concerning the software to `cchris13@cs.ubc.ca`.

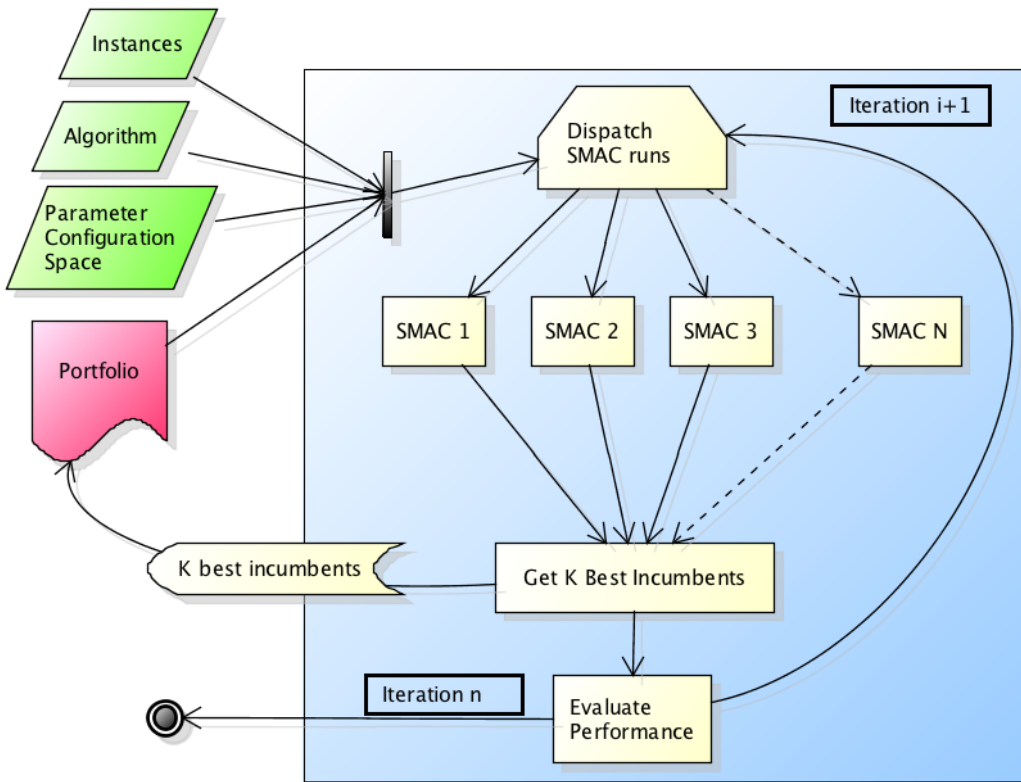


Figure 1: Basic Hydra pipeline.

## 1.3 System Requirements

Hydra only requires Java 7 or greater to run, and should work on both Windows and UNIX based operating systems. There are also a few example scenarios in `./example_scenarios` that can be run out-of-the-box that depend on either python or ruby.

## 2 Usage

### 2.1 Unpacking

Download the `hydra-1.0-development-{commit}-dist..zip` and unzip. This can be done by:

```
unzip Hydra-1.0-development-{commit}-dist.zip
```

A new folder will be created containing source code, executables, and example scenarios. From this folder, navigate to the `./bin` folder to execute hydra. From `./bin`, you can run Hydra using the simple bash script `./hydra`. On windows please use the corresponding `./hydra.bat` file. The following subsection contains instructions on how to execute a simple example scenario.

### 2.2 Example Execution

To get started, this section will walk the user through the execution of a very simple example. To execute the example locally, execute the following from the root of the folder:

```
> cd example_scenarios/saps
> ./run_saps-local.sh
```

After the java call string, you will see the following printed to stdout:

```
[INFO ] Running Hydra for 4 iterations
[INFO ] *****Hydra iteration: 0*****
[INFO ] Generating SMAC runs
[INFO ] Executing 2 SMAC runs
```

Next, 2 algorithm configurator (SMAC) runs will be ran, which find a best configuration to be added to the portfolio. For each SMAC run, you will see logging to stdout similar to:

```
SMAC started at: 18-Jun-2015 10:16:24 PM. Minimizing mean runtime.
[INFO ] First incumbent: config 1 (internal ID: 0xA31E1), with mean runtime: 0.03;
       estimate based on 1 runs.
[INFO ] Sample call for new incumbent config 1 (internal ID: 0xA31E1):
.....
.....
.....
[INFO ] SMAC Execution Completed because: wall-clock time limit (53.79 s)
       has been reached.
```

After the SMAC executions complete, the best 1 out of 2 configurations will be added to the portfolio based on the training performance. The user can choose how many configuration to run and how many to add to their portfolio with command-line parameters described in the subsequent subsections.

```
[INFO ] SMAC runs complete. Processing run results...
[INFO ] Finding best 1 incumbents out 2 SMAC runs
[INFO ] Incumbents sorted by training performance on runs executed by SMAC...
[INFO ] Performance: 0.06941176470588237, Configuration:
-alpha '1.35' -ps '0.089' -rho '0.454' -wp '0.008'
[INFO ] Performance: 3.311111111111112, Configuration:
-alpha '1.189' -ps '0.1' -rho '0.5' -wp '0.03'
[INFO ] Found best incumbents.
[INFO ] Setting Best Single configuration. Only occurs in first Hydra iteration
[INFO ] Best single configuration:
-alpha '1.35' -ps '0.089' -rho '0.454' -wp '0.008'
[INFO ] Adding new incumbent configurations...
[INFO ] Adding configuration: -alpha '1.35' -ps '0.089' -rho '0.454' -wp '0.008'
```

Test performance will then be calculated based on a set of provided test instances. The user has the option to use VBS or zilla to test portfolio performance. The test performance of the portfolio will be printed once the test instances have been solved.

```
[INFO ] Getting portfolio performance
[INFO ] Test performance of hydra after iteration 0: 0.148
```

After the new configurations are added to the portfolio, the next iteration will begin in the same fashion with algorithm performance representing the VBS of all the configurations contained in the portfolio.

```
[INFO ] *****Hydra iteration: 1*****
.....
.....
.....
```

After 4 iterations, the hydra run terminated and prints out the final portfolio and test performance:

```
[INFO ] Writing current portfolio to:
./hydra-output/Hydra_Saps/Hydra-1/Portfolios/portfolio-3.txt
[INFO ] Best single configuration: -alpha '1.3570229625905657'
-ps '0.08955468111361481' -rho '0.4549265007328307' -wp '0.00844758766969784'
[INFO ] Final portfolio configurations
[INFO ] 1. Portfolio Configuration : -alpha '1.3570229625905657'
-ps '0.08955468111361481' -rho '0.4549265007328307' -wp '0.00844758766969784'
[INFO ] 2. Portfolio Configuration : -alpha '1.1223601730778767'
-ps '0.07888475911986773' -rho '0.8047314559241363' -wp '0.00628190839060591'
[INFO ] 3. Portfolio Configuration : -alpha '1.3778947812738795'
```

```
-ps '0.14691893797877198' -rho '0.29807307383168247' -wp '0.03949665635243472'
[INFO ] 4. Portfolio Configuration : -alpha '1.260589798073366'
-ps '0.03020086893063898' -rho '0.23304825431044507' -wp '0.0010791984714169822'
[INFO ] Hydra performance by iteration:
{0=0.148, 1=0.036000000000000004, 2=0.024, 3=0.022}
```

## 2.3 Getting Ready to Run

For a description of the parameters, you have the following options to use at the command line:

```
./hydra --help or ./hydra --help-level basic :Basic parameters for running hydra

./hydra --help-level intermediate :Includes more advanced parameters for experimental use

./hydra --help-level developer :Includes more advanced parameters that are not recommended to be altered
```

You likely won't need to deal with any parameters beyond `--help-level basic`.

## 2.4 Required Parameters

Running Hydra requires two parameters, which must be set to file paths of two different option files. One that specifies the configuration scenario for the algorithm configurator used within hydra (`smac-options`) and the other that informs Hydra how to compute the algorithm configurator executions (`smac-execution-options`).

### 2.4.1 SMAC options

SMAC (Sequential Model-based Algorithm Configuration) is the automatic algorithm configurator that the current Hydra implementation uses. You will need to provide a file containing options to run SMAC using the following parameter:

```
--smac-options <path-to-file>
```

The basic options that the file must contain are shown below.

```
--algo String to execute algorithm from the command line.

--paramfile File containing parameters of the algorithm in .pcs format.

--instance_file File containing a list of paths to problem instance files.
```

An example `smac_options` file can be found at `./example_scenarios/saps/smac_options.pcs`. See Section 4 of the SMAC manual for details on how to create a SMAC scenario file.

### 2.4.2 SMAC execution options

Because SMAC is a randomized algorithm, its performance differs across runs. Therefore Hydra increases the likelihood of finding good configurations by running many SMAC executions in each iteration and retaining configurations with best training performance. You will need to create a file informing Hydra how to run SMAC executions using the following parameter:

```
--smac-execution-options <path-to-file>
```

The basic parameter within the option file is described below. It provides the user the choice between executing SMAC locally or in a distributed fashion that can scale arbitrarily. You can leave the other parameters as default unless you are familiar with the underlying AEATK software and you need to setup a specific configuration.

```
--tae {SMAC, MYSQLDB}
```

The two choices: **SMAC** (local execution) and **MYSQLDB** (distributed execution) are described in the following subsections:

### 2.4.3 SMAC

Setting the tae parameter to **SMAC** will execute the SMAC runs sequentially on your local machine. It is recommended that you use this option when debugging your experimental setup before dispatching a large scale experiment. However, this is inefficient for large numbers of SMAC runs.

### 2.4.4 MYSQLDB

Setting the tae parameter to **MYSQLDB** gives the user a distributed computation option through MySQL for large scale experiments. To use this, you must first have a MySQL database set up. You will provide Hydra with information needed to contact your database and Hydra will use the database as a holding place for input/output of **SMAC** executions.

You will need to dispatch ‘worker’ jobs which look in the database for SMAC inputs and execute SMAC when a SMAC input is found. The distributed setup proceeds in the following steps (refer to Figure 2):

1. Hydra will store information needed to run SMAC in the database, and waits.
2. ‘worker’ jobs on your cluster grab the SMAC information when available
3. ‘worker’ jobs execute SMAC, and then put the SMAC result back in the database
4. After all SMAC runs have completed, Hydra gathers the results and continues to the next iteration.

See Section 2.8 for instructions on how to run ‘worker’ jobs on a cluster.

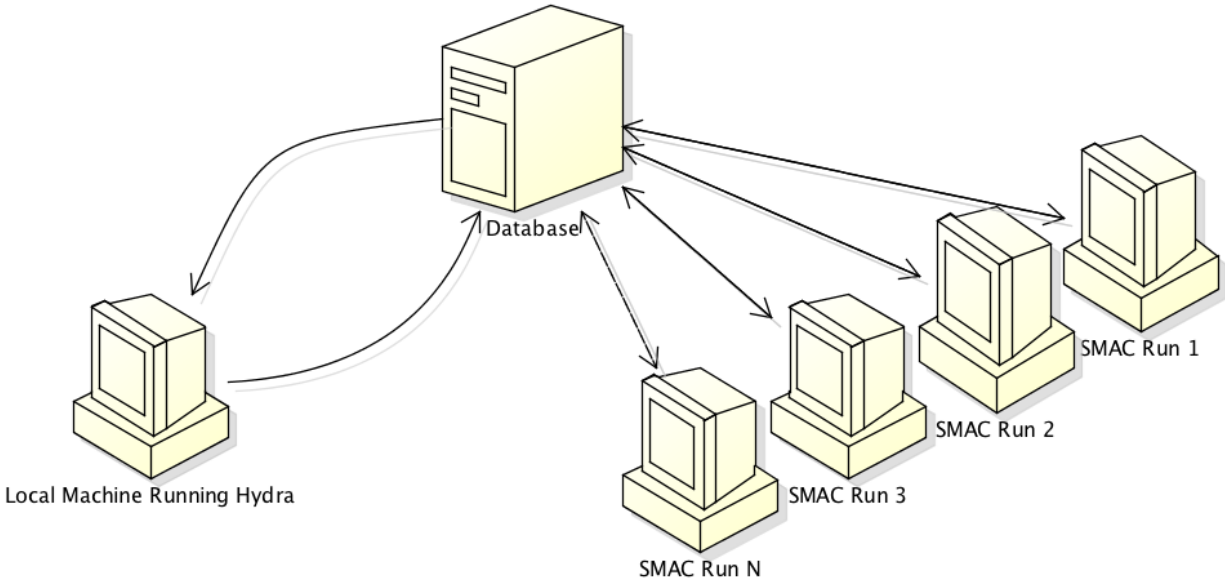


Figure 2: Basic mysqlldb configuration

## 2.5 Important parameters

To tell hydra where to store the output of your hydra experiments, you have the following parameters:

`--run-group` Name of group of experiments

`--num-run` Run number within `run-group` experiment directory.

The experimental output can be found at:

`{your-execution-directory}/hydra-output/{rungroup}/Hydra-{numrun}`

Below are important options that you might want to change based on your computational resources and experiment.

`--num-iterations` How many iterations of Hydra before termination.

`--num-smac-runs` How many SMAC runs at each Hydra iteration. This also will seed your Hydra run. SMAC runs in the database are cached using this parameter.

`--num-configs-per-iter` How many SMAC configurations to add to the portfolio at each Hydra iteration.

Hydra will select `num-configs-per-iter` configurations out of the `num-smac-runs` configurations based on SMAC's internal estimate on training instances.



## 2.6 Running Hydra

A final call to Hydra might look like:

```
./hydra --smac-options <path-to-smac-options-file>
--smac-execution-options <path-to-smac-execution-options>
--num-iterations <number-of-iterations>
--num-smac-runs <number-of-smac-runs-per-iter>
--num-configs-per-iter <number-of-smac-incumbent-configurations-added-per-iter>
```

For instance, to run the included example scenario on Linux you could execute from the root of the folder:

```
> cd example_scenarios/saps
> ../../bin/hydra --smac-options saps-scenario.txt
--smac-execution-options ./example_scenario/spear/smac-executions-options.txt
--num-iterations 4 --num-smac-runs 2 --num-configs-per-iter 1
```

## 2.7 Restarting Hydra

If Hydra crashes or terminates unexpectedly, you can restart Hydra from the last completed iteration by providing the output directory of the crashed run. You can do this by adding the following parameter to your hydra call.

```
--restart-folder <path-to-output-folder>
```

You can find the output folder of your crashed/terminated run at location:

```
{your-execution-directory}/hydra-output/{rungroup}/Hydra-{numrun}
```

## 2.8 Using MySQL database

After setting up your database, you need to tell Hydra:

1. How to contact the database.
2. What pool name to use which will inform the naming of the tables that will be created to store all the information.
3. How to run ‘worker’ jobs on your cluster that will solve the SMAC problems stored in the database.

### 2.8.1 Contacting database

To use the MYSQLDB option for smac executions, you need to provide information to contact the database. Navigate to the `./aeatk` directory and change the `mysqlbdae.opt` and `mysql.opt` files to reflect your database. Then, grab the `./aeatk` directory and copy to your home directory. On linux:

```
> cp -r .aeatk $HOME
```

### 2.8.2 Pool Parameter

The only additional parameter you need to set in the hydra call string is the pool name, which names the tables that are created within the database:

```
mysqldbtae-pool <name-of-pool>
```

For example, to run the included example scenario with the database on Linux you could execute from the root of the folder:

```
> cd example_scenarios/saps
> ../../bin/hydra --smac-options saps-scenario.txt
--smac-execution-options ./example_scenario/spear/smac-executions-options-database.txt
--num-iterations 4 --num-smac-runs 2 --num-configs-per-iter 1
--mysqldbtae-pool hydra_saps --mysqldbtae-run-partition 1
```

### 2.8.3 Running worker

The run jobs that retrieve SMAC input from database and execute SMAC runs, you need to run `./bin/mysql-worker`. A sample call to the executable is shown below:

```
./bin/mysql-worker --pool hydra_testing --logOutputDir ./output --runsToBatch 1
--tae SMAC
```

The pool parameter ***\*must\**** be consistent with `mysqldbtae-pool` parameter in the Hydra call string. If you are using a TORQUE cluster, you can use the worker script at `./scripts/launchworker` and run the `./dispatch_workers` script in order to dispatch a set of `./mysql-worker` jobs. The call string looks like:

```
./dispatch_workers --wall-time <walltime> --num-workers <num-workers>
--pool <pool-name> --log-output-dir <output-directory>
```

For example, to dispatch 25 workers for two days, you would run:

```
> ./dispatch_workers --wall-time 2:00:00:00 --num-workers 25 --pool hydra_test
--log-output-dir $PWD
```

Note: If you have relative paths in your experimental setup, make sure the workers are running in the correct experiment directory. You may want to add `cd {experimental directory}` to the beginning of the `./launchworker` script. The worker will be given all the information in the smac scenario file. Therefore, be careful that the workers are executing in the correct directory if there are relative paths in the smac scenario.

### 2.8.4 Useful option

**Disclaimer:** If you submit jobs to your database, they will not automatically be removed if your job crashes. Therefore, you might execute another run and your workers could be solving jobs from an old hydra run that crashed.

To avoid this problem, you can either

1. Switch pool name for every Hydra run.

2. Delete runs in the database after hydra completes. To do this, set the `--mysqldbtae-run-partition` parameter to an integer  $> 0$ . Then set the `--mysqldbtae-delete-partition-on-shutdown` parameter to `true`.
3. Or, the information in the database could be manually deleted.

## 2.9 Zilla Portfolio Evaluation

To evaluate the test performance of the portfolio in a sequential setting, you may want to use zilla [6], which builds a model that maps instance features to configurations in the portfolio at runtime. To use the zilla portfolio evaluator, set the following hydra command line parameters:

```
--portfolio-evaluator ZILLA
--zilla-options <path-to-file>
```

To see the `saps` example with zilla as the portfolio evaluator, execute the following from the root folder:

```
> cd example_scenarios/saps
> ./run_saps-zilla.sh
```

Below is the example zilla options file:

```
solver-subset-selector-choice=NONE
cheap-feature-manager="./cheap-feature-manager.txt"
aem-choice=LOOKUP
num-presolvers=2
```

Zilla has a complex set of parameters to be used. Here, we limit discussion to how to use zilla with its default parameters for Hydra experiments. For a full comprehensive description of the parameters, see the zilla manual, which will be released shortly. It is recommended that the user can use a zilla scenario similar to the one printed above while needing to change the following parameters:

`--aem-choice LOOKUP` This tells zilla to use algorithm executions from a lookup table rather than file. This option will be removed in the next release.

`--cheap-feature-manager` Path to file containing feature options (ex. `./cheap-feature-manager.txt`). Inside that file, there is a `fromfile-feature-manager-file` options that must point to a file containing features of all the training/testing instances. The feature file must be a .csv file with format:

```
instance, runtime , feature1, feature2, ..., featuren
inst1,1,2,...,3
inst2,1,2,...,3
...
instn,1,2,...,3
```

(Note that if a feature cannot be computed for an instance, give the feature a value of -512.) See the `./features.csv` file within `./example_scenarios/saps` directory for an example.

The cheap feature file should contain a subset of the complete features that are cheap to compute. These features are used to predict if the rest of features are computable within a designated feature cutoff time. If using the cheap features, zilla predicts that the rest of the features will not be computable for an instance, zilla will go straight to a backup solver.

### 2.9.1 Log output

Here we walk you through the log output you will see with zilla as the portfolio evaluator.

#### Training phase

Initially, a set of presolver schedules will be generated. Presolvers are ran for a short amount of time before features are computed (possibly expensive) to ensure very easy instances are solved fast.

```
[INFO ] Creating 16 pre-solving schedule of 2 pre-solvers out of 2 solvers
and 3 runtimes.
[INFO ] Pre-solving policies:
...
```

Next, for every presolving schedule, a classification model will be built on all the training instances that both can't be solved by presolving schedule and have a complete set of features (no missing features):

```
[INFO ] Trying pre-solving schedule 1/16: [].
[INFO ] Building Zilla stage.
[INFO ] 5 instances to train on.
[INFO ] Using pre-solving policy []
[INFO ] 5 instances to train on, after pre-solving.
[INFO ] 0 instances with incomplete features to train on, after pre-solving.
[INFO ] Finding a backup solver for these instances...
[INFO ] ...done!
[INFO ] Backup solver -alpha '1.150772766802867' -ps '0.151208835650751'
-rho '0.9536596755518831' -wp '0.016415262593045724' identified.
[INFO ] 5 instances with complete features to train on after pre-solving.
[INFO ] Considering 2 solvers: [-alpha '1.2622748357849096' -ps '0.061225064277181024'
-rho '0.6406855588973956' -wp '0.05793399015684799', -alpha '1.150772766802867'
-ps '0.151208835650751' -rho '0.9536596755518831' -wp '0.016415262593045724'].
[INFO ] Training solver pairs models for these instances...
[INFO ] Building model 1 out of 1.
[INFO ] ...done!
[INFO ] Doing solver subset selection ...
[INFO ] Performing solver subset selection ...
[INFO ] No optimization performed. Returning the full set as an answer.
[INFO ] ...done!
```

```
[INFO ] ...done!
[INFO ] Solvers selected: [-alpha '1.150772766802867' -ps '0.151208835650751'
-rho '0.9536596755518831' -wp '0.016415262593045724',
-alpha '1.2622748357849096' -ps '0.061225064277181024'
-rho '0.6406855588973956' -wp '0.05793399015684799']
```

Then, all the models from all the presolving schedules will be evaluated with the validation instance set, and the best pre solving scenario and corresponding model will be selected:

```
[INFO ] Zilla building for parameter selection completed.
[INFO ] Best Zilla selector (scenario: [Execute -alpha '1.150772766802867'
-ps '0.151208835650751' -rho '0.9536596755518831'
-wp '0.016415262593045724' for 0.25], solver subset:
[-alpha '1.2622748357849096' -ps '0.061225064277181024'
-rho '0.6406855588973956' -wp '0.05793399015684799',
-alpha '1.150772766802867' -ps '0.151208835650751'
-rho '0.9536596755518831' -wp '0.016415262593045724'])
had performance 0.03788821660229363.
```

## Testing phase

Online, zilla will use the pre solving schedule and model found from training and solve the testing instances. For each, you get an output similar to:

```
[INFO ] 1/5 - solving instance ./instances/test/SWlin2006.12713.cnf.
[INFO ] Setting instance's base features.
[INFO ] Selecting solving policy.
[INFO ] Solving policy selected:
[INFO ] Zilla Solving Policy -
presolving [Execute -alpha '1.150772766802867' -ps '0.151208835650751'
-rho '0.9536596755518831' -wp '0.016415262593045724' for 0.25],
main solving selector AlgorithmListClassificationWrapperSolvingPolicySelector
with model Constant Model Returning [-alpha '1.150772766802867'
-ps '0.151208835650751' -rho '0.9536596755518831'
-wp '0.016415262593045724', -alpha '1.2622748357849096'
-ps '0.061225064277181024' -rho '0.6406855588973956'
-wp '0.05793399015684799']., backup solving
[Execute -alpha '1.150772766802867' -ps '0.151208835650751'
-rho '0.9536596755518831' -wp '0.016415262593045724' up to termination].
[INFO ] Executing solving policy.
[INFO ] Result: 0.03 (5.0), SAT, 0.0.
```

## 2.10 Output

Once Hydra completes, it should print out a string for all the configurations in the final portfolio. For example:

```

Best single configuration: -x '3.64' -y '3.05' -z '1.38'
Final portfolio configurations
1. Portfolio Configuration : -x '3.64' -y '3.05' -z '1.38'
2. Portfolio Configuration : -x '4.49' -y '2.91' -z '2.23'
3. Portfolio Configuration : -x '3.68' -y '1.11' -z '1.13'
4. Portfolio Configuration : -x '2.5' -y '2.5' -z '2.5'
5. Portfolio Configuration : -x '4.41' -y '3.97' -z '2.02'
6. Portfolio Configuration : -x '2.95' -y '4.10' -z '3.13'
Iteration      Performance
-----
0              6.45
1              5.38
2              5.12
3              5.12
-----

```

The last line for the output shows the performance history across hydra iterations. The performance was 6.45 at iteration 0, 5.38 at iteration 1, and stagnated at iteration 2 at 5.12.

### 2.10.1 Directory Structure

By default, Hydra writes the following logging files out to disk (NOTE: The N represents the `--seed` setting): `log-runN.txt` contains a full dump of all the information logged, and where it was logged from. `log-warnN.txt` contains the same information as the above file, except only from warning and higher level messages. `log-errN.txt` contains the same information as the above file, except only from error messages. Below contains the directory structure from a sample hydra scenario.

```

{execution-directory}
├── hydra-output
│   └── {rungroup}
│       └── Hydra-{num-run}
│           ├── log-run.txt (main logging messages)
│           ├── log-warn.txt (warning logging messages)
│           ├── log.err.txt (error logging messages)
│           ├── Portfolios (Configurations in JSON format after each hydra iteration)
│           │   ├── Portfolio_0.txt (portfolio after iteration 0)
│           │   └── Portfolio_1.txt (portfolio after iteration 1)
│           ├── smac_runs
│           │   ├── Iteration_0_1
│           │   │   └── smac output from smac run 1 form hydra iteration 0
│           │   ├── Iteration_1_1
│           │   │   └── smac output from smac run 1 form hydra iteration 1

```

## 2.11 Building

Hydra is built using Gradle. If you would like to make changes to the software and rebuild hydra, execute the following steps from the hydra root folder:

First create a folder for the software and create a directory within that will contain the source code. For example:

```
mkdir -p Hydra/src/main/java
```

Then you need to copy both the build files within `./build` and the hydra source jar within the `./lib` folder into the directory you just created.

```
cp ./build/* Hydra/  
cp ./lib/Hydra-1.0-development-{commit}-sources.jar Hydra/src/main/java
```

Next, uncompress the source .jar

```
cd Hydra/src/main/java  
jar xf Hydra-1.0-development-{commit}-sources.jar  
cd ../../..
```

Then after making any changes to the source code, you can build with:

```
./gradlew :installDist
```

Running `./gradlew :instDist` will download gradle, all the necessary dependencies and compile the project. Expect this to take a while for the first build. Subsequent builds will be much faster after you have all the dependencies downloaded and cached locally.

Note: Use `./gradlew.bat` if you are a windows user.

Good Luck ☺

## References

- [1] Frank Hutter. Paramils: an automatic algorithm configuration framework. *Journal of Artificial Intelligence Research*, 36(1):267–306, 2009.
- [2] Frank Hutter, Holger H Hoos, and Kevin Leyton-Brown. Sequential model-based optimization for general algorithm configuration. In *Learning and Intelligent Optimization*, pages 507–523. Springer, 2011.
- [3] Frank Hutter, Holger H Hoos, and Kevin Leyton-Brown. Parallel algorithm configuration. In *Learning and Intelligent Optimization*, pages 55–70. Springer, 2012.
- [4] Frank Hutter and Steve Ramage. Manual for smac version v2. 08.00-master. 2014.
- [5] Lin Xu, Holger Hoos, and Kevin Leyton-Brown. Hydra: Automatically configuring algorithms for portfolio-based selection. 2010.
- [6] Lin Xu, Frank Hutter, Holger H Hoos, and Kevin Leyton-Brown. Satzilla: portfolio-based algorithm selection for sat. *Journal of Artificial Intelligence Research*, pages 565–606, 2008.
- [7] Lin Xu, Frank Hutter, Holger H Hoos, and Kevin Leyton-Brown. Hydra-mip: Automated algorithm configuration and selection for mixed integer programming. In *RCRA workshop on experimental evaluation of algorithms for solving problems with combinatorial explosion at the international joint conference on artificial intelligence (IJCAI)*, pages 16–30, 2011.