

## 0.1 Running SMAC

To get started with an existing configuration scenario you simply need to execute smac as follows:

```
./smac --scenarioFile <file> --numRun 0
```

This will execute SMAC with the default options on the scenario specified in the file. Some commonly-used non-default options of SMAC are described in this section. The **--numRun** argument controls the seed and names of output files (to support parallel independent runs)

## 0.2 Testing the Wrapper

SMAC includes a method of Testing Algorithm Execution, via the `smac-algotest` utility. It takes the required scenario options <sup>1</sup> **--execDir**, **--paramFile**, **--algo**, **--cutoff\_time** the instance, and configuration to run on.

For example:

```
./smac-algotest --execDir <dir> --paramFile <file> --algo <file>
--cutoff_time 300 --instance <instance> --config <config string>
-P[name]=[value] -P[name]=[value]...
```

Some parameters deserve special mention:

1. The config string syntax is a single string with “-name=‘value’ ” ... you can also specify `RANDOM` which will generate a random configuration or `DEFAULT` which will generate the default configuration.
2. The `-P` parameters are optional and allow overriding specific values in the configuration (this is useful primarily for `RANDOM` and `DEFAULT`, to allow you to set certain values). To set the `sortalgo` to `merge` you would specify `Psortalgo=merge`.

## 0.3 ROAR Mode

```
./smac --scenarioFile <file> --executionMode ROAR --numRun 0
```

This will execute the ROAR algorithm, a special case of SMAC that uses an empty model and random selection of configurations. See [?] for details on ROAR.

## 0.4 Adaptive Capping

```
./smac --scenarioFile <file> --adaptiveCapping true --numRun 0
```

Adaptive Capping (originally introduced for ParamILS [? ], but also applicable in SMAC [? ]) will cause SMAC to only schedule algorithm runs for as long as is needed to determine whether they are better than the current incumbent. Without this option, each target algorithm runs up to the runtime specified in the configuration scenario file **--cutoffTime**.

NOTE: Adaptive Capping should only be used when the **--runObj** is `RUNTIME`. Adaptive capping can drastically improve SMAC’s performance for scenarios with a large difference between **--cutoffTime** and the runtime of the best-performing configurations. Related configuration options are **--capSlack**, **--capAddSlack**, and **--imputationIterations**.

---

<sup>1</sup>Unfortunately it cannot read scenario files currently

## 0.5 Wall-Clock Limit

```
./smac --scenarioFile <file> --runtimeLimit <seconds> --numRun 0
```

SMAC offers the option to terminate after using up a given amount of wall-clock time. This option is useful to limit the overheads of starting target algorithm runs, which are otherwise unaccounted for. This option does not override **--tunerTimeout** or other options that limit the duration of the configuration run; whichever termination criterion is reached first triggers termination.

## 0.6 Change Initial Incumbent

```
./smac --scenarioFile <file> --numRun 0 --initialIncumbent <config string>
```

SMAC offers the option to specify the initial incumbent, and by default uses the default configuration specified in the parameter file. The argument to **--initialIncumbent** follows the same conventions as in Section 0.2.

## 0.7 State Restoration

```
./smac --scenarioFile <file> --restoreStateFrom <dir>  
      --restoreIteration <iteration> --numRun 0
```

SMAC will read the files in the specified directory and restore its state to that of the saved SMAC run at the specified iteration. Provided the remaining options (e.g. **--seed**, **--overall\_obj**) are set identically, SMAC should continue along the same trajectory.

This option can also be used to restore runs from SMAC v1.xx (although due to the lossy nature of Matlab files and differences in random calls you will not get the same resulting trajectory). By default the state can be restored to iterations that are powers of 2, as well as the 2 iterations prior to the original SMAC run stopping. If the original run crashed, additional information is saved, often allowing you to replay the crash.

NOTE: When you restore a SMAC state, you are in essence preloading a set of runs and then running the scenario. In certain cases, if the scenario has been changed in the meantime, this may result in undefined behavior. Changing something like **--tunerTimeout** is usually a safe bet, however changing something central (such as **--runObj**) would not be.

To check the available iterations that can be restored from a saved directory, use:

```
./smac-possible-restores <dir>
```

To disable saving any state information to disk, use

```
./smac --scenarioFile <file> --stateSerializer NULL --numRun 0
```

## 0.8 Concurrent Algorithm Execution Requests

```
./smac --scenarioFile <file> --maxConcurrentAlgoExecs <num> --numRun 0
```

In certain circumstances, it may be much faster to allow more than one target algorithm execution at once, (e.g., when multiple cores are available or when actual algorithm execution is I/O bound). To exploit this, you can have SMAC schedule multiple runs at a time. If **--adaptiveCapping** is not set, this will result in the same trajectory as a sequential version (when **--maxConcurrentAlgoExecs** is set to 1). When adaptive capping is enabled, concurrent runs are scheduled with cutoff times as if each were the first of the runs to be scheduled.

## 0.9 Named Rungroups

```
./smac --scenarioFile <file> --runGroupName <foldername> --numRun 0
```

All output is written to the folder <foldername>; runs differing in **--numRun** will yield different output files in that folder.

## 0.10 Offline Validation

SMAC includes a tool for the offline assessment of incumbents selected during the configuration process. By default, given a test instance file with  $N$  instances, SMAC performs  $\approx 1\,000$  target algorithm validation runs per configuration (rounded up to the nearest multiple of  $N$ ).

By default, SMAC limits the number of seeds used in validation runs to 1 000 seeds per instance. This number can be changed as in the following example:

```
./smac --scenarioFile <file> --numSeedsPerTestInstance 50
```

(This parameter does not have any effect in the case of instance/seed files.)

### 0.10.1 Limiting the Number of Instances Used in a Validation Run

To use only some of the instances or instance seeds specified you can limit them with the **--numTestInstances** parameter. When this parameter is specified, SMAC will only use the specified number of lines from the top of the file, and will keep repeating them until enough seeds are used:

```
./smac --scenarioFile <file> --numTestInstances 10
```

For instance files containing seeds, this option will only use the specified number of instance seeds in the file.

### 0.10.2 Disabling Validation

Validation can be skipped altogether as follows:

```
./smac --scenarioFile <file> --skipValidation
```

### 0.10.3 Standalone Validation

SMAC also includes a method of validating configurations outside of a smac run. You can supply a configuration using the **-configuration** option. All scenario options are applicable to the standalone validator, but check the usage screen to see all the options available NOTE: Some options while present are not applicable for validation but are presented anyway.

Here is an example call:

```
./smac-validate --scenarioFile <file> --numValidationRuns 10000  
--configuration <config string> --maxConcurrentAlgoExecs 8 --numRun 0
```

Usage notes for the offline validation tool:

1. This validates against the test set only; the training instance set is not used.
2. By default this outputs into the current directory; you can change the output directory with the option **--runGroupName**.
3. You can also validate against a trajectory file issued by **--trajectoryFile** option.