Manual for SMAC version v2.10.03-master

Frank Hutter & Steve Ramage
Department of Computer Science
University of British Columbia
Vancouver, BC V6T 1Z4, Canada
{hutter,seramage}@cs.ubc.ca

July 19, 2015

Contents

1 Introduction

This document is the manual for SMAC [**?**] (an acronym for *Sequential Model-based Algorithm Configuration*). SMAC aims to solve the following *algorithm configuration* problem: Given a binary of a parameterized algorithm $\mathcal{A}$, a set of instances $\mathcal{S}$ of the problem $\mathcal{A}$ solves, and a performance metric m, find parameter settings of $\mathcal{A}$ optimizing m across $\mathcal{S}$.

In slightly more detail, users of SMAC must provide:

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

SMAC then executes algorithm $\mathcal{A}$ with different *parameter configurations* (combinations of parameters $\langle \theta_1, \ldots, \theta_n \rangle \in \Theta_1 \times \cdots \times \Theta_n$, on instances $\pi \in \Pi$), searching for the configuration that yields overall best performance across the benchmark instances under the supplied objective. For more details please see [**?**]; if you use SMAC in your research, please cite that article. It would also be nice if you sent us an email – we are always interested in additional application domains.

1.1 License

This version of SMAC is released under the AGPLv3, please read `LICENSE-AGPLv3.txt` for more information. For other license options please e-mail Frank Hutter. Note that versions of SMAC prior to v2.10 were released under a different license.

1.2 System Requirements

SMAC itself requires only Java 7 or newer to run. SMAC is primarily intended to run on Unix like platforms, but now includes start up scripts so that it can run on Windows. You should be able to follow the commands exactly as laid out in this document but note that you may need to change the ending of the script to `.bat` (for instance `smac` becomes `smac.bat`), additionally you may want to substitute backslashes instead of forward slashes for paths.

Most of the included scenarios (in `./example_scenarios/` require ruby and Linux 32-bit libraries to run. The scenarios in `./example_scenarios/branin/` require only python and are cross platform. The saps scenario should work on Windows, Linux and Mac OS X (no BSD or Solaris).

1.3 Version

This version of the manual is for SMAC v2.10.03-master-778.

Project	Version	Commit	Dirty Flag
aeatk	v2.10.01-master-803	d97311a573a01ad81b40b818502fb29daa6e9f05	0
aeatk	v2.10.02-development-804	c4b45729e04087639b294c9783cf1261391befb2	0
aeatk	v2.10.02-master-805	9c0ad1b3eea679d95755e1444e3f4e38626aae4d	0
aeatk	v2.10.02-development-806	c4b45729e04087639b294c9783cf1261391befb2	1
aeatk	v2.10.02-development-807	c4b45729e04087639b294c9783cf1261391befb2	1
aeatk	v2.10.02-development-808	c4b45729e04087639b294c9783cf1261391befb2	1
aeatk	v2.10.02-development-809	c4b45729e04087639b294c9783cf1261391befb2	1
aeatk	v2.10.03-development-810	c61caa5344123dc91c718aff94ed91d7e881e838	1
aeatk	v2.10.03-master-811	4d93d555f3d162028bcbd0044530605ce8c6a693	1
aeatk	v2.10.03-master-812	ebf4c8c913d509bce45e7e65c3ed79ea9d0829ba	0
aeatk	v2.10.03-master-813	ebf4c8c913d509bce45e7e65c3ed79ea9d0829ba	0
aeatk	v2.10.03-master-814	ebf4c8c913d509bce45e7e65c3ed79ea9d0829ba	0
smac	v2.10.03-master-778	3ee628ef9bf258142ef0e6aa8d73093888283fa3	0

NOTE: For non-`master` builds these commits may not contain everything in the build. (*i.e.*,non-`master` builds can be built with uncommitted changes). If the dirty flag is 0 that means the commit contains this exact copy, 1 means there were some uncommitted changes, and something else means some other error occurred when we tried to generate this.

2 Differences Between SMAC and ParamILS

There are a number of differences between SMAC and ParamILS, including the following.

- **Support for continuous parameters:** While ParamILS was limited to categorical parameters, SMAC also natively handles continuous and integer parameters. See Section 4.4.1 for details.
- **Run objectives:** Not all of ParamILS's run objectives are supported at this time. If you require an unsupported objective please let us know.
- **Order of instances:** In contrast to ParamILS, the order of instances in the instance file does not matter in SMAC.
- **Configuration time budget and runtime overheads:** Both ParamILS and SMAC accept a time budget as an input parameter. ParamILS only keeps track of the CPU time the target algorithm reports and terminates once the sum of these runtimes exceeds the time budget; it does *not* take into account overheads due to e.g. command line calls of the target algorithm. In cases where the reported CPU time of each target algorithm run was very small (e.g. milliseconds), these unaccounted overheads could actually dominate ParamILS's wall-clock time. SMAC offers a more flexible management of

its runtime overheads through the options **--use-cpu-time-in-tunertime** and **--wallclock-limit**. See Section 3.4 for details on the wall clock time limit.

- **Resuming previous runs:** While this was not possible in ParamILS, in SMAC you can resume previous runs from a saved state. Please refer to Section 3.6 for how to use the state restoration feature. Section 6.2 describes the file format for saved states.
- **Feature files:** SMAC accepts as an optional input a feature file providing additional information about the instances in the training set; see Section 4.3.
- **Algorithm wrappers:** The wrapper syntax has been extended in SMAC to support additional results in the "solved" field. Specifically, there is a new result **ABORT** signalling that the configuration process should be aborted (e.g. because the wrapper is in an inconsistent state that should never be reached). A similar behaviour is triggered if option **--abort-on-first-run-crash** is set and the first run returns **CRASHED**. Additionally, the wrapper can also return additional data to SMAC that is associated with the run [1]. For more information see Section 5.1.2.
- **Instance files vs. instance/seed files:** The **instance_file** parameter now auto-detects whether the file conforms to ParamILS's **instance_file** or **instance_seed_file** format. SMAC treats the latter option as an alias for the former. See Section 4.2 for details. While SMAC is backwards compatible with previous (space-separated) files, the preferred format is now `.csv`.

3 Commonly Used Options

3.1 Running SMAC

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

```
./smac --scenario-file <file> --seed 1
```

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 **--seed** argument controls the seed and names of output files (to support parallel independent runs). The **--seed-offset** argument lets you keep the output folders names simple while varying the actual seed of SMAC. The seed argument is also optional and will automatically be chosen if not set.

3.2 Testing the Wrapper

SMAC includes a method of Testing Algorithm Execution, via the `algotest` utility. It takes the required scenario options [2]

For example:

```
./algotest --scenario-file <scenario> --instance <instance>
--config <config string> -P[name]=[value] -P[name]=[value]...
```

Some parameters deserve special mention:

[1] This data will be saved in the run and results file (Section 6.2) that is used in state saving

[2] Unfortunately it cannot read scenario files currently

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` parameter to `merge` you would specify `Psortalgo=merge`.

3.3 Verifying the Scenario

SMAC includes a utility that allows you to test the scenario. It is currently `BETA` but does a bit more sanity checks than SMAC will normally do.

For example:

```
./verify-scenario --scenarios ./scenarios/*.txt --verify-instances true
```

The utility has some limitations however:

1. It currently does not check test instances

2. Scenario files can specify non-scenario options in SMAC (and some of the example scenarios in fact do), this utility is not aware of them, and will report an error.

3.4 Wall-Clock Limit

```
./smac --scenario-file <file> --wallclock-limit <seconds> --seed 1
```

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 **--tunertime-limit** or other options that limit the duration of the configuration run; whichever termination criterion is reached first triggers termination.

3.5 Change Initial Incumbent

```
./smac --scenario-file <file>  --initial-incumbent <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 **--initial-incumbent** follows the same conventions as in Section 3.2.

3.6 State Restoration

```
./smac --scenario-file <file> --restore-scenario <dir>
```

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 identicially, 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 behaivor. Changing something like **--tunertime-limit** is usually a safe bet, however changing something central (such as **--run-obj**) would not be.

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

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

3.7 Warm-Starting the Model

```
./smac --scenario-file <file> --warmstart <foldername>
```

Using the same state data as in Section 3.6, you can also just choose to warm-up the model with previous runs. Instead of the **--restore-scenario** option use **--warmstart** instead. SMAC will operate normally, but when building the model the above data will also be used. Please keep in mind the following.

NOTE: If the execution mode is ROAR, this option has no effect.

WARNING: Due to design limitations of the state restoration format in this version of SMAC you cannot / should not have any differences between the instance distribution used to warmstart the model, and the instance distribution we are configuring against. In the best case you will simply get a random exception at some point (perhaps a `NullPointerException`), and in the worst case it will just load the model with junk.

TIP: The included state-merge utility allows you to easily merge a bunch of different runs of SMAC into one state that you can use for a warm start.

3.8 Named Rungroups

```
./smac --scenario-file <file> --rungroup <foldername>
```

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

3.9 More Options

By default SMAC only displays BASIC usage options, other options are INTERMEDIATE, ADVANCED, and DEVELOPER. Be warned that there are a bunch of options and some of the more advanced and developer options may cause SMAC to perform very poorly.

```
./smac --help-level INTERMEDIATE
```

3.10 Shared Model Mode (Experimental)

NOTE: Please read this full section before deciding to use this option

SMAC has an experimental option that essentially allows multiple runs of SMAC to share data and construct better models quickly.

```
./smac --scenario-file <file> --shared-model-mode true
```

There are a couple of things to keep in mind when running with this option:

1. The first is that the different SMAC runs need to be using the exact same scenario, that MUST have different seeds. Even small inconsequential differences may cause this to fail (for instance if the location on different machines and the path to execute the target algorithm is different). SMAC should in most cases recover gracefully and just ignore the incompatible run data, but it is possible that the SMAC run may be corrupted.

2. The shared file system between clusters needs to allow a file that is being written on one machine to be read on another machine. We've had reports that on some file systems (AFS) with some locking policies you cannot do this. In this case you can still benefit from this mode but it might need to be a little more coarse. After doing a first set of runs, you can then do a second batch with the –share-run-data, they won't be able to read the second batches data, but they should be able to read the first batches.

3. The frequency with which runs are re-read is controlled via the **--shared-model-mode-frequency** which defaults to 5 minutes, depending on your scenario and the amount of data you need you may want to set it lower. If your runs take considerably longer than 5 minutes there is no need to increase this, the data is only re-read at most once for every local algorithm run, and the above is designed to prevent hitting the file system too frequently

4. You probably will need to increase the amount of memory you give to SMAC, using the `SMAC_MEMORY` environment variable. In your bash shell script this can be accomplished via export SMAC_MEMORY=2048, and in Windows SET SMAC_MEMORY=2048.

5. This mode turns N independent SMAC runs into N dependent runs of SMAC. This mode is designed to help get better performing configurations, it be inappropriate to treat these runs as independent samples from the same distribution for experimental purposes. What may be appropriate is to compare the experimental protocols, selecting the best performing run of independent SMAC on the training set, versus the best performing run of dependent SMAC on the training set, and reporting their values on the testing set.

6. This mode does not require that different runs of SMAC execute concurrently at all. At one other extreme case runs could happen sequentially, this mode would just be an easier way of running SMAC, flushing the run data, but warm starting the model with the previous. At the other extreme case, and the case we have some preliminary experiments for, all the runs are started at the exact same time. In this case, there was a substantial boost in median performance (but see previous point why this is misleading), but also selecting the run with the best training instance over time generally resulted in as good or better performance. Another possibility is to have some runs not use this mode and have other runs with this enabled. Depending on the scenario, this might allow the models to maintain more diversity. Unfortunately the benefits and best practices of this option is unexplored at this time.

One advantage of this approach is that if you only care about getting a good configuration quickly (without worrying about reproducibility), you can schedule the runs of SMAC to the cluster independently, which should make them quicker to dispatch and yet still benefit from the shared data.

7. While SMAC is running in shared model mode, you may see sporadic errors about corrupted files, etc. These are generally safe to ignore and are likely caused by writing and reading happening simultaneously. Upon reading an error, SMAC will continue trying to read the file. Until the file is successfully read, no further errors or warnings will be presented.

8. When in this mode you should see messages like the following which indicate a new source of runs was detected:

```
[INFO ] Detected new sources of shared runs :
 [live-rundata-1.json, live-rundata-2.json]
```

   This indicates that we have started to read runs from the above files.

9. At the end of a run you will see a line like:

```
[INFO ] At shutdown: ./smac-output/branin-scenario/live-rundata-3.json
had 15 runs added to it
[INFO ] At shutdown: we retrieved atleast 20 runs and added them to
 our current data set [live-rundata-1.json=>10, live-rundata-2.json=>10]
```

3.11 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 --scenario-file <file> --num-seeds-per-test-instance 50
```

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

3.11.1 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 **--num-test-instances** 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 --scenario-file <file> --num-test-instances 10
```

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

3.11.2 Disabling Validation

Validation can be skipped alltogether as follows:

```
./smac --scenario-file <file> --seed 1 --validation false
```

3.11.3 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 --scenario-file <file> --num-validation-runs 100
     --configuration <config string> --cli-cores 8 --seed 1
```

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 **--rungroup**.
3. You can also validate against a trajectory file issued by **--trajectory-file** option.

4 File Format Reference

4.1 Option Files

Option Files are a way of saving a different set of values frequently used with SMAC without having to specify them on every execution. The general format for an option file is the name of the configuration option (without the two dashes), an equal sign, and then the value (for booleans it should be true or false, lowercase). Currently options that take multiple arguments are not supported. Additionally you can not use aliases that are single dashed (*e.g.* to override the Experiment Directory, you must use **–experiment-dir** and not **-e**)

When using Option Files it is important that no two files (including the Scenario File), specify the same option, the resulting configuration is undefined, and in general this will not throw an error.

4.1.1 Scenario File

The Scenario Option File, or Scenario File, is the recommended way of configuring SMAC [3]. The Scenario Files used in SMAC are backwards compatible with ParamILS and the name of option names here reflect that[4]. NOTE: **cutoff_length** is not currently supported.

algo An algorithm executable or wrapper script around an algorithm that conforms with the input/output format specified in section 5.1. The string here should be invokable via the system shell.

[3] Nothing in general prevents you from specifying non-scenario options in these files, but in general you should restrict your files to these.

[4] Every option name listed here is in fact an alias for an existing option listed in the section 9.5 and it is entirely possible to use SMAC without using Scenario Files.

execdir Directory to execute `<algo>` from: (*i.e.* "cd `<execdir>`; `<algo>`")

deterministic A boolean that governs whether or not the algorithm should be treated as deterministic. For backwards compatibility with ParamILS, this option also supports using 0 for false, and 1 for true. SMAC will never invoke the target algorithm more than once for any given instance, seed and configuration. If this is set to `true`, SMAC will never invoke the target algorithm more than once for any given instance and configuration.

run_obj Determines how to convert the resulting output line (as defined in Section 5.1.2) into a scalar quantifying how "good" a single algorithm execution is, (*e.g.* how long it took to execute, how good of a solution it found, etc...). SMAC will attempt to *minimize* this objective.

Currently implemented objectives are the following:

Name	Description
RUNTIME	**Minimize the reported runtime of the algorithm.**
QUALITY	**Minimize the reported quality of the algorithm.**

overall_obj While **run_obj** defines the objective function for a single algorithm run, **overall_obj** defines how those single objectives are combined to reach a single scalar value to compare two parameter configurations. Implemented examples for this are as follows:

Name	Description
MEAN	**The mean of the values**
MEAN1000	**Unsuccessful runs are counted as 1000 $\times$ target_run_cputime_limit**
MEAN10	**Unsuccessful runs are counted as 10 $\times$ target_run_cputime_limit**

target_run_cputime_limit The CPU time after which a single algorithm execution will be terminated as unsuccess (and treated as a **TIMEOUT**). This is an important parameter: If chosen too high, lots of time will be wasted with unsuccessful runs. If chosen too low the optimization is biased to perform well on easy instances only.

cputime_limit The limit of the CPU time allowed for configuration (*i.e.*The sum of all algorithm runtimes, and by default the sum of the CPU time of SMAC itself).

wallclock_limit The limit of the amount of wallclock (or real) time allowed for configuration.

paramfile Specifies the file with the parameters of the algorithm. The format of this file is covered in Section 4.4.

outdir Specifies the directory SMAC should write its results to.

instance_file Specifies the file containing the list of problem instances (and possibly seeds) for SMAC to use during the *Automatic Configuration Phase*. The ParamILS parameter **instance_seed_file** aliases this one and the format is auto-detected. The format of these files is covered in section 4.2.

test_instance_file Specifies the file containing the list of problem instances (and possibly seeds) for SMAC to use during *Validation Phase*. The ParamILS parameter **test_instance_seed_file** aliases this one and the format is auto-detected. The format of these files is covered in section 4.2.

feature_file Specifies the a file with the features for the instances in the **instance_file** and possibly the **test_instance_file** [5]. The format of this file is covered in section 4.3.

4.2 Instance File Format

The files used by the **instance_file** & **test_instance_file** options come in four potential formats, all of which are CSV based[6]. Before specifying the formats it is important to note the three kinds of information that are specified with instances [7].

Instance Name The name of the instance that was selected. This should be meaningful to the target algorithm we are configuring [8].

Instance Specific Information A free form text string (with no spaces or line breaks) that will be passed to the Target Algorithm whenever executed.

Seed A specific seed to use when executing the target algorithm.

The possible formats are as follows, and depend on what information you'd like to specify.

1. Each line specifies only a unique **Instance Name**. No **Instance Specific Information** will be used, and **Seed**'s will be automatically generated.
2. Each line specifies a **Seed** followed by the **Instance Name**. Every line must be unique, but for each **Instance Name** additional seeds will be used in order, when that instance is selected.
3. Each line specifies a **Instance Name** followed by the **Instance Specific Information**. Every **Instance Name** must be unique, **Seed**'s will be automatically generated.
4. Each line specifies a **Seed** followed by the **Instance Name** followed by the **Instance Specific Information**. Every line must be unique, and furthermore, for all **Instance Name**'s the **Instance Specific Information** must be the same for all **Seed** values (*i.e.*You cannot specify different instance specific information that is a function of the seed used).

4.3 Feature File Format

The **feature_file** specifies features that are to be used for instances. Feature Files are specified in CSV format, the first column of every row should list an **Instance Name** as it appears in the **instance_file**. The subsequent columns should list `double` values specifying a computed continuous feature. By convention the value -512, and -1024 are used to signify that a feature value is missing or not applicable. All instances must have the same number of features.

At the top of the file there MUST appear a header row, the cell that appears above the instance names is unimportant, but for each feature a unique and *non-numeric* (*i.e.* it must contain atleast one letter) feature name must be specified.

[5] The Validator will load features into memory for test instances if they exist.

[6] Specifically each cell should be double-quoted (*i.e.*"), and use a comma as a cell delimiter. SMAC also supports the old method of reading files that use space as a cell delimiter and do not enclose values. However these files cannot handle **Instance Name**'s that contain spaces.

[7] Features which are required for SMAC but not ParamILS are specified in a seperate file see section 4.3.

[8] Generally **Instance Names** reference specific files on disk.

4.4 Parameter Configuration Space Format

The PCS format requires each line to contain one of the following 3 clauses, or only whitespace/comments.

- **Parameter Declaration Clauses** specify the names of parameters, their type, their domains, and default values.
- **Conditional Parameter Clauses** specify when a parameter is active/inactive.
- **Forbidden Parameter Clauses** specify when a combination of parameter settings is illegal.

Comments are allowed throughout the file; they begin with a #, and run to the end of a line.

4.4.1 Parameter Declaration Clauses

The PCS format supports four types of parameters: Categorical, Ordinal, Integer, or Real.

Categorical parameters

Categorical parameters take one of a finite set of values. Each line specifying an categorical parameter should be of the form:

```
<parameter_name> categorical {<value 1>, ..., <value N>} [<default value>]
```

where '`<default value>`' has to be one of the set of possible values.

Example 1

```
decision-heuristic categorical {1,2,3} [1]
```

This means that the parameter '`decision-heuristic`' can be given one of three possible values, with the default assignment being '`1`'.

Example 2

```
@1:loops categorical {common,distinct,shared,no}[no]
```

In this example, the somewhat cryptic parameter name '`@1:loops`' is perfectly legal; the only forbidden characters in parameter names are spaces, commas, quotes, and parentheses.

Categorical parameter values are also strings with the same restrictions; in particular, there is no restriction for categorical parameter values to be numbers.

Restrictions

When using the Advanced Forbidden Syntax, parameter name and values are restricted to starting with an alphabetic character or an underscore, and can only contain alphabetic, digits, or underscores.

Example 3

```
DS categorical {TinyDataStructure, FastDataStructure}[TinyDataStructure]
```

As this example shows, the parameter values can even be Java class names (to be used, e.g., via reflection).

Example 4

```
random-variable-frequency categorical {0, 0.05, 0.1, 0.2} [0.05]
```

Finally, as this example shows, numerical parameters can trivially be treated as categorical ones by simply discretizing their domain (selecting a subset of reasonable values).

Ordinal parameters

Ordinal parameters take one of a finite set of values. Each line specifying a ordinal parameter should be of the form:

```
<parameter_name> ordinal {<value 1>, ..., <value N>} [<default value>]
```

where '`<default value>`' has to be one of the set of possible values. An ordinal parameter differs from a categorical parameter, in that individual values are comparable (under $>$ and $<$ operators).

Example 5

```
random-variable-frequency ordinal {0, 0.05, 0.1, 0.2} [0.05]
```

This example encodes the same variable as Example 4, and has the same set of allowed values but also specifies that the following is true: $0 < 0.05 < 0.1 < 0.2$.

Example 6

```
annealing-temperature ordinal {cold, cool, medium, warm, hot} [medium]
```

In this example the set of variables is the same, but also informs us that `cold` $<$ `cool` $<$ `medium` $<$ `warm` $<$ `hot`.

Example 7

```
alpha-heuristic ordinal { 1, 10, 100, 1000, INFINITE } [1]
```

Finally in this example it is shown that you can mix and match numerical and string values.

Tip

The primary benefit of encoding something as an ordinal is that it can allow better inferences about unseen parameter values. With a categorical parameter, the knowledge of one value doesn't tell one much (if anything) about other values, where as with an ordinal value we would expect closer values (with respect to ordering) to be more related (or similar).

Numeric parameters

Numerical parameters (both Real and Integer) are specified as follows:

Integer

```
<parameter_name> integer [<min value>, <max value>] [<default value>] [log]
```

Real

```
<parameter_name> real [<min value>, <max value>] [<default value>] [log]
```

Example 8

```
sp-rand-var-dec-scaling real [0.3, 1.1] [1]
```

Parameter `sp-rand-var-dec-scaling` is real-valued with a default value of 1, and we can choose values for it from the (closed) interval `[0.3, 1.1]`. Note that there may be other parameter values outside this interval that are in principle legal values for the parameter (e.g., your solver might accept any positive floating point value for the parameter). What you specify here is the range that automated configuration procedures should search (i.e., a range you expect a priori to contain good values); of course, every value in the specified range must be legal. There is a tradeoff in choosing the best range size.

Example 9

```
mult-factor integer [2, 15] [5]
```

Parameter `mult-factor` is integer-valued, takes any integer value between `2` and `15` (inclusive), and has a default value of 5. Parameter `mult-factor` could also be specified as an ordinal parameter with values {`2`,`3`,`4`,`5`,`6`,`7`,`8`,`9`,`10`,`11`,`12`,`13`,`14`,`15`}. In most situations the representations are identical (the integer setting might be slightly faster), if you are using forbidden or conditional parameter settings however the distinction matters.

Example 10

```
DLSc real [0.00001, 0.1] [0.01] log
```

Parameter `DLSc` is real-valued with a default value of `0.01`, and we can choose values for it from the (closed) interval `[0.00001, 0.1]`. The trailing '`log`' denotes that this parameter naturally varies on a log scale. If we were to discretize the parameter, a natural choice would be {`0.00001, 0.0001, 0.001, 0.01, 0.1`}. That means, a priori the distance between parameter values `0.001` and `0.01` is identical to that between `0.01` and `0.1` (after a $\log_{10}$ transformation, `0.001`, `0.01`, and `0.1` become -3, -2, and -1, respectively). We express this natural variation on a log scale by the '`log`' flag.

Example 11

```
first-restart integer [10, 1000] [100] log
```

Parameter `first-restart` is integer-valued with a default value of `100`, and we can choose values for it from the (closed) interval `[10, 1000]`. It also varies naturally on a logarithmic scale. For example, due to this logarithmic scale, after the transformation drawing a uniform random value of `first-restart` will yield a number below `100` half the time.

Restrictions

- Numerical integer parameters must have their lower and upper bounds specified as integers, and the default must also be an integer.
- The bounds for parameters with a log scale must be strictly positive.

4.4.2 Conditional Parameter Clauses

Depending on the instantiation of some 'higher-level' parameters, certain 'lower-level' parameters may not be active. For example, the subparameters of a heuristic are not important (i.e., active) if the heuristic is not selected. All parameters are considered to be active by default, and conditional parameter clauses express under which conditions a parameter is active. The syntax for conditional parameter clauses is as follows:

```
<child name> | <parent name1> <operator1> <value1> (&& or ||)
<parent name2> <operator2> <value2>
```

`<operator>` can be one of `==, !=, <, >, in`. The $>$ and $<$ operators are only defined for ordinal, integer, or real types. `<value>` is either a specific allowed value for the parameter for the `==, !=, <, >` operators, or for the `in` operator it is a set of values (i.e., {`a,b,c`}). Finally, you can specify conjunctions or disjunctions for parent parameters. There is no support for parenthesis with conditionals. The `&&` connective has higher precedence than `||`, so `a||b&& c||d` is the same as: `a||(b&&c)||d`.

This can be read as "The child parameter `<child name>` is only active if the parent parameter `<parent name>` satisfies the logical conditional."

Tip

- Parameters that are not listed as a child parameter in any conditional parameter clause are always active.
- A child's name can appear only once.

Example 12

```
sort-algo categorical {quick,insertion,merge,heap,stooge,bogo} [bogo]
quick-selection-method categorical {first, random, median-of-medians} [random]
quick-selection-method | sort-algo in {quick}
```

In this example, `quick-selection-method` is conditional on the `sort-algo` parameter being set to `quick`, and will be ignored otherwise.

Example 13

```
heur1 categorical { off, on } [on]
heur2 categorical { off, on } [on]
heur_order categorical  { heur1then2, heur2then1 } [heur1then2]
heur_order | heur1 == on && heur2 == on
```

In this example, the `heur_order` parameter requires both of `heur1` and `heur2` to have the value of `on`.

Example 14

```
temperature real [-273.15, 100] [10]
rain real [0, 200] [0]
gloves ordinal { none, yarn, leather, gortex } [none]
gloves | rain > 0 || temperature < 5
```

In this example, `gloves` is only active if the `rain` is greater than 0 or `temperature` is less than 5.

4.4.3 Forbidden Parameter Clauses

Forbidden Parameters are combinations of parameter values which are invalid (e.g., a certain data structure may be incompatible with a lazy heuristic that does not update the data structure, resulting in incorrect algorithm behaviour). Configuration methods should never try to run an algorithm with a forbidden parameter configuration. The syntax for forbidden parameter combinations is as follows:

Classic Syntax

```
{<parameter name 1>=<value 1>, ..., <parameter name N>=<value N>}
```

Example 15

```
DSF categorical {DataStructure1, DataStructure2, DataStructure3}[DataStructure1]
PreProc categorical {NoPreProc, SimplePreproc, ComplexPreproc}[ComplexPreproc]
{DSF=DataStructure2, PreProc=ComplexPreproc}
{DSF=DataStructure2, PreProc=SimplePreproc}
{DSF=DataStructure3, PreProc=ComplexPreproc}
```

In this example, there are different data structures and different simplifications. `DataStructure2` is incompatible with `ComplexPreproc`, and `DataStructure2` is incompatible with both `SimplePreproc` and `ComplexPreproc`.

Warning

The default parameter setting is not allowed to contain a forbidden combination of parameter values. This is true even if the parameters in question are inactive.

Advanced Syntax An advanced syntax for forbidden clauses is available that allows specifying any expression of parameter settings as a forbidden.

The syntax is:

```
{ <expression> }
```

Example 16 If only the points within a unit sphere are desired one can use the following PCS specification:

```
x real [-1,1] [0]
y real [-1,1] [0]
{ x^2+y^2 > 1 }
```

Example 17 Additionally to specify a range of parameters like $0 < a < b < c < 1$

```
a real [0,1]
b real [0,1]
c real [0,1]
{ (a > b) || (b > c) }
```

Using the Advanced Syntax To understand all the limitations and restrictions of the advanced syntax it is probably simplest to explain how it is implemented. The advanced syntax utilizes exp4j (http://www.objecthunter.net/exp4j/) to parse and evaluate mathematical expressions. The expression which consists of variables (parameters above) are populated with their corresponding values, and then evaluated, if the expression evaluates to 0.0 then it is considered `false`, otherwise it is treated as `true`. In this case `true` means that the parameter setting is forbidden. With categorical and ordinal parameters, exp4j will evaluate the expression as a constant that is either the value of the parameter, if it is a number, or a suitable constant that will ensure proper ordering (e.g., an ordinal ranking of cold, warm, hot will have values assigned to variables cold, warm and hot such that cold $<$ warm $<$ hot).

The following lists the available operators[9]:

Arithmetic Operators	+, - , * , , ^, Unary +/-, and %
Functions	abs,acos,asin,atan,cbrt,ceil,cos,cosh,exp,floor,log,log10,log2,sin,sinh,sqrt,tan,tanh
Logical Operators	$>=$, $<=$, $>$, $<$, $==$, $!=$, `||`, `&&`

[9]The Logical Operators are not included by in exp4j but have been added here

Important!

As a result of how forbidden parameters are implemented there are some things to keep in mind when using them.

1. Many procedures rely on generating random configurations, and the procedure is not aware of forbidden parameters. If a random configuration is forbidden, another random sample is obtained until it is valid. This can cause a significant slowdown (e.g., with integer parameters the expression $\{a^2 + b^2 == c^2\}$ which would only select pythagorean triples), or cause the procedure to stop entirely (e.g., $\{a^3 + b^3 == c^3\}$ which is never satisfiable). Try to minimize the area of the space that is forbidden. For example given the forbidden expression $\{a < -1 || a > 1\}$ the parameter settings `a real [-1,1] [0]` and `a real [-10,10] [0]` are equivalent, but the latter will be 10 times slower.

2. Even if forbidden clauses are contradictions and thus could never be true, evaluating them involves significant time (in a toy example, we observed it resulted in a $4\times$ slowdown) in the time it takes to generate a random configuration. If your procedure does get many target algorithm runs completed, or they are especially short, this time may become significant. Additionally, a much smaller but still significant effect was observed between having multiple advanced forbidden statements versus using the `&&` operator, with the latter being slightly faster.

3. Errors involving categorical and ordinal parameters are not caught. Using Example 14, the following is a legal expression { `gloves+log(leather) <= rain^2` }, but its result is undefined, as far as this specification is concerned. For categorical only the $==$ and $!=$ operators should be used. For ordinals you can additionally use $>=$, $<=$, $>$, $<$.

4. Real parameters should not use the $==$ and $!=$ operators, as they will almost always evaluate to false and true respectively.

5. The names of parameters, as well as the values for categorical and ordinal parameters are restricted. All parameter names must start with a letter or the underscore and can only include letters, digits or underscores. The same restriction applies to values, except they can additionally be numeric.

6. When expressions are evaluated all variables exist within the same name space, this means that all of the parameter names as well as the non-numeric values for ordinal and categorical parameters must be treated the same. Again delving into the implementation a bit, what this means is that every parameter name, and all the values for ordinals and categoricals (including the numeric values) must be topologically sortable, considering the ordinal parameters. For instance the following two lines when put together violate this requirement `a ord { off, on } [on]` and `b ord { on, off } [on]`, because in the first on is greater than off, and in the second the order is reversed. An additional example is that the following is illegal when using advanced forbidden syntax `a ord { 1000,100,10,1 } [1]`, this is because as specified $1 > 10$, but numerically $10 > 1$. Like the restrictions on names, these restrictions only apply when a forbidden clause using the advanced syntax is specified.

7. You can use the `pcs-check` utility included in SMAC v2.10 or later which will check for many of these restrictions.

5 Wrappers

5.1 Algorithm executable / wrapper

The target algorithm as specified by the **algo** parameter must obey the following general contracts. While modifying your own code to directly achieve this is one option there are other methods outlined in section 5.3.

5.1.1 Invocation

The algorithm must be invokable via the system command-line using the following command with arguments:
`<algo_executable> <instance_name> <instance_specific_information> <cutoff_time> <cutoff_length> <seed> <param> <param> <param>...`

algo_executable Exactly what is specified in the **algo** argument in the scenario file.

instance_name The name of the problem instance we are executing against.

instance_specific_information An arbitrary string associated with this instance as specified in the **instance_file** . If no information is present then a "0" is always passed here.

cutoff_time The amount of time in seconds that the target algorithm is permitted to run. It is the responsibility of the callee to ensure that this is obeyed. It is not necessary that that the actual algorithm execution time (wall clock time) be below this value (*e.g.*If the algorithm needs to cleanup, or it's only possible to terminate the algorithm at certain stages).

cutoff_length A domain specific measure of when the algorithm should consider itself done.

seed A positive integer that the algorithm should use to seed itself (for reproducibility). "-1" is used when the algorithm is **deterministic**

param A setting of an active parameter for the selected configuration as specified in the Algorithm Parameter File. SMAC will only pass parameters that are active. Additionally SMAC is not guaranteed to pass the parameters in any particular order. The exact format for each parameter is:
`-name value`[10]

All of the arguments above will always be passed, even if they are inapplicable, in which case a dummy value will be passed.

Environment Variables

Recent versions of SMAC also set the following environment variables, which shouldn't be considered input to the solver but relate to the execution in some way. When implementing your wrapper you can entirely disregard this section unless you need some advanced features.

[10]The target algorithm will see the value as a single argument, even if it contains spaces or should otherwise be treated as more than one argument, i.e. to execute this in a shell you would see -name 'value', to ensure that the value is passed as a single argument. Older versions also passed the single quote

Environment Variable	Purpose
AEATK_CONCURRENT_TASK_ID	A zero indexed value of which concurrent run SMAC is executing. No two concurrent runs will see the same value, but subsequent runs will see the same value. This is mainly intended to allow the wrapper to manage CPU affinities.
AEATK_SET_TASK_AFFINITY	This environment variable is NOT set by SMAC, or used by SMAC but is used internally by some wrappers to ensure that AEATK_CONCURRENT_TASK_ID is read and the task is tied to a specific core. It is strongly recommended that you set this value to "1" which your wrapper then reads to know to set the affinity properly. This isn't enabled by default, because some clusters, notably SGE do not set job affinities properly and so parallel jobs will get tied to the same core.
AEATK_PORT	The wrapper can send in progress up dates to SMAC to the localhost via UDP on this port. The message format is a single double value. While SMAC doesn't directly use this presently, other utilities such as `algo-test` do, and future versions may, and some advanced options may preform better with this set.
AEATK_CPU_TIME_FREQUENCY	Signifies how often updates to the runtime should be sent. This is only a hint, and roughly should be treated as: there is no point in sending updates more frequently than this value in seconds.
AEATK_EXECUTION_UUID	A UUID associated with the particular invocation of the wrapper. The primary purpose of this is facilitate identify every process associated with a specific invocation of a wrapper on a particular computer. This is primarily used in conjunction with the **–cli-kill-by-environment-cmd** option. If this environment variable already exists, then another one will be chosen, so do not rely on this particular variable being set. You should ensure that environment variables are passed to all sub processes however, so that they can be killed accordingly.

5.1.2 Output

The Target Algorithm is free to output anything, which will be ignored but must at some point output a line (only once) in the following format[11]:

`Result of this algorithm run: <status>, <runtime>, <runlength>, <quality>, <seed>, <additional rundata>`

status Must be one of `SAT` (signifying a successful run that was satisfiable), `UNSAT` (signifying a successful run that was unsatisfiable), `TIMEOUT` if the algorithm didn't finish within the allotted time, `CRASHED` if something untoward happened during the algorithm run, or `ABORT` if something prevents the target algorithm for successfully executing and it is believed that further attempts would be futile.

SMAC does not differentiate between `SAT` and `UNSAT` responses, and the primary use of these is historical and serves as a check that the algorithm is executing correctly by outputting whether the

[11] Other strings are also permissible, but will one day be replaced. Most notably "Result for ParamILS:"

instance in question is satisfiable or not. See the **--verify-sat** option for information on how to utilize this feature.
NOTE: SMAC by default crashes if the wrapper ever reports SAT and UNSAT for the same instance across runs. Occasionally edge cases in exposed parameters are tripped and turn a solver buggy, and so this safe guard exists to help detect if this is occurring. To change this behaviour use the **--check-sat-consistency** and **--check-sat-consistency-exception** options.

SMAC also supports reporting `SATISFIABLE` and `SUCCESS` for `SAT` and `UNSATISFIABLE` for `UNSAT`.
NOTE: These are only aliases and SMAC will not preserve which alias was used in the log or state files.

`ABORT` can be useful in cases where the target algorithm cannot find required files, or a permission problem prevents access to them. This will cause SMAC to stop running immediately. Use this option with care, it should only be reported when the algorithm knows for CERTAIN that subsequent results may fail. For things like sporadic network failures, and other cosmic-ray induced failures, one should consider using `CRASHED` in combination with the **--retry-crashed-count** and **–abort-on-crash** options, to mitigate these.

In other files or the log you may see the following following additional types used. `RUNNING` which signifies a result that is currently in the middle of being run, and `KILLED` which signifies that SMAC internally decided to terminate the run before it finished. These are internal values only, and wrappers are NOT permitted to output these values. If these values are reported by the wrapper, it will be treated as if the run had status `CRASHED`.

runtime The amount of CPU time used during this algorithm run. SMAC does not measure the CPU time directly, and this is the amount that is used with respect to **tunerTimeout**. You may get unexpected performance degradation when this amount is heavily under reported [12].

NOTE:The **runtime** should always be strictly less than the requested **cutoff_time** when reporting `SAT`or `UNSAT`. The runtime must be strictly greater than zero (and not NaN).

If an algorithm reports `TIMEOUT` or `CRASHED` the algorithm can report the actual CPU time used, and SMAC will treat it correctly as a timeout for optimization purposes, but count the actual time for **--tunertime-limit** purposes.

runlength A domain specific measure of how far the algorithm progressed. This value must be from the set: $-1 \cup [0, +\infty]$.

quality A domain specific measure of the quality of the solution. This value needs to be from the set: $(-\infty, +\infty)$.

NOTE: Keep in mind that SMAC will attempt to *minimize* this value. If you would like to maximize this value, your wrapper should subtract your quantity from some constant known to be larger than any value.

NOTE: In certain cases, such as when using log transforms in the model, this value must be: $(0, +\infty)$.

seed The seed value that was used in this target algorithm execution.

[12] This typically happens when targeting very short algorithm runs with large overheads that aren't accounted for.

NOTE: This seed argument is ignored by SMAC as of version 2.06.02. SMAC will always use the requested seed internally and so you can ignore this output. Note previous versions, as well as ParamILS and other applications may still require that this matches.

additional rundata A string (not containing commas, or newline characters) that will be associated with the run as far as SMAC is concerned. This string will be saved in run and results file (Section 6.2). NOTE:**additional rundata** is not compatible with ParamILS at time of writing, and so wrappers should not include this or the preceding comma if they wish to be compatible.

All fields except for **additional rundata** are mandatory. If the field is not applicable for your scenario a 0 can be substituted.

5.2 Wrapper Output Semantics

As SMAC is entirely insulated from the target algorithm execution by the wrapper it is up to the wrapper to ensure that constraints with respect to the cutoff and runlength are enforced. Occasionally wrappers may not properly enforce these constraints and SMAC will need to somehow handle these cases. The following table outlines how SMAC transforms these values and details what value is used in various parts of SMAC. In future versions some parts of this table may in fact change, and so it is best to ensure that your wrapper is well behaved.

NOTE: The cutoff time in the table is the amount of time SMAC schedules the run for, the scenario cutoff time is denoted as κ_{max}.

A description of the columns is as follows:

Tuner Time The amount of time that will be subtracted from the remaining tuner time limit given in the scenario.

PAR10 Score The value that will be used for empirical comparisons between configurations.

Model The value that will be used to build the model.

5.2.1 Runtime Optimization Semantics

status	**cutoff** (κ)	**runtime** (r)	Tuner Time	PAR10 Score	Model
*	*	$(-\infty, 0)$	**EXCEPTION THROWN**		
ABORT	*	$[0, \infty)$	**EXCEPTION THROWN**		
CRASHED	*	$[0, \infty)$	r	$10 \cdot \kappa_{max}$	$10 \cdot \kappa_{max}$
SAT, UNSAT	$\kappa \leq \kappa_{max}$	$[0, 0.1]$	0.1	r	r
SAT, UNSAT	$\kappa \leq \kappa_{max}$	$[0.1, \kappa)$	r	r	r
SAT, UNSAT	$\kappa \leq \kappa_{max}$	$[\kappa, \kappa_{max})$	r	r	r
SAT, UNSAT	$\kappa \leq \kappa_{max}$	$[\kappa_{max}, \infty)$	r	$10 \cdot \kappa_{max}$	$10 \cdot \kappa_{max}$
TIMEOUT	$\kappa < \kappa max$	$[0, \infty)$	r	κ	κ
TIMEOUT	$\kappa = \kappa max$	$[0, \infty)$	r	$10 \cdot \kappa_{max}$	$10 \cdot \kappa_{max}$

5.2.2 Solution Quality Optimization Semantics

For solution quality optimization, the Tuner Time calculation is the same as runtime. For the Model we always report the reported quality, except for `CRASHED` runs, in which case we change the value reported to $\min(10^9, \texttt{observedValue})$. You can disable this transformation by **--transform-crashed-quality** to `false`, you can modify the value that is used with **--transform-crashed-quality-value**.

NOTE: Unlike runtime optimization, if a `TIMEOUT` value has a low quality score, it will be considered good by SMAC. As such TIMEOUT values should generally have a high value.

5.3 Wrappers & Native Libraries

In order to optimize an algorithm, SMAC needs a method of invoking it. While modifying the code to manage the timing and input mechanisms manually is possible, this can sometimes be invasive and difficult to manage. There exist three other methods that one could consider using.

Wrappers Executable Scripts that manage the resource limits automatically and format the specified string into something usable by the actual target algorithm. This approach is probably the most common, but among its drawbacks are the fact that they often rely on third party scripting languages, and for smaller execution times have a large amount of overhead that may not be accounted for as far as the **tunerTimeout** limit is concerned. Most of the examples included in SMAC use this approach, and the wrappers included can be adapted for your own projects.

NOTE: When writing wrappers it is important not to poll the output stream of the target algorithm, especially if there is lots of output. Doing so often results in lock-contention and significantly modifies the runtime performance of the algorithm enough that the resulting configuration is not well tuned to the real algorithm's performance.

Inter Process Communication Introduced in SMAC v2.06.02, the **IPC** TAE can be selected via the **--tae** option. It will essentially use various forms of interprocess communication to notify some other process about the run to do, and will wait for it to respond. Presently the only existing mechanism is over UDP, but other methods are possible. For example:

```
./smac --scenario-file <file> --seed 1 --tae IPC
--ipc-mechanism UDP --ipc-remote-port 5050
--ipc-remote-host localhost
```

Will use sent the required information to localhost on port 5050. It will then wait for a response, and parse that response as a string. For more details see the section "Inter-Process Communication Target Algorithm Evaluator Options", in the Appendix.

Target Algorithm Evaluators This is probably the most powerful, but also the most complicated approach. SMAC is architected in a way that makes it fairly simple to replace the mechanism for execution with something completely custom. This can be done without even recompiling SMAC by creating a new implementation of the `TargetAlgorithmEvalutor` interface, which is responsible for converting run requests (`RunConfig` objects) into run results (`AlgorithmRun` objects). Both the input and output objects are simple *Value Objects* so the coupling between SMAC and the rest of your code is almost zero with this approach. For more information see **??**

6 Interpreting SMAC's Output

SMAC outputs a variety of information to log files, trajectory files, and state files. Most of the files are human readable, and this section describes these files.
NOTE: All output is written to the **outdir** in the **--rungroup** sub-directory.

6.1 Logging Output

SMAC uses slf4j (http://www.slf4j.org/), a library that allows for abstracting and replacing the logging system with ease, and uses logback (http://logback.qos.ch/) as the default logging system. While there is limited ability to change logging options via the command line (*e.g.*, **--log-level**, **--console-log-level**, **--log-all-call-strings**,**--log-all-process-output**). It is possible that by setting a system property[13] you can override the configuration by using `-Dlogback.configurationFile=/path/to/config.xml`

NOTE: If you replace the logger in SMAC or modify the configuration file, the logging command line options may no longer work.

By default SMAC writes the following logging files out to disk (NOTE: The N represents the **--seed** setting):

log-run$N$.txt A log file that contains a full dump of all the information logged, and where it was logged from.

log-warn$N$.txt Contains the same information as the above file, except only from warning and higher level messages.

log-err$N$.txt Contains the same information as the above file, except only from error messages.

6.1.1 Interpreting the Log File

SMAC basically goes through three phases when executing:

- Setup Phase Input files are read, and their arguments validated. Everything necessary to execute the Automatic Configuration Phase is constructed. This phase ends, once the following message appears:

  ```
  SMAC started at: 10-Apr-2014 10:01:40 AM. Minimizing penalized average runtime (PAR10)
  ```

- Automatic Configuration Phase: SMAC is now actively configuring the target algorithm. SMAC will spend most of it's time here, and outputs it's progress.

  There are two types of messages you will see here:

  ```
  1) Incumbent changed to: config 2 (internal ID: 0x7), with penalized average runtime (PAR10): 7.1;
   estimate based on 2 runs.
   Sample call for new incumbent config 2 (internal ID: 0x7):
  cd saps; ruby saps_wrapper.rb instances/train/SWlin2006.19724.cnf 0 10.0 2147483647 -1
   -alpha '1.1' -ps '0.1' -rho '0.84' -wp '0.06'
  ```

[13] You'd have to edit the start up script smac or smac.bat

This signifies that the incumbent (the best configuration found so far), has been changed to configuration 2 (this ID is used in some files that SMAC will output). It also gives a sample estimate of the performance of this configuration on the instances we have seen already.

Next a sample call is given for this configuration so that you can test it yourself. From this you can determine the actual configuration selected, in this example it is:

-alpha '1.1' -ps '0.1' -rho '0.84' -wp '0.06'

```
2) Updated estimated penalized average runtime (PAR10) of the same incumbent: 3.86;
estimate now based on 4 runs.
```

As SMAC continues to run it will continue refining the estimate by making more samples of the incumbents configuration, and it will occasionally provide you with an update. This number can vary wildly as SMAC learns more about your instance distribution.

- Once SMAC is completed, it will output some summary statistics:

```
================================================================================
SMAC has finished. Reason: total CPU time limit (600.0 s) has been reached.
SMAC's final incumbent: config 45 (internal ID: 0xBB),
 with estimated penalized average runtime (PAR10): 0.12 ,
 based on 5 run(s) on 5 training instance(s).
Total number of runs performed: 171, total CPU time used: 604 s,
 total wallclock time used: 607 s, total configurations tried: 113.
================================================================================
```

The first line indicates why SMAC terminated, in this case the CPU time limit was exceeded. It provides an updated estimate of the objective, in this case 0.12.

- Offline Validation Phase, depending on the options used this can also take a large fraction of SMAC's runtime. The logic here is actually quite simple, as it largely only requires running many algorithm runs and computing the objectives from them.

```
-----------------------------------------------------
Minimized penalized average runtime (PAR10):
Final time: 607 config 45 (internal ID: 0xBB):  0.14 on the test set

Sample call for the final incumbent:
cd saps; ruby saps_wrapper.rb instances/train/SWlin2006.19724.cnf 0 10.0 2147483647 -1
 -alpha '1.1' -ps '0.1' -rho '0.84' -wp '0.06'

Additional information about run 1 in: smac-output/run1/
-----------------------------------------------------
```

6.2 State Files

State files allow you to examine and potentially restore the state of SMAC at a specific point of it's execution. The files are written to the state-run$N$/ sub-directory, where N is the value of **--seed** option.

All files have the following convention as a suffix either `it` or `CRASH` followed by either the iteration number M, or in some cases `quick` or `quick-bak`.

The state is saved for every iteration m, where $m = 2^n$ $n \in \mathbb{N}$, additionally it is saved when SMAC completes whether successfully or due to crash.

The following files are saved in this state directory (ignoring the suffix):

java_obj_dump-v2-it$M$.obj Stores (Java) serialized versions of the the incumbent and the random object state. In general there is no need to look at this file, and it is not human readable.

paramstrings-it$M$.txt Stores a human readable setting of each configuration ran, with a prefix of the numeric id of the configuration (as used in the logs, and other state files).

uniq_configurations-M.csv Stores the configurations ran in a more concise but effectively un-human readable form. The first column again is the numeric id of the configuration (as used in the logs, and other state files).

runs_and_results-it$M$.csv Stores the result of every run of the target algorithm that SMAC has done. The first 13 columns (after the header row are designed to be backwards compatible with SMAC versions 1.xx. Each column is labelled with what data it contains, the following columns deserve some description.

`Instance ID` This is the instance used, and is the n^{th} **Instance Name** specified in the **instance_file** option.

`Response Value(y)` This is the value determined by the **run_obj** on the run.

`Censored` Indicates whether the `Cutoff Time Used` field is less than the **cutoff_time** in the original run. 0 means `false`, 1 means `true`.

`Run Result Code` This is a mapping from the `Run Result` to an integer for use with previous versions.

param-file If **--save-context** is enabled, a copy of the **paramfile** will be in the state folder

instances If **--save-context** is enabled, a copy of the **instance_file** will be in the state folder

instance-features If **--save-context** is enabled, and SMAC is running with features, then a copy of the **feature_file** will be in the state folder.

scenario If **--save-context** is enabled, and SMAC is using a scenario file, then a copy of the **--scenario-file** will be in the state folder.

6.3 Trajectory File

SMAC also outputs a trajectory file into the `detailed-traj-run-`N`.csv` and outline the incumbent (by id) over the course of execution and it's performance. The first line gives the **–rungroup**, and then the **–seed**.

The rest of the file follows the following format:

Column Name	Description
CPU Time Used	Sum of all execution times and CPU time of SMAC
Estimated Training Performance	Performance of the Incumbent under the given **–run-obj** and **–overall-obj**
Wallclock Time	Time of entry with respect to wallclock time.
Incumbent ID	The ID of the incumbent as listed in the **param_strings** file in §6.2, and the logs
Automatic Configurator (CPU) Time	CPU Time used of SMAC
Full Configuration	The full configuration of this incumbent

NOTE: SMAC also outputs `traj-run-`N`.txt` the first five columns are the same, but the remaining

columns represent the configuration, with each cell being a key and value. This is identical the trajectory file outputted by ParamILS.

6.4 Validation Output

When Validation is completed four files are outputted, (again N is the value of the **--seed** argument). Finally depending on which options are used the, especially with `smac-validate`, the actual file name outputted may vary.

1. `validationResults-traj-run-`N`-walltime.csv`: A CSV file that contains a summary of the results of validation, the *Validation Configuration ID* maps to a line in the next file
2. `validationCallStrings-traj-run-`N`-walltime.csv`: A CSV file containing a mapping between *Validation Configuration ID* to the actual configuration, and a sample call string.
3. `validationPerformanceDebug-traj-run-`N`-walltime.csv`: A CSV file that contains a detailed breakdown of how the final validation score was obtained. This file is meant for human consumption, and not for parsing.
4. `validationObjectiveMatrix-traj-run-`N`-walltime.csv`: A CSV file that contains a table, for each row (configuration) the objective for the problem instance seed pair as given in the column.
5. `validationRunMatrix-traj-run-`N`-walltime.csv`: A CSV file that contains a table, for each row (configuration) the response from the wrapper (ignoring the prefix).

6.5 JSON Output

For each run of SMAC, SMAC will also output a file `live-rundata-N.json`. The format of the file consists of an array representation of all of the problem instances. Then the individual runs are reported one after the other. The actual JSON representation is meant to be a mostly semantic view of the objects and is enough to construct all of the runs data.

7 Developer Reference

This section is meant as a guide to those who need to modify the SMAC code base for whatever reason.

7.1 Design Overview

The SMAC Application is broken up into three distinct projects as follows:

SMAC Contains all of the logic that is specific to SMAC, (*e.g.*Validation, the SMAC algorithm, construction of SMAC Objects). In essence it stitches together components of the Automatic Configurator Library. The sources are included in `smac-src.jar`.

Algorithm Execution & Abstraction Toolkit Contains all of the primary abstractions/models used by SMAC (*e.g.*Object representations for Instances, Target Algorithm Configurations & methods for executing algorithms,...). 90% of the code that SMAC uses lives in this library. It also contains code

for converting the data from these abstractions into input needed to build the model. These are shipped with SMAC in the `aclib-src.jar` file. **Note**: Historically this was called "aclib", and in this version of SMAC internally the code still referred to it as aclib. The next version of SMAC will likely rename it.

Random Forests The Random Forest model code. The sources are included in `fastrf-src.jar`.

The scope of this document governs only the first two projects. At the time of writing the **Algorithm Execution & Abstraction Toolkit** has no published documentation, but if you e-mail the above a draft version is available.

- The bulk of the code necessary to run SMAC lives in four classes `AbstractAlgorithmFramework`, `SequentialModelBasedAlgorithmConfiguration`, `SMACBuilder` and finally, `SMACExecutor`.

7.2 Class Overview

The most important classes to SMAC are as follows:

Algorithm Execution & Abstraction Toolkit	
Name	Description
AbstractOptions	Base class for creating new options for SMAC. While not important in and of itself, you will generally be implementing or modifying one of it's subtypes to implement options.
AlgorithmRun	Interface that represents the results of a target algorithm run. These are created by a TargetAlgorithmEvaluator. Outside of the TargetAlgorithmEvaluator these classes are generally immutable.
AlgorithmExecutionConfig	Immutable object containing the information required to invoke a target algorithm.
InstanceSeedGenerator	Interface that gets seeds for a ProblemInstance. These objects are constructed by ProblemInstanceHelper
ModelBuilder	Interface whose implementations should result in a constructed model.
OverallObjective	Aggregates many RunObjective values under some statistic (*e.g.*mean), to produce a value to be optimized.
ParamConfiguration	Class that represents a specific setting of the target algorithm's parameters. This class also implements the Map interface, though does not support all the required operations. The ID associated with is object, is used only for logging and should not be used in the code. Finally although this class is not immutable the general life cycle is that the object is created, given specific values, and then never changed again. In future this may be augmented with the ability to prevent writes. These objects are always constructed via the ParamConfigurationSpace.
ParamConfigurationSpace	(Almost immutable) class that represents the entire configuration space of a target algorithm. This class is constructed with the **Algorithm Parameter File** described in section 4.4. This class also contains the specifics of each parameter (*e.g.*domains, defaults, etc...). Currently the Random object used is the only portion that is mutable, and this will change in the future.
ProblemInstance	Immutable class that represents a specific problem instance, constructed by ProblemInstanceHelper.
ProblemInstanceSeedPair	Immutable class that represents a problem instance and seed. Decisions of which seed to use when scheduling a run are made in RunHistory.
RunConfig	Immutable class that represents a problem instance seed pair, and configuration to execute.
RunHistory	Interface that saves all the runs performed, and allows various queries against this information.
RunObjective	Converts an AlgorithmRun into a scalar value for optimization
SanitizedModelData	Converts the run data into a format to use when building the model. Other things such as PCA, and other data filtering are done here. This interface and mechanism will likely be refactored in the future as it is brittle at the moment.
SeedableRandomSingleton	A global random object whose existence is a convincing case that Singleton's are Anti-Patterns. This will, thankfully, go the way of the dodo bird at some point.
StateFactory	Interface that constructs StateSerializer & StateDeserializer to manage saving and restoring state respectively.
TargetAlgorithmEvaluator	Interface whose implementations should be able to run the algorithm (*i.e.* Implementations should convert RunConfig objects to AlgorithmRun objects). See section **??** for more information.

SMAC Library Classes	
Name	Description
`AbstractAlgorithmFramework`	*Non-abstract* class that provides a default Automatic Configurator (ROAR)
`SequentialModelBasedAlgorithmConfiguration`	Class that subtypes `AbstractAlgorithmFramework` and implements the methods required for SMAC
`SMACExecutor`	Parses command line options and creates some of the objects SMAC needs to execute (SMAC entrypoint)
`SMACBuilder`	Takes the options parsed by SMACExecutor or some other utility, and builds everything necessary to create an instance of `AbstractAlgorithmFramework`. If you want to plug smac into your application, you generally want to mimic what SMACExecutor does to invoke SMACBuilder.
`Validator`	Performs Validation of selected configurations
`ValidatorExecutor`	Entry point to stand alone validation utility

7.3 SMAC Sequence Diagram

See Figure 1 on page 32 for a sequence diagram relating the 5 main classes.

7.4 Algorithm Execution & Abstraction Toolkit

At some future point a better guide will be available, in the interim please e-mail the authors above for a draft.

7.5 Running SMAC in Eclipse

Depending on what you would like to do it might be better to ask for git repository access, which contains ant build scripts and an existing eclipse project. The following procedure however will get you a working installation in eclipse.

NOTE: This guide is pretty straight forward except for step 10 & 11, so if you are comfortable with Eclipse you should just skip to those steps.

To start the smac project project in eclipse do the following:

1. Create a new project in Eclipse, ensure that the JDK is 1.7 or higher.
2. Create a new source folder: `aeatk`
3. Create a new source folder: `smac`
4. Create a new source folder: `fastrf`
5. Create a new folder: `lib`

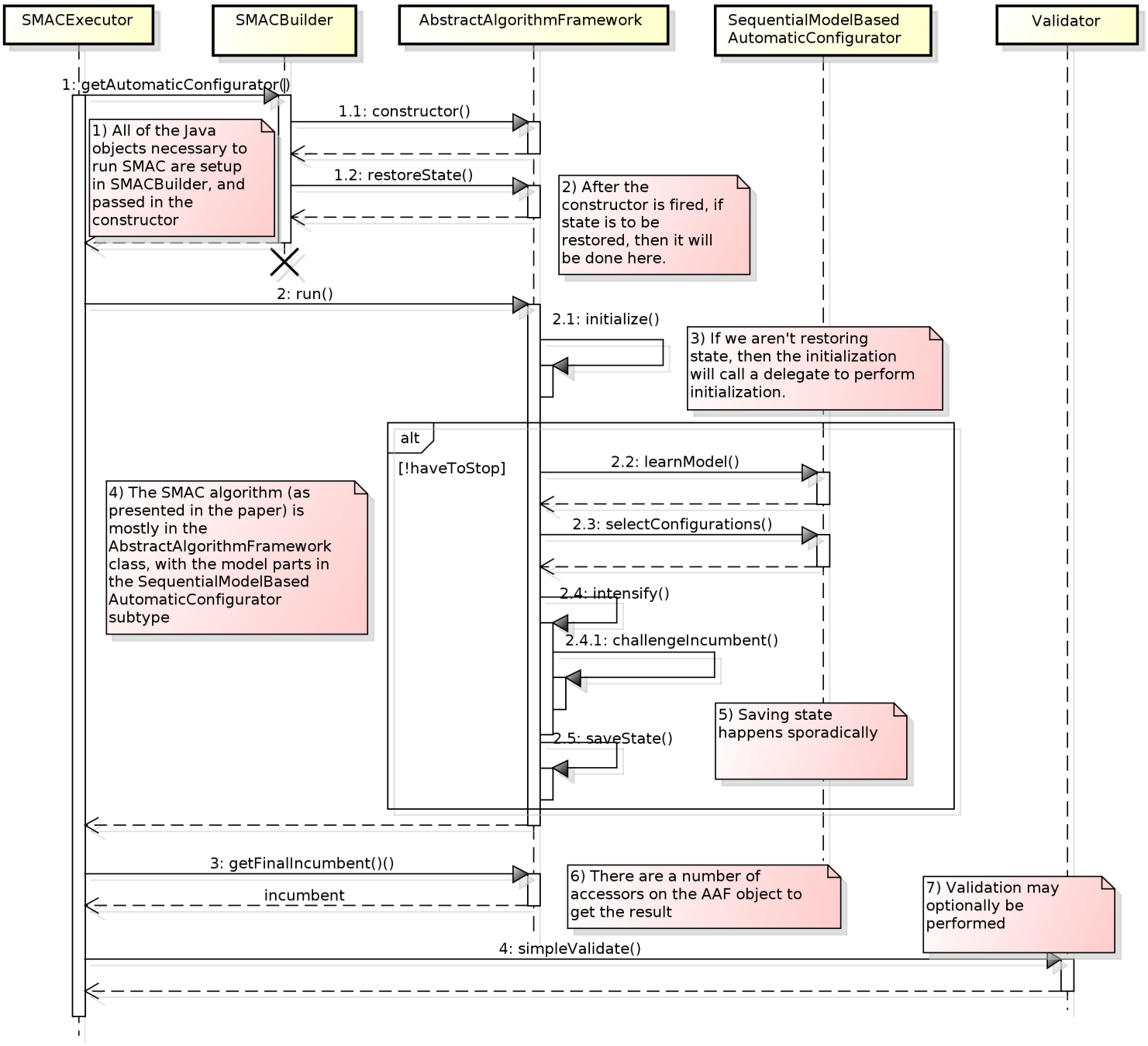

Figure 1: Sequence diagram detailing the execution of a standard SMAC run

6. Copy all the jar files from the `lib` folder of the SMAC release into the lib folder of the eclipse project, except for `smac.jar`, `smac-src.jar`, `aeatk.jar`, `aeatk-src.jar`, `fastrf.jar` and `fastrf-src.jar`.

7. Unzip the `smac-src.jar` into the smac source folder

8. Unzip the `aeatk-src.jar` into the aeatk source folder

9. Unzip the `fastrf-src.jar` into the fastrf source folder.

10. Right click on the project and go to Properties → Java Compiler → Annotation Processing and check the *Enable project specific settings* and *Enable annotation processing*.

11. Then in the project properties navigate to Java Compiler → Annotation Processing → Factory Path and hit *Add Jars* then select `lib/spi-0.2.4.jar` and hit OK.

    The entry point of any application can be retrieved from the shell script folder, for instance by opening the smac file we can see that the entry point is: `ca.ubc.cs.beta.smac.executors.SMACExecutor` and smac-validate is : `ca.ubc.cs.beta.smac.executors.ValidatorExecutor`

    Many of the helper utilities are contained in the `ca.ubc.cs.beta.aeatk.example.` subpackages.

NOTE: If you try and run existing scenarios packaged with SMAC they contain paths relative to the root of the smac dir. So in your *Run Configuration* you should set the *Working Directory* to the root of some smac release to run it as you would on the command line.

If when running SMAC you see either of the following errors:

WARNING: I could not find ANY Target Algorithm Evaluators on the classpath.
If you made this JAR (or setup the classpath) yourself chances are you did not setup SPI correctly. You must ensure that in the JAR (or on the classpath) there is a
META-INF/services/ca.ubc.cs.beta.aeatk.targetalgorithmevaluator.TargetAlgorithmEvaluatorFactory file
In this file should list every implementation of that interface

This means you did not follow steps 10 & 11 properly.

8 Acknowledgements

We are indebted to Jonathan Shen for porting our random forest code from C to Java in preparation for a Java port of all of SMAC. Alexandre Fréchette and Chris Thornton for their constant feedback and patches to SMAC. We would also like to thank Marius Lindauer for many valuable early bug reports and suggestions for improvements, as well as subsequent patches.

9 Appendix

9.1 Return Codes

NOTE: All error conditions besides 255 are fixed. However in future some exceptions that previously reported 255 may be changed to a non 255 value as needed / requested

Value	Error Name	Description
0	**Success**	Everything completed successfully
1	**Parameter Error**	There was a problem with the input arguments or files
2	**Trajectory Divergence**	For some reason SMAC has taken a unexpected path (*e.g.* SMAC executes a run that does not match a run in the **–runHashCodeFile**)
3	**Serialization Exception**	A problem occurred when saving or restoring state
255	**Other Exceptions**	Some other error occurred

9.2 Version History of Java SMAC

Version 2.00 (Aug-2012) First Internal Release of Java SMAC (this is a port and extension of the original Matlab version).

Version 2.02 (Oct-2012) First Public Release of SMAC v2 and contained many fixes from the previous release.

Version 2.04 (Dec-2012) Second Release of Java SMAC including the following improvements:

1. Validation file output times consistent with Tuner Times
2. Some **INFO** log statements have been moved to **DEBUG** and some **DEBUG** to **TRACE**
3. Added support for verifying whether responses of SAT and UNSAT are consistent with Instance Specific Information see **--verifySAT** option for more information
4. Added support for the SMAC_MEMORY environment variable to control how much RAM (in MB) SMAC will use when executed via the supplied shell scripts.
5. Context is now added to the state folders to make it easier to debug issues later, to disable consult the **--saveContext** option.
6. Greatly improved memory usage in State Serialization code, and now we free the existing model prior to building a new one, so for some JVMs this may improve memory usage.

Version 2.04.01 (Feb-2013) Minor Bug Fix of Java SMAC

1. Added option to validate over training set instances
2. Can now use <DEFAULT> as a configuration to validate against
3. Fixed bug where **TIMEOUT** runs below our requested cutoff time are not counted properly when considering incumbent changes
4. Can now specify the initial incumbent with the **--initialIncumbent** option.
5. Wallclock time is now saved in the trajectory file instead of -1
6. FAQ Improvements
7. Git commit hash is now documented in Manual, FAQ, and Version strings
8. **(BETA)** Support for bash auto-completion of arguments for `smac` and `smac-validate`. You can load the file by running:

   ```
   . ./util/bash_autocomplete.sh
   ```

Version 2.04.02 (Aug-2013) Minor Bug Fix of Java SMAC

1. Incumbent Performance now displayed when validation is turned off.
2. **--runtimeLimit** option is no longer just for show.

Version 2.06.00 (Aug-2013) Significant Feature Enhancements

1. New `algo-test` utility allows easy invocation of wrappers.
2. New `verify-scenario` utility preforms extra validation on scenario files.
3. Scenario now ends if the configuration space is exhausted
4. SMAC now lets you search only a subspace for good configurations
5. Validation output formats improved with headers
6. Option to always compare with the initial incumbent (to prevent an early poor choice from derailing the run) (See **--always-run-initial-config**)
7. SMAC reports an error if runs give different answers for **SAT** and **UNSAT** now
8. New **--restore-scenario** option to make restoring scenarios easier
9. New **--warmstart** option makes it possible to preload the model with additional SMAC runs.
10. Can now set seeds to different parts of SMAC using **-S**
11. Runtime Statistics and Termination Reasons now rewritten
12. New validation options **--validate-all**, **--validate-only-if-tunertime-reached** (See the validation options for all of them)
13. SMAC now checks limits *before* scheduling a run, rather than immediately after the run as in previous versions. (This means that if the last run went over, but changed the incumbent it will be logged.)
14. Instances can now be ordered deterministically (that is in the order they are declared in the instance file via **--determinstic-instance-ordering**.
15. Usage improved via new help levels which are displayed with **--help-level** and new usage screens.
16. Improvements to bash auto completion.
17. Target Algorithm Evaluators now take options.
18. Fixes for CPU Time calculation in SMAC.
19. Example scenarios cleaned up, new ones provided.
20. SMAC should be more forgiving with relative paths in a scenario file.
21. Default option files now supported (SMAC will read from `~/.aclib/smac.opt`, `~/.aclib/tae.opt` and `~/.aclib/help.opt`. It will also read from defaults for plugins that are available. **NOTE:** A future version changed the files to `~/.aeatk/`.
22. Rungroup name is now configurable.
23. Logging of some objects is cleaned up.
24. Windows Startup scripts, and improved Unix start up scripts.
25. Fixed lock-up issue with wrappers launching unterminating subprocesses.

26. Fixed ConvergenceException error message.
27. Options now have a primary non-camel case format.
28. Manual now has a basic options section, before listing all the options.
29. Significant API changes to the Target Algorithm Evaluators so previous plugins will need to be refactored (and another change will come either in v2.06 or v2.08).
30. SMAC will now match capitalization of words in the Result String of wrappers.
31. New **--validation-seed** option should cause the validation at the end of SMAC to behave the same as the stand-alone utility.

Version 2.06.01 (Oct-2013) Minor Bug Fix Release of Java SMAC

1. Fixed a bug introduced in 2.06.00 that caused validation to be performed against the training instance distribution instead of the test instance distribution.
2. Default acquisition function for solution quality optimization is now Expected Improvement (instead of Exponential Expected Improvement).
3. Fixed exception if Scenario file doesn't have extension.
4. New option **--terminate-on-delete** will cause SMAC to abort the procedure before the next set of runs (as if it had hit it's CPU time limit) if the file specified is deleted.
5. New option **--kill-runs-on-file-delete** will cause SMAC to kill any runs in progress . This option should be used with care, as it may cause SMAC to select the wrong incumbent, and it should always be used with **--terminate-on-delete**.
6. New option **--save-runs-every-iteration** will cause SMAC to output the runs and results file necessary to restore state every run. This is useful if your cluster or environment is particularly unreliable. It should NOT be used when runtimes in the scenario can grow very small as the amount of time SMAC will spend writing to disk loosely [14] changes from $O(n)$ to $O(n^2)$, where n is the number of runs it performs.
7. If SMAC is shutting down for an unexpected reason (e.g. `OutOfMemoryError` ,or it received a `SIGTERM`), SMAC will now try its best to write a final batch of state data with the "SHUTDOWN" prefix.
   NOTE: This state may be corrupted for a variety of reasons, and even if it is written correctly you may not be able to restore it properly as the snap shot may be from the middle of an iteration.
8. Fixed typo in error message that mistakenly reported that instances where missing, when in fact it was the test instances that were missing.

Version 2.08.00 (Aug-2014) Usability and Validation Changes

1. SMAC is now more picky about instance names and feature names matching.
2. New `sat-check` utility allows determination of the satisfiability for each instance of a instance file.
3. Environment variable **AEATK_CONCURRENT_TASK_ID** is now set when executing the wrapper, containing an index into the number of concurrent jobs. This is primarily used to allow the wrapper to determine CPU affinities correctly. See the wrapper section for more information.-

[14] Assuming the number of iterations scales linearly with the number of runs.

4. SMAC has been made drastically less verbose. The default level `INFO` now only contains the final information, and information about changes to the incumbent. `DEBUG` contains most of the old info level, `TRACE` contains most of the old `DEBUG` levels. The old `TRACE` level was never used and has been removed.
5. Instances can now be specified by folder using the **--instances** and **--test-instances** option. You can restrict which instances are used via the **--instance-suffix** and **--test-instance-suffix**
6. **--exec-dir** option now defaults to current working directory.
7. New option **--use-instances** will use a dummy instance instead of the instance file (useful for black box optimization).
8. New advanced option **--shared-model-mode** may improve performance in some cases, see Section 3.10.
9. **[BETA]** Target Algorithm Evaluator implementation allows integrating with a TAE using UDP/TCP (more to come).
10. Implemented a work around to a bug where configurations with censored early runs could become the incumbent erroneously. It's still suboptimal, but in fact it probably would never happen. See Known Issue #1 in section 9.3.
11. Validation rounding mode now changes the number of runs on deterministic runs, or runs with set problem instance seed pairs.
12. New option **--cli-kill-by-environment-cmd** allows terminating all processes by an environment variable. See section 5.1.1 for more information.
13. Target algorithms no longer see quoted arguments for parameter values. The option **--cli-call-params-with-quotes** can be used to get the old behaviour back, this option will likely be removed in future.
14. New option **--quick-saves** controls whether to make any quick save states or not.
15. New option **--intermediary-saves** controls whether to many any save states at all while SMAC is running (if false SMAC will still save information at the end)
16. Revamped Quickstart guide
17. After validation SMAC prints correct termination reason message.
18. SMAC will now terminate all outstanding runs when exiting prematurely (for instance due to **CTRL+C**)
19. Standardized scenario options (no new ones), but scenario options can be used in ParamILS versions 2.3.7 and later.
20. Mitigated bug that caused deterministic instances to take forever to load from file. This may still happen in some cases, if feature file names and instance file names do not perfectly match up.
21. Auto detect restore scenario option now made more robust in case files are missing
22. The state merge utility no longer crashes if merging runs that don't have a run for every instance
23. Renamed many references of ACLib to AEAToolkit to reflect change of name of the toolkit SMAC is built with.
24. Default options are now read from `~/.aeatk` instead of `~/.aclib`.
25. Fixed an issue with absolute paths on windows not being handled correctly.

26. Validation now performs 1 run per instance by default instead of next multiple after 1000.
27. Can now specify the number of cores that SMAC validate will use (only when using the local command line), using the **--validation-cores** option.
28. Previous state folder is now renamed to something that preserves the run name and is no longer a warning.
29. Emphasized in many places that SMAC is minimizing the objective functions.
30. SMAC now ignores the seed output in the response of wrappers entirely (it automatically substitutes the requested value. If your wrapper doesn't set this value correctly, you may notice discrepancies in SMAC.
31. A few validation options have been deprecated and removed
32. Can now validate multiple trajectory files in one pass using the **--trajectory-files** option.
33. Output format of validation has been completely changed to be more useful.
34. `traj-run-N.csv` is now `detailed-traj-run-N.csv` and has a slightly different format.
35. SMAC now requires Java version 7 to run.
36. `conf/logback.xml` is no longer used, and the file is stored internally. To override the configuration, set the java system property `logback.configurationFile=/path/to/config.xml`
37. Some columns in the trajectory file have been renamed for clarity. The order is still the same.
38. Changed default wrapper string to "Result of this algorithm run:".

Version 2.10.00 (May-2015) Feature Improvements

1. In shared model mode, SMAC can now reuse an existing run from another run, instead of re-running it.
2. Detailed Trajectory File now outputs predicted performance of the incumbent.
3. Fixed a bug with the LCB acquisition function that (probably) causes poor performance.
4. Added Mac OS X support for SAPS example scenario (thanks Chris Fawcett and Alexandre Fréchette).
5. Removed explicit SAPS Windows scenario, and changed the Linux one to detect windows.
6. Fixed a bug with solution quality optimization and large values greater than the cutoff time. **CRASHED** runs will now be penalized. For more details about how CRASHED runs are handled see Section 5.2.
7. Shared model mode will now log a message when a new file to read from is detected.
8. Fixed spear scenario to use instance features again.
9. SMAC will now throw an error if feature file specified and no features found.
10. Runs that are terminated because of taking too long will now be treated as **CRASHED** instead of **TIMEOUT**.
11. New PCS Syntax (old syntax still works fine) [Thanks to Marius].
12. Support for ordinals [Thanks to Marius].
13. More advanced support for conditionals [Thanks to Marius].

14. New Advanced Forbidden Syntax.
15. Vastly improved memory usage.
16. Fixed bug related to integer parameters being rerun.
17. Can now set **--num-ei-random** to zero.
18. Can now specify an initial list of challengers for SMAC using the **--initial-challengers**. [Thanks to Matthias]
19. SMAC will no longer call algorithm with zero cut off when using other TAEs.
20. Fixed `IllegalStateException` when dealing with large response values.
21. SMAC no longer leaves temp files on disk.
22. Fixed walltime reported incorrectly for some **CRASHED** runs.
23. New option **--num-ls-random** allows using random points for starting local search (defaults to zero).
24. License changed to AGPLv3.

Version 2.10.01 (June-2015) Minor Fix

1. Undeprecated **--use-scenario-outdir** after an outcry from users.

Version 2.10.02 (July-2015) Minor Fixes

1. Fixed bug with **--initial-incumbent-runs** not being able to be set higher than the number of instances (as opposed to number of instance and seed pairs.
2. Fixed bug with **--initial-challenger-runs** causing a crash in rare circumstances.
3. Fixed minor bug with pseudo-random number generation (PRNG) which caused random sequences used in the first iteration, to be replayed in the second. The most pronounced effect of this, was that random configurations generated in the second iteration would be identical to those in the first.
4. Added default directory for **--file-cache-source** and **--file-cache-output** of the folder runcache in the current working directory.

Version 2.10.03 (July-2015) Simple Bug Fix

1. Fixed issue with `smac-validate` not being able to validate detailed trajectory files.
2. **--use-scenario-outdir** is no longer hidden in UI.

9.3 Known Issues

1. In a rare case, configurations that are reinspected by SMAC after initially being rejected may continue their challenge when they otherwise shouldn't. If the configuration continues it's challenges successfully, prior to being the incumbent we will presently check all the runs, which is strictly more expensive than necessary.

2. Using any alias for **--showHiddenParameters**, **--help**, or **--version** as values to other arguments (*e.g.* Setting --runGroupName --help) does not parse correctly (This is unlikely to be fixed until someone complains).

3. Using large parameter values in continuous integral parameters, may cause loss of precision, and or crashes if the values are too big.
4. ArrayOutOfBoundsException occurs if not all instances have features
5. **--num-seeds-per-test-instance** and **--num-test-instances** are both broken currently and will probably be removed in the future.

9.4 Basic Options Reference

The following sections outline only the basic options

9.4.1 SMAC Options

General Options for Running SMAC

BASIC OPTIONS

--help show help

--help-level Show options at this level or lower

Default Value: BASIC
Domain: {BASIC, INTERMEDIATE, ADVANCED, DEVELOPER}

--validation perform validation when SMAC completes

Default Value: true
Domain: {true, false}

-v print version and exit

9.4.2 Scenario Options

Standard Scenario Options for use with SMAC. In general consider using the –scenarioFile directive to specify these parameters and Algorithm Execution Options

BASIC OPTIONS

--feature-file file that contains the all the instances features

--instances File or directory containing the instances to use for the scenario. If it's a file it must coform a specific format (see Instance File Format section of the manual), if it's a directory it you must also use the --instance-suffix option to restrict the match (unless all files have the same extension), and the instance list will be in sorted order.

REQUIRED
Default Value: null

--run-obj per target algorithm run objective type that we are minimizing

REQUIRED
Default Value: null
Domain: {RUNTIME, QUALITY}

--scenario-file scenario file

Domain: FILES

--skip-features If true the feature file will be ignored (if the feature file is required, this will cause an error, as if it was not supplied

Default Value: false
Domain: {true, false}

--test-instances File or directory containing the instances to use for the scenario. If it's a file it must coform a specific format (see Instance File Format section of the manual), if it's directory you must also use the --test-instance-suffix option to restrict the match (unless all files have the same extension), , and the instance list will be in sorted order

Default Value: null

9.4.3 Scenario Configuration Limit Options

Options that control how long the scenario will run for

BASIC OPTIONS

--cputime-limit limits the total cpu time allowed between SMAC and the target algorithm runs during the automatic configuration phase

Default Value: 2147483647
Domain: [0, 2147483647]

--runcount-limit limits the total number of target algorithm runs allowed during the automatic configuration phase

Default Value: 9223372036854775807
Domain: (0, 9223372036854775807]

--wallclock-limit limits the total wall-clock time allowed during the automatic configuration phase

Default Value: 2147483647
Domain: (0, 2147483647]

9.4.4 Algorithm Execution Options

Options related to invoking the target algorithm

BASIC OPTIONS

--algo-cutoff-time CPU time limit for an individual target algorithm run

Default Value: 1.7976931348623157E308
Domain: $(0, \infty)$

--algo-deterministic treat the target algorithm as deterministic

Default Value: true
Domain: {true, false}

--algo-exec command string to execute algorithm with

REQUIRED
Default Value: null

--pcs-file File containing algorithm parameter space information in PCS format (see Algorithm Parameter File in the Manual). You can specify "SINGLETON" to get a singleton configuration space or "NULL" to get a null one.

REQUIRED
Default Value: null

-T additional context needed for target algorithm execution (see TAE documentation for possible values, generally rare)

Default Value:

9.5 Complete Options Reference

9.5.1 SMAC Options

General Options for Running SMAC

BASIC OPTIONS

--help show help

Aliases: --help, -?, /?, -h

--help-level Show options at this level or lower

Aliases: --help-level
Default Value: BASIC
Domain: {BASIC, INTERMEDIATE, ADVANCED, DEVELOPER}

--validation perform validation when SMAC completes

Aliases: --validation, --doValidation
Default Value: true
Domain: {true, false}

-v print version and exit

Aliases: -v, --version

INTERMEDIATE OPTIONS

--adaptive-capping Use Adaptive Capping

Aliases: --adaptive-capping, --ac, --adaptiveCapping
Default Value: Defaults to true when –runObj is RUNTIME, false otherwise
Domain: {true, false}

--always-run-initial-config if true we will always run the default and switch back to it if it is better than the incumbent

Aliases: --always-run-initial-config, --alwaysRunInitialConfiguration
Default Value: false
Domain: {true, false}

--console-log-level default log level of console output (this cannot be more verbose than the logLevel)

Aliases: --console-log-level, --consoleLogLevel
Default Value: INFO
Domain: {TRACE, DEBUG, INFO, WARN, ERROR, OFF}

--deterministic-instance-ordering If true, instances will be selected from the instance list file in the specified order

Aliases: --deterministic-instance-ordering, --deterministicInstanceOrdering
Default Value: false
Domain: {true, false}

--exec-mode execution mode of the automatic configurator

Aliases: --exec-mode, --execution-mode, --executionMode

Default Value: SMAC
Domain: {SMAC, ROAR, PSEL}

--experiment-dir root directory for experiments Folder

Aliases: --experiment-dir, --experimentDir, -e
Default Value: <current working directory>

--initial-challenger-runs initial amount of runs to request when intensifying on a challenger

Aliases: --initial-challenger-runs, --initialN, --initialChallenge
Default Value: 1
Domain: (0, 2147483647]

--initial-challengers Can be specified multiple times. Every item is one additional initial challenger which will be used to challenge the incumbent prior to starting the actual optimization method. For the syntax, please see --initialIncumbent.

Aliases: --initial-challengers, --initialChallengers
Default Value: []

--initial-incumbent Initial Incumbent to use for configuration (you can use RANDOM, or DEFAULT as a special string to get a RANDOM or the DEFAULT configuration as needed). Other configurations are specified as: -name 'value' -name 'value' ... For instance: --quick-sort 'on'

Aliases: --initial-incumbent, --initialIncumbent
Default Value: DEFAULT

--initial-incumbent-runs initial amount of runs to schedule against for the default configuration

Aliases: --initial-incumbent-runs, --initialIncumbentRuns, --defaultConfigRuns
Default Value: 1
Domain: (0, 2147483647]

--log-level messages will only be logged if they are of this severity or higher.

Aliases: --log-level, --logLevel
Default Value: INFO
Domain: {TRACE, DEBUG, INFO, WARN, ERROR, OFF}

--num-run number of this run (used for file generation, etc). This also controls the seed.

Aliases: --num-run, --numrun, --numRun, --seed
Default Value: Randomly generated
Domain: [0, 2147483647]

--restore-scenario Restore the scenario & state in the state folder

Aliases: --restore-scenario, --restoreScenario
Default Value: null
Domain: FILES

--rungroup name of subfolder of outputdir to save all the output files of this run to

Aliases: --rungroup, --rungroup-name, --runGroupName

Default Value:

--save-runs-every-iteration if true will save the runs and results file to disk every iteration. Useful if your runs are expensive and your cluster unreliable, not recommended if your runs are short as this may add an unacceptable amount of overhead

Aliases: --save-runs-every-iteration
Default Value: false
Domain: {true, false}

--show-hidden show hidden parameters that no one has use for, and probably just break SMAC (no-arguments)

Aliases: --show-hidden, --showHiddenParameters

--validation-cores Number of cores to use when validating (only applicable when using local command line cores). Essentially this changes the value of --cli-cores and --cores after SMAC has run. The use of this parameter is undefined if the TargetAlgorithmEvaluator being used is not the CLI

Aliases: --validation-cores
Default Value: The value of –cores
Domain: (0, 2147483647]

--warmstart location of state to use for warm-starting

Aliases: --warmstart, --warmstart-from
Default Value: N/A (No state is being warmstarted)

ADVANCED OPTIONS

--ac-add-slack amount to increase computed adaptive capping value of challengers by (post scaling)

Aliases: --ac-add-slack, --capAddSlack
Default Value: 1.0
Domain: $(0, \infty)$

--ac-mult-slack amount to scale computed adaptive capping value of challengers by

Aliases: --ac-mult-slack, --capSlack
Default Value: 1.3
Domain: $(0, \infty)$

--acq-func acquisition function to use during local search, NOTE: The LCB acquisition function mu+k$\times$sigma will have k sampled from an exponential distribution with mean 1.

Aliases: --acq-func, --acquisition-function, --ei-func, --expected-improvement-function, --expectedImprovementFuncti
Default Value: EXPONENTIAL if minimizing runtime, EI otherwise.
Domain: {EXPONENTIAL, SIMPLE, LCB, EI, LCBEIRR}

--clean-old-state-on-success will clean up much of the useless state files if smac completes successfully

Aliases: --clean-old-state-on-success, --cleanOldStateOnSuccess
Default Value: true
Domain: {true, false}

--config-tracking Take measurements of configuration as it goes through it's lifecycle and write to file (in state folder)

Aliases: --config-tracking
Domain: {true, false}

--doubling-capping-challengers Number of challengers to use with the doubling capping mechanism

Aliases: --doubling-capping-challengers
Default Value: 2
Domain: (0, 2147483647]

--doubling-capping-runs-per-challenger Number of runs each challenger will get with the doubling capping initilization strategy

Aliases: --doubling-capping-runs-per-challenger
Default Value: 2
Domain: (0, 2147483647]

--help-default-file file that contains default settings for SMAC

Aliases: --help-default-file, --helpDefaultsFile
Default Value: /.aeatk/help.opt
Domain: FILES

--imputation-iterations amount of times to impute censored data when building model

Aliases: --imputation-iterations, --imputationIterations
Default Value: 2
Domain: [0, 2147483647]

--init-mode Initialization Mode

Aliases: --init-mode, --initialization-mode, --initMode, --initializationMode
Default Value: CLASSIC
Domain: {CLASSIC, ITERATIVE_CAPPING, UNBIASED_TABLE}

--initial-challengers-intensification-time Time to spend on intensify for the initial challengers.

Aliases: --initial-challengers-intensification-time, --initialChallengersIntensificationTime
Default Value: 2147483647

--intensification-percentage percent of time to spend intensifying versus model learning

Aliases: --intensification-percentage, --intensificationPercentage, --frac_rawruntime
Default Value: 0.5
Domain: (0, 1)

--intermediary-saves determines whether to make any intermediary-saves or not (if false, no quick saves will be made either). The state will still be saved at the end of the run however

Aliases: --intermediary-saves
Default Value: true
Domain: {true, false}

--iterativeCappingBreakOnFirstCompletion In Phase 2 of the initialization phase, we will abort the first time something completes and not look at anything else with the same kappa limits

Aliases: --iterativeCappingBreakOnFirstCompletion
Default Value: false
Domain: {true, false}

--iterativeCappingK Iterative Capping K

Aliases: --iterativeCappingK
Default Value: 1

--mask-censored-data-as-kappa-max Mask censored data as kappa Max

Aliases: --mask-censored-data-as-kappa-max, --maskCensoredDataAsKappaMax
Default Value: false
Domain: {true, false}

--mask-inactive-conditional-parameters-as-default-value build the model treating inactive conditional values as the default value

Aliases: --mask-inactive-conditional-parameters-as-default-value, --maskInactiveConditionalParametersAsDefaultValu
Default Value: true
Domain: {true, false}

--max-incumbent-runs maximum number of incumbent runs allowed

Aliases: --max-incumbent-runs, --maxIncumbentRuns, --maxRunsForIncumbent
Default Value: 2000
Domain: (0, 2147483647]

--num-challengers number of challengers needed for local search

Aliases: --num-challengers, --numChallengers, --numberOfChallengers
Default Value: 10
Domain: (0, 2147483647]

--num-ei-random number of random configurations to evaluate during EI search

Aliases: --num-ei-random, --numEIRandomConfigs, --numberOfRandomConfigsInEI, --numRandomConfigsInEI, --numberOfEIRandomConfigs
Default Value: 10000
Domain: [0, 2147483647]

--num-ls-random Number of random configurations that will be used as potential starting points for local search

Aliases: --num-ls-random, --num-local-search-random
Default Value: 0
Domain: [0, 2147483647]

--num-pca number of principal components features to use when building the model

Aliases: --num-pca, --numPCA
Default Value: 7

Domain: (0, 2147483647]

--option-file read options from file

Aliases: --option-file, --optionFile
Domain: FILES

--option-file2 read options from file

Aliases: --option-file2, --optionFile2, --secondaryOptionsFile
Domain: FILES

--print-rungroup-replacement-and-exit print all the possible replacements in the rungroup and then exit

Aliases: --print-rungroup-replacement-and-exit
Default Value: false
Domain: {true, false}

--quick-saves determines whether to make quick saves or not

Aliases: --quick-saves
Default Value: true
Domain: {true, false}

--restore-iteration iteration of the state to restore, use "AUTO" to automatically pick the last iteration

Aliases: --restore-iteration, --restoreStateIteration, --restoreIteration
Default Value: N/A (No state is being restored)

--restore-state-from location of state to restore

Aliases: --restore-state-from, --restoreStateFrom
Default Value: N/A (No state is being restored)

--save-context saves some context with the state folder so that the data is mostly self-describing (Scenario, Instance File, Feature File, Param File are saved)

Aliases: --save-context, --saveContext, --saveContextWithState
Default Value: true
Domain: {true, false}

--shared-model-mode If true the run data will be read from other runs in the output dir periodically (the runs need have a specific filename)

Aliases: --shared-model-mode, --share-model-mode, --shared-run-data, --share-run-data
Default Value: false
Domain: {true, false}

--shared-model-mode-frequency How often to poll for new run data (in seconds)

Aliases: --shared-model-mode-frequency, --share-model-mode-frequency, --shared-run-data-frequency, --share-run-data-frequency
Default Value: 300 seconds
Domain: (0, 2147483647]

--shared-model-mode-write-data If true we will write run data to a JSON file

Aliases: --shared-model-mode-write-data, --write-json-data
Default Value: true
Domain: {true, false}

--smac-default-file file that contains default settings for SMAC
Aliases: --smac-default-file, --smacDefaultsFile
Default Value: /.aeatk/smac.opt
Domain: FILES

--state-deserializer determines the format of the files that store the saved state to restore
Aliases: --state-deserializer, --stateDeserializer
Default Value: LEGACY
Domain: {NULL, LEGACY}

--state-serializer determines the format of the files to save the state in
Aliases: --state-serializer, --stateSerializer
Default Value: LEGACY
Domain: {NULL, LEGACY}

--treat-censored-data-as-uncensored builds the model as-if the response values observed for cap values, were the correct ones [NOT RECOMMENDED]
Aliases: --treat-censored-data-as-uncensored, --treatCensoredDataAsUncensored
Default Value: false
Domain: {true, false}

--unbiased-capping-challengers Number of challengers we will consider during initialization
Aliases: --unbiased-capping-challengers
Default Value: 2
Domain: (0, 2147483647]

--unbiased-capping-cpulimit Amount of CPU Time to spend constructing table in initialization phase
Aliases: --unbiased-capping-cpulimit
Default Value: 0
Domain: (0, 2147483647]

--unbiased-capping-runs-per-challenger Number of runs we will consider during initalization per challenger
Aliases: --unbiased-capping-runs-per-challenger
Default Value: 2
Domain: (0, 2147483647]

--validation-seed Seed to use for validating SMAC
Aliases: --validation-seed
Default Value: 0 which should cause it to run exactly the same as the stand-alone utility.

--warmstart-iteration iteration of the state to use for warm-starting, use "AUTO" to automatically pick the last iteration

Aliases: --warmstart-iteration
Default Value: AUTO (if being restored)

DEVELOPER OPTIONS

--seed-offset offset of numRun to use from seed (this plus --numRun should be less than INTEGER_MAX)

Aliases: --seed-offset, --seedOffset
Default Value: 0

--shared-model-mode-asymetric If set to true, then (based on the order of the file names) we will only read from runs that are transitively 2N and 2N+1 from our ID. So for instance if there were 16 runs, 0-15, runs 8-15 would be independent. Run 4 would read from 8,9. Run 5 would read from 10,11. Run 2 would read from 4,5,8,9,10,11, etc...

Aliases: --shared-model-mode-asymetric
Default Value: false
Domain: {true, false}

--shared-model-mode-default-handling If set to USE_ALL then all runs of the default configuration will be used, If set to SKIP_FIRST_TWO then then first two runs (presumably the default) will not be read, If set to IGNORE_ALL then we will always ignore runs with the default configuration

Aliases: --shared-model-mode-default-handling
Default Value: USE_ALL
Domain: {USE_ALL, SKIP_FIRST_TWO, IGNORE_ALL}

--shared-model-mode-tae If true and shared model mode is enabled, then we will also try and share run data at the TAE level

Aliases: --shared-model-mode-tae
Default Value: true
Domain: {true, false}

-S Sets specific seeds (by name) in the random pool (e.g. -SCONFIG=2 -SINSTANCE=4). To determine the actual names that will be used you should run the program with debug logging enabled, it should be output at the end.

Aliases: -S

9.5.2 Random Forest Options

Options used when building the Random Forests

ADVANCED OPTIONS

--rf-full-tree-bootstrap bootstrap all data points into trees

Aliases: --rf-full-tree-bootstrap, --fullTreeBootstrap
Default Value: false
Domain: {true, false}

--rf-ignore-conditionality ignore conditionality for building the model

Aliases: --rf-ignore-conditionality, --ignoreConditionality

Default Value: false
Domain: {true, false}

--rf-impute-mean impute the mean value for the all censored data points

Aliases: --rf-impute-mean, --imputeMean
Default Value: false
Domain: {true, false}

--rf-log-model store response values in log-normal form

Aliases: --rf-log-model, --log-model, --logModel
Default Value: true if optimizing runtime, false if optimizing quality
Domain: {true, false}

--rf-min-variance minimum allowed variance

Aliases: --rf-min-variance, --minVariance
Default Value: 1.0E-14
Domain: $(0, \infty)$

--rf-num-trees number of trees to create in random forest

Aliases: --rf-num-trees, --num-trees, --numTrees, --nTrees, --numberOfTrees
Default Value: 10
Domain: (0, 2147483647]

--rf-penalize-imputed-values treat imputed values that fall above the cutoff time, and below the penalized max time, as the penalized max time

Aliases: --rf-penalize-imputed-values, --penalizeImputedValues
Default Value: false
Domain: {true, false}

--rf-ratio-features ratio of the number of features to consider when splitting a node

Aliases: --rf-ratio-features, --ratioFeatures
Default Value: 0.8333333333333334
Domain: (0, 1]

--rf-shuffle-imputed-values shuffle imputed value predictions between trees

Aliases: --rf-shuffle-imputed-values, --shuffleImputedValues
Default Value: false
Domain: {true, false}

--rf-split-min minimum number of elements needed to split a node

Aliases: --rf-split-min, --split-min, --splitMin
Default Value: 10
Domain: [0, 2147483647]

DEVELOPER OPTIONS

--rf-preprocess-marginal build random forest with preprocessed marginal

Aliases: --rf-preprocess-marginal, preprocessMarginal
Default Value: true
Domain: {true, false}

--rf-store-data store full data in leaves of trees

Aliases: --rf-store-data, --rf-store-data-in-leaves, --storeDataInLeaves
Default Value: false
Domain: {true, false}

--rf-subsample-memory-percentage when free memory percentage drops below this percent we will apply the subsample percentage

Aliases: --rf-subsample-memory-percentage, --freeMemoryPecentageToSubsample
Default Value: 0.25
Domain: (0, 1]

--rf-subsample-percentage multiply the number of points used when building model by this value

Aliases: --rf-subsample-percentage, --subsamplePercentage
Default Value: 0.9
Domain: (0, 1]

--rf-subsample-values-when-low-on-memory subsample model input values when the amount of memory available drops below a certain threshold (see --subsampleValuesWhenLowMemory) (Not Tested)

Aliases: --rf-subsample-values-when-low-on-memory, --subsampleValuesWhenLowOnMemory, --subsampleValuesWhenLowMemory
Default Value: false
Domain: {true, false}

9.5.3 Scenario Options

Standard Scenario Options for use with SMAC. In general consider using the –scenarioFile directive to specify these parameters and Algorithm Execution Options

BASIC OPTIONS

--feature-file file that contains the all the instances features

Aliases: --feature-file, --instanceFeatureFile, --feature_file

--instances File or directory containing the instances to use for the scenario. If it's a file it must coform a specific format (see Instance File Format section of the manual), if it's a directory it you must also use the --instance-suffix option to restrict the match (unless all files have the same extension), and the instance list will be in sorted order.

REQUIRED
Aliases: --instances, --instance-file, --instance-dir, --instanceFile, -i, --instance_file, --instance_seed_file
Default Value: null

--run-obj per target algorithm run objective type that we are minimizing

REQUIRED

Aliases: --run-obj, --run-objective, --runObj, --run_obj
Default Value: null
Domain: {RUNTIME, QUALITY}

--scenario-file scenario file
Aliases: --scenario-file, --scenarioFile, --scenario
Domain: FILES

--skip-features If true the feature file will be ignored (if the feature file is required, this will cause an error, as if it was not supplied
Aliases: --skip-features, --ignore-features
Default Value: false
Domain: {true, false}

--test-instances File or directory containing the instances to use for the scenario. If it's a file it must coform a specific format (see Instance File Format section of the manual), if it's directory you must also use the --test-instance-suffix option to restrict the match (unless all files have the same extension), , and the instance list will be in sorted order
Aliases: --test-instances, --test-instance-file, --test-instance-dir, --testInstanceFile, --test_instance_file, --test_instance_seed_file
Default Value: null

INTERMEDIATE OPTIONS

--instance-suffix A suffix that all instances must match when reading instances from a directory. You can optionally specify a (java) regular expression but be aware that it is suffix matched (internally we take this string and append a $ on it)
Aliases: --instance-suffix, --instance-regex
Default Value: null

--intra-obj objective function used to aggregate multiple runs for a single instance
Aliases: --intra-obj, --intra-instance-obj, --overall-obj, --intraInstanceObj, --overallObj, --overall_obj, --intra_instance_obj
Default Value: MEAN if –run-obj is QUALITY and MEAN10 if it is runtime
Domain: {MEAN, MEAN1000, MEAN10}

--output-dir Output Directory
Aliases: --output-dir, --outputDirectory, --outdir
Default Value: <current working directory>/____-output

--test-instance-suffix A suffix that all instances must match when reading instances from a directory. You can optionally specify a (java) regular expression but be aware that it is suffix matched (internally we take this string and append a $ on it)
Aliases: --test-instance-suffix, --test-instance-regex
Default Value: null

--use-instances If false skips reading the instances and just uses a dummy instance
Aliases: --use-instances

Default Value: true
Domain: {true, false}

ADVANCED OPTIONS

--check-instances-exist check if instances files exist on disk

Aliases: --check-instances-exist, --checkInstanceFilesExist
Default Value: false
Domain: {true, false}

--inter-obj objective function used to aggregate over multiple instances (that have already been aggregated under the Intra-Instance Objective)

Aliases: --inter-obj, --inter-instance-obj, --interInstanceObj, --inter_instance_obj
Default Value: MEAN
Domain: {MEAN, MEAN1000, MEAN10}

9.5.4 Scenario Configuration Limit Options

Options that control how long the scenario will run for

BASIC OPTIONS

--cputime-limit limits the total cpu time allowed between SMAC and the target algorithm runs during the automatic configuration phase

Aliases: --cputime-limit, --cputime_limit, --tunertime-limit, --tuner-timeout, --tunerTimeout
Default Value: 2147483647
Domain: [0, 2147483647]

--runcount-limit limits the total number of target algorithm runs allowed during the automatic configuration phase

Aliases: --runcount-limit, --runcount_limit, --totalNumRunsLimit, --numRunsLimit, --numberOfRunsLimit
Default Value: 9223372036854775807
Domain: (0, 9223372036854775807]

--wallclock-limit limits the total wall-clock time allowed during the automatic configuration phase

Aliases: --wallclock-limit, --wallclock_limit, --runtime-limit, --runtimeLimit, --wallClockLimit
Default Value: 2147483647
Domain: (0, 2147483647]

ADVANCED OPTIONS

--iteration-limit limits the number of iterations allowed during automatic configuration phase

Aliases: --iteration-limit, --numIterations, --numberOfIterations
Default Value: 2147483647
Domain: (0, 2147483647]

--max-norun-challenge-limit if the parameter space is too small we may get to a point where we can make no new runs, detecting this condition is prohibitively expensive, and this heuristic controls the number of times we need to try a challenger and get no new runs before we give up

Aliases: --max-norun-challenge-limit, --maxConsecutiveFailedChallengeIncumbent
Default Value: 1000

--terminate-on-delete Terminate the procedure if this file is deleted

Aliases: --terminate-on-delete
Default Value: null

--use-cpu-time-in-tunertime include the CPU Time of SMAC as part of the tunerTimeout

Aliases: --use-cpu-time-in-tunertime, --countSMACTimeAsTunerTime
Default Value: true
Domain: {true, false}

9.5.5 Algorithm Execution Options

Options related to invoking the target algorithm

BASIC OPTIONS

--algo-cutoff-time CPU time limit for an individual target algorithm run

Aliases: --algo-cutoff-time, --target-run-cputime-limit, --target_run_cputime_limit, --cutoff-time, --cutoffTime, --cutoff_time
Default Value: 1.7976931348623157E308
Domain: $(0, \infty)$

--algo-deterministic treat the target algorithm as deterministic

Aliases: --algo-deterministic, --deterministic
Default Value: true
Domain: {true, false}

--algo-exec command string to execute algorithm with

REQUIRED
Aliases: --algo-exec, --algoExec, --algo
Default Value: null

--pcs-file File containing algorithm parameter space information in PCS format (see Algorithm Parameter File in the Manual). You can specify "SINGLETON" to get a singleton configuration space or "NULL" to get a null one.

REQUIRED
Aliases: --pcs-file, --param-file, -p, --paramFile, --paramfile
Default Value: null

-T additional context needed for target algorithm execution (see TAE documentation for possible values, generally rare)

Aliases: -T
Default Value:

INTERMEDIATE OPTIONS

--algo-exec-dir working directory to execute algorithm in

Aliases: --algo-exec-dir, --exec-dir, --execDir, --execdir
Default Value: current working directory

ADVANCED OPTIONS

--continous-neighbours Number of neighbours for continuous parameters

Aliases: --continous-neighbours, --continuous-neighbors, --continuousNeighbours
Default Value: 4

DEVELOPER OPTIONS

--search-subspace Only generate random and neighbouring configurations with these values. Specified in a "name=value,name=value,..." format (Overrides those set in file)

Aliases: --search-subspace, --searchSubspace
Default Value: null

--search-subspace-file Only generate random and neighbouring configurations with these values. Specified each parameter on each own line with individual value

Aliases: --search-subspace-file, --searchSubspaceFile
Default Value: null
Domain: FILES

9.5.6 Target Algorithm Evaluator Options

Options that describe and control the policy and mechanisms for algorithm execution

INTERMEDIATE OPTIONS

--abort-on-crash treat algorithm crashes as an ABORT (Useful if algorithm should never CRASH). NOTE: This only aborts if all retries fail.

Aliases: --abort-on-crash, --abortOnCrash
Default Value: false
Domain: {true, false}

--abort-on-first-run-crash if the first run of the algorithm CRASHED treat it as an ABORT, otherwise allow crashes.

Aliases: --abort-on-first-run-crash, --abortOnFirstRunCrash
Default Value: true
Domain: {true, false}

--bound-runs [DEPRECATED] (Use the option on the TAE instead if available) if true, permit only --cores number of runs to be evaluated concurrently.

Aliases: --bound-runs, --boundRuns
Default Value: false
Domain: {true, false}

--check-sat-consistency Ensure that runs on the same problem instance always return the same SAT/UNSAT result

Aliases: --check-sat-consistency, --checkSATConsistency

Default Value: true
Domain: {true, false}

--check-sat-consistency-exception Throw an exception if runs on the same problem instance disagree with respect to SAT/UNSAT
Aliases: --check-sat-consistency-exception, --checkSATConsistencyException
Default Value: true
Domain: {true, false}

--cores [DEPRECATED] (Use the TAE option instead if available) maximum number of concurrent target algorithm executions
Aliases: --cores, --numConcurrentAlgoExecs, --maxConcurrentAlgoExecs, --numberOfConcurrentAlgoExecs
Default Value: 1

--kill-run-exceeding-captime Attempt to kill runs that exceed their captime by some amount
Aliases: --kill-run-exceeding-captime
Default Value: true
Domain: {true, false}

--kill-run-exceeding-captime-factor Attempt to kill the run that exceed their captime by this factor
Aliases: --kill-run-exceeding-captime-factor
Default Value: 10.0
Domain: $(1, \infty)$

--retry-crashed-count number of times to retry an algorithm run before reporting crashed (NOTE: The original crashes DO NOT count towards any time limits, they are in effect lost). Additionally this only retries CRASHED runs, not ABORT runs, this is by design as ABORT is only for cases when we shouldn't bother further runs
Aliases: --retry-crashed-count, --retryCrashedRunCount, --retryTargetAlgorithmRunCount
Default Value: 0
Domain: [0, 2147483647]

--tae Target Algorithm Evaluator to use when making target algorithm calls
Aliases: --tae, --targetAlgorithmEvaluator
Default Value: CLI
Domain: {ANALYTIC, BLACKHOLE, CLI, CONSTANT, IPC, PRELOADED, RANDOM}

--track-scheduled-runs If true outputs a file in the output directory that outlines how many runs were being evaluated at any given time
Aliases: --track-scheduled-runs
Default Value: false
Domain: {true, false}

--verify-sat Checks SAT/UNSAT/UNKNOWN responses of algorithm with the value stored as instance specific information, logging an error if there is a discrepancy. The default value is auto-detected based on the value of the instance specific information of every problem instance. If every instance has an instance specific information in the following set SAT, UNSAT, UNKNOWN, SATISFIABLE, UNSATISFIABLE, this will be set to true, otherwise it will be false.

Aliases: --verify-sat, --verify-SAT, --verifySAT
Default Value: Auto detected (see description)
Domain: {true, false}

ADVANCED OPTIONS

--call-observer-before-completion Ensure that the TAE observer is called on runs before completion

Aliases: --call-observer-before-completion
Default Value: true
Domain: {true, false}

--file-cache If true runs will be either written or read from the specified input and output files. If directories are specified, then input will be from all files in the directory, and output will be to a new random file in the directory. Note: This cache is static, we do not re-read from the cache over time

Aliases: --file-cache
Default Value: false
Domain: {true, false}

--file-cache-output Where to write files from

Aliases: --file-cache-output
Default Value: Current Directory/runcache/

--file-cache-source Where to read files from

Aliases: --file-cache-source
Default Value: Current Directory/runcache/

--filter-zero-cutoff-runs If true runs that are requested with 0 cutoff will be internally completed as TIMEOUT.

Aliases: --filter-zero-cutoff-runs
Default Value: true
Domain: {true, false}

--log-requests-responses If set to true all evaluation requests will be logged as they are submitted and completed

Aliases: --log-requests-responses
Default Value: false
Domain: {true, false}

--log-requests-responses-rc-only If set to true we will only log the run configuration when a run completes

Aliases: --log-requests-responses-rc-only, --log-requests-responses-rc
Default Value: false
Domain: {true, false}

--observer-walltime-delay How long to wait for an update with runtime information, before we use the walltime. With the 5 seconds and an scale of 0.95, it means we will see 0,0,0,0...,4.95...

Aliases: --observer-walltime-delay
Default Value: 5.0

Domain: $(0, \infty)$

--observer-walltime-if-no-runtime If true and the target algorithm doesn't update us with runtime information we report wallclock time

Aliases: --observer-walltime-if-no-runtime
Default Value: true
Domain: {true, false}

--observer-walltime-scale What factor of the walltime should we use as the runtime (generally recommended is the 0.95 times the number of cores)

Aliases: --observer-walltime-scale
Default Value: 0.95
Domain: $(0, \infty)$

--tae-default-file file that contains default settings for Target Algorithm Evaluators

Aliases: --tae-default-file
Default Value: /.aeatk/tae.opt
Domain: FILES

--track-scheduled-runs-resolution We will bucket changes into this size

Aliases: --track-scheduled-runs-resolution
Default Value: 1.0
Domain: $(0, \infty)$

--transform-crashed-quality If true we will transform the solution quality reported to the MAX(quality, --transform-crashed-quality-value).

Aliases: --transform-crashed-quality
Default Value: true
Domain: {true, false}

--transform-crashed-quality-value The minimum quality value that a CRASHED run can have

Aliases: --transform-crashed-quality-value
Default Value: 1.0E9

DEVELOPER OPTIONS

--cache-runs If true we will cache runs internally, so that subsequent requests are not re-executed [EXPERIMENTAL]

Aliases: --cache-runs
Default Value: false
Domain: {true, false}

--cache-runs-debug If true we will print the state of the cache every so often for debug purposes.

Aliases: --cache-runs-debug
Default Value: false
Domain: {true, false}

--cache-runs-strictly-increasing-observer If true then we will enforce that all runtimes seen externally always have strictly increasing times. (Internally if the run is restarted for some reason, the observed time may in fact go down).

Aliases: --cache-runs-strictly-increasing-observer
Default Value: false
Domain: {true, false}

--check-for-unclean-shutdown If true, we will try and detect an unclean shutdown of the Target Algorithm Evaluator

Aliases: --check-for-unclean-shutdown
Default Value: true
Domain: {true, false}

--check-for-unique-runconfigs Checks that all submitted Run Configs in a batch are unique

Aliases: --check-for-unique-runconfigs
Default Value: true
Domain: {true, false}

--check-for-unique-runconfigs-exception If true, we will throw an exception if duplicate run configurations are detected

Aliases: --check-for-unique-runconfigs-exception
Default Value: true
Domain: {true, false}

--check-result-order-consistent Check that the TAE is returning responses in the correct order

Aliases: --check-result-order-consistent, --checkResultOrderConsistent
Default Value: false
Domain: {true, false}

--exception-on-prepost-command Throw an abort

Aliases: --exception-on-prepost-command, --exceptionOnPrePostCommand
Domain: {true, false}

--exit-on-failure If true, when a failure is detected the process will try its best to shutdown, potentially not cleanly

Aliases: --exit-on-failure
Default Value: false
Domain: {true, false}

--file-cache-crash-on-cache-miss Application will crash on cache miss, this is for debugging

Aliases: --file-cache-crash-on-cache-miss, --file-cache-crash-on-miss
Default Value: false
Domain: {true, false}

--kill-runs-on-file-delete All runs will be forcibly killed if the file is deleted. This option may cause the application to enter an infinite loop if the file is deleted, so care is needed. As a rule, you need to set this and some other option to point to the same file, if there is another option, then the application will probably shutdown nicely, if not, then it will probably infinite loop.

Aliases: --kill-runs-on-file-delete
Default Value: null

--post-scenario-command Command that will run on shutdown

Aliases: --post-scenario-command, --postScenarioCommand, --post_cmd

--pre-scenario-command Command that will run on startup

Aliases: --pre-scenario-command, --preScenarioCommand, --pre_cmd

--prepost-exec-dir Execution Directory for Pre/Post commands

Aliases: --prepost-exec-dir, --prePostExecDir
Default Value: Current Working Directory
Domain: {readabledirectories}

--prepost-log-output Log all the output from the pre and post commands

Aliases: --prepost-log-output, --logOutput
Domain: {true, false}

--run-hashcode-file file containing a list of run hashes one per line: Each line should be: "Run Hash Codes: (Hash Code) After (n) runs". The number of runs in this file need not match the number of runs that we execute, this file only ensures that the sequences never diverge. Note the n is completely ignored so the order they are specified in is the order we expect the hash codes in this version. Finally note you can simply point this at a previous log and other lines will be disregarded

Aliases: --run-hashcode-file, --runHashCodeFile
Domain: FILES

--skip-outstanding-eval-tae If set to true code, the TAE will not be wrapped by a decorator to support waiting for outstanding runs

Aliases: --skip-outstanding-eval-tae
Default Value: false
Domain: {true, false}

--tae-stop-processing-on-shutdown If true, then once JVM Shutdown is triggered either within the application or externally all further requests will be silently dropped. This is recommended since otherwise applications may see unexpected results as the TAE may be unable to continue processing.

Aliases: --tae-stop-processing-on-shutdown
Default Value: true
Domain: {true, false}

--tae-warn-if-no-response-from-tae If greater than 0, it is the number of seconds to wait for the TAE to respond before issuing a warning

Aliases: --tae-warn-if-no-response-from-tae
Default Value: 120
Domain: [0, 2147483647]

--use-dynamic-cutoffs If true then we change all cutoffs to the maximum cutoff time and dynamically kill runs that exceed there cutoff time. This is useful because cache hits require the cutoff time to match

Aliases: --use-dynamic-cutoffs
Default Value: false
Domain: {true, false}

9.5.7 Transform Target Algorithm Evaluator Decorator Options

This Target Algorithm Evaluator Decorator allows you to transform the response value of the wrapper according to some rules. Expressions that can be used by exp4j (http://www.objecthunter.net/exp4j/), can be specified and will cause the returned runs to be transformed accordingly. The variables in the expression can be S which will be -1 if the run was UNSAT, 1 if SAT, and 0 otherwise, R which is the original reported runtime, Q which is the original reported quality, and C which was the requested cutoff time. Care should be taken when transforming values to obey wrapper semantics. If you don't know what you are doing, we recommend that SAT and UNSAT values should be kept in the range between 0 and cutoff, and the TIMEOUT value shouldn't be transformed at all. A very special thanks to the original author Alexandre Fréchette.

ADVANCED OPTIONS

--tae-transform Set to true if you'd like to transform the result, if false the other transforms have no effect
Aliases: --tae-transform
Default Value: false.
Domain: {true, false}

--tae-transform-SAT-quality Function to apply to an algorithm run's quality if result is SAT.
Aliases: --tae-transform-SAT-quality
Default Value: Identity transform.
Domain: Calculable string using a run's associated variables: S run result (SAT=1,UNSAT=-1,other=0), R runtime, Q quality, C cutoff.

--tae-transform-SAT-runtime Function to apply to an algorithm run's runtime if result is SAT.
Aliases: --tae-transform-SAT-runtime
Default Value: Identity transform.
Domain: Calculable string using a run's associated variables: S run result (SAT=1,UNSAT=-1,other=0), R runtime, Q quality, C cutoff.

--tae-transform-TIMEOUT-quality Function to apply to an algorithm run's quality if result is TIMEOUT.
Aliases: --tae-transform-TIMEOUT-quality
Default Value: Identity transform.
Domain: Calculable string using a run's associated variables: S run result (SAT=1,UNSAT=-1,other=0), R runtime, Q quality, C cutoff.

--tae-transform-TIMEOUT-runtime Function to apply to an algorithm run's runtime if result is TIMEOUT.
Aliases: --tae-transform-TIMEOUT-runtime
Default Value: Identity transform.
Domain: Calculable string using a run's associated variables: S run result (SAT=1,UNSAT=-1,other=0), R runtime, Q quality, C cutoff.

--tae-transform-UNSAT-quality Function to apply to an algorithm run's quality if result is UNSAT.

Aliases: --tae-transform-UNSAT-quality

Default Value: Identity transform.

Domain: Calculable string using a run's associated variables: S run result (SAT=1,UNSAT=-1,other=0), R runtime, Q quality, C cutoff.

--tae-transform-UNSAT-runtime Function to apply to an algorithm run's runtime if result is UNSAT.

Aliases: --tae-transform-UNSAT-runtime

Default Value: Identity transform.

Domain: Calculable string using a run's associated variables: S run result (SAT=1,UNSAT=-1,other=0), R runtime, Q quality, C cutoff.

--tae-transform-other-quality Function to apply to an algorithm run's quality if result is not SAT, UNSAT or TIMEOUT.

Aliases: --tae-transform-other-quality

Default Value: Identity transform.

Domain: Calculable string using a run's associated variables: S run result (SAT=1,UNSAT=-1,other=0), R runtime, Q quality, C cutoff.

--tae-transform-other-runtime Function to apply to an algorithm run's runtime if result is not SAT, UNSAT or TIMEOUT.

Aliases: --tae-transform-other-runtime

Default Value: Identity transform.

Domain: Calculable string using a run's associated variables: S run result (SAT=1,UNSAT=-1,other=0), R runtime, Q quality, C cutoff.

DEVELOPER OPTIONS

--tae-transform-valid-values-only If the transformation of runtime results in a value that is too large, the cutoff time will be returned, and the result changed to TIMEOUT. If the result is too small it will be set to 0

Aliases: --tae-transform-valid-values-only

Default Value: true

Domain: {true, false}

9.5.8 Forking Target Algorithm Evaluator Decorator Options

This Target Algorithm Evaluator Decorator allows you to delegate some runs to another TAE, denoted the slave TAE. Several policies are implemented (or will be upon request/need). The first two duplicate the run on the slave, and the primary motivation is performance of very short runs, where overhead of dispatch to the primary might be surprisingly high. The next two (to be implemented), would allow some runs to simply done by the slave, either before the master or after the master.

ADVANCED OPTIONS

--fork-to-tae If not null, runs will also be submitted to this other TAE at the same time. The first TAE that returns an answer is used.

Aliases: --fork-to-tae

Default Value: Forking of requests is disabled

Domain: {ANALYTIC, BLACKHOLE, CLI, CONSTANT, IPC, PRELOADED, RANDOM}

--fork-to-tae-duplicate-on-slave-quick-timeout What timeout to use when the DUPLICATE_ON_SLAVE_QUICK policy.

Aliases: --fork-to-tae-duplicate-on-slave-quick-timeout
Default Value: 5 seconds
Domain: (0, 2147483647]

--fork-to-tae-policy Selects the policy that we will fork with. For instance DUPLICATE_ON_SLAVE will simply submit runs to the slave as well. DUPLICATE_ON_SLAVE_QUICK will submit the runs to the slave, but with a reduced cutoff time

Aliases: --fork-to-tae-policy
Default Value: Must be explicitly set if the forkToTAE is not null
Domain: {DUPLICATE_ON_SLAVE, DUPLICATE_ON_SLAVE_QUICK}

9.5.9 Validation Options

Options that control validation

INTERMEDIATE OPTIONS

--max-timestamp maximimum relative timestamp in the trajectory file to configure against. -1 means auto-detect

Aliases: --max-timestamp, --maxTimestamp
Default Value: Auto Detect
Domain: $[0, \infty) \bigcup \{-1\}$

--min-timestamp minimum relative timestamp in the trajectory file to configure against.

Aliases: --min-timestamp, --minTimestamp
Default Value: 0.0
Domain: $[0, \infty)$

--num-validation-runs approximate number of validation runs to do

Aliases: --num-validation-runs, --numValidationRuns, --numberOfValidationRuns
Default Value: 1
Domain: [0, 2147483647]

--save-state-file Save a state file consisting of all the runs we did

Aliases: --save-state-file, --saveStateFile
Default Value: false
Domain: {true, false}

--validate-by-wallclock-time Validate runs by wall-clock time

Aliases: --validate-by-wallclock-time, --validateByWallClockTime
Default Value: true
Domain: {true, false}

--validate-only-if-tunertime-reached If the walltime in the trajectory file hasn't hit this entry we won't bother validating

Aliases: --validate-only-if-tunertime-reached, --validateOnlyIfTunerTimeReached
Default Value: 0.0
Domain: $[0, \infty)$

--validate-only-if-walltime-reached If the walltime in the trajectory file hasn't hit this entry we won't bother validating

Aliases: --validate-only-if-walltime-reached, --validateOnlyIfWallTimeReached
Default Value: 0.0
Domain: $[0, \infty)$

--validate-only-last-incumbent validate only the last incumbent found

Aliases: --validate-only-last-incumbent, --validateOnlyLastIncumbent
Default Value: true
Domain: {true, false}

ADVANCED OPTIONS

--mult-factor base of the geometric progression of timestamps to validate (for instance by default it is maxTimestamp, maxTimestamp/2, maxTimestamp/4,... whiletimestamp $\geq$ minTimestamp)

Aliases: --mult-factor, --multFactor
Default Value: 2.0
Domain: $(0, \infty)$

--output-file-suffix Suffix to add to validation run files (for grouping)

Aliases: --output-file-suffix, --outputFileSuffix

--validate-all Validate every entry in the trajectory file (overrides other validation options)

Aliases: --validate-all, --validateAll
Default Value: false
Domain: {true, false}

--validation-rounding-mode selects whether to round the number of validation (to next multiple of numTestInstances

Aliases: --validation-rounding-mode, --validationRoundingMode
Default Value: UP
Domain: {UP, NONE}

DEVELOPER OPTIONS

--num-seeds-per-test-instance Deprecated/Broken: number of test seeds to use per instance during validation

Aliases: --num-seeds-per-test-instance, --numSeedsPerTestInstance, --numberOfSeedsPerTestInstance
Default Value: 1000
Domain: (0, 2147483647]

--num-test-instances Deprecated/Broken: Check results carefully: number of instances to test against (will execute min of this, and number of instances in test instance file). To disable validation in SMAC see the --doValidation option

Aliases: --num-test-instances, --numTestInstances, --numberOfTestInstances
Default Value: 2147483647
Domain: (0, 2147483647]

--validation-headers put headers on output CSV files for validation

Aliases: --validation-headers, --validationHeaders
Default Value: true
Domain: {true, false}

9.5.10 Analytic Target Algorithm Evaluator Options

This Target Algorithm Evaluator uses an analytic function to generate a runtime. Most of the function definitions come from Test functions for optimization needs, by Marcin Molga, Czesaw Smutnicki (http://www.zsd.ict.pwr.wroc.pl/files/docs/functions.pdf). NOTE: Some functions have been shifted vertically so that there response values are always positive.

ADVANCED OPTIONS

--analytic-function Which analytic function to use

Aliases: --analytic-function
Default Value: CAMELBACK
Domain: {ZERO, ADD, CAMELBACK, BRANINS, SINEPLUSONE}

DEVELOPER OPTIONS

--analytic-observer-frequency How often to notify observer of updates (in milli-seconds)

Aliases: --analytic-observer-frequency
Default Value: 100
Domain: (0, 2147483647]

--analytic-scale-simulate-delay Divide the simulated delay by this value

Aliases: --analytic-scale-simulate-delay
Default Value: 1.0
Domain: (0, ∞)

--analytic-simulate-cores If set to greater than 0, the TAE will serialize requests so that no more than these number will execute concurrently.

Aliases: --analytic-simulate-cores
Default Value: 0
Domain: [0, 2147483647]

--analytic-simulate-delay If set to true the TAE will simulate the wallclock delay

Aliases: --analytic-simulate-delay
Default Value: false
Domain: {true, false}

9.5.11 Blackhole Target Algorithm Evaluator Options

This Target Algorithm Evaluator simply never returns any runs

DEVELOPER OPTIONS

--blackhole-warnings Suppress warning that is generated

Aliases: --blackhole-warnings
Default Value: true
Domain: {true, false}

9.5.12 Command Line Target Algorithm Evaluator Options

This Target Algorithm Evaluator executes commands via the command line and the standard wrapper interface.

INTERMEDIATE OPTIONS

--cli-concurrent-execution Whether to allow concurrent execution

Aliases: --cli-concurrent-execution
Default Value: true
Domain: {true, false}

--cli-cores Number of cores to use to execute runs. In other words the number of requests to run at a given time.

Aliases: --cli-cores
Default Value: 1
Domain: (0, 2147483647]

--cli-log-all-call-strings log every call string

Aliases: --cli-log-all-call-strings, --log-all-call-strings, --logAllCallStrings
Default Value: false
Domain: {true, false}

--cli-log-all-calls log all the call strings and result lines

Aliases: --cli-log-all-calls, --cli-log-all-call-strings-and-results, --log-all-calls, --log-all-call-strings-and-results
Default Value: false
Domain: {true, false}

--cli-log-all-process-output log all process output

Aliases: --cli-log-all-process-output, --log-all-process-output, --logAllProcessOutput
Default Value: false
Domain: {true, false}

--cli-log-all-results log all the result lines

Aliases: --cli-log-all-results, --cli-log-all-call-results, --log-all-call-results, --log-all-results
Default Value: false
Domain: {true, false}

ADVANCED OPTIONS

--cli-call-params-with-quotes If true calls to the target algorithm will have parameters that are quoted "'3'" instead of "3". Older versions of the code passed arguments with '. This has been removed and will be deprecated in the future

Aliases: --cli-call-params-with-quotes
Default Value: false
Domain: {true, false}

--cli-default-file file that contains default settings for CLI Target Algorithm Evaluator (it is recommended that you use this file to set the kill commands)

Aliases: --cli-default-file
Default Value: /.aeatk/cli-tae.opt
Domain: FILES

--cli-kill-by-environment-cmd If not null, this script will be executed with three arguments, the first a key, the second a value, the third our best guess at a pid (-1 means we couldn't guess). They represent environment name and value, and the script should find every process with that name and value set and terminate it. Do not assume that the key is static as it may change based on existing environment variables. Example scripts may be available in example_scripts/env_kill/

Aliases: --cli-kill-by-environment-cmd
Default Value: null

--cli-listen-for-updates If true will create a socket and set environment variables so that we can have updates of CPU time

Aliases: --cli-listen-for-updates
Default Value: true
Domain: {true, false}

--cli-pg-force-kill-cmd Command to execute to try and ask the process group to terminate nicely (generally a SIGKILL in Unix). Note

Aliases: --cli-pg-force-kill-cmd
Default Value: bash -c "kill -s KILL -

--cli-pg-nice-kill-cmd Command to execute to try and ask the process group to terminate nicely (generally a SIGTERM in Unix). Note

Aliases: --cli-pg-nice-kill-cmd
Default Value: bash -c "kill -s TERM -

--cli-proc-force-kill-cmd Command to execute to try and ask the process to terminate nicely (generally a SIGTERM in Unix). Note

Aliases: --cli-proc-force-kill-cmd
Default Value: kill -s KILL

--cli-proc-nice-kill-cmd Command to execute to try and ask the process to terminate nicely (generally a SIGTERM in Unix). Note

Aliases: --cli-proc-nice-kill-cmd

Default Value: kill -s TERM

DEVELOPER OPTIONS

--cli-observer-frequency How often to notify observer of updates (in milli-seconds)

Aliases: --cli-observer-frequency
Default Value: 500
Domain: (0, 2147483647]

9.5.13 Constant Target Algorithm Evaluator Options

Parameters for the Constant Target Algorithm Evaluator

DEVELOPER OPTIONS

--constant-additional-run-data Additional Run Data to return

Aliases: --constant-additional-run-data

--constant-run-length Runlength to return

Aliases: --constant-run-length
Default Value: 0.0

--constant-run-quality Quality to return

Aliases: --constant-run-quality
Default Value: 0.0

--constant-run-result Run Result To return

Aliases: --constant-run-result
Default Value: SAT
Domain: {TIMEOUT, SAT, UNSAT, CRASHED, ABORT, RUNNING, KILLED}

--constant-runtime Runtime to return

Aliases: --constant-runtime
Default Value: 1.0

9.5.14 Inter-Process Communication Target Algorithm Evaluator Options

This Target Algorithm Evaluator hands the requests off to another process. The current encoding mechanism is the same as on the command line, except that we do not specify the algo executable field. The current mechanism can only execute one request to the server at a time. A small code change would be required to handle the more general case, so please contact the developers if this is required.

ADVANCED OPTIONS

--ipc-async-threads Number of asynchronous threads to use

Aliases: --ipc-async-threads
Default Value: One more than the number of available processors

--ipc-default-file file that contains default settings for IPC Target Algorithm Evaluator (it is recommended that you use this file to set the kill commands)

Aliases: --ipc-default-file
Default Value: /.aeatk/ipc-tae.opt
Domain: FILES

--ipc-encoding How the message is encoded
Aliases: --ipc-encoding
Default Value: CALL_STRING
Domain: {CALL_STRING, JAVA_SERIALIZATION}

--ipc-exec-on-start-up This script will be executed on start up of the IPC TAE. A final argument will be appended which is the server port if our IPCMechanism is REVERSE_TCP
Aliases: --ipc-exec-on-start-up, --ipc-exec
Default Value: null

--ipc-exec-output If true we will log all output from the script
Aliases: --ipc-exec-output
Default Value: false
Domain: {true, false}

--ipc-local-port Local server port for some kinds of IPC mechanisms (if 0, this will be automatically allocated by the operating system)
Aliases: --ipc-local-port
Default Value: 0
Domain: [1,65535]

--ipc-mechanism Mechanism to use for IPC
Aliases: --ipc-mechanism
Default Value: UDP
Domain: {UDP, TCP, REVERSE_TCP}

--ipc-remote-host Remote Host for some kinds of IPC mechanisms
Aliases: --ipc-remote-host
Default Value: 127.0.0.1

--ipc-remote-port Remote Port for some kinds of IPC mechanisms
Aliases: --ipc-remote-port
Default Value: 5050
Domain: [0,65535]

--ipc-report-persistent Whether the TAE should be treated as persistent, loosely a TAE is persistent if we could ask it for the same request later and it wouldn't have to redo the work from scratch.
Aliases: --ipc-report-persistent
Default Value: false
Domain: {true, false}

--ipc-reverse-tcp-pool-connections If true we will pool all the connections instead of closing them
Aliases: --ipc-reverse-tcp-pool-connections

Default Value: false
Domain: {true, false}

--ipc-udp-packetsize Remote Port for some kinds of IPC mechanisms
Aliases: --ipc-udp-packetsize
Default Value: 4096
Domain: [0,65535]

9.5.15 Preloaded Response Target Algorithm Evaluator

Target Algorithm Evaluator that provides preloaded responses

DEVELOPER OPTIONS

--preload-additional-run-data Additional Run Data to return
Aliases: --preload-additional-run-data

--preload-quality Quality to return on all values
Aliases: --preload-quality
Default Value: 0.0

--preload-response-data Preloaded Response Values in the format [SAT,UNSAT,...=x], where x is a runtime (e.g. [SAT=1],[UNSAT=1.1]...
Aliases: --preload-response-data, --preload-responseData

--preload-run-length Runlength to return on all values
Aliases: --preload-run-length, --preload-runLength
Default Value: -1.0

9.5.16 Random Target Algorithm Evaluator Options

This Target Algorithm Evaluator randomly generates responses from a uniform distribution

DEVELOPER OPTIONS

--random-additional-run-data Additional Run Data to return
Aliases: --random-additional-run-data

--random-max-response The maximum runtime we will generate
Aliases: --random-max-response
Default Value: 10.0
Domain: [0, ∞)

--random-min-response The minimum runtime we will generate (values less than 0.01 will be rounded up to 0.01)
Aliases: --random-min-response
Default Value: 0.0
Domain: [0, ∞)

--random-observer-frequency How often to notify observer of updates (in milli-seconds)

Aliases: --random-observer-frequency
Default Value: 500
Domain: (0, 2147483647]

--random-sample-seed Seed to use when generate random responses

Aliases: --random-sample-seed
Default Value: Current Time in Milliseconds

--random-scale-simulate-delay Divide the simulated delay by this value

Aliases: --random-scale-simulate-delay
Default Value: 1.0
Domain: $(0, \infty)$

--random-simulate-cores If set to greater than 0, the TAE will serialize requests so that no more than these number will execute concurrently.

Aliases: --random-simulate-cores
Default Value: 0
Domain: [0, 2147483647]

--random-simulate-delay If set to true the TAE will simulate the wallclock delay

Aliases: --random-simulate-delay
Default Value: false
Domain: {true, false}

--random-trend-coefficient The Nth sample will be drawn from Max(0,Uniform(min,max) + N$\times$(trend-coefficient)) distribution. This allows you to have the response values increase or decrease over time.

Aliases: --random-trend-coefficient
Default Value: 0.0