Quickstart Guide for SMAC version v2.06.01-development

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

September 24, 2013

1 Introduction

This document is the manual for SMAC [1] (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 $\mathcal{S}$. The goal of this quickstart guide is to get you off the ground quickly; for more detailed information, please see the comprehensive manual.

1.1 License

SMAC is to be released under a dual usage license. Academic & non-commercial usage is permitted free of charge. Please contact us to discuss commercial usage.

2 Usage

2.1 System Requirements

SMAC itself requires only Java 6 or newer to run. The included scripts are currently only available for Unix-compatible operating systems. The included example scenarios require Ruby.

2.2 Running SMAC

Download and extract the SMAC archive from:

`http://www.cs.ubc.ca/labs/beta/Projects/SMAC/smac-VERSION.tar.gz`

This will create a new folder named `smac-VERSION` containing SMAC and several example scenarios. From this folder, you can run SMAC using the simple bash script `./smac`. On windows please use the corresponding .bat files, if bash is otherwise unavailable you will need to manually construct a java call string. The class name needed can be retrieved by opening up the smac file.

2.3 Basic Usage Example

SMAC comes with some small configuration scenarios you can run to ensure everything is set up properly. We will use one of them as an example:

```
example_scenarios/spear/spear-scenario.txt
```

This configures the systematic tree search algorithm Spear for a small training set of 5 SAT instances. (Normally, one would use a much larger set of training instances to avoid over-tuning to the specific instances in the training set – we usually use 1 000 instances if available; here, we did not include as many instances to keep the release small in size.) From `smac-v2.06.01-development-619`, you can start SMAC on this scenario with the following command line:

```
./smac --scenario-file ./example_scenarios/spear/spear-scenario.txt --seed 1 --num-validation-runs 10
```

This command executes SMAC on this simple configuration scenario (which should take just above 30 seconds) and writes several output files to a subfolder of:

```
smac-VERSION/smac-output/
```

In this case the folder will look like:

```
smac-VERSION/smac-output/spear-scenario-SMAC-ac-true-cores1-cutoff5.0-DATE
```

The `--seed` parameter controls SMAC's random seed and output filenames in that folder; since SMAC is highly randomized and its performance differs across runs, we recommend to perform several parallel runs (with different values for `--seed`) and use the one with the best training performance (see [2] for details). The parameter `--num-validation-runs` controls the number of runs to perform for validating the final incumbent SMAC finds at the end of its runtime budget (again, here we set this to a small number of 10 to facilitate a quick example run, but in practice one would use a much larger number, *e.g.* the default of 1 000 validation runs). Note that validation runs are carried out on a benchmark test set disjoint from the training set; this is done to guard against over-tuning since performance on the training set is a biased (optimistic) estimator of generalization performance.

2.4 Interpreting output files

SMAC writes several output files; not all of these are important in everyday use, but if you encounter a problem please zip up and send all of them. The most important output file for users is the log file, here:

```
smac-VERSION/example_out/log-run0.txt
```

This file lists a lot of information about the run and at its end includes an example command line call for the final incumbent, as well as its training and test performance. For example, in the run we performed, this output was:

```
[INFO ] Total Objective of Final Incumbent 65 (0xBD98E) on training set: 0.0316; on test set: 0.036
[INFO ] Sample Call for Final Incumbent 65 (0xBD98E)
cd ./example_scenarios/spear; ruby spear_wrapper.rb instances/qcplin2006.10408.cnf 0 5.0 2147483647
15643122 -sp-update-dec-queue '1' -sp-rand-var-dec-scaling '0.9192384779217873'
-sp-clause-decay '1.3443711377045175' -sp-variable-decay '1.1655745641981787'
```

```
-sp-orig-clause-sort-heur '5' -sp-rand-phase-dec-freq '0.0001' -sp-clause-del-heur '2'
-sp-learned-clauses-inc '1.3325108654268105' -sp-restart-inc '1.1118337573922332'
-sp-resolution '0' -sp-clause-activity-inc '0.7748363656557136' -sp-learned-clause-sort-heur '13'
-sp-var-activity-inc '0.6522039283230536' -sp-rand-var-dec-freq '0.01' -sp-use-pure-literal-rule '1'
-sp-learned-size-factor '1.0325284104478474' -sp-var-dec-heur '5'
-sp-phase-dec-heur '6' -sp-rand-phase-scaling '0.3537413161162636' -sp-first-restart '345'
```

Note that due to SMAC's randomization, the result will likely differ when you execute this command. Also note that training and test performance are rather different from each other here because of the small number of training/test instances used. With larger training and test sets sampled from the same underlying instance distribution, these numbers should be much closer. A better training than test performance can be a sign of over-tuning.

3 Configuring your own algorithms

We will refer to the algorithm to be configured as the *target algorithm*. The quickest way to set up everything needed to configure a new target algorithm is to copy-paste and edit one of the provided examples. The following subsections briefly describe the parts of a configuration scenario and give an example for each. Full details are given in Section 4 of the SMAC manual. All filenames below are relative to the SMAC folder `smac-VERSION/`.

3.1 Scenario File

This is the central file describing all ingredients of a configuration scenario. This file is not actually required since all its contents can also be specified on the command line, but it is still often useful to have as a central specification saved in this file. It references an exectuable for calling the target algorithm (and the path to execute it from), a file describing the target algorithm's parameters, a file containing a list of problem instances, a performance metric, an optional file specifying characteristics for each training instance, and some additional information for the configuration procedure. In the Spear example above, this file is

```
example_scenarios/spear/spear-scenario.txt
```

3.2 Parameter File

This file, referenced in the scenario file, specifies the configurable parameters your target algorithm accepts as an input, along with their domains. In the SPEAR example above, this file is

```
example_scenarios/spear/spear-params-mixed.pcs
```

In this file, notice the different syntax for specifying the domain of categorical parameters (a set of possible values in curly braces) and of numerical parameters (an interval of possible values). For both of these parameter types, the default is given after the domain specification. Also notice the two optional modifiers of numerical parameters given right after the default; 'i' stands for integer parameters and 'l' stands for parameters that naturally vary on a log-scale. For example, Spear parameter *sp-first-restart* is an integer parameter that would naturally be discretized to $25, 50, \ldots, 3200$, with default 100. As a categorical parameter, this would be written as `{25,50,100,200,400,800,1600,3200}[100]`; as an integer parameter, it is now written as `[25, 3200][100]il`.

3.3 Algorithm wrapper

The scenario file references an exectuable of the target algorithm. Since this executable has to follow a rigid input/output format that we don't want to impose on algorithm developers, we typically specify a *wrapper* around the target algorithm that translates between SMAC's and the target algorithm's input/output format. The input specifies (in this order) instance name, instance specifics (which is often empty but could, *e.g.*, hold an optimal solution quality to reach), a cutoff time for the run, a cutoff length for the run (often empty), a seed, and a list of parameter value pairs in the format `-param value`. In the Spear example above, the wrapper is the Ruby script

```
example_scenarios/spear/spear_wrapper.rb
```

Out of its inputs, this wrapper script only uses the instance name, cutoff time, seed, and parameter values; it ignores the two inputs instance specifics and cutoff length. It then runs Spear with the given parameters on the named instance, instructing it to terminate unsuccessfully after the given cutoff time. It then parses Spear's output and prints the following output line, from which SMAC parses the run's performance (in this case `measured_runtime`):

```
Result for ParamILS: <solved>, <measured_runtime>, <measured_runlength>,
<best_solution>, <seed>
```

Note that while this output line is compatible with our previous ParamILS software, SMAC can also parse other result lines. See Section 4.5 in the manual for this and further details on the wrapper.

3.4 Instance File

This file, referenced in the scenario file, specifies the list of training instances to configure the target algorithm on. In the Spear example above, this file is

```
example_scenario/spear/instances-train.txt
```

There is also a file with test instances that will not be used at configuration time, but are used for offline validation of the final incumbent SMAC found; in the Spear example above, this file is

```
example_scenario/spear/instances-test.txt
```

More detailed information about this file format is in Section 4.2 of the manual.

3.5 Feature file

In order to optimize performance across a set or distribution of instances, SMAC accepts an optional input file that contains additional information for each instance in the training set. If such a feature file is given it has to contain an entry for every instance in the training set. Further, the file has to hold the same features (in the same order) for each instance. Any domain-specific instance characteristics (or "features") that might correlate with performance can be useful. If the file contains entries for additional instances these will be ignored by SMAC. (This feature enables users to, *e.g.*, have one file containing the features for all their SAT instances.) In the Spear example above, the feature file is

```
example_scenario/spear/features.csv
```

More detailed information about this file format is in Section 4.3 of the manual.

4 Differences Between SMAC and ParamILS

There are various differences between SMAC and ParamILS. The following differences are described in some more detail in Section 2 of the SMAC manual.

- SMAC natively supports continuous and integer parameters.
- SMAC so far only implements ParamILS's most popular run objectives.
- The order of instances in SMAC's instance file is not important.
- SMAC counts its own overhead time as part of the configuration budget and allows for wall clock timeouts. This can fix ParamILS's large overheads in cases where target algorithm runs are extremely fast.
- SMAC can resume previous runs.
- SMAC accepts optional feature files as input.
- SMAC supports new options for the wrappers.
- SMAC auto-detects instance files vs. instance/seed files.

References

[1] Hutter, F., Hoos, H. H., and Leyton-Brown, K. (2011). Sequential model-based optimization for general algorithm configuration. In *Proc. of LION-5*, LNCS, pages 507–523.

[2] Hutter, F., Hoos, H. H., and Leyton-Brown, K. (2012). Parallel algorithm configuration. In *Proc. of LION-6*, LNCS. To appear.