Quickstart Guide for SMAC version vDEV

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

September 15, 2012

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-vDEV-000.tar.gz
```

This will create a new folder named `smac-vDEV-000` containing SMAC and several example scenarios. From `smac-vDEV-000`, you can run SMAC using the simple bash script `./smac`. If bash is not available (*e.g.*, on Windows), you can run SMAC directly using `java`.[1]

2.3 Basic Usage Example

SMAC comes with some small configuration scenarios you can run to ensure everything is set up properly. The simplest scenario

```
example_saps/scenario-Saps-SWGCP-sat-small-train-small-test.txt
```

configures the local search algorithm SAPS for a small training set of SAT instances.[2] From `smac-vDEV-000`, you can start SMAC on this scenario with the following command line:

```
./smac --scenarioFile
example_saps/scenario-Saps-SWGCP-sat-small-train-small-test.txt
--runGroupName first_example_out --numRun 0 --numberOfValidationRuns 10
```

This command executes SMAC on this simple configuration scenario (which should take just above 30 seconds) and writes several output files to folder `smac-vDEV-000/first_example_out`. The `numRun` 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 `numRun`) and use the one with the best training performance (see [2] for details). The parameter `numberOfValidationRuns` controls the number of runs to perform for validating the final incumbent SMAC finds at the end of its runtime budget; here we set it to a small number of 10 to facilitate a quick example run, but in practice one would use a much larger number (the default is 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

[1]The main entry point class is `ca.ubc.cs.beta.smac.executors.AutomaticConfigurator` and you need to include every jar in the folder in the classpath. You would call it using: `java -cp <jar files> ca.ubc.cs.beta.smac.executors.AutomaticConfigurator`.

[2]Normally, to avoid over-tuning to the specific instances in the training set, one would use a much larger set of training instances – we usually use 1 000 instances if available.

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-vDEV-000/first_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 is

```
[INFO ] Total Objective of Final Incumbent
on training set: 0.7999999999999999; on test set: 1.143
[INFO ] Sample Call for Final Incumbent 5 (0xEACB)
cd example\_saps; ˜ruby saps\_wrapper.rb
example\_data/SWGCP-satisfiable-instances/SWlin2006.8287.cnf 0 5.0
2147483647 3724427 -alpha '1.126' -ps '0.066' -rho '0.333' -wp '0.02'
```

(Note that due to SMAC's randomization, the result will likely differ on your machine.)

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 files are part of a configuration scenario:

- **Scenario file:** This is the central file describing all ingredients of the configuration scenario, referencing the files that follow.
- **Parameter file:** This file specifies the target algorithm's configurable parameters.
- **Instance file:** This file specifies the list of training instances to configure the target algorithm on.
- **Algorithm wrapper:** This is a wrapper that executes the target algorithm and returns a result.

- **Feature file:** This optional file provides additional information characterizing each problem instance.

These files are described by example in the following subsections.

3.1 Scenario File

This is the central file describing all ingredients of the configuration scenario. This file is not required since all its ingredients can also be specified on the command line, but it is still often useful to have as a central specification. 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, and some additional information for the configuration procedure. In the SAPS example above, this file is `example_saps/scenario-Saps-SWGCP-sat-small-train-small-test.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 SAPS example above, this file is `example_saps/saps-params.txt`.

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.

3.3.1 Wrapper Changes

Compared to ParamILS, we allow a new possible result code *ABORT* that is used to signify that not only did the program fail, but that if this occurs the configuration procedure should abort.

3.4 Instance File

This file, referenced in the scenario file, specifies the list of training instances to configure your algorithm on. In the SAPS example above, this file is `example_saps/example_data/SWGCP-satisfiable-small-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 SAPS example above, this file is
`example_data/SWGCP-satisfiable-small-test.txt`.

3.5 Feature file

In order to optimize performance across instance set $\mathcal{S}$, SMAC accepts an optional input file that contains additional information for each instance $s \in \mathcal{S}$. Any domain-specific instance characteristics (or "features") that might correlate with performance can be useful. The file can also contain entries for additional instances; those will be ignored by SMAC. In the SAPS example above, the feature file is
`example_saps/SWGCP-small-train-features.csv`.

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.