Quickstart Guide for SMAC version v2.08.00-development

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

April 10, 2014

1 Introduction

This document is the quick start guide for SMAC [2] and is designed to get you off the ground quickly; for more detailed information please see the accompanying manual. Additionally, some questions are answered in the accompanying FAQ document.

2 Usage

2.1 Prerequisites

SMAC only requires Java 7 to run, and should work on both Windows and UNIX based operating systems. Some of the included examples require ruby version 1.9.3 or later.

2.2 Downloading SMAC

Download and unpack the SMAC archive from:

```
http://www.cs.ubc.ca/labs/beta/Projects/SMAC/smac-v2.08.00-development-676.tar.gz
```

This will create a new folder named `smac-v2.08.00-development-676` 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 `smac.bat` file.

Additionally, SMAC includes some example scenarios that you can use in the `example_scenarios` folder.

2.3 Getting Ready to Run

2.3.1 Creating a PCS File

First you will need a Parameter Configuration Space (PCS) file. The format is outlined in the section of the SMAC Manual entitled "Algorithm Parameter File". It contains a description of the space of allowable configurations for your algorithm and supports both categorical and numeric parameter types.

Included with SMAC are some examples that we will refer to, example PCS files are located in:

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

You will pass this file to SMAC via the **--pcs-file** argument.

2.3.2 Algorithm Wrapper

Next you will need to create a *wrapper*. A wrapper is a program that acts as a bridge between SMAC and your algorithm. SMAC invokes the wrapper with a predefined format, the wrapper translates the call into the appropriate call for your algorithm, and then invokes it. The wrapper is responsible for ensuring that the algorithm does not exceed it's time budget, and finally it translates the output of the algorithm into the format required by SMAC. The specifics of the input and output required of the wrapper are detailed in the manual section "Wrappers".

The AClib project (https://www.aclib.net) maintains a good template wrapper that is designed to be easy to adapt, and we recommend that you use it as a basis for writing your wrapper as there are many subtleties.

2.3.3 Readying Problem Instances

An *instance* in SMAC terminology is a particular instantiation of a problem for your algorithm, for example a graph colouring algorithm might have a set of graphs to color, each individual graph is referred to as an instance. If your algorithm doesn't use instances, then you can add **--no-instances** to `true` on the command line, and skip the remainder of this subsection (SMAC will pass a dummy argument in this case for the instance).

When using SMAC it is recommended that you partition your instances into two sets, a training set, and a test set. If your instances exist on the file system, the easiest way to do this is to move them into two folders, one called train and one called test. Then invoke smac with:

--instances `<train-path>` **--test-instances** `<test-path>`

Once SMAC completes, it will perform validation to obtain a better measure of the selected configurations performance. If you do not specify the **--test-instances** argument, no validation will be performed.

2.4 Running SMAC

Once you have the previous three steps completed, then invoking SMAC is fairly straight forward. A more detailed description of the following options is available in the manual under the "Scenario File" section.

The first option which must always be set, is how much CPU time each invocation of the target algorithm should be allowed[1]. This limit is set with the **--algo-cutoff-time** option and takes an integer in seconds.

Secondly, you must decide whether you'd like SMAC to *minimize* your algorithms' runtime, or some other measure of solution quality. To minimize runtime set the argument **--run-obj** to `RUNTIME` otherwise set it to `QUALITY`. A few things change depending on which option you pick internally in SMAC. Of particular note, is that when set to `QUALITY` SMAC tries to minimize the mean across instances, but when set to `RUNTIME` SMAC minimizes the *PAR10*[2]. There are other options that are affected by this choice as well, again consult the manual for a more detailed description.

[1] It is entirely up to the wrapper to measure, police and ultimately report this value.

[2] The PAR10 score is the the arithmetic mean of each runtime, except that runtimes that hit the limit **--algo-cutoff-time** are multiplied by 10.

Finally you must decide how long SMAC itself should run. You can do that by setting some combination of the following three arguments [3]:

--wallclock-limit How long SMAC should run for in seconds.

--cputime-limit How many CPU seconds SMAC and the algorithm are allowed to consume. Loosely speaking, SMAC will terminate once the sum of the reported runtimes and SMAC's internal measurement of it's own CPU time passes this threshold.

--runcount-limit How many invocations of the target algorithm are allowed to occur before SMAC terminates.

So a final call to SMAC might look like

```
./smac --pcs-file <pcsfile> --instances <instances-folder>
--algo <call-to-wrapper> --seed <seed> --wallclock-limit <wallclock-time>
--algo-cutoff-time <algo-cutoff-time> --run-obj <run-obj>
```

For instance to run the included SPEAR scenario on Linux you could execute:

```
./smac --pcs-file ./example_scenarios/spear/spear-params-mixed.pcs --instances
./example_scenarios/spear/instances/train/ --algo "ruby
 ./example_scenarios/spear/spear_wrapper.rb" --seed 1 --wallclock-limit 30
--algo-cutoff-time 5 --run-obj RUNTIME
```

And to run the included SAPS scenario on Windows you could execute:

```
smac.bat --pcs-file ./example_scenarios/saps/saps-params.pcs --instances
.\example_scenarios\saps\instances\train --algo "ruby
.\example_scenarios\saps\win_saps_wrapper.rb" --test-instances .\example_scenari
os\saps\instances\test --seed 1 --wallclock-limit 30 --algo-cutoff-time 5 --run-
obj RUNTIME
```

Note: Because SMAC is a randomized algorithm its performance differs across runs, thus we recommend that you perform several parallel runs (with different values of **--seed**) and use the one with the best reported training performance (see [1] for details).

2.5 Interpreting SMACs output

Once SMAC completes it should print out a sample call string to your algorithm, as well as the performance of the configuration selected (which SMAC denotes the *final incumbent*), on the training instances and test instances (if available).

For example a SMAC run on the SAPS scenario above, yields the following output near the end.[4]:

[3] SMAC will terminate once the first limit is reached

[4] Because SMAC is a randomized algorithm, the exact output you see will likely be different.

```
[INFO ] Total Objective of Final Incumbent 3 (0x754D) on training set: 0.0480733
3310445453; on test set: 0.022399997711184
[INFO ] Sample Call for Final Incumbent 3 (0x754D)
cd C:\smac-v2.06.02-development-660& ruby .\example_scenarios\saps\win_saps_wrapper
.\example_scenarios\saps\instances\train\SWlin2006.10286.cnf 0 5.0 2147483647
15686777 -alpha '1.2552471818554198' -wp '0.011088485535736967' -rho '0.312
2844671947156' -ps '0.038973215638773054'
```

In the above example, the training performance is approximately 0.04 and the test performance is approximately 0.02 (Since this is a `RUNTIME` run, this is the PAR10 score for each set).

The optimized parameter values for SAPS are:

Parameter	Value (rounded)
alpha	1.25524
ps	0.03897
rho	0.31228
wp	0.01108

SMAC will also spit out more details and information to files, including a log and various CSV files that detail its performance in more detail. See the section "Interpreting SMAC's Output" in the manual for more information.
Good Luck ☺

References

[1] Hutter, F., Hoos, H. H., and K.Leyton-Brown (2012). Parallel algorithm configuration. In *Proc. of LION-6*, pages 55–70.

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