Sparkle user guide

March 31, 2021

1 Quick start

Note: Sparkle currently relies on Slurm (`https://slurm.schedmd.com/`), and does not work without it.

Follow these steps:

1.1 Install Sparkle

1.2 Prepare your configuration environment; or 1.3 Prepare your selection environment

1.4 Execute commands

1.1 Installing Sparkle

1. Copy the Sparkle files to your desired directory
2. Install Python 3.9 (other 3.x versions may work, but were not tested with the packages included in the `requirements_first.txt` and `requirements_second.txt` files.
   With Anaconda:
   `conda create -n <env_name> python=3.9`
3. Install Swig 3.0
   With Anaconda:
   `conda install swig=3.0`
4. Navigate into the Sparkle directory
5. Install `requirements_first.txt`:
   `pip install -r requirements_first.txt`
   With Anaconda:
   `/home/<username>/<anaconda_dir>/envs/<env_name>/bin/pip install -r requirements_first.txt`

6. Install `requirements_second.txt`:
   `pip install -r requirements_second.txt`
   With Anaconda:
   `/home/<username>/<anaconda_dir>/envs/<env_name>/bin/pip install -r requirements_second.txt`
7. Install `epstopdf`
   (if manually, for instance on a cluster, as described in Section 6.1.2)
8. Install other requirements if they are not on your system yet:
   `LaTeX`
   `BibTeX`
   `gnuplot`

1.2 Algorithm Configuration

Configuring an algorithm has the following minimal requirements for the algorithm (for an example of a solver directory see Section 2.2):

- A working solver executable

1.2.1 An algorithm wrapper called `sprakle_smac_wrapper.py`

1.2.2 A PCS (parameter configuration space) file

- The runsolver binary (e.g. from `Examples/Resources/Solvers/PbO-CCSAT-Generic/`)

Further, training and testing instance sets are needed (for an example of an instances directory see Section 2.1). For the purpose of testing whether your configuration setup works with Sparkle, it is advised to primarily use instances that are solved (relatively) quickly even with the default parameters.

1.2.1 Creating a wrapper for your algorithm

A template for the wrapper that connects your algorithm with Sparkle is available at `Examples/Resources/Solvers/template/sparkle_smac_wrapper.py`. Within this template a number of `TODO`s are indicated where you are likely to need to make changes for your specific algorithm. You can also compare the different example solvers to get an idea for what kind of changes are needed.

1.2.2 Parameter configuration space (PCS) file

The PCS (parameter configuration space) format[1] is used to pass the possible parameter ranges of an algorithm to Sparkle in a `.pcs` file. For an example see e.g. `Examples/Resources/Solvers/PbO-CCSAT-Generic/PbO-CCSAT-params_test.pcs`.

In this file you should enter all configurable parameters of your algorithm. Note that parameters such as the random seed used by the algorithm should not be configured and therefore should also not be included in the PCS file.

[1] See: `http://aclib.net/cssc2014/pcs-format.pdf`

1.3 Algorithm Selection

Creating a portfolio selector requires multiple algorithms with the following minimal requirements (for an example of a solver directory see Section 2.3):

- A working solver executable

1.3.1 An algorithm wrapper called `sprakle_run_default_wrapper.py`

Further, training and testing instance sets are needed (for an example of an instances directory see Section 2.1). For the purpose of testing whether your selection setup works with Sparkle, it is advised to primarily use instances that are solved (relatively) quickly.

1.3.1 Creating a wrapper for your algorithm

A template for the wrapper that connects your algorithm with Sparkle is available at `Examples/Resources/Solvers/template/sparkle_run_default_wrapper.py`. Within this template a number of `TODO`s are indicated where you are likely to need to make changes for your specific algorithm. You can also compare the different example solvers to get an idea for what kind of changes are needed.

1.4 Executing commands

Executing commands in Sparkle is as simple as running them in the top directory of Sparkle, for example:

```
Commands/initialise.py
```

Do note that when running on a cluster additional arguments may be needed, for instance under Slurm the above command would change to something like:

```
srun -N1 -n1 -p graceTST Commands/initialise.py
```

In the `Examples/` directory a number of common command sequences are given. For instance, for configuration with specified training and testing sets see e.g. `Examples/configuration.md` for an example of a sequence of commands to execute. Note that some command run in the background and need time to complete before the next command is executed. To see whether a command is still running the Slurm command `squeue` can be used.

In the `Output/` directory paths to generated scripts and logs are gathered per executed command.

2 File structure

2.1 A typical instance directory

An instance directory should look something like this:

```
Instances/
  Example_Instance_Set/
    instance_a.cnf
    instance_b.cnf
    ...         ...
    instance_z.cnf
```

This directory simply contains a collection of instances, as example here SAT instances in the CNF format are given.

For instances consisting of multiple files one additional file should be included in the `Example_Instance_Set` directory, describing which files together form an instance. The format is a signle instance per line with each file separated by a space, as shown below.

```
instance_a_part_one.abc instance_a_part_two.xyz
instance_b_part_one.abc instance_b_part_two.xyz
...                     ...
instance_z_part_one.abc instance_z_part_two.xyz
```

2.2 A typical solver directory (configuration)

A solver directory should look something like this:

```
Solver/
  Example_Solver/
    solver
    sparkle_smac_wrapper.py
    parameters.pcs
    runsolver
```

Here `solver` is a binary executable of the solver that is to be configured. The `sprakle_smac_wrapper.py` is a wrapper that Sparkle should call to run the solver with specific settings, and then returns a result for the configurator. In `parameters.pcs` the configurable parameters are described in the PCS format. Finally, `runsolver` is a binary executable of the runsolver tool. This allows Sparkle to make fair time measurements for all configuration experiments.

Note: Currently the runsolver binary has to be in every solver directory, it can be found in the `Examples/Resources/Solvers/PbO-CCSAT-Generic/` directory.

2.3 A typical solver directory (selection)

A solver directory should look something like this:

```
Solver/
  Example_Solver/
    solver
    sparkle_run_default_wrapper.py
```

Here `solver` is a binary executable of a solver that is to be included in a portfolio selector. The `sprakle_run_default_wrapper.py` is a wrapper that Sparkle should call to run the solver on a specific instance.

3 Wrappers

3.1 `sparkle_run_default_wrapper.py`

The `sparkle_run_default_wrapper.py` has two functions that need to be implemented for each algorithm:

- `print_command(instance_file, seed_str: str, cutoff_time_str: str)`
- `print_output(terminal_output_file: str)`

`print_command(...)` should print a command line call that Sparkle can use to run the algorithm on a given instance file. Ideally, for reproducibility purposes, the seed provided by Sparkle should also be passed to the algorithm. If the algorithm requires this, the cutoff time can also be passed to the algorithm. However, in this case the cutoff time should be made very large. For instance by multiplying by ten with: `cutoff_time_str = str(int(cutoff_time_str) * 10)`. This is necessary to ensure Sparkle stops the algorithm after the cutoff time, rather than the algorithm itself. By doing this it is ensured runtime measurements are always done by Sparkle, and thus consistent between algorithms that might measure time differently.

`print_output(...)` should process the algorithm output. If the performance measure is `RUNTIME`, this function only needs to output the algorithm status. For all `QUALITY` performance measures both the algorithm status and the solution quality have to be given. Sparkle internally measures `RUNTIME`, while it can be overwritten by the user if desired, for consistent runtime measurements between solvers this is not recommended. The output should be printed and formatted as in the example below.

```
quality 8734
status SUCCESS
```

Status can hold the following values {`SUCCESS, TIMEOUT, CRASHED`}. If the status is not known, reporting `SUCCESS` will allow Sparkle to continue, but may mean that Sparkle does not know when the algorithm crashed, and continues with faulty results.

4 Commands

Currently the commands below are available in Sparkle (listed alphabetically). Every command can be called with the `--help` option to get a description of the required arguments and other options.

about.py
add_feature_extractor.py
add_instances.py
4.1 add_solver.py
cleanup_current_sparkle_platform.py
cleanup_temporary_files.py
compute_features_parallel.py
compute_features.py
compute_marginal_contribution.py
4.2 configure_solver.py
construct_sparkle_portfolio_selector.py
4.3 generate_report.py
4.4 initialise.py
load_record.py
remove_feature_extractor.py
remove_instances.py
remove_record.py
remove_solver.py
run_ablation.py
run_solvers.py
run_sparkle_portfolio_selector.py
run_status.py
save_record.py
system_status.py
4.6 validate_configured_vs_default.py

Arguments in [square brackets] are optional, arguments without brackets are mandatory. Input in <chevrons> indicate required text input, {curly brackets} indicate a set of inputs to choose from.

4.1 add_solver.py

Add a solver to the Sparkle platform.

Arguments:

```
[--run-solver-later]
[--parallel]
--deterministic {0, 1}
<solver_source_directory>
```

4.2 configure_solver.py

Configure a solver in the Sparkle platform.

Arguments:

```
--solver <solver>
--instance-set-train <instance-set-train>
```

```
[--instance-set-test <instance-set-test>]
--validate
--ablation
```

Note that the test instance set is only used if the `--ablation` or `--validation` flags are given.

4.3 `generate_report.py`

Without any arguments a report for the most recent algorithm selection or algorithm configuration procedure is generated.

4.3.1 Generate a configuration report

Generate a report describing the configuration results for a solver and specific instance sets in the Sparkle platform.

Arguments:

```
--solver <solver>
[--instance-set-train <instance-set-train>]
[--instance-set-test <instance-set-test>]
```

Note that if a test instance set is given, the training instance set must also be given.

4.4 `initialise.py`

Initialise the Sparkle platform, this command does not have any arguments.

4.5 `run_ablation.py`

Runs parameter importance between the default and configured parameters with ablation. This command requires a finished configuration for the solver instance pair.

Arguments:

```
--solver <solver>
[--instance-set-train <instance-set-train>]
[--instance-set-test <instance-set-test>]
```

Note that if no test instance set is given, the validation is performed on the training set.

4.6 `validate_configured_vs_default.py`

Test the performance of the configured solver and the default solver by doing validation experiments on the training and test sets.

Arguments:

```
--solver <solver>
--instance-set-train <instance-set-train>
[--instance-set-test <instance-set-test>]
```

5 Settings

5.1 Sparkle settings

Most settings can be controlled through `Settings/sparkle_settings.ini`. Possible settings are summarised per category in Sect. 5.1.2. For any settings that are not provided the defaults will be used. Meaning, in the extreme case, that if the settings file is empty (and nothing is set through the command line) everything will run with default values.

For convenience after every command `Settings/latest.ini` is written with the used settings. This can, for instance, be used to provide the same settings to the next command in a chain. E.g. for `validate_configured_vs_default` after `configure_solver`. The used settings are also recorded in the relevant `Output/` subdirectory. Note that when writing settings Sparkle always uses the name, and not an alias.

5.1.1 Example `sparkle_settings.ini`

This is a short example to show the format, see the settings file in `Settings/sparkle_settings.ini` for more.

```
[general]
performance_measure = RUNTIME
target_cutoff_time = 60

[configuration]
number_of_runs = 25

[slurm]
number_of_runs_in_parallel = 25
```

5.1.2 Names and possible values

[general]

`performance_measure`
aliases: `smac_run_obj`
values: {`RUNTIME, QUALITY_ABSOLUTE` (also: `QUALITY`)}

`target_cutoff_time`
aliases: `smac_each_run_cutoff_time, cutoff_time_each_performance_computation`
values: integer

`extractor_cutoff_time`
aliases: `cutoff_time_each_feature_computation`
values: integer

`penalty_multiplier`
aliases: `penalty_number`
values: integer

`solution_verifier`
aliases: N/A
values: {`NONE, SAT`}
note: Only available for SAT solving.

[**configuration**]

`budget_per_run`
aliases: `smac_whole_time_budget`
values: integer

`number_of_runs`
aliases: `num_of_smac_runs`
values: integer

[**smac**]
`target_cutoff_length`
aliases: `smac_each_run_cutoff_length`
values: {`max`} (other values: whatever is allowed by SMAC)

[**ablation**]

`racing`
aliases: `ablation_racing`
values: boolean

[**slurm**]

`number_of_runs_in_parallel`
aliases: `smac_run_obj`
values: integer

`clis_per_node`
aliases: N/A
values: integer
note: Not really a Slurm option, will likely be moved to another section.

5.2 Priorities

Settings provided through different channels have different priorities as follows:

low Default – Default values will be overwritten if a value is given through any other mechanism;

medium File – Settings form the `Settings/sparkle_settings.ini` overwrite default values, but are overwritten by settings given through the command line;

high-1 Command line file – Settings files provided through the command line, overwrite default values and other settings files.

high-2 Command line – Settings given through the command line overwrite all other settings, including settings files provided through the command line.

5.3 Slurm (focused on Grace)

Slurm settings can be specified in the `Settings/sparkle_slurm_settings.txt` file. Currently these settings are inserted *as is* in any `srun` or `sbatch` calls done by Sparkle. This means that any options exclusive to one or the other currently should not be used (see Section 5.3.2).

5.3.1 Tested options

Below a list of tested Slurm options for `srun` and `sbatch` is included. Most other options for these commands should also be safe to use (given they are valid), but have not been explicitly tested. Note that any options related to commands other than `srun` and `sbatch` should not be used with Sparkle, and should not be included in `Settings/sparkle_slurm_settings.txt`.

```
--partition / -p
--exclude
--nodelist
```

5.3.2 Disallowed options

The options below are exclusive to `sbatch` and are thus disallowed:

```
--array
--clusters
--wrap
```

The options below are exclusive to `srun` and are thus disallowed:

```
--label
```

5.3.3 Nested `srun` calls

A number of Sparkle commands internally call the `srun` command, and for those commands the provided settings need to match the restrictions of your call to a Sparkle command. Take for instance the following command:

```
srun -N1 -n1 -p graceTST Commands/configure_solver.py
   --solver Solvers/PbO-CCSAT-Generic --instances-
   train Instances/PTN/
```

This call restricts itself to the `graceTST` partition (the `graceTST` partition only consists of node 22). So if the settings file contains the setting `--exclude=ethnode22`, all available nodes are excluded, and the command cannot execute any internal `srun` commands it may have.

Finally, Slurm ignores nested partition settings for `srun`, but not for `sbatch`. This means that if you specify the `graceTST` partition (as above) in your command, but the `graceADA` partition in the settings file, Slurm will still execute any nested `srun` commands on the `graceTST` partition only.

6 Required packages

6.1 Sparkle on Grace

Grace is the computing cluster of the ADA group[2] at LIACS, Leiden University. Since not all packages required by Sparkle are installed on the system, some have to be installed local to the user.

6.1.1 Making your algorithm run on Grace

Shell and Python scripts should work as is. If a compiled binary does not work, you may have to compile it on Grace and manually install packages on Grace that are needed by your algorithm.

6.1.2 `epstopdf`

The `epstopdf` package (or a package containing it) is required for Sparkle's reporting component to work (e.g. `generate_report, generate_report_for_configuration`), it can be installed in your user directory as follows:

1. Download `epstopdf`

   ```
   wget http://mirrors.ctan.org/support/epstopdf.zip
   ```

2. Unzip the package (ideally somewhere static, rather than a `/Downloads/` directory)

   ```
   unzip epstopdf.zip
   ```

[2] `http://ada.liacs.nl/`

3. Rename `epstopdf.pl` (inside the directory you just unzipped)
   `mv epstopdf.pl epstopdf`
4. Add this line to your `.bashrc` (open with e.g. `vim ~/.bashrc`)
   `export PATH="/<directory>/epstopdf:$PATH"`
   (replace "`<directory>`" with the path to the `epstopdf` directory, e.g.: `home/blomkvander/bin`)
5. Reload `.bashrc` to make sure everything is updated
   `source ~/.bashrc`

6.1.3 General requirements

Other software used by Sparkle:

```
pdflatex,
latex,
bibtex,
gnuplot,
gnuplot-x11
```

To manually install `gnuplot` see for instance the instructions on their website `http://www.gnuplot.info/development/`

7 Installation and compilation of examples

7.1 Solvers

7.1.1 CSCCSat

CSCCSat can be recompiled as follows in the `Examples/Resources/Solvers/CSCCSat/` directory:

```
unzip src.zip
cd src/CSCCSat_source_codes/
make
cp CSCCSat ../../
```

7.1.2 MiniSAT

MiniSAT can be recompiled as follows in the `Examples/Resources/Solvers/MiniSAT/` directory:

```
unzip src.zip
cd minisat-master/
make
cp build/release/bin/minisat ../
```

7.1.3 PbO-CCSAT

PbO-CCSAT can be recompiled as follows in the `Examples/Resources/Solvers/PbO-CCSAT-Generic/` directory:

```
unzip src.zip
cd PbO-CCSAT-master/PbO-CCSAT_process_oriented_version_source_code/
make
cp PbO-CCSAT ../../
```

7.1.4 TCA and FastCA

The TCA and FastCA solvers, require `GLIBCXX_3.4.21`. This library comes with `GCC 5.1.0` (or greater). Following installation you may have to update environment variables such as `LD_LIBRARY_PATH, LD_RUN_PATH, CPATH` to point to your installation directory.

TCA can be recompiled as follows in the `Examples/Resources/CCAG/Solvers/TCA/` directory:

```
unzip src.zip
cd TCA-master/
make clean
make
cp TCA ../
```

FastCA can be recompiled as follows in the `Examples/Resources/CCAG/Solvers/FastCA/` directory:

```
unzip src.zip
cd fastca-master/fastCA/
make clean
make
cp FastCA ../../
```

7.1.5 VRP_SISRs

VRP_SISRs can be recompiled as follows in the `Examples/Resources/CVRP/Solvers/VRP_SISRs/` directory:

```
unzip src.zip
cd src/
make
cp VRP_SISRs ../
```