Sparkle Documentation

Release 0.0

ADA Research Group, LIACS

May 19, 2022

CONTENTS:

CHAPTER

ONE

SPARKLE USER GUIDE

1.1 Quick start

Note: Sparkle currently relies on Slurm, and does not work without it.

Follow these steps:

1. *Install Sparkle*
2. Prepare an *algorithm configuration* or an *algorithm selection*.
3. *Execute commands*

1.2 Installing Sparkle

Note: The installation process use the `conda` command available in Anaconda or Miniconda to manage some dependencies.

1.2.1 Get a copy of Sparkle

To get a copy of Sparkle you can clone the repository.

If `git` is available on your system, this will clone the Sparkle repository and create a subdirectory named `sparkle` :

```
$ git clone https://bitbucket.org/sparkle-ai/sparkle.git
```

You can also download the stable version here: https://bitbucket.org/sparkle-ai/sparkle/get/main.zip

1.2.2 Install dependencies

Sparkle depends on Python 3.9+, swig 3.0, gnuplot, LaTeX, and multiple Python packages. An easy way to install most of what is needed is to use the `conda` package manager (https://docs.conda.io/en/latest/miniconda.html).

Note: LaTeX is used to create the reports and the documentation and must be installed manually on the system. If you don't plan to use the reports or recreate the documentations, you can skip it.

You can install the base requirements with

```
$ conda env create -f environment.yml
```

This will create an environment named `sparkle` that contains everything needed to run Sparkle, except LaTeX that needs to be installed manually.

To activate the environment in the current shell, execute:

```
$ conda activate sparkle
```

Note: You will need to reactivate the environment every time you log in, before using Sparkle.

The file `environment.yml` contains a tested list of Python packages with fixed versions required to execute Sparkle. We recommended using it.

The file `environment-dev.txt` contains unpinned packages and the dependencies are not resolved. It is used for development and may cause problems.

The two environments can be created in parallel since one is named `sparkle` and the other `sparkle-dev`. If you want to update an environment it is better to do a clean installation by removing and recreating it. For example:

```
$ conda deactivate
$ conda env remove -n sparkle
$ conda env create -f environment.yml
$ conda activate sparkle
```

This should be fast as both `conda` and `pip` use local cache for the packages.

1.3 Algorithm Configuration

Configuring an algorithm has the following minimal requirements for the algorithm (for an example of a solver directory see Section 1.6.2):

- A working solver executable
- An algorithm wrapper called `sprakle_smac_wrapper.py`
- 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 1.6.1). 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.

Note: See the *example* page for a walk-through on how to perform configuration with Sparkle.

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_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.

1.3.2 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.

1.4 Algorithm Selection

Creating a portfolio selector requires multiple algorithms with the following minimal requirements (for an example of a solver directory see Section 1.6.3):

- A working solver executable
- 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 1.6.1). For the purpose of testing whether your selection setup works with Sparkle, it is advised to primarily use instances that are solved (relatively) quickly.

Note: See the *example* page for a walk-through on how to perform selection with Sparkle.

1.4.1 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] See: http://aclib.net/cssc2014/pcs-format.pdf

1.5 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 the Slurm workload manager the above command would change to something like:

```
srun -N1 -n1 -c1 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.

1.6 File structure

1.6.1 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 called `sparkle_instance_list.txt` should be included in the `Example_Instance_Set` directory, describing which files together form an instance. The format is a single 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
```

1.6.2 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.

1.6.3 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.

1.7 Wrappers

1.7.1 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.

1.8 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*
- *add_solver.py*
- *cleanup_current_sparkle_platform.py*
- *cleanup_temporary_files.py*
- *compute_features.py*
- *compute_features_parallel.py*
- *compute_marginal_contribution.py*
- *configure_solver.py*
- *construct_sparkle_portfolio_selector.py*
- *generate_report.py*
- *initialise.py*
- *load_record.py*
- *remove_feature_extractor.py*
- *remove_instances.py*
- *remove_record.py*
- *remove_solver.py*
- *run_ablation.py*
- *run_configured_solver.py*
- *run_solvers.py*
- *run_sparkle_portfolio_selector.py*
- *run_status.py*
- *save_record.py*
- *sparkle_wait.py*
- *system_status.py*
- *validate_configured_vs_default.py*

Note: 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.

1.8.1 about.py

```
usage: about.py [-h]
```

-h, --help
show this help message and exit

1.8.2 add_feature_extractor.py

```
usage: add_feature_extractor.py [-h] [--run-extractor-now | --run-extractor-later] [--
↪nickname NICKNAME] [--parallel] extractor-path
```

extractor-path
path to the feature extractor

-h, --help
show this help message and exit

--run-extractor-now
immediately run the newly added feature extractor on the existing instances

--run-extractor-later
do not immediately run the newly added feature extractor on the existing instances (default)

--nickname <nickname>
set a nickname for the feature extractor

--parallel
run the feature extractor on multiple instances in parallel

1.8.3 add_instances.py

```
usage: add_instances.py [-h] [--run-extractor-now | --run-extractor-later] [--run-solver-
↪now | --run-solver-later] [--nickname NICKNAME] [--parallel] instances-path
```

instances-path
path to the instance set

-h, --help
show this help message and exit

--run-extractor-now
immediately run the feature extractor(s) on the newly added instances

--run-extractor-later
do not immediately run the feature extractor(s) on the newly added instances (default)

--run-solver-now
immediately run the solver(s) on the newly added instances

--run-solver-later
do not immediately run the solver(s) on the newly added instances (default)

--nickname <nickname>
: set a nickname for the instance set

--parallel
: run the solvers and feature extractor on multiple instances in parallel

1.8.4 add_solver.py

Add a solver to the Sparkle platform.

```
usage: add_solver.py [-h] --deterministic {0,1} [--run-solver-now | --run-solver-later]␣
↪[--nickname NICKNAME] [--parallel] solver-path
```

solver-path
: path to the solver

-h, --help
: show this help message and exit

--deterministic {0,1}
: indicate whether the solver is deterministic or not

--run-solver-now
: immediately run the newly added solver on all instances

--run-solver-later
: do not immediately run the newly added solver on all instances (default)

--nickname <nickname>
: set a nickname for the solver

--parallel
: run the solver on multiple instances in parallel

1.8.5 cleanup_current_sparkle_platform.py

```
usage: cleanup_current_sparkle_platform.py [-h]
```

-h, --help
: show this help message and exit

1.8.6 cleanup_temporary_files.py

```
usage: cleanup_temporary_files.py [-h]
```

-h, --help
: show this help message and exit

1.8.7 compute_features.py

```
usage: compute_features.py [-h] [--recompute] [--parallel] [--settings-file SETTINGS_
↪FILE]
```

-h, --help
show this help message and exit

--recompute
re-run feature extractor for instances with previously computed features

--parallel
run the feature extractor on multiple instances in parallel

--settings-file
specify the settings file to use in case you want to use one other than the default

1.8.8 compute_features_parallel.py

```
usage: compute_features_parallel.py [-h] [--recompute]
```

-h, --help
show this help message and exit

--recompute
re-run feature extractor for instances with previously computed features

1.8.9 compute_marginal_contribution.py

```
usage: compute_marginal_contribution.py [-h] (--perfect | --actual) [--recompute] [--
↪performance-measure {RUNTIME,QUALITY_ABSOLUTE,QUALITY}] [--settings-file SETTINGS_FILE]
```

-h, --help
show this help message and exit

--perfect
compute the marginal contribution for the perfect selector

--actual
compute the marginal contribution for the actual selector

--recompute
force marginal contribution to be recomputed even when it already exists in file for for the current selector

--performance-measure
the performance measure, e.g. runtime

--settings-file
specify the settings file to use in case you want to use one other than the default

1.8.10 configure_solver.py

Configure a solver in the Sparkle platform.

```
usage: configure_solver.py [-h] [--validate] [--ablation] --solver SOLVER --instance-set-
↪train INSTANCE_SET_TRAIN [--instance-set-test INSTANCE_SET_TEST] [--performance-
↪measure {RUNTIME,QUALITY_ABSOLUTE,QUALITY}]
                           [--target-cutoff-time TARGET_CUTOFF_TIME] [--budget-per-run␣
↪BUDGET_PER_RUN] [--number-of-runs NUMBER_OF_RUNS] [--settings-file SETTINGS_FILE]
```

-h, --help
show this help message and exit

--validate
validate after configuration

--ablation
run ablation after configuration

--solver <solver>
path to solver

--instance-set-train <instance_set_train>
path to training instance set

--instance-set-test <instance_set_test>
path to testing instance set (only for validating)

--performance-measure
the performance measure, e.g. runtime

--target-cutoff-time
cutoff time per target algorithm run in seconds

--budget-per-run
configuration budget per configurator run in seconds

--number-of-runs
number of configuration runs to execute

--settings-file
specify the settings file to use instead of the default

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

1.8.11 construct_sparkle_portfolio_selector.py

```
usage: construct_sparkle_portfolio_selector.py [-h] [--recompute-portfolio-selector] [--
↪recompute-marginal-contribution] [--performance-measure {RUNTIME,QUALITY_ABSOLUTE,
↪QUALITY}]
```

-h, --help
show this help message and exit

--recompute-portfolio-selector

force the construction of a new portfolio selector even when it already exists for the current feature and performance data. NOTE: This will also result in the computation of the marginal contributions of solvers to the new portfolio selector.

--recompute-marginal-contribution

force marginal contribution to be recomputed even when it already exists in file for the current selector

--performance-measure

the performance measure, e.g. runtime

1.8.12 generate_report.py

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

```
usage: generate_report.py [-h] [--solver SOLVER] [--instance-set-train INSTANCE_SET_
↪TRAIN] [--instance-set-test INSTANCE_SET_TEST] [--no-ablation [FLAG_ABLATION]] [--
↪selection] [--test-case-directory TEST_CASE_DIRECTORY]
                          [--performance-measure {RUNTIME,QUALITY_ABSOLUTE,QUALITY}] [--
↪settings-file SETTINGS_FILE]
```

-h, --help

show this help message and exit

--solver <solver>

path to solver for an algorithm configuration report

--instance-set-train <instance_set_train>

path to training instance set included in Sparkle for an algorithm configuration report

--instance-set-test <instance_set_test>

path to testing instance set included in Sparkle for an algorithm configuration report

--no-ablation <flag_ablation>

turn off reporting on ablation for an algorithm configuration report

--selection

set to generate a normal selection report

--test-case-directory <test_case_directory>

Path to test case directory of an instance set for a selection report

--performance-measure

the performance measure, e.g. runtime

--settings-file

specify the settings file to use in case you want to use one other than the default

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

1.8.13 initialise.py

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

```
usage: initialise.py [-h]
```

-h, --help
show this help message and exit

1.8.14 load_record.py

```
usage: load_record.py [-h] record-file-path
```

record-file-path
path to the record file

-h, --help
show this help message and exit

1.8.15 remove_feature_extractor.py

```
usage: remove_feature_extractor.py [-h] [--nickname] extractor-path
```

extractor-path
path to or nickname of the feature extractor

-h, --help
show this help message and exit

--nickname
if set to True extractor_path is used as a nickname for the feature extractor

1.8.16 remove_instances.py

```
usage: remove_instances.py [-h] [--nickname] instances-path
```

instances-path
path to or nickname of the instance set

-h, --help
show this help message and exit

--nickname
if given instances_path is used as a nickname for the instance set

1.8.17 remove_record.py

```
usage: remove_record.py [-h] record-file-path
```

record-file-path

path to the record file

-h, --help

show this help message and exit

1.8.18 remove_solver.py

```
usage: remove_solver.py [-h] [--nickname] solver-path
```

solver-path

path to or nickname of the solver

-h, --help

show this help message and exit

--nickname

if set to True solver_path is used as a nickname for the solver

1.8.19 run_ablation.py

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

```
usage: run_ablation.py [-h] [--solver SOLVER] [--instance-set-train INSTANCE_SET_TRAIN]␣
↪[--instance-set-test INSTANCE_SET_TEST] [--ablation-settings-help] [--performance-
↪measure {RUNTIME,QUALITY_ABSOLUTE,QUALITY}]
                        [--target-cutoff-time TARGET_CUTOFF_TIME] [--budget-per-run␣
↪BUDGET_PER_RUN] [--number-of-runs NUMBER_OF_RUNS] [--racing RACING] [--settings-file␣
↪SETTINGS_FILE]
```

-h, --help

show this help message and exit

--solver <solver>

path to solver

--instance-set-train <instance_set_train>

path to training instance set

--instance-set-test <instance_set_test>

path to testing instance set

--ablation-settings-help

prints a list of setting that can be used for the ablation analysis

--performance-measure

the performance measure, e.g. runtime

--target-cutoff-time
cutoff time per target algorithm run in seconds

--budget-per-run
configuration budget per configurator run in seconds

--number-of-runs
number of configuration runs to execute

--racing
performs abaltion analysis with racing

--settings-file
specify the settings file to use in case you want to use one other than the default

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

1.8.20 run_configured_solver.py

```
usage: run_configured_solver.py [-h] [--settings-file SETTINGS_FILE] [--performance-
↪measure {RUNTIME,QUALITY_ABSOLUTE,QUALITY}] [--parallel] instance_path [instance_path .
↪..]
```

instance_path
Path(s) to instance file(s) (when multiple files are given, it is assumed this is a multi-file instance) or instance directory.

-h, --help
show this help message and exit

--settings-file <settings_file>
settings file to use instead of the default (default: Settings/sparkle_settings.ini)

--performance-measure {RUNTIME,QUALITY_ABSOLUTE,QUALITY}
the performance measure, e.g. runtime (default: RUNTIME)

--parallel
run the solver on multiple instances in parallel

1.8.21 run_solvers.py

```
usage: run_solvers.py [-h] [--recompute] [--parallel] [--performance-measure {RUNTIME,
↪QUALITY_ABSOLUTE,QUALITY}] [--target-cutoff-time TARGET_CUTOFF_TIME] [--also-construct-
↪selector-and-report] [--verifier {NONE,SAT}]
                      [--settings-file SETTINGS_FILE]
```

-h, --help
show this help message and exit

--recompute
recompute the performance of all solvers on all instances

--parallel
run the solver on multiple instances in parallel

--performance-measure
: the performance measure, e.g. runtime

--target-cutoff-time
: cutoff time per target algorithm run in seconds

--also-construct-selector-and-report
: after running the solvers also construct the selector and generate the report

--verifier
: problem specific verifier that should be used to verify solutions found by a target algorithm

--settings-file
: specify the settings file to use in case you want to use one other than the default

1.8.22 run_sparkle_portfolio_selector.py

```
usage: run_sparkle_portfolio_selector.py [-h] [--settings-file SETTINGS_FILE] [--
↪performance-measure {RUNTIME,QUALITY_ABSOLUTE,QUALITY}] instance_path [instance_path ..
↪.]
```

instance_path
: Path to instance or instance directory

-h, --help
: show this help message and exit

--settings-file
: settings file to use instead of the default

--performance-measure
: the performance measure, e.g. runtime

1.8.23 run_status.py

```
usage: run_status.py [-h] [--verbose]
```

-h, --help
: show this help message and exit

--verbose, -v
: output run status in verbose mode

1.8.24 save_record.py

```
usage: save_record.py [-h]
```

-h, --help
: show this help message and exit

1.8.25 sparkle_wait.py

```
usage: sparkle_wait.py [-h]
                       [--job-id JOB_ID | --command {ABOUT,ADD_FEATURE_EXTRACTOR,ADD_
↪INSTANCES,ADD_SOLVER,CLEANUP_CURRENT_SPARKLE_PLATFORM,CLEANUP_TEMPORARY_FILES,COMPUTE_
↪FEATURES,COMPUTE_MARGINAL_CONTRIBUTION,CONFIGURE_SOLVER,CONSTRUCT_SPARKLE_PORTFOLIO_
↪SELECTOR,GENERATE_REPORT,INITIALISE,LOAD_RECORD,REMOVE_FEATURE_EXTRACTOR,REMOVE_
↪INSTANCES,REMOVE_RECORD,REMOVE_SOLVER,RUN_ABLATION,RUN_SOLVERS,RUN_SPARKLE_PORTFOLIO_
↪SELECTOR,RUN_STATUS,SAVE_RECORD,SPARKLE_WAIT,SYSTEM_STATUS,VALIDATE_CONFIGURED_VS_
↪DEFAULT,RUN_CONFIGURED_SOLVER}]
```

-h, --help
show this help message and exit

--job-id <job_id>
job ID to wait for

--command {ABOUT,ADD_FEATURE_EXTRACTOR,ADD_INSTANCES,ADD_SOLVER,
CLEANUP_CURRENT_SPARKLE_PLATFORM,CLEANUP_TEMPORARY_FILES,COMPUTE_FEATURES,
COMPUTE_MARGINAL_CONTRIBUTION,CONFIGURE_SOLVER,CONSTRUCT_SPARKLE_PORTFOLIO_SELECTOR,
GENERATE_REPORT,INITIALISE,LOAD_RECORD,REMOVE_FEATURE_EXTRACTOR,REMOVE_INSTANCES,
REMOVE_RECORD,REMOVE_SOLVER,RUN_ABLATION,RUN_SOLVERS,RUN_SPARKLE_PORTFOLIO_SELECTOR,
RUN_STATUS,SAVE_RECORD,SPARKLE_WAIT,SYSTEM_STATUS,VALIDATE_CONFIGURED_VS_DEFAULT,
RUN_CONFIGURED_SOLVER}
command you want to run. Sparkle will wait for the dependencies of this command to be completed

1.8.26 system_status.py

```
usage: system_status.py [-h] [--verbose]
```

-h, --help
show this help message and exit

--verbose, -v
output system status in verbose mode

1.8.27 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.

```
usage: validate_configured_vs_default.py [-h] --solver SOLVER --instance-set-train␣
↪INSTANCE_SET_TRAIN [--instance-set-test INSTANCE_SET_TEST] [--performance-measure
↪{RUNTIME,QUALITY_ABSOLUTE,QUALITY}]
                                         [--target-cutoff-time TARGET_CUTOFF_TIME] [--
↪settings-file SETTINGS_FILE]
```

-h, --help
show this help message and exit

--solver <solver>
path to solver

--instance-set-train <instance_set_train>
: path to training instance set

--instance-set-test <instance_set_test>
: path to testing instance set

--performance-measure
: the performance measure, e.g. runtime

--target-cutoff-time
: cutoff time per target algorithm run in seconds

--settings-file
: specify the settings file to use instead of the default

1.9 Sparkle settings

Most settings can be controlled through `Settings/sparkle_settings.ini`. Possible settings are summarised per category in Section 1.9.2. 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.

1.9.1 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
```

1.9.2 Names and possible values

[general]

performance_measure

aliases: `smac_run_obj`

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

description: The type of performance measure that sparkle uses. `RUNTIME` focuses on runtime the solver requires, `QUALITY_ABSOLUTE` focuses on the average absolute improvements on the instances and `QUALITY` does the same a the former.

`target_cutoff_time`

aliases: `smac_each_run_cutoff_time`, `cutoff_time_each_performance_computation`

values: integer

description: The time a solver is allowed to run before it is terminated.

`extractor_cutoff_time`

aliases: `cutoff_time_each_feature_computation`

values: integer

description: The time a feature extractor is allowed to run before it is terminated. In case of multiple feature extractors this budget is divided equally.

`penalty_multiplier`

aliases: `penalty_number`

values: integer

description: In case of not solving an instance within the cutoff time the runtime is set to be the `penalty_multiplier * cutoff_time`.

`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

description: The wallclock time one configuration run is allowed to use for finding configurations.

`number_of_runs`

aliases: `num_of_smac_runs`

values: integer

description: The number of separate configurations runs.

[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

description: Use racing when performing the ablation analysis between the default and configured parameters

[slurm]

`number_of_runs_in_parallel`

aliases: `smac_run_obj`

values: integer

description: The number of configuration runs that can run in parallel.

`clis_per_node`

aliases: N/A

values: integer

note: Not really a Slurm option, will likely be moved to another section.

description: The number of parallel processes that can be run on one compute node. In case a node has 32 cores and each solver uses 2 cores, the `cli_per_node` is at most 16.

1.9.3 Priorities

Sparkle has a large flexibility with passing along settings. Settings provided through different channels have different priorities as follows:

- Default –- Default values will be overwritten if a value is given through any other mechanism;
- File –- Settings form the `Settings/sparkle_settings.ini` overwrite default values, but are overwritten by settings given through the command line;
- Command line file -– Settings files provided through the command line, overwrite default values and other settings files.
- Command line –- Settings given through the command line overwrite all other settings, including settings files provided through the command line.

1.9.4 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 1.9.4).

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`

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`

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.

1.10 Required packages

1.10.1 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.

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.

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/

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

1.11 Installation and compilation of examples

1.11.1 Solvers

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 ../../
```

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 ../
```

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 ../../
```

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
```

(continues on next page)

(continued from previous page)

```
make
cp FastCA ../../
```

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 ../
```

CHAPTER

TWO

EXAMPLES

2.1 Use Sparkle for algorithm configuration

These steps can also be found as a Bash script in `Examples/configuration.sh`

2.1.1 Initialise the Sparkle platform

```
Commands/initialise.py
```

2.1.2 Add instances

Add train, and optionally test, instances (in this case in CNF format) in a given directory, without running solvers or feature extractors yet

```
Commands/add_instances.py Examples/Resources/Instances/PTN/
```

```
Commands/add_instances.py Examples/Resources/Instances/PTN2/
```

2.1.3 Add a configurable solver

Add a configurable solver (here for SAT solving) with a wrapper containing the executable name of the solver and a string of command line parameters, without running the solver yet

The solver directory should contain the solver executable, the `sparkle_smac_wrapper.py` wrapper, and a `.pcs` file describing the configurable parameters

```
Commands/add_solver.py --deterministic 0 Examples/Resources/Solvers/PbO-CCSAT-Generic/
```

If needed solvers can also include additional files or scripts in their directory, but keeping additional files to a minimum speeds up copying.

2.1.4 Configure the solver

Perform configuration on the solver to obtain a target configuration

```
Commands/configure_solver.py --solver Solvers/PbO-CCSAT-Generic/ --instance-set-train
Instances/PTN/
```

2.1.5 Validate the configuration

To make sure configuration is completed before running validation you can use the `sparkle_wait` command

```
Commands/sparkle_wait.py
```

Validate the performance of the best found parameter configuration. The test set is optional.

```
Commands/validate_configured_vs_default.py --solver Solvers/PbO-CCSAT-Generic/
--instance-set-train Instances/PTN/ --instance-set-test Instances/PTN2/
```

2.1.6 Generate a report

Wait for validation to be completed

```
Commands/sparkle_wait.py
```

Generate a report detailing the results on the training (and optionally testing) set. This includes the experimental procedure and performance information; this will be located in a `Configuration_Reports/` subdirectory for the solver, training set, and optionally test set like `PbO-CCSAT-Generic_PTN/Sparkle-latex-generator-for-configuration/`

```
Commands/generate_report.py
```

By default the `generate_report` command will create a report for the most recent solver and instance set(s). To generate a report for older solver-instance set combinations, the desired solver can be specified with `--solver Solvers/PbO-CCSAT-Generic/`, the training instance set with `--instance-set-train Instances/PTN/`, and the testing instance set with `--instance-set-test Instances/PTN2/`.

2.1.7 Run ablation to determine parameter importance based on default (from the `.pcs` file) and configured parameters

Run ablation

Run ablation using the training instances and validate the parameter importance with the test set

```
Commands/run_ablation.py --solver Solvers/PbO-CCSAT-Generic/ --instance-set-train
Instances/PTN/ --instance-set-test Instances/PTN2/
```

Generate a report

Wait for ablation to be completed

```
Commands/sparkle_wait.py
```

Generate a report including ablation, and as before the results on the train (and optionally test) set, the experimental procedure and performance information; this will be located in a `Configuration_Reports/` subdirectory for the solver, training set, and optionally test set like `PbO-CCSAT-Generic_PTN/Sparkle-latex-generator-for-configuration/`

```
Commands/generate_report.py
```

The ablation section can be suppressed with `--no-ablation`

2.1.8 Immediate ablation and validation after configuration

By adding `--ablation` and/or `--validate` to the `configure_solver.py` command, ablation and respectively validation will run directly after the configuration is finished.

There is no need to execute `run_ablation.py` and/or `validate_configured_vs_default.py` when these flags are given with the `configure_solver.py` command

Training set only

```
Commands/configure_solver.py --solver Solvers/PbO-CCSAT-Generic/ --instance-set-train
Instances/PTN/ --ablation --validate
```

Training and testing sets

Wait for the previous example to be completed

```
Commands/sparkle_wait.py
```

```
Commands/configure_solver.py --solver Solvers/PbO-CCSAT-Generic/ --instance-set-train
Instances/PTN/ --instance-set-test Instances/PTN2/ --ablation --validate
```

2.2 Use Sparkle for algorithm configuration

These steps can also be found as a Bash script in `Examples/configuration_qualty.sh`

2.2.1 Initialise the Sparkle platform

```
Commands/initialise.py
```

2.2.2 Add instances

Add train, and optionally test, instances (in this case for the VRP) in a given directory, without running solvers or feature extractors yet

```
Commands/add_instances.py Examples/Resources/CVRP/Instances/X-1-10/
```

```
Commands/add_instances.py Examples/Resources/CVRP/Instances/X-11-20/
```

2.2.3 Add a configurable solver

Add a configurable solver (here for vehicle routing) with a wrapper containing the executable name of the solver and a string of command line parameters, without running the solver yet

The solver directory should contain the solver executable, the `sparkle_smac_wrapper.py` wrapper, and a `.pcs` file describing the configurable parameters

```
Commands/add_solver.py --deterministic 0 Examples/Resources/CVRP/Solvers/VRP_SISRs/
```

If needed solvers can also include additional files or scripts in their directory, but keeping additional files to a minimum speeds up copying.

2.2.4 Configure the solver

Perform configuration on the solver to obtain a target configuration. For the VRP we measure the absolute quality performance by setting the `--performance-measure` option, to avoid needing this for every command it can also be set in `Settings/sparkle_settings.ini`.

```
Commands/configure_solver.py --solver Solvers/VRP_SISRs/ --instance-set-train Instances/
X-1-10/ --performance-measure QUALITY_ABSOLUTE
```

2.2.5 Validate the configuration

To make sure configuration is completed before running validation you can use the `sparkle_wait` command

```
Commands/sparkle_wait.py
```

Validate the performance of the best found parameter configuration. The test set is optional. We again set the performance measure to absolute quality.

```
Commands/validate_configured_vs_default.py --solver Solvers/VRP_SISRs/
--instance-set-train Instances/X-1-10/ --instance-set-test Instances/X-11-20/
--performance-measure QUALITY_ABSOLUTE
```

2.2.6 Generate a report

Wait for validation to be completed

```
Commands/sparkle_wait.py
```

Generate a report detailing the results on the training (and optionally testing) set. This includes the experimental procedure and performance information; this will be located in a `Configuration_Reports/` subdirectory for the solver, training set, and optionally test set like `VRP_SISRs_X-1-10_X-11-20/Sparkle-latex-generator-for-configuration/`. We again set the performance measure to absolute quality.

```
Commands/generate_report.py --performance-measure QUALITY_ABSOLUTE
```

By default the `generate_report` command will create a report for the most recent solver and instance set(s). To generate a report for older solver-instance set combinations, the desired solver can be specified with `--solver Solvers/VRP_SISRs/`, the training instance set with `--instance-set-train Instances/X-1-10/`, and the testing instance set with `--instance-set-test Instances/X-11-20/`.

2.3 Use Sparkle for algorithm selection

These steps can also be found as a Bash script in `Examples/selection.sh`

2.3.1 Initialise the Sparkle platform

```
Commands/initialise.py
```

2.3.2 Add instances

Add instance files (in this case in CNF format) in a given directory, without running solvers or feature extractors yet

```
Commands/add_instances.py Examples/Resources/Instances/PTN/
```

2.3.3 Add solvers

Add solvers (here for SAT solving) with a wrapper containing the executable name of the solver and a string of command line parameters, without running the solvers yet

Each solver directory should contain the solver executable and a wrapper

```
Commands/add_solver.py --deterministic 0 Examples/Resources/Solvers/CSCCSat/
```

```
Commands/add_solver.py --deterministic 0 Examples/Resources/Solvers/PbO-CCSAT-Generic/
```

```
Commands/add_solver.py --deterministic 0 Examples/Resources/Solvers/MiniSAT/
```

2.3.4 Add feature extractor

Similarly, add a feature extractor, without immediately running it on the instances

```
Commands/add_feature_extractor.py Examples/Resources/Extractors/SAT-features-competition2012_revised_wi
```

2.3.5 Compute features

Compute features for all the instances; add the `--parallel` option to run in parallel

```
Commands/compute_features.py
```

2.3.6 Run the solvers

Run the solvers on all instances; add the `--parallel` option to run in parallel

```
Commands/run_solvers.py
```

2.3.7 Construct a portfolio selector

To make sure feature computation and solver performance computation are done before constructing the portfolio use the `sparkle_wait` command

```
Commands/sparkle_wait.py
```

Construct a portfolio selector, using the previously computed features and the results of running the solvers

```
Commands/construct_sparkle_portfolio_selector.py
```

2.3.8 Generate a report

Generate an experimental report detailing the experimental procedure and performance information; this will be located at `Components/Sparkle-latex-generator/Sparkle_Report.pdf`

```
Commands/generate_report.py
```

2.4 Run the portfolio selector (e.g. on a test set)

2.4.1 Run on a single instance

Run the portfolio selector on a *single* testing instance; the result will be printed to the command line

```
Commands/run_sparkle_portfolio_selector.py Examples/Resources/Instances/PTN2/plain7824.
cnf
```

2.4.2 Run on an instance set

Run the portfolio selector on a testing instance *set*

```
Commands/run_sparkle_portfolio_selector.py Examples/Resources/Instances/PTN2/
```

2.4.3 Generate a report including results on the test set

Wait for the portfolio selector to be done running on the testing instance set

```
Commands/sparkle_wait.py
```

Generate an experimental report that includes the results on the test set, and as before the experimental procedure and performance information; this will be located at `Components/Sparkle-latex-generator/Sparkle_Report_For_Test.pdf`

```
Commands/generate_report.py
```

By default the `generate_report` command will create a report for the most recent instance set. To generate a report for an older instance set, the desired instance set can be specified with: `--test-case-directory Test_Cases/PTN2/`

2.5 Use Sparkle for algorithm selection with multi-file instances

2.5.1 Initialise the Sparkle platform

```
Commands/initialise.py
```

2.5.2 Add instances

Add instance files in a given directory, without running solvers or feature extractors yet. In addition to the instance files, the directory should contain a file `sparkle_instance_list.txt` where each line contains a space separated list of files that together form an instance.

```
Commands/add_instances.py Examples/Resources/CCAG/Instances/CCAG/
```

2.5.3 Add solvers

Add solvers (here for the constrained covering array generation (CCAG) problem) with a wrapper containing the executable name of the solver and a string of command line parameters, without running the solvers yet

Each solver directory should contain the solver executable and a wrapper

```
Commands/add_solver.py --deterministic 0 Examples/Resources/CCAG/Solvers/TCA/
```

```
Commands/add_solver.py --deterministic 0 Examples/Resources/CCAG/Solvers/FastCA/
```

2.5.4 Add feature extractor

Similarly, add a feature extractor, without immediately running it on the instances

```
Commands/add_feature_extractor.py Examples/Resources/CCAG/Extractors/
CCAG-features_sparkle/
```

2.5.5 Compute features

Compute features for all the instances; add the `--parallel` option to run in parallel

```
Commands/compute_features.py
```

2.5.6 Run the solvers

Run the solvers on all instances; add the `--parallel` option to run in parallel. For the CCAG (Constrained Covering Array Generation) problem we measure the absolute quality performance by setting the `--performance-measure` option, to avoid needing this for every command it can also be set in `Settings/sparkle_settings.ini`.

```
Commands/run_solvers.py --performance-measure QUALITY_ABSOLUTE
```

2.5.7 Construct a portfolio selector

To make sure feature computation and solver performance computation are done before constructing the portfolio use the `sparkle_wait` command

```
Commands/sparkle_wait.py
```

Construct a portfolio selector, using the previously computed features and the results of running the solvers. We again set the performance measure to absolute quality.

```
Commands/construct_sparkle_portfolio_selector.py --performance-measure QUALITY_ABSOLUTE
```

2.5.8 [Coming soon] Generate a report

This is not yet implemented for quality performance

Generate an experimental report detailing the experimental procedure and performance information; this will be located at `Components/Sparkle-latex-generator/Sparkle_Report.pdf`. We again set the performance measure to absolute quality.

```
Commands/generate_report.py --performance-measure QUALITY_ABSOLUTE
```

2.6 [Coming soon] Run the portfolio selector (e.g. on the test set)

This is not yet implemented for quality performance

This is not yet implemented for multi-file instances

2.6.1 Run on a single instance

Run the portfolio selector on a *single* testing instance; the result will be printed to the command line. We again set the performance measure to absolute quality.

```
Commands/run_sparkle_portfolio_selector.py Examples/Resources/CCAG/Instances/
CCAG2/Banking2.model Examples/Resources/CCAG/Instances/CCAG2/Banking2.constraints
--performance-measure QUALITY_ABSOLUTE
```

2.6.2 Run on an instance set

Run the portfolio selector on a testing instance *set*. We again set the performance measure to absolute quality.

```
Commands/run_sparkle_portfolio_selector.py Examples/Resources/CCAG/Instances/CCAG2/
--performance-measure QUALITY_ABSOLUTE
```

2.6.3 Generate a report including results on the test set

Wait for the portfolio selector to be done running on the testing instance set

```
Commands/sparkle_wait.py
```

Generate an experimental report that includes the results on the test set, and as before the experimental procedure and performance information; this will be located at `Components/Sparkle-latex-generator/Sparkle_Report_For_Test.pdf`. We again set the performance measure to absolute quality.

```
Commands/generate_report.py --performance-measure QUALITY_ABSOLUTE
```

By default the `generate_report` command will create a report for the most recent instance set. To generate a report for an older instance set, the desired instance set can be specified with: `--test-case-directory Test_Cases/CCAG2/`

INDEX

Symbols

A

C

E

G

I

L

R

S

V