

Accessible meta-algorithms with Sparkle

Koen van der Blom^{1,2}, Holger Hoos^{3,2,4}, Chuan Luo⁵, Jeroen Rook⁶

¹Sorbonne Université, ²Leiden University, ³RWTH Aachen University,
⁴University of British Columbia, ⁵Beihang University, ⁶University of Twente

Meta-algorithms

- Algorithm configuration
- Algorithm selection
- Parallel algorithm portfolios
- Get the best performance out of algorithms

Adoption of meta-algorithms

- Adoption is limited, even in ML research [Bouthillier & Varoquaux, 2020]
- Meta-algorithms are complex and difficult for non-experts
- Substantial pitfalls, e.g., in AAC [Eggenberger et al., 2019]
- Errors are costly, e.g., re-running AAC is computationally expensive

Expanding the reach of meta-algorithms

- COSEAL community
- ...
- Laypeople from any other field
- Sparkle platform → Reach wider audiences, one step at a time

Goals of Sparkle

- Simplify the use of meta-algorithms
 - Increase the adoption of meta-algorithms
 - Prevent common pitfalls and often-made errors
 - Ensure proper experimentation pipelines
-
- Improve our ability to assess, access and improve the SOTA in computational problem solving

Command line and scripting interface

```
1: Commands/initialise.py
2: Commands/add_instances.py path/to/PTN/
3: Commands/add_solver.py --deterministic 0 path/to/PbO-CSCCSAT/
4: Commands/add_solver.py --deterministic 0 path/to/MiniSAT/
5: Commands/add_feature_extractor.py path/to/Extractor/
6: Commands/compute_features.py
7: Commands/run_solvers.py
8: Commands/construct_sparkle_portfolio_selector.py
9: Commands/generate_report.py
```

Reports

Configuration Report for the Solver PbO-CCSAT-Generic on the Training Instance Set PTN in Sparkle

Solver, instance set(s)

Automatic Configuration by Sparkle (version: 1.0.0)
8th March 2022

1 Introduction

Sparkle [3] is a multi-agent problem-solving platform (PbO) [2], and would provide a number of effective algorithm optimisation techniques (such as automated algorithm configuration, portfolio-based algorithm selection, etc) to accelerate the existing solvers. This experimental report is automatically generated by Sparkle. This report presents experimental results on the scenario of configuring the solver PbO-CCSAT-Generic on the training instance set PTN.

2 Information about the Instance Set(s)

- Training set: PTN, consisting of 12 instances

3 Information about the Configurator, protocols, budgets

The configurator used in Sparkle is SMAC (Sequential Model-based Algorithm Configuration) [4], and the version of SMAC used in Sparkle is 2.10.03.

During the configuration process, Sparkle performs 10 independent SMAC runs for configuring the solver PbO-CCSAT-Generic on the training instance set PTN; the configuration objective is RUNTIME; the whole configuration time budget is 3600 seconds; the cutoff time for each run is 120 seconds.

Each independent SMAC run would obtain 10 optimised configurations. As a result, Sparkle would obtain 100 optimised configurations. Each of these was then evaluated on the entire training set, with one solver run per instance and a cutoff time of 120 seconds, and the configuration with the lowest PAR10 value was selected as the result of the configuration process.

4 Information about the Final configuration

After the configuration process is finished, Sparkle would output the optimised configuration. The details of the optimised configuration are described as below.

```
-gamma_hscore2 '351' -init_solution '1' -p_swt '0.20423712003341465' -perform_aspiration '1' -perform_double -perform_first_div '0' -prob_new -perform_double -perform_first_div '0' -prob_new
```

1

Performance measures

5 Comparison between Configured and Default Version on the Training Instance Set

In order to investigate the performance on the training instance set, Sparkle would run the configured version of PbO-CCSAT-Generic and the default version of PbO-CCSAT-Generic on the training instance set. During this phase, each version was performed one run per instance with a cutoff time of 120 seconds. The results are presented in Table 1.

- PbO-CCSAT-Generic (configured), PAR10: 621.2856854001681
- PbO-CCSAT-Generic (default), PAR10: 621.2856854001681

The empirical comparison between the PbO-CCSAT-Generic (configured) and PbO-CCSAT-Generic (default) on the training set of PTN is presented in Figure 1.

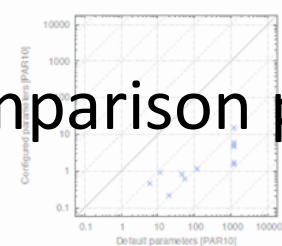


Figure 1: Empirical comparison between the PbO-CCSAT-Generic (configured) and PbO-CCSAT-Generic (default) on the training set of PTN.

Table 1 shows on how many instances the PbO-CCSAT-Generic (configured) and PbO-CCSAT-Generic (default) timed out (different from the PAR10 value) on the training set of PTN, as well as on how many instances both timed out.

configured	default	overlap
0	6	0

Table 1: Number of time-outs for PbO-CCSAT-Generic (configured), PbO-CCSAT-Generic (default), and for how many instances both timed out on the training set of PTN.

6 Parameter importance via Ablation

Ablation analysis is performed from the PbO-CCSAT-Generic (default) PbO-CCSAT-Generic (configured) on the training set of PTN. The analysis is performed by flipping parameters one by one from the default configuration to the configured configuration. The flipped parameter is added to the configuration and the next round starts with the remaining parameters. This repeats until all parameters are flipped, which is the best found configuration. The analysis resulted in the ablation path presented in Table 2.

2

parameters that differ in the two configurations will form the ablation path. Starting from the default configuration, the path is computed by performing a sequence of rounds. In a round, each available parameter is flipped in the configuration and is validated on its performance. The flipped parameter with the best performance in that round, is added to the configuration and the next round starts with the remaining parameters. This repeats until all parameters are flipped, which is the best found configuration. The analysis resulted in the ablation path presented in Table 2.

Table 2: Ablation path from PbO-CCSAT-Generic (default) to PbO-CCSAT-Generic (configured) where parameters with higher importance are ranked higher.

Round	Flipped parameter	Source value	Target value	Validation result
0	-source-	N/A	N/A	610.48433
1	sd_var_div	5	2	
	sd_var_break_tk_greedy	2	4	
	gamma_hscore2	1000	351	116.32313
2	perform_pac	0	1	
	prob_pac	5.800000000000000E-4	0.005730374136488115	18.91441
3	p_swt	0.3	0.20423712003341465	122.56680
4	q_swt	0.0	0.68072071796774418	17.40350
5	threshold_swt	300	32	103.07659
6	perform_double_cc	1	0	3.85326
7	-target-	N/A	N/A	3.85717

References

- [1] Chris Fawcett and Holger H. Hoos. Analysing differences between algorithm configurations through ablation. *J. Heuristics*, 23(4):453–504, 2014.
- [2] Holger H. Hoos. *Multi-Agent Problem Solving*. CM, 55(2):70–80, 2012.
- [3] Holger H. Hoos. Sparkle: A pbo-based multi-agent problem-solving platform. Technical report, Department of Computer Science, University of British Columbia, 2015.
- [4] Frank Hutter, Holger H. Hoos, and Kevin Leyton-Brown. Sequential model-based optimization for general algorithm configuration. In *Proceedings of the 5th International Conference on Learning and Intelligent Optimization (LION 5)*, pages 507–523, 2011.

3

Going forward

- Slurm clusters only → Local execution in progress
- SMAC only → Extending to other configurators
- Simplify, improve, extend ... in many directions

Sparkle makes meta-algorithms accessible
for improving the state of the art
in solving challenging problems in AI.

Try Sparkle yourself!

`bitbucket.org/sparkle-ai/sparkle`

Opening for research engineer @ RWTH Aachen