AClib: A Benchmark Library for Algorithm Configuration

Frank Hutter[1(✉)], Manuel López-Ibáñez[2], Chris Fawcett[3], Marius Lindauer[4], Holger H. Hoos[3], Kevin Leyton-Brown[3], and Thomas Stützle[2]

[1] Department of Computer Science, Freiburg University, Freiburg, Germany
fh@informatik.uni-freiburg.de
[2] IRIDIA, Université Libre de Bruxelles, Brussel, Belgium
{manuel.lopez-ibanez,stuetzle}@ulb.ac.be
[3] Department of Computer Science,
University of British Columbia, Vancouver, Canada
{fawcettc,hoos,kevinlb}@cs.ubc.ca
[4] Institute of Computer Science, Potsdam University, Potsdam, Germany
manju@cs.uni-potsdam.de

1 Introduction

Modern solvers for hard computational problems often expose parameters that permit customization for high performance on specific instance types. Since it is tedious and time-consuming to manually optimize such highly parameterized algorithms, recent work in the AI literature has developed automated approaches for this *algorithm configuration problem* [1,3,10,11,13,16]. This line of work has already led to improvements in the state of the art for solving a wide range of problems, including propositional satisfiability (SAT) [2,7,12,20], mixed integer programming (MIP) [9], timetabling [4], AI planning [6,21], answer set programming (ASP) [18], bi-objective TSP [14], and bi-objective flowshop scheduling problems [5].

As the field of algorithm configuration matures and the number of available configuration procedures grows, so does a need for standardized problem definitions, interfaces, and benchmarks. Such a benchmark library would encourage reproducible research, facilitate the empirical evaluation of new and existing configuration procedures, reduce obstacles faced by researchers new to the community, and allow an objective scientific evaluation of strengths and weaknesses of different methods. We therefore introduce AClib (www.aclib.net), a library of algorithm configuration benchmarks.

2 Design Criteria and Summary of Benchmarks

Instances of the algorithm configuration problem (called *configuration scenarios*) comprise various components: a parameterized algorithm A (*target algorithm*) to be configured, a distribution D of problem instances $\mathcal{I}$ (*target instances*) and a performance metric $m(\boldsymbol{\theta}, \pi)$ capturing A's performance with parameter settings

P.M. Pardalos et al. (Eds.): LION 2014, LNCS 8426, pp. 36–40, 2014.
DOI: 10.1007/978-3-319-09584-4_4

$\boldsymbol{\theta} \in \boldsymbol{\Theta}$ on instances $\pi \in \mathcal{I}$. The objective is then to find a configuration $\boldsymbol{\theta}^*$ that minimizes a statistic (often the mean) of m across instances sampled from D. In practice, we typically have a finite set of instances from distribution D, which is partitioned into disjoint training and test sets in order to obtain an unbiased estimate of generalization performance for the configuration selected.

Table 1 summarizes the benchmarks we selected for AClib 1.0. One of our design criteria was to achieve diversity across the following dimensions:

Table 1. Overview of algorithm configuration benchmarks in AClib.

Problem	Solvers	#Scenarios		#Parameters	#Instances	Citation
		Runtime	Quality			
SAT	12 different solvers	126	0	2–270	500–2064	[7,8,10]
MIP	CPLEX	4	4	76	100–2000	[9]
ASP	Clasp	3	0	85	480–3133	[18]
AI Planning	LPG & Fast Downward	20	0	45–66	60–559	[6,21]
Time-tabling	CTT	1	1	7–18	24	[4]
TSP	ACOTSP, ACOTSP-VAR	0	2	11–28	50–100	[15]
bTSP	MOACO	0	1	16	60	[14]
Machine Learning	AutoWEKA	0	21	768	10 (CV folds)	[19]

- **Target problems:** decision and optimization problems, as well as machine learning;
- **Algorithm types:** stochastic local search (SLS), tree search, machine learning;
- **Number of parameters:** from 2 (in some SLS solvers) to 768 (in Auto-WEKA [19]);
- **Parameter types:** from purely continuous (several SLS algorithms) to mixed discrete/continuous with occasional conditional parameters (most algorithms) to massively conditional spaces (SATenstein [12] and Auto-WEKA [19]);
- **Different objectives:** runtime required to reach an optimal solution (most scenarios) and solution quality achieved in a given time (timetabling, TSP, MIP, and machine learning);
- **Target instance homogeneity:** from quite homogeneous instance distributions (most scenarios) to distributions that are heterogeneous at least in instance hardness;
- **Instance features:** from scenarios for which no characteristic features have been defined so far (timetabling) to those where 138 features exist for every problem instance included (most SAT scenarios).

Another design criterion was to select configuration scenarios that will enable the assessment of different components of algorithm configuration procedures, such as their search procedures (deciding which configuration will be selected next), their intensification/racing components (deciding how many runs to perform on which instances), and, in case of runtime optimization, their capping

procedures (deciding after which time a run is terminated as unsuccessful). We achieved this by including scenarios where:

- the racing component is more important than the search component (because most configurations are good, but instances are fairly heterogeneous; e.g., Spear-SWV);
- the search component is more important than the racing component (because few configurations are good, but instances are homogeneous; e.g., the CPLEX scenarios);
- capping is unimportant: all scenarios optimizing solution quality, and scenarios with maximum runtimes that are already low enough for capping to not yield large gains;
- capping is very important: large captimes and configurations that finish in orders of magnitude below the captime (*e.g.*, CPLEX scenarios, ASP-Riposte, and Spear-SWV).

On a more technical level, since the evaluation of target algorithm A's performance with configuration $\boldsymbol{\theta}$ requires us to execute A with $\boldsymbol{\theta}$ on several target instances, it is important to ensure that time and memory limits are respected and that different configuration procedures C_1 and C_2 call A identically. If that is not guaranteed, spurious performance differences may be measured — *e.g.*, we once measured a 20 % performance difference just because C_1 used relative paths and C_2 absolute paths to call a particular target algorithm A (the target algorithm saved its callstring in the heap space before the instance data, such that the callstring length affected memory locality and thus the number of cache misses). To avoid such problems, for each configuration scenario we defined a wrapper that deterministically maps parameter settings to target algorithm command line call strings, executes those call strings and parses their results. This wrapper also kills target algorithm runs if these do not respect their runtime or memory limits (and in this case returns the worst possible performance); this mechanism is important to avoid "hung" runs of a configuration procedure that are waiting for a particular target algorithm call to finish, as well as cases where excessive memory consumption leads to swapping or to jobs being killed (e.g., when executing on a cluster). For this purpose, we modified the runsolver tool [17] used to control algorithm runs in the international SAT competition.

Regarding usage and maintenance, we designed AClib to be lightweight and extensible. Because instance files for some scenarios are extremely large, AClib allows users to download subsets of scenarios. This can be done (automatically) on the basis of problems (e.g., all TSP scenarios), of algorithms (e.g., all scenarios that can be used with Clasp), or by requesting individual scenarios. All downloadable pieces are hashed to guarantee the integrity of downloads. All current sequential algorithm configuration procedures that support discrete variables and multiple instances (ParamILS [11], SMAC [10], and irace [13]) can be run on the scenarios via a common interface. New configuration scenarios and configuration procedures can be contributed through a streamlined process.

3 Future Work

In future work, we would like to grow AClib to include other configuration scenarios from the literature that are complementary to the current set, including polynomial-time algorithms. We plan to use AClib to assess strengths and weaknesses of existing configuration procedures and to use it as the basis for the first competition of such procedures. We also plan to expand the instance feature portion of AClib and to use AClib as a source for generating benchmarks for algorithm selection.

Acknowledgments. We gratefully acknowledge all authors of algorithms and instance distributions for making their work available (they are cited on the webpage, acknowledged in README files, and will be cited in a future longer version of this paper). We thank Kevin Tierney and Yuri Malitsky for modifying GGA [1] to support AClib's format; Lin Xu for generating several instance distributions and writing most feature extraction code for SAT and TSP; Adrian Balint and Sam Bayless for contributing SAT benchmark distributions; Mauro Vallati for exposing many new parameters in LPG; the developers of Fast Downward for helping define its configuration space; and Steve Ramage for helping diagnose and fix problems with several wrappers and runsolver. M. Lindauer acknowledges support by DFG project SCHA 550/8-3, and M. López-Ibáñez acknowledges support from a "Crédit Bref Séjour à l'étranger" from the Belgian F.R.S.-FNRS.

References

1. Ansótegui, C., Sellmann, M., Tierney, K.: A gender-based genetic algorithm for the automatic configuration of algorithms. In: Gent, I.P. (ed.) CP 2009. LNCS, vol. 5732, pp. 142–157. Springer, Heidelberg (2009)
2. Balint, A., Fröhlich, A., Tompkins, D., Hoos, H.: Sparrow 2011. In: Booklet of SAT-2011 Competition (2011)
3. Birattari, M., Stützle, T., Paquete, L., Varrentrapp, K.: A racing algorithm for configuring metaheuristics. In: Proceedings of GECCO-02, pp. 11–18 (2002)
4. Chiarandini, M., Fawcett, C., Hoos, H.: A modular multiphase heuristic solver for post enrolment course timetabling. In: Proceedings of PATAT-08 (2008)
5. Dubois-Lacoste, J., López-Ibáñez, M., Stützle, T.: A hybrid TP+PLS algorithm for bi-objective flow-shop scheduling problems. Comput. Oper. Res. **38**(8), 1219–1236 (2011)
6. Fawcett, C., Helmert, M., Hoos, H.H., Karpas, E., Röger, G., Seipp, J.: FD-autotune: domain-specific configuration using fast-downward. In: Proceedings of ICAPS-PAL11 (2011)
7. Hutter, F., Babić, D., Hoos, H.H., Hu, A.J.: Boosting verification by automatic tuning of decision procedures. In: Proceedings of FMCAD-07, pp. 27–34 (2007)
8. Hutter, F., Balint, A., Bayless, S., Hoos, H., Leyton-Brown, K.: Configurable SAT solver challenge (CSSC) (2013), riptsizehttp://www.cs.ubc.ca/labs/beta/Projects/CSSC2013/
9. Hutter, F., Hoos, H.H., Leyton-Brown, K.: Automated configuration of mixed integer programming solvers. In: Lodi, A., Milano, M., Toth, P. (eds.) CPAIOR 2010. LNCS, vol. 6140, pp. 186–202. Springer, Heidelberg (2010)

10. Hutter, F., Hoos, H.H., Leyton-Brown, K.: Sequential model-based optimization for general algorithm configuration. In: Coello, C.A.C. (ed.) LION 2011. LNCS, vol. 6683, pp. 507–523. Springer, Heidelberg (2011)
11. Hutter, F., Hoos, H.H., Leyton-Brown, K., Stützle, T.: ParamILS: an automatic algorithm configuration framework. JAIR **36**, 267–306 (2009)
12. KhudaBukhsh, A., Xu, L., Hoos, H.H., Leyton-Brown, K.: SATenstein: automatically building local search SAT solvers from components. In: Proceedings of IJCAI-09, pp. 517–524 (2009)
13. López-Ibáñez, M., Dubois-Lacoste, J., Stützle, T., Birattari, M.: The irace package, iterated race for automatic algorithm configuration. Technical report TR/IRIDIA/2011-004, IRIDIA, Université Libre de Bruxelles, Belgium (2011)
14. López-Ibáñez, M., Stützle, T.: The automatic design of multi-objective ant colony optimization algorithms. IEEE Trans. Evol. Comput. **16**(6), 861–875 (2012)
15. López-Ibáñez, M., Stützle, T.: Automatically improving the anytime behaviour of optimisation algorithms. Eur. J. Oper. Res. (2013)
16. Nannen, V., Eiben, A.: Relevance estimation and value calibration of evolutionary algorithm parameters. In: Proc. of IJCAI-07, pp. 975–980 (2007)
17. Roussel, O.: Controlling a solver execution with the runsolver tool. JSAT **7**(4), 139–144 (2011)
18. Silverthorn, B., Lierler, Y., Schneider, M.: Surviving solver sensitivity: an ASP practitioner's guide. In: Proceedings of ICLP-LIPICS-12, pp. 164–175 (2012)
19. Thornton, C., Hutter, F., Hoos, H.H., Leyton-Brown, K.: Auto-WEKA: combined selection and hyperparameter optimization of classification algorithms. In: Proceedings of KDD-2013, pp. 847–855 (2013)
20. Tompkins, D.A.D., Balint, A., Hoos, H.H.: Captain Jack: new variable selection heuristics in local search for SAT. In: Sakallah, K.A., Simon, L. (eds.) SAT 2011. LNCS, vol. 6695, pp. 302–316. Springer, Heidelberg (2011)
21. Vallati, M., Fawcett, C., Gerevini, A.E., Hoos, H.H., Saetti, A.: Generating fast domain-optimized planners by automatically configuring a generic parameterised planner. In: Proceedings of ICAPS-PAL11 (2011)