On the Potential of Automatic Algorithm Configuration

Frank Hutter
University of British Columbia, 2366 Main Mall
Vancouver, BC, V6T1Z4, Canada
hutter@cs.ubc.ca

Abstract

Design and implementation of efficient and robust algorithms are core topics of computer science and operations research, and the determination of appropriate values for free algorithm parameters is a challenging and tedious task in the design of effective algorithms for hard problems. Such parameters include categorical choices (e.g., neighborhood structure in local search or variable/value ordering heuristics in tree search), as well as numerical parameters (e.g., noise or restart timing). In practice, tuning of these parameters is largely carried out manually by applying rules of thumb and crude heuristics, while more principled approaches are only rarely used. In this paper, we study some tuning scenarios in more detail and demonstrate the large potential of even very simple automatic algorithm configuration approaches.

1 Introduction

The problem of setting an algorithm's free parameters for maximal performance on a class of problem instances is ubiquitous in the design and empirical analysis of algorithms. Examples of parameterised algorithms can be found in tree search [6] and local search [9]; commercial solvers, such as ILOG CPLEX[1], also offer a plethora of parameter settings. Considerable effort is often required to find a default parameter configuration that yields high and robust performance across all or at least most instances within a given set or distribution [3, 1].

The use of automated tools for finding performance-optimising parameter settings has the potential to liberate algorithm designers from the tedious task of manually searching the parameter space. Notice that the task of constructing an algorithm by combining various building blocks can be seen as a special case of algorithm configuration: Consider, for example, two tree search algorithms for SAT

[1] http://www.ilog.com/products/cplex/

that only differ in their preprocessing and variable ordering heuristics – in fact, these can be seen as a single algorithm with two nominal parameters.

Algorithm configuration is commonly (either implicitly or explicitly) treated as an optimisation problem, where the objective function captures performance on a fixed set of benchmark instances. Depending on the number and type of parameters, the methods used to solve this optimisation problem include exhaustive enumeration, beam search [15], experimental design [5, 2], genetic programming [16], the application of racing algorithms [4, 3], and combinations of fractional experimental design and local search [1].

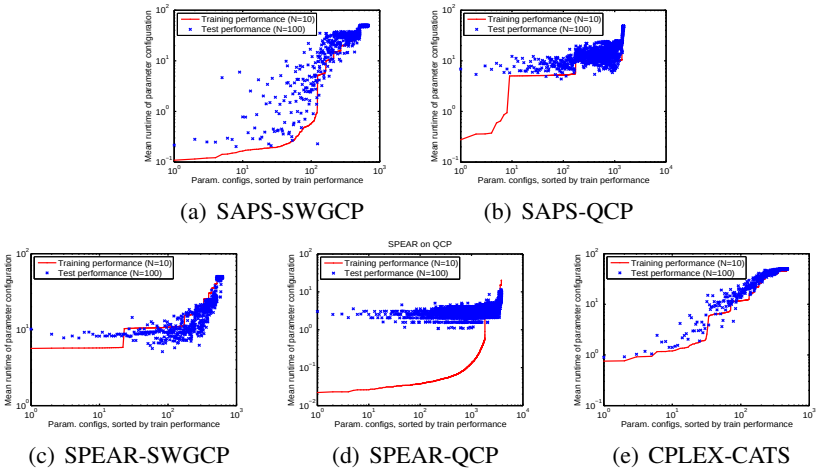
Recently, we have introduced an iterated local search approach for algorithm configuration [13]. This approach has subsequently lead to enormous speed-ups of tree search algorithms for SAT for solving SAT-encoded software verification (speedups of a factor of 500) and bounded model-checking instances (speedups of a factor of 4.5) [10].

2 Tuning scenarios

In this paper, we study tuning scenarios including tree search and local search for propositional satisfiability (SAT) and mixed integer programming. In particular, we study the tree search algorithm SPEAR [10] with 26 mixed discrete/continuous parameters, the local search algorithm SAPS [12] with four continuous parameters, and the commercial software CPLEX for mixed integer programming (MIP)[2] with 80 mixed discrete/continuous parameters. All continous parameters are discretized to seemingly meaningful values spread around the algorithm defaults. As SAT domains, we employ a set of SAT-encoded quasi-group completion (QCP) problems [8] and a set of SAT-encoded graph colouring problems based on small-world graphs (SWGCP) [7]. For MIP, we employ a set of combinatorial auction (CATS) instances from the combinatorial auctions test set [14].

In order to get an idea about the potential of improvement for these tuning scenarios, and about the potential for overtuning to be expected, we sampled a number of parameter configurations uniformly at random, evaluating them on a small training set with $N = 10$ instances, and iterated this process up to a total CPU time usage of five hours. In Figure 1, we plot both performance on the small training set and on an independent test set with $M = 100$ instances; configurations are ordered with respect to training quality. Note that the cutoff time for each domain is five seconds, unsuccesful runs are counted as taking ten times this time, and we plot average performance over the training/test instances; thus, a performance of 50 is the absolute worst a parameter configuration can achieve, and zero is the best. We note that the differences between parameter configrations vary between tuning scenarios, as does the correlation between training and test performance.

[2] http://www.ilog.com/products/cplex/



(a) SAPS-SWGCP (b) SAPS-QCP

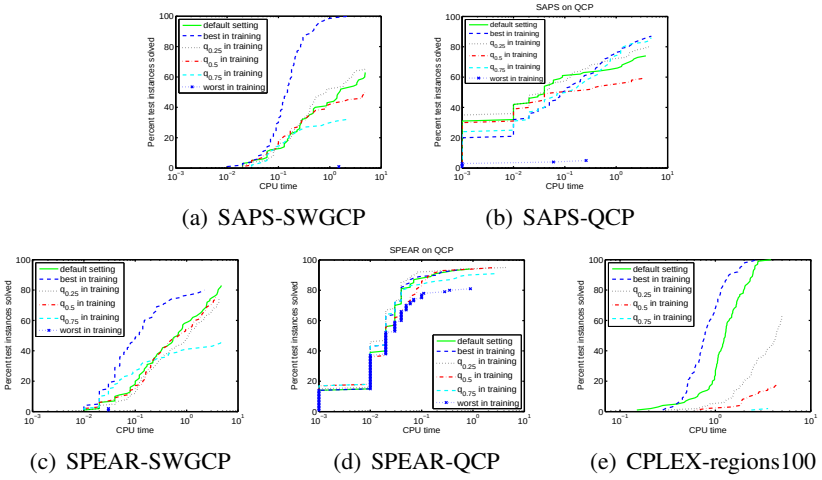
(c) SPEAR-SWGCP (d) SPEAR-QCP (e) CPLEX-CATS

Figure 1: Performance of randomly sampled parameter configurations on a small training set of $N = 10$ instances and a test set of 100 instances. For details see text.

Figure 2 takes a closer look at test performance of a number of selected parameter configurations, namely the default parameter configuration of the algorithms, and five of the randomly sampled configurations: the best and worst in terms of training performance, as well as the 25%, 50%, and 75% quantiles. For each of these parameter configurations, we plot the cumulative distribution of the probability of solving the instances in the test set. We note that in each single scenario, the best parameter configuration based on the small training set of ten instances already performs better than the default parameter setting. Of course, we do not anticipate this to generalize to arbitrary tuning scenarios, but it at least speaks for the potential of automatic tuning.

3 Experiments with iterated local search

In this section, we study the effectiveness of our iterated local search from [13] in the above tuning scenarios. We compare test performances of pure random sampling based on a training set of 100 instances, BasicILS based on the same training set, and FocusedILS (which uses a different number of instances to evaluate each parameter configuration). Compared to [13], we implemented one important improvement for all approaches, namely a pruning technique that stops evaluations of parameter configurations when they are already proovably worse than a previously seen parameter configurations. This technique improves random sampling the most, followed by BasicILS, and only improves FocusedILS marginally. Without this pruning technique BasicILS and FocusedILS outperformed random



(a) SAPS-SWGCP (b) SAPS-QCP

(c) SPEAR-SWGCP (d) SPEAR-QCP (e) CPLEX-regions100

Figure 2: Test performance for a number of selected parameter configurations; for details, see text.

sampling in our experiments for [13] and FocusedILS outperformed BasicILS. In Table 1 we see that the picture becomes less clear when pruning is used: while FocusedILS performs statistically significantly better than the other approaches on two domains, the other two approaches reach similar or better (albeit not statistically significantly better) performance in the remaining three scenarios.

Based on our above analysis of correlations between training and test set, we expected overtuning effects to be strongest for the tuning scenarios involving the QCP domain. This expectation is confirmed by the results in Table 1. Finally, note that the automatically found parameter configurations always clearly outperform the default configuration, much more so than the random parameter configurations, sampled without pruning.

4 Conclusions

Automatic algorithm configuration can greatly improve performance for all tuning scenarios we studied here, and even a simple random sampling of parameter configurations shows very good performance when combined with a simple pruning technique that stops algorithm evaluations once they are proovably worse (on the training set) than the incumbent solution.

In future work, we plan to study model-based approaches in order to speed up the search for good parameter configurations. We also plan to integrate computationally cheap instance features into this model and use it to perform per-instance algorithm configuration, where we automatically choose an appropriate parameter configuration for each given instance [11].

Scenario	Default	Random(100)	BasicILS(100)	FocusedILS
SAPS-SWGCP	45.41	0.21 ± 0.03	0.21 ± 0.03	0.35 ± 0.05
		0.32 ± 0.05	0.32 ± 0.06	0.32 ± 0.05
SPEAR-SWGCP	9.74	6.71 ± 1.2	6.65 ± 1.48	8.26 ± 0.73
		7.97 ± 1.14	8.05 ± 0.9	8.3 ± 1.06
SAPS-QCP	15.80	3.4 ± 1.53	$\mathbf{2.78 \pm 1.28}$	3.95 ± 0.27
		5.92 ± 0.44	5.5 ± 0.53	$\mathbf{5.21 \pm 0.39}$
SPEAR-QCP	2.65	0.45 ± 0.51	0.36 ± 0.41	1.08 ± 0.18
		1.2 ± 0.18	1.39 ± 0.33	1.29 ± 0.2
CPLEX-regions100	1.61	0.7 ± 0.12	$\mathbf{0.39 \pm 0.12}$	0.35 ± 0.04
		0.71 ± 0.12	0.4 ± 0.11	$\mathbf{0.35 \pm 0.04}$

Table 1: Performance for our tuning scenarios; the top row for each scenario gives training performance, the bottom row test performance, mean$\pm$stddev over 25 executions of the tuning approach. The training performance of Random(100) and BasicILS(100) is directly comparable since they use the same training set: bold face indicates statistically better performance for BasicILS, but due to limited representativeness of the training set this does not transfer to statistically better performance on the test set. For FocusedILS, bold face indicates statistically better performance than the other approaches on the test set; the cases where Random performs best are not statistically significant.

References

[1] B. Adenso-Diaz and M. Laguna. Fine-tuning of algorithms using fractional experimental design and local search. *Operations Research*, 54(1):99–114, Jan–Feb 2006.

[2] T. Bartz-Beielstein. *Experimental Research in Evolutionary Computation*. Springer Verlag, 2006.

[3] M. Birattari. *The Problem of Tuning Metaheuristics as Seen from a Machine Learning Perspective*. PhD thesis, Université Libre de Bruxelles, Brussels, Belgium, 2004.

[4] M. Birattari, T. Stützle, L. Paquete, and K. Varrentrapp. A racing algorithm for configuring metaheuristics. In *Proc. of GECCO-02*, pages 11–18, 2002.

[5] S. P. Coy, B. L. Golden, G. C. Runger, and E. A. Wasil. Using experimental design to find effective parameter settings for heuristics. *Journal of Heuristics*, 7(1):77–97, 2001.

[6] N. Eén and N. Sörensson. An extensible SAT solver. In *Proc. of SAT-03*, pages 502–518, 2003.

[7] I. P. Gent, H. H. Hoos, P. Prosser, and T. Walsh. Morphing: Combining structure and randomness. In *Proc. of AAAI-99*, pages 654–660, Orlando, Florida, 1999.

[8] Carla P. Gomes and Bart Selman. Problem structure in the presence of perturbations. In *Proc. of AAAI-97*, 1997.

[9] H. H. Hoos and T. Stützle. *Stochastic Local Search – Foundations & Applications*. Morgan Kaufmann, 2005.

[10] F. Hutter, D. Babić, H. H. Hoos, and A. J.Hu. Boosting verification by automatic tuning of decision procedures. In *Formal Methods in Computer Aided Design (FMCAD'07)*, 2007. To appear.

[11] F. Hutter, Y. Hamadi, H. H. Hoos, and K. Leyton-Brown. Performance prediction and automated tuning of randomized and parametric algorithms. In *Proc. of CP-06*, pages 213–228, 2006.

[12] F. Hutter, D. A. D. Tompkins, and H. H. Hoos. Scaling and probabilistic smoothing: Efficient dynamic local search for SAT. In *Proc. of CP-02*, pages 233–248, 2002.

[13] Frank Hutter, Holger H. Hoos, and Thomas Stützle. Automated algorithm configuration based on local search. In *Proceedings of the Twenty-Second Conference on Artificial Intelligence (AAAI-07)*, 2007. To appear.

[14] K. Leyton-Brown, M. Pearson, and Y. Shoham. Towards a universal test suite for combinatorial auction algorithms. In *ACM Conference on Electronic Commerce (EC-00)*, 2000.

[15] S. Minton. Automatically configuring constraint satisfaction programs: A case study. *Constraints*, 1(1):1–40, 1996.

[16] M. Oltean. Evolving evolutionary algorithms using linear genetic programming. *Evolutionary Computation*, 13(3):387–410, 2005.