Automatic Algorithm Configuration based on Local Search

Frank Hutter and **Holger H. Hoos**
University of British Columbia, 2366 Main Mall
Vancouver, BC, V6T1Z4, Canada
{hutter, hoos}@cs.ubc.ca

Thomas Stützle
Université Libre de Bruxelles, IRIDIA, CoDE
Av. F. Roosevelt 50 B-1050 Brussels, Belgium
stuetzle@ulb.ac.be

Abstract

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 present a local search approach for algorithm configuration and prove its convergence to the globally optimal parameter configuration. Our approach is very versatile: it can, e.g., be used for minimising run-time in decision problems or for maximising solution quality in optimisation problems. It further applies to arbitrary algorithms, including heuristic tree search and local search algorithms, with no limitation on the number of parameters. Experiments in four algorithm configuration scenarios demonstrate that our automatically determined parameter settings always outperform the algorithm defaults, sometimes by several orders of magnitude. Our approach also shows better performance and greater flexibility than the recent CALIBRA system. Our ParamILS code, along with instructions on how to use it for tuning your own algorithms, is available on-line at http://www.cs.ubc.ca/labs/beta/Projects/ParamILS.

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 (Eén & Sörensson 2003) and local search (Hoos & Stützle 2005); 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 (Birattari 2004; Adenso-Diaz & Laguna 2006).

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 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 (Minton 1996), experimental design (Coy *et al.* 2001; Bartz-Beielstein 2006), genetic programming (Oltean 2005), the application of racing algorithms (Birattari *et al.* 2002; Birattari 2004), and combinations of fractional experimental design and local search (Adenso-Diaz & Laguna 2006).

Our work presented in the following offers three main contributions:

- We define the first iterated local search (ILS) algorithm for the algorithm configuration problem. Our approach works for both, deterministic and randomised algorithms and can be applied regardless of tuning scenario and optimisation objective.
- We study the effects of over-confidence and over-tuning that occur when an algorithm is tuned based on a finite number of training instances (the dominant approach in the literature). While the possibility of these effects was shown previously by Birattari (2004), we give the first precise statistical arguments and experimental results for the fact that cost estimates can be orders of magnitude too optimistic, which leads to strongly impaired performance.
- We extend our basic ILS algorithm in order to avoid over-confidence and over-tuning. A theoretical analysis of the resulting algorithm proves its convergence to the optimal parameter configuration.

In an empirical study comprising four tuning scenarios and three algorithms, our automatically determined parameter configurations always outperform the algorithms' default settings, sometimes by several orders of magnitude. In all experiments, our approach reaches or exceeds the performance of the recent CALIBRA system (Adenso-Diaz & Laguna 2006), while also being more broadly applicable.

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

Problem Definition and Notation

In typical tuning scenarios, we are given an algorithm $\mathcal{A}$, whose parameters are to be optimised for a distribution (or set) of problem instances, $\mathcal{D}$. With each of the algorithm parameters, a domain of possible values is associated, and the parameter configuration space, Θ, is the cross-product of these domains (or a subset thereof, if some combinations of parameter values are excluded). The elements of Θ are called *parameter configurations*, and we use $\mathcal{A}(\theta)$ to denote algorithm $\mathcal{A}$ with parameter configuration $\theta \in \Theta$. The objective of the *algorithm configuration problem* is then to find the parameter configuration $\theta \in \Theta$ resulting in the best performance of $\mathcal{A}(\theta)$ on distribution $\mathcal{D}$.

The meaning of “best” performance depends on the application scenario, which we deliberately leave open. The only assumption we make is that it is possible to assign a scalar cost, $sc(\mathcal{A}, \theta, I, s)$, to any single run of $\mathcal{A}(\theta)$ on an instance I, in the case of an randomised algorithm using seed s. This cost could, for example, be run-time, approximation error or the improvement achieved over an instance-specific reference cost. A distribution over instances and, in the case of randomised algorithms, a distribution over random seeds then induce a cost distribution $CD(\mathcal{A}, \theta, \mathcal{D})$.

Thus, the formalised objective in algorithm configuration is to minimise a statistic, $c(\theta)$, of this cost distribution, such as the mean, median or some other quantile. This is a stochastic optimisation problem, because the cost distributions $CD(\mathcal{A}, \theta, \mathcal{D})$ are typically unknown, and we can only acquire approximations of their statistics $c(\theta)$ based on a limited number of samples, that is, the cost of single executions of $\mathcal{A}(\theta)$. We denote an approximation of $c(\theta)$ based on N samples by $\hat{c}_N(\theta)$. Birattari (2004) proved that if the statistic of interest is expected cost, then the variance of $\hat{c}_N(\theta)$ is minimised by sampling N instances and performing a single run on each. We follow this approach here: when our finite training set has $M \geq N$ instances, we base our approximation $\hat{c}_N(\theta)$ on one run each for the first N instances. For deterministic algorithms, $\hat{c}_N$ is only of interest for $N \leq M$; for randomised algorithms, if $N > M$, we loop through the M instances, using multiple runs per instance to compute $\hat{c}_N(\theta)$.

Iterated Local Search in Parameter Configuration Space

Depending on the number of algorithm parameters and their type (categorical, ordinal, continuous), the task of finding the best parameter configuration for an algorithm can be easy or hard. If there are only very few parameters, parameter tuning tends to be easy. In this case, it is often most convenient to allow a certain number of values for each parameter (discretization) and then to try each combination of these parameter values; this approach is also known as *full factorial design*.

If an algorithm has too many parameters, one often reverts to local optimisation, because full factorial design becomes intractable as the number of possible configurations grows exponentially with the number of parameters. In such situations, the manual optimisation process is typically started with some parameter configuration, and then parameter values are modified one at a time, where new configurations are accepted whenever they result in improved performance. These sequential modifications are typically continued until no change of a single parameter yields an improvement anymore.

Algorithm 1: ParamILS

Outline of iterated local search in parameter configuration space; the specific variants of ParamILS we study, *BasicILS(N)* and *FocusedILS*, differ in the procedure *better*.

```
Input  : Parameter configuration space $\Theta$, neighbourhood
         relation $\mathcal{N}$, procedure better (compares $\theta, \theta' \in \Theta$).
Output : Best parameter configuration $\theta$ found.
$\theta_0 \leftarrow$ default parameter configuration $\theta \in \Theta$;
for $i \leftarrow 1...R$ do
    $\theta \leftarrow$ random $\theta \in \Theta$;
    if better($\theta$, $\theta_0$) then $\theta_0 \leftarrow \theta$;
$\theta_{ils} \leftarrow$ IterativeFirstImprovement ($\theta_0, \mathcal{N}$);
while not TerminationCriterion() do
    $\theta \leftarrow \theta_{ils}$;
    // ===== Perturbation
    for $i \leftarrow 1...s$ do $\theta \leftarrow$ random $\theta' \in \mathcal{N}(\theta)$;
    // ===== LocalSearch
    $\theta \leftarrow$ IterativeFirstImprovement ($\theta, \mathcal{N}$);
    // ===== AcceptanceCriterion
    if better($\theta$, $\theta_{ils}$) then $\theta_{ils} \leftarrow \theta$;
    with probability $p_{\text{restart}}$ do $\theta_{ils} \leftarrow$ random $\theta \in \Theta$;
return overall best $\theta$ found;

Procedure IterativeFirstImprovement ($\theta, \mathcal{N}$)
repeat
    $\theta' \leftarrow \theta$;
    foreach $\theta'' \in \mathcal{N}(\theta')$ in randomised order do
        if better($\theta''$, $\theta'$) then $\theta \leftarrow \theta''$; break;
until $\theta' = \theta$;
return $\theta$;
```

This widely adopted procedure corresponds to a manually executed local search in parameter configuration space: the search space is the set of all possible configurations, the objective function quantifies the performance achieved when using a given configuration, the initial configuration is the one used at the start of the manual search process, the neighbourhood is a one-exchange neighbourhood (one parameter is modified in each search step) and the search strategy is iterative first improvement. Viewing this manual procedure as a local search process is advantageous for at least two reasons: it suggests an automation of the procedure as well as improvements based on ideas from the stochastic local search community.

Simple iterative improvement search terminates in the first local optimum it encounters (i.e., in a parameter configuration that cannot be improved by modifying a single parameter value); to overcome this problem, in the following we use iterated local search (ILS) (Lourenço, Martin, & Stützle 2002) to search for performance-optimising parameter configurations. ILS is a stochastic local search method that builds a chain of local optima by iterating through a main loop consisting of, in this order, (i) a solution perturbation to escape from local optima, (ii) a subsidiary local search procedure and (iii) an acceptance criterion that is used to decide whether to keep or reject a newly obtained candidate solution.

ParamILS (Algorithm 1) is an ILS method that searches parameter configuration space. It uses a combination of default and random settings for initialisation, employs iterative first improvement as a subsidiary local search procedure, uses a fixed number, s, of random moves for perturbation, and always accepts better or equally good parameter configurations, but re-initialises the search at random with probability p_{restart}.[2] Furthermore, it is based on a one-exchange neighbourhood, that is, neighbouring parameter configurations differ in one parameter value only.

In practice, many heuristic algorithms have a number of parameters that are only relevant when some other "higher-level" parameters take certain values; we call such parameters *conditional*. (For example, SAT4J implements six different clause learning mechanisms, three of which are parameterised by conditional parameters governing percentages or clause lengths.) ParamILS deals with these conditional parameters by excluding all configurations from the neighbourhood of a configuration θ which only differ in a conditional parameter that is not relevant in θ.

In the basic variant of ParamILS, *BasicILS(N)*, procedure *better*(θ_1, θ_2) compares cost approximations $\hat{c}_N(\theta_1)$ and $\hat{c}_N(\theta_2)$ based on exactly N samples from the respective cost distributions $CD(\mathcal{A}, \theta_1, \mathcal{D})$ and $CD(\mathcal{A}, \theta_2, \mathcal{D})$. For each configuration θ, these sample costs are gathered by executing $\mathcal{A}(\theta)$ on the same N instances and recording the N costs; they are reused in later computations of $\hat{c}_N(\theta)$.

Over-confidence and Over-tuning

Like many other approaches in the literature (Minton 1996; Adenso-Diaz & Laguna 2006), BasicILS employs an optimisation approach, trying to minimise cost approximations $\hat{c}_N(\theta)$ based on a fixed number of N samples. In analogy to machine learning, we call these N samples the *training set*; the *test set* consists of additional samples, that is, the cost of additional runs on independently sampled instances.

There are two main problems associated with a fixed-size training set. We refer to the first one as *over-confidence*: intuitively, this problem concerns the fact that the cost estimated for the best training configuration, θ^*_{train}, underestimates the cost of the same configuration on the test set. This essentially happens because θ^*_{train} has the lowest observed cost on the training set, that is, $\hat{c}_N(\theta^*_{train}) = \min\{\hat{c}_N(\theta_1), \ldots, \hat{c}_N(\theta_n)\}$, and this is a biased estimator of $c(\theta^*)$, where $\theta^* \in \text{argmin}_{\theta_i}\{c(\theta_1), \ldots, c(\theta_n)\}$. In fact, $E[\min\{\hat{c}_N(\theta_1), \ldots, \hat{c}_N(\theta_n)\}] \leq E[c(\theta^*)]$ with equality only holding in pathological cases (Birattari 2004). As a result, the optimisation typically "hallucinates" a much better performance than it will achieve on an independent test set.

Consider the following example. Let the objective be to minimise average cost using a budget that only allows a single run per configuration $\theta \in \Theta$, and assume an exponential cost distribution $CD(\mathcal{A}, \theta, \mathcal{D})$ for each configuration $\theta \in \Theta$ (an assumption which, in the case of run-time distributions, is often not far from reality – see, e.g., Hoos & Stützle (2005)). Then, we can express the distribution of $\hat{c}_1(\theta^*_{train})$ in closed form due to the following Lemma.

Lemma 1 (Bias for exponential distributions)**.** *Let $\mathcal{A}$ be an algorithm with parameter configuration space Θ, and for each $\theta \in \Theta$, let $CD(\mathcal{A}, \theta, \mathcal{D}) \sim Exp(\lambda_\theta)$ for some λ_θ. Then, $\hat{c}_1(\theta^*_{train}) \sim Exp(\sum_{\theta \in \Theta} \lambda_\theta)$.*[3]

The training cost $\hat{c}_N(\theta^*_{train})$ of θ^*_{train} can thus underestimate the real cost $c(\theta^*_{train})$ by far. In the extreme case of $\lambda_{\theta_1} = \cdots = \lambda_{\theta_n}$ in Lemma 1, a cost approximation based on $N = 1$ samples underestimates the true mean cost by a factor of n in expectation, where n is the number of parameter configurations evaluated. This effect increases with the number of configurations evaluated and the variance of the cost distributions , and it decreases as N is increased.

Over-confidence has many undesirable effects on empirical studies and algorithm development. One consequence is that if algorithm costs are only reported for the best out of many parameter configurations, then these results are typically underestimates. Another issue is that there is a high chance for the best training configuration to just be a lucky one that resulted in a large cost underestimate; hence, the best training configuration θ^*_{train} often performs much worse than the true best configuration θ^*.

This can lead to the second problem encountered by typical parameter optimisation approaches, dubbed *over-tuning* (Birattari 2004): the unfortunate situation where additional tuning actually impairs test performance. This is analogous to over-fitting in supervised machine learning, which can lead to arbitrarily poor test results. In an earlier study, Birattari (2004) could only demonstrate a very small over-tuning effect using an extremely large number of 10^9 pseudo-runs; his analysis was qualitative, and he did not suggest a solution to the problem. We demonstrate that over-confidence and over-tuning can in fact severely impair the performance of parameter optimisation methods (see Figures 1(a) and 1(b)) and introduce a variant of ParamILS that provably does not suffer from these effects.

Our approach is based on the fact that when $\hat{c}_N(\theta)$ is a consistent estimator of $c(\theta)$,[4] the cost approximation becomes more and more reliable as N approaches infinity, eventually eliminating over-confidence and the possibility of mistakes in comparing two parameter configurations (and thus, over-tuning). This is captured in the following Lemma.

Lemma 2 (No mistakes)**.** *Let $\theta_1, \theta_2 \in \Theta$ be any two parameter configurations with $c(\theta_1) < c(\theta_2)$. Then, for consistent estimators $\hat{c}_N$, $\lim_{N\to\infty} P(\hat{c}_N(\theta_1) \geq \hat{c}_N(\theta_2)) = 0$.*

In order to optimise with respect to the true objective function, this Lemma suggests to base the cost approximation on a large number of samples, N. However, using large sample sizes N has the disadvantage that the evaluation of a parameter configuration's cost becomes very expensive.

Focused ILS

In this section, we introduce FocusedILS, a variant of ParamILS that moves through configuration space quickly but avoids the problems of over-confidence and over-tuning. It achieves this goal by focusing samples on promising parameter configurations. In the following, we denote the

[2] Our choices of $\langle R, s, p_{\text{restart}}\rangle = \langle 10, 3, 0.01\rangle$ are somewhat arbitrary – we have not experimented with other settings but would expect performance to be fairly robust with respect to their choice.

[3] The straight-forward proofs of this and the following lemmata are provided in a companion technical report.

[4] Recall that $\hat{c}_N(\theta)$ is a consistent estimator for $c(\theta)$ iff $\forall \epsilon > 0 : \lim_{N\to\infty} P(|\hat{c}_N(\theta) - c(\theta)| < \epsilon) = 1$.

number of samples used to approximate a cost distribution $CD(\mathcal{A}, \theta, \mathcal{D})$ by $N(\theta)$.

In FocusedILS, procedure *better*(θ_1, θ_2) is somewhat more complicated than in BasicILS. We say that θ_1 *dominates* θ_2 if, and only if, $N(\theta_1) \geq N(\theta_2)$ and the performance of $\mathcal{A}(\theta_1)$ using the first $N(\theta_2)$ samples is better than that of $\mathcal{A}(\theta_2)$; formally, $\hat{c}_{N(\theta_2)}(\theta_1) < \hat{c}_{N(\theta_2)}(\theta_2)$.[5] Procedure *better*$(\theta_1, \theta_2)$ starts by acquiring one sample for the configuration with smaller $N(\theta_i)$ (one each in the case of ties). Then, it iteratively acquires samples for the configuration with smaller $N(\theta_i)$ until one configuration dominates the other and returns true if, and only if, θ_1 dominates θ_2.

We also keep track of the total number B of samples acquired since the last improving step (the last time procedure *better* returned true). Whenever *better*(θ_1, θ_2) returns true, we acquire B "bonus" samples of $CD(\mathcal{A}, \theta_1, \mathcal{D})$ and reset B. This mechanism ensures that we acquire many samples of good configurations, and that the error made in every comparison of two configurations θ_1 and θ_2 decreases in expectation. Following Lemma 2, eventually the truly better θ_i will be preferred; based on this fact, we can prove convergence to the true best parameter configuration.

Lemma 3 (Unbounded number of evaluations)**.** *Let $N(J, \theta)$ denote the number of samples used by FocusedILS at the end of iteration J to approximate $c(\theta)$. Then, for any constant K and configuration $\theta \in \Theta$ (with finite Θ), $\lim_{J\to\infty} [P(N(J, \theta)) \geq K] = 1$.*

The proof of this lemma essentially exploits the probabilistic random restarts performed within ParamILS, which have the effect that the number of samples for any configuration grows arbitrarily large as the number of iterations approaches infinity.

Theorem 4 (Convergence of FocusedILS)**.** *As the number of iterations approaches infinity, FocusedILS converges to the true optimal parameter configuration θ^*.*

Proof. According to Lemma 3, $N(\theta)$ grows unboundedly for each $\theta \in \Theta$. For each θ_1, θ_2, as $N(\theta_1)$ and $N(\theta_2)$ approach infinity, Lemma 2 states that in a pairwise comparison, the truly better configuration will be preferred. The theorem follows from the fact that eventually, FocusedILS visits all finitely many parameter configurations and prefers the best one over all others with probability one. □

Empirical evaluation

We report computational experiments for tuning three different algorithms: SAT4J (http://www.sat4j.org), a tree search solver for SAT; SAPS (Hutter, Tompkins, & Hoos 2002), a local search algorithm for SAT; and GLS$^+$, a local search algorithm for solving the Most Probable Explanation (MPE) problem (Hutter, Hoos, & Stützle 2005). Their respective parameters are summarised in Table 1 (for details, see the original publications). For discretisation, we picked lower and upper bounds for each parameter (same as for CALIBRA) and discretised the interval uniformly or uniformly on a log scale. For SAT4J, the algorithm developer provided a discretisation.

Parameter	Default value	Values considered for tuning
SAPS: $7^4 = \mathbf{2\,401}$ parameter configurations		
α	1.3	1.01, 1.066, 1.126, 1.189, 1.256, 1.326, 1.4
ρ	0.8	0, 0.17, 0.333, 0.5, 0.666, 0.83, 1
P_{smooth}	0.05	0, 0.033, 0.066, 0.1, 0.133, 0.166, 0.2
wp	0.01	0, 0.01, 0.02, 0.03, 0.04, 0.05, 0.06
GLS$^+$: $2 \times 6 \times 5 \times 4 \times 7 = \mathbf{1\,680}$ parameter configurations		
init	MB-w(10^5)	random, MB-w(10^5)
ρ	0.9	0.7, 0.8, 0.9, 0.99, 0.999, 1.00
N_ρ	200	50, 200, 10^3, 10^4, ∞
w	10^4	10^2, 10^3, 10^4, 10^5
b	10^2	0, 10^2, 3×10^2, 10^3, 3×10^3, 10^4, 10^5
SAT4J: $(5{+}1{\times}5){\times}(3{+}3{\times}5){\times}6{\times}3{\times}5{\times}7{\times}5 = \mathbf{567\,000}$ conf.		
Variable order	OrderHeap	6 categorical values, 1 of them parameterised
Learning strategy	Minisat	6 categorical values, 3 of them parameterised
Data structure	mixed	6 categorical values
Clause minimisation	expensive	3 cat. values: none, simple, expensive
Variable decay	0.95	0.8, 0.85, 0.9, 0.95, 1.0
First restart	100	100, 200, 500, 1000, 5000, 30000, 100000
Restart multiplication	1.5	1.1, 1.3, 1.5, 1.7, 2.0

Table 1: Algorithms, their parameters and parameter values considered for tuning. For SAT4J, "X of them parameterised" for variable order and learning strategy means that for X values of that parameter, a conditional continuous parameter becomes relevant; we consider five values for each such conditional parameter.

We carried out experiments for tuning SAT4J and SAPS for a variety of instance distributions, including instances from previous SAT competitions; an instance mix from SATLIB; a single "quasi-group with holes" instance, *QWH*; and a set of $20\,000$ graph colouring instances based on small-world graphs, *SW-GCP* (Gent *et al.* 1999). Due to space limitations, we only report results for QWH (a fairly easy instance allowing us to perform many runs) and SW-GCP. We constructed two subclasses of SW-GCP: *SW-GCP-sat4j*, containing all $2\,202$ instances for which SAT4J takes between five and ten seconds; and *SW-GCP-saps*, containing all 213 satisfiable instances with median SAPS run-time between five and ten seconds.[6]

For GLS$^+$, we optimise performance for instance set *GRID*, a set of 100 grid-structured Bayesian networks with 25×25 binary variables, random conditional probability tables and 10 evidence variables. GLS$^+$ is the best-performing algorithm for these instances (Hutter, Hoos, & Stützle 2005). Each instance set with the exception of QWH is split into independent training and test sets; for QWH, test performance is computed from independent runs on the same instance.

We consider four parameter optimisation scenarios:

- **SAT4J-SW**: Minimise SAT4J median run-time for set SW-GCP-sat4j, using a cutoff of ten seconds.
- **SAPS-SW**: Minimise SAPS median run-time for set SW-GCP-saps, using a cutoff of ten seconds.
- **SAPS-QWH**: Minimise the median run-length of SAPS on the single instance QWH, using a cutoff of one second.
- **GLS-GRID**: Maximise the average probability GLS$^+$ finds for the networks in GRID within 10 seconds, relative to the best solution GLS$^+$ finds with its default settings in

[5] Note that some of the samples for $CD(\mathcal{A}, \theta_1, \mathcal{D})$ are not used in this comparison, because for heterogeneous instance distributions, $\mathcal{D}$, it is more robust to base the comparison only on the first $N(\theta_2)$ sampled instances for which both configurations have been evaluated.

[6] All experiments were carried out on a cluster of 50 dual 3.2GHz Intel Xeon PCs with 2MB cache and 2GB RAM, running OpenSuSE Linux 10.1. Reported runtimes are CPU times.

Scenario	Default	CALIBRA(100)	BasicILS(100)	FocusedILS	p-value
GLS-GRID	$\epsilon = 1.81$	1.240 ± 0.469	0.965 ± 0.006	0.968 ± 0.002	
		1.234 ± 0.492	0.951 ± 0.004	$\mathbf{0.949 \pm 0.0001}$	$\mathbf{.0016}$
SAPS-QWH	$85.5K$	$8.6K \pm 0.7K$	$8.7K \pm 0.4K$	$10.2K \pm 0.4K$	
	steps	$10.7K \pm 1.1K$	$10.9K \pm 0.6K$	$\mathbf{10.6K \pm 0.5K}$	.52
SAPS-SW	5.60	0.044 ± 0.005	0.040 ± 0.002	0.043 ± 0.005	
	seconds	0.053 ± 0.010	0.046 ± 0.01	$\mathbf{0.043 \pm 0.005}$	$\mathbf{.0003}$
SAT4J-SW	7.02	N/A (too many	0.96 ± 0.59	0.62 ± 0.21	
	seconds	parameters)	1.19 ± 0.58	$\mathbf{0.65 \pm 0.2}$	$\mathbf{.00007}$

Table 2: Performance of CALIBRA(100), BasicILS(100) and FocusedILS in four tuning scenarios. For each scenario and approach, we list training and test performance in the top and bottom row, respectively (mean ± stddev over 25 independent repetitions); "p-value" refers to the p-value of a Wilcoxon signed rank test for testing the null hypothesis "there is no difference between CALIBRA and FocusedILS results" (BasicILS *vs* FocusedILS for SAT4J).

one hour. We equivalently *minimise* the reciprocal value ϵ; values $\epsilon < 1$ mean that tuned parameters find better solutions in 10 seconds than the default finds in one hour.

We compare the results of BasicILS and FocusedILS to the algorithms' default parameters as well as to the parameter configurations found by the recent CALIBRA system (Adenso-Diaz & Laguna 2006). For each optimisation scenario, we perform multiple independent training repetitions of CALIBRA, BasicILS and FocusedILS; each repetition consisted of an independent execution of the algorithm to be tuned. Test performance for each approach and repetition was computed on the same independent test set.

CALIBRA(100) and BasicILS(100) both use a cost approximation based on $N = 100$ samples for each configuration. Each repetition is terminated after $n = 200$ configurations have been visited (which is the CALIBRA default). This leads to a total of $n \times N = 20\,000$ algorithm executions per repetition, which took about 10h for SAPS-SW, 1h for SAPS-QWH and 55h for GLS-GRID; the same cutoff times were used for FocusedILS. For SAT4J-SW, we chose a cutoff time of 10h.

Table 2 shows that CALIBRA(100) and BasicILS(100) both lead to large improvements over the algorithm defaults, with a slight edge for BasicILS, whose main advantage over CALIBRA is that it supports arbitrary numbers and types of parameters. In contrast, CALIBRA is limited to a total of five continuous or integer parameters, rendering it inapplicable for tuning SAT4J.

Even for cost approximations based on $N = 100$ samples, we notice a significant difference between training and test performance for both CALIBRA and BasicILS. FocusedILS judges its progress more realistically, showing more similar training and test performance, and more importantly, always the best test performance.

In practical applications, one typically cannot afford as many as $20\,000$ algorithm executions (each of which may take hours), and may thus be tempted to use a lower value of N for CALIBRA or BasicILS. We now demonstrate that using small values of N can lead to very poor behaviour due to over-confidence and over-tuning; for this, we employed tuning scenario SAPS-QWH and BasicILS(1). In each repetition of BasicILS(1), we evaluated each visited configuration θ by a single run of SAPS(θ) on instance QWH, keeping track of the incumbent configuration $\theta_{inc}(t)$ with the shortest run-length across the configurations visited up to time t – this constitutes training performance. Off-line, at every time step t, we then computed test performance $c(\theta_{inc}(t))$ using $1\,000$ independent runs of SAPS on QWH. We repeated this process 100 times with different runs used for training, leading to 100 training and 100 test performance measurements for each time step; Figure 1(a) shows the median, 10% and 90% quantiles of these. Training and test performance diverge, and especially in the 90% quantile of test performance we see evidence for over-tuning. The same over-tuning effect occurs for CALIBRA(1).

FocusedILS avoids the problem of over-tuning gracefully by performing more runs for promising parameter configurations: after ten seconds, its training performance for SAPS-QWH is virtually identical to its test performance. One may expect this to come at the cost of slow movement through configuration space, but as we demonstrate in Figure 1(b), this is not the case: FocusedILS starts its progress as quickly as BasicILS(1), but also always performs better than BasicILS(10) and BasicILS(100).

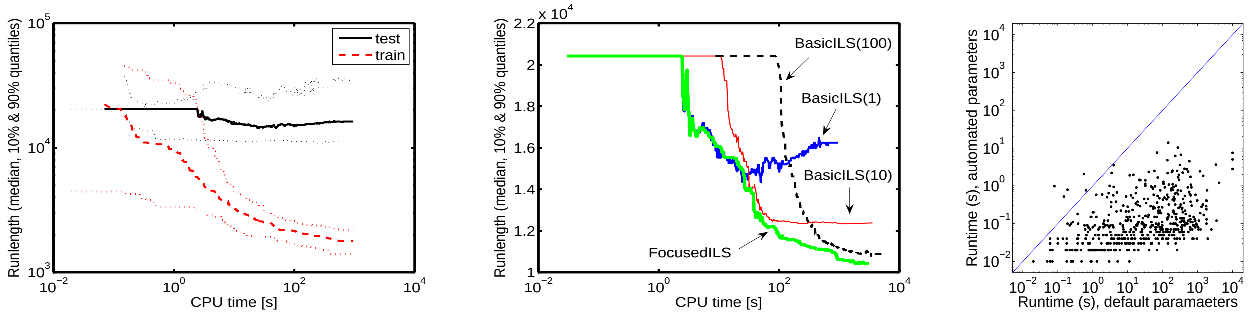
We further studied whether a parameter setting tuned for an easy subclass (SW-GCP-saps) of a fairly homogeneous problem class (SW-GCP) can lead to good performance on the entire class. Figure 1(c) shows that in our particular example, automatically tuned parameters for SW-GCP-saps (found by FocusedILS in one hour) drastically outperform the SAPS default setting for the entire set, with speedups up to four orders of magnitude. This is an extremely encouraging result for automatic parameter tuning on homogeneous instance classes.

Finally, ParamILS was used by Thachuk et al. to tune a replica exchange Monte Carlo algorithm for protein structure prediction (Thachuk, Shmygelska, & Hoos 2007) and achieved performance improvements of about 50%. It was also applied to several solvers submitted to the 2007 SAT Competition, where it achieved a speedup of about an order of magnitude for SAPS on random instances and a 50-fold speedup for tuning a new state-of-the-art tree search algorithm on SAT-encoded software verification problems (Hutter *et al.* 2007).

Related Work

There already exist several approaches for automatic algorithm configuration for a given problem distribution (Minton 1996; Coy *et al.* 2001; Adenso-Diaz & Laguna 2006; Oltean 2005). Like BasicILS, these techniques are all subject to over-tuning. The only methods of which we are aware that would avoid over-tuning if given enough time are racing algorithms (Birattari *et al.* 2002; Birattari 2004); however, these are limited to tuning tasks with a fairly small number of configurations.

Recent work on finding the best parameter configuration on a per instance basis (see, e.g., the work by Hutter et al. (2006) and the references therein) is in principle more powerful than the per distribution approach we follow, but it is much less general since it requires instance features and makes various assumptions about the shape of run-time distributions. Orthogonal to our off-line tuning approach is the idea of tuning algorithm parameters online, at execution time. This strategy is followed by so-called reactive search methods (Battiti & Brunato 2005). On-line and off-line strategies can be seen as complementary: even reactive search methods tend to have a number of parameters that remain fixed during the search and can hence be tuned by off-line approaches.



(a) Over-tuning for BasicILS(1) and SAPS-QWH.

(b) Test performance for SAPS-QWH.

(c) SAPS speedup on SW-GCP.

Figure 1: (a) and (b): Over-confidence and over-tuning effects in tuning scenario SAPS-QWH. The over-tuning in (a) causes BasicILS(1) to perform very poorly in (b). At each time point, we plot training and test performance of the incumbent (training) parameter configuration found so far. FocusedILS dominates all versions of BasicILS. (c) Performance of SAPS, $\langle \alpha, \rho, P_{smooth}, wp \rangle = \langle 1.189, 0.666, 0.066, 0.01 \rangle$ (tuned on SW-GCP-saps) *vs* $\langle 1.3, 0.8, 0.05, 0.01 \rangle$ (SAPS defaults) on 500 randomly sampled satisfiable instances from data set SW-GCP. The average speedup is between three and four orders of magnitude: average run-time 0.43 *vs* 263 seconds, median 0.09 *vs* 33 seconds.

Conclusion and Future Work

ParamILS is a powerful, new method for the automated configuration and tuning of parameterised algorithms. Its underlying principles are easy to understand, and it is a versatile tool that can be applied to effectively configure a wide variety of algorithms, including those that have many parameters (some of which can be conditional). One particular variant of this method, FocusedILS, provably converges to the true optimal parameter configuration and effectively avoids the problems of over-confidence and over-tuning.

We have demonstrated the practical usefulness of ParamILS in a number of tuning scenarios, in all of which it outperforms the algorithm defaults, sometimes by orders of magnitude. It also performs better than or on par with the recent CALIBRA system and is, in contrast to CALIBRA, directly applicable in tuning scenarios involving a large number of parameters. ParamILS proved to be very helpful in various recent parameter tuning tasks in the development of new algorithms for SAT and protein structure prediction.

There are a number of avenues for further research. By integrating statistical testing procedures into our algorithm, unnecessary evaluations of poorly performing configurations can be avoided. Furthermore, it would be interesting to consider other ways of searching the neighbourhood and to investigate specialised local search methods for continuous parameters. The need for discretisation is a limitation of ParamILS, and we will investigate how this limitation can be overcome by using response surface models.

Overall, we are convinced that the increasingly effective tools produced by this line of research will allow researchers to focus on the essential and scientifically valuable tasks in designing algorithms for solving hard problems, and help practitioners to solve application problems more effectively.

Acknowledgements

We would like to thank Kevin Leyton-Brown and Kevin Murphy for discussions and helpful comments on an earlier version of this manuscript.

References

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

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

Battiti, R., and Brunato, M. 2005. Reactive search: machine learning for memory-based heuristics. Technical Report DIT-05-058, Università Degli Studi Di Trento, Trento, Italy.

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

Birattari, M. 2004. *The Problem of Tuning Metaheuristics as Seen from a Machine Learning Perspective*. Ph.D. Dissertation, Université Libre de Bruxelles, Brussels, Belgium.

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

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

Gent, I. P.; Hoos, H. H.; Prosser, P.; and Walsh, T. 1999. Morphing: Combining structure and randomness. In *Proc. of AAAI-99*, 654–660.

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

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

Hutter, F.; Babić, D.; Hoos, H. H.; and Hu, A. J. 2007. Boosting verification by automatic tuning of decision procedures. Manuscript in preparation.

Hutter, F.; Hoos, H. H.; and Stützle, T. 2005. Efficient stochastic local search for MPE solving. In *Proc. of IJCAI-05*, 169–174.

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

Lourenço, H. R.; Martin, O. C.; and Stützle, T. 2002. Iterated local search. In *Handbook of Metaheuristics*, 321–353. Kluwer.

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

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

Thachuk, C.; Shmygelska, A.; and Hoos, H. H. 2007. Replica exchange Monte Carlo for protein folding in the HP model. Submitted to BMC Bioinformatics.