# Analysing differences between algorithm configurations through ablation

Chris Fawcett, Holger H. Hoos

Computer Science Department
University of British Columbia
201-2366 Main Mall, Vancouver, BC, Canada
{fawcettc,hoos}@cs.ubc.ca

**Abstract**

Developers of high-performance algorithms for hard computational problems increasingly take advantage of automated algorithm configuration tools, and consequently often create solvers with many parameters and vast configuration spaces. However, there has been very little work to help these algorithm developers answer questions about the high-quality configurations produced by these tools, specifically about which parameter changes contribute most to improved performance. In this work, we present an automated technique for answering such questions by performing *ablation analysis* between two algorithm configurations. We perform an extensive empirical analysis of our technique on five scenarios from propositional satisfiability, mixed-integer programming and AI planning, and show that in all of these scenarios more than 95% of the performance gains between default configurations and configurations obtained by using automated configuration tools can be explained by modifying the values of a small number of parameters (1–4 in the scenarios we studied).

## 1 Introduction

High-performance solvers for hard computational problems such as propositional satisfiability or mixed-integer programming are typically run by users on classes of problem instances from different application domains (such as random 3-SAT, hardware verification or software verification). The existence of such varied domains provides an incentive to the developers of such solvers to parameterise aspects of their implementation to customise the behaviour on each target problem domain. Finding good values manually for these algorithm parameters is difficult, as even human experts have trouble predicting which configurations will result in high performance due to interactions between parameters and the sheer size of the combinatorial configuration spaces involved.

While tools specifically designed for automatically tuning the parameters of such algorithms have been in use for at least a decade (see, *e.g.*, [3]), the introduction of advanced procedures capable of dealing with dozens of parameters, such as ParamILS [14, 15], GGA [1], irace [16] and SMAC [11], has generated great interest in the area of automated algorithm configuration. The success of these automatic algorithm configurators in practice has inspired a software design paradigm called *Programming by Optimisation* (PbO) [8], which encourages developers to expose design choices and actively seek alternatives for key parts of their algorithms, leading to highly parametric designs that are then automatically optimised for specific use contexts.

However, many configurations are sampled by these configuration tools, and developers are often left wondering *why* their algorithm parameters were set to specific values by the automated configuration process, or whether the modification of some parameters from their default settings was truly necessary to achieve substantially improved performance. Given a highly parameteric algorithm, after making many parameter changes as a result of automated configuration, how can an algorithm developer know which of the parameter changes were actually important? The ability to answer questions like these will allow developers to focus their efforts on the aspects of their solvers that are providing the most performance gains (or losses), in an iterative algorithm development process.

In this work, we introduce the concept of *ablation analysis*, a procedure investigating the path of configurations obtained by iteratively modifying parameter settings from a source configuration (*e.g.*, an expert-defined default) to those from a target configuration (*e.g.*, one obtained from an automatic configurator). Parameter values are modified one at a time, and at each stage the configuration with the

best performance is retained. We present a brute-force approach to this analysis, as well as an accelerated version that takes advantage of racing methods for algorithm selection. We demonstrate the effectiveness of this approach with an empirical study on five well-studied algorithm configuration scenarios that involve high-performance solvers for propositional satisfiability, mixed integer programming and AI planning problems, and we show that for these scenarios, more than 95% of the performance gains from automated configuration can be obtained by the modification of at most 4 (out of 26–76) algorithm parameters.

The remainder of this paper is structured as follows: In Section 2, we place our contribution in context with related work in parameter importance, and we then provide an in-depth explanation of both variants of our ablation analysis procedure in Section 3. Section 4 presents the details of the experimental study that we performed, with the results of that study shown and discussed in Section 5. Finally, we conclude in Section 6 with a discussion of possible extensions and future work in this area.

## 2 Background and Related Work

Compared to the work on algorithm configuration, there has been little progress on addressing the question of parameter importance. The most closely related area of related work is that of sensitivity analysis in statistics, especially analysis of variance (ANOVA) and functional ANOVA [7] approaches to decomposing model or function response variance into low-order components. There has also been other related work on interactive parameter exploration using contour plot visualization [2], on evolutionary algorithms for parameter relevance estimation [18] and on experimental design for analysing optimization algorithms [4]. These techniques each have difficulties with the high dimensionality and discrete nature of the configuration spaces of typical highly-parameterised algorithms. Many individual applications of automated algorithm configuration to specific solvers include statements from the authors about the modified parameters, as a post-hoc subjective justification without formal analysis. Examples of this include the configuration of a state-of-the-art industrial SAT solver [9], as well as the automated design of general-purpose frameworks for AI planning [20].

Very recently, Hutter *et al.* have been using model-based techniques to investigate the problems of parameter importance and parameter interaction directly, using forward selection [13] and functional ANOVA [12]. Both approaches require an initial data-gathering step to obtain algorithm performance data, which is then partitioned into training and test sets. In [13], this data was obtained by sampling 1 000 – 10 000 pairs of configurations and instances uniformly at random, while in [12] 10 000 randomly sampled runs were combined with the algorithm runs performed during 25 executions of the SMAC configurator (62 861 – 1 354 189 additional runs [1]).

In the forward selection approach [13] this performance data is used to iteratively build a regression model by greedily adding, at each iteration, the parameter or instance feature which results in a model with the lowest root mean squared error on the validation set. The work in [12], on the other hand, introduces an efficient technique for applying functional ANOVA to random forest models. This variance decomposition takes a random forest model constructed from the precomputed data, and expresses the performance variation in terms of components, with one component for every subset of parameters of size up to $k$ (for small $k$). These two contributions differ from our own in several fundamental ways.

Current versions of the forward selection and functional ANOVA approaches construct models wholly or partially based on thousands of configurations sampled uniformly at random from the configuration space. The CPU time required to obtain this data, as well as the time required to build the models themselves, can be significant. The CPU time requirements for model construction are especially significant for forward selection, which typically requires the construction of thousands of models.

More importantly, this random sampling of configurations means that many of the configurations used to build the model are from parts of the configuration space that are unlikely to contain high quality configurations. Furthermore, both methods have so far been used only to measure parameter importance globally on expectation across the entire configuration space, with the exception of one set of functional

[1] Personal communication with the authors.

**Algorithm 1:** Ablation $(\mathcal{A}, \theta_{\mathsf{source}}, \theta_{\mathsf{target}}, I, m)$

**Input**: Parameterised algorithm $\mathcal{A}$, two parameter configurations of $\mathcal{A}$, $\theta_{\mathsf{source}}$ and $\theta_{\mathsf{target}}$, benchmark instance set $I$, performance metric $m$
**Output**: An ordered list $(\theta_0, \theta_1, \theta_2, \theta_3, \ldots, \theta_l)$ of configurations of $\mathcal{A}$ chosen during each round of ablation. $\theta_0 = \theta_{\mathsf{source}}$ and $\theta_l = \theta_{\mathsf{target}}$

$\theta \leftarrow \theta_{\mathsf{source}}$
activeParameters $\leftarrow$ set of parameters of $\mathcal{A}$ with different values in $\theta_{\mathsf{source}}$ and $\theta_{\mathsf{target}}$
ablationRoundsBest $\leftarrow (\theta_{\mathsf{source}})$
**while** activeParameters $\neq \emptyset$ **do**
  $\mathcal{A}' \leftarrow$ set of algorithms with configurations obtained from $\theta$ via flipping 1 parameter in activeParameters to the value in $\theta_{\mathsf{target}}$, ignoring configurations that are prohibited in the configuration space or that are equal to $\theta$ due to parameter conditionality.
  $\theta' \leftarrow$ determine_best $(\mathcal{A}', I, m)$
  $\theta \leftarrow \theta'$
  activeParameters $\leftarrow$ set of parameters of $\mathcal{A}$ with different values in $\theta$ and $\theta_{\mathsf{target}}$
  append (ablationRoundsBest, $\theta'$)
**end**
**return** ablationRoundsBest

ANOVA results where model samples were restricted to configurations with better performance than the default configuration. The importance values derived from those experiments are still global measures, and can be averages across many regions of very different high-performance configurations. There is no guarantee that these importance measures apply to any individual algorithm configuration, specifically to any high-performance configuration and the local neighbourhoods around such configurations. Finally, the functional ANOVA work relies on the assumption that accurate models of algorithm performance can be obtained at a reasonable computational cost. This appears to be the case for the experiments reported by Hutter *et al.* [12], but there is no guarantee that on other scenarios, models with similar parameter importance accuracy can be practically obtained. Our approach does not require model construction, and is therefore not constrained by this assumption.

The most important distinguishing factor between our work presented here and these earlier studies lies in the fact that we are interested in explaining the importance of differences between two algorithm configurations that are of interest to an algorithm developer and user – for example, between the default configuration and one produced by applying an automated algorithm configuration tool such as ParamILS. Using ablation analysis, we can quantify the performance losses (or gains) along the "ablation path" (see Section 3.3) from one configuration to another. This allows algorithm developers or users to find a minimal set of parameter modifications from a given default configuration, while maintaining most or all of the performance gains achieved by automated algorithm configuration. We believe that this approach can be complementary to the recent model-based techniques of Hutter *et al.* [12, 13], as the local information provided by our approach can strengthen and validate (or invalidate) the results obtained with those techniques. We also believe that there are ways to combine the two lines of work (see Section 6).

# 3 Ablation Analysis

Given a parameterised algorithm $\mathcal{A}$ with $d$ parameters and configuration space $\Theta$, along with a source and target configuration ($\theta_{\mathsf{source}}$, $\theta_{\mathsf{target}} \in \Theta$) of that algorithm, our ablation procedure works as follows. Given a set of benchmark instances $I$ and a performance metric $m$ (*e.g.*, penalised average runtime or mean solution quality), we first compute the set of parameters whose values differ between $\theta_{\mathsf{source}}$ and $\theta_{\mathsf{target}}$. Then, beginning from $\theta_{\mathsf{source}}$, we proceed through a series of rounds: in each round, we use a subprocedure determine_best to choose a configuration from the set of all configurations obtained by flipping one parameter in the current configuration to its value in $\theta_{\mathsf{target}}$. Algorithm 1 further outlines the details of this procedure.

In each round of ablation, *i.e.*, in each iteration of the while-loop in Algorithm 1, the procedure

determine_best $(\mathcal{A}', I, m)$ selects the configuration in $\mathcal{A}'$ with the best performance on $I$ w.r.t. $m$. In the case where the source configuration has better performance on $I$ than the target configuration, each configuration selected by determine_best $(\mathcal{A}', I, m)$ will be the one with *minimum loss* compared to the configuration $\theta$ from the previous round. Conversely, when $\theta_{\mathsf{target}}$ has better performance than $\theta_{\mathsf{source}}$ on $I$, the configuration selected by determine_best will be that with *maximum gain* over the previous $\theta$. Some parameterised algorithms have conditional parameters, *i.e.*, parameters that only exist (or whose values only affect algorithm performance) if one or more other parameters (parents) are set to specific values. Configurations obtained by modifying the values of inactive conditional parameters are ignored in our procedure, as these configurations are by definition identical to the configuration from which they were produced.

In the experiments we present in Section 4, we perform ablation analysis in both directions for every pair of configurations. By performing ablation in the direction of minimum loss, we can gauge the relative extent (by number of parameter modifications) of the local area around $\theta_{\mathsf{source}}$ with roughly equal performance. In the direction of maximum gain, we find the minimal number of parameter modifications required to achieve roughly equal performance to $\theta_{\mathsf{target}}$. As a greedy approach (not unlike forward selection), ablation in either direction may produce suboptimal results at any distance except 1 from $\theta_{\mathsf{source}}$. In light of this, performing the analysis in two directions provides additional robustness. In the following, we describe two variants of determine_best $(\mathcal{A}', I, m)$: a naïve brute-force method, which is easy to implement but slow, and a greatly accelerated version based on a racing method.

### 3.1 Brute-Force Ablation

Our brute-force implementation of determine_best $(\mathcal{A}', I, m)$ involves performing a full empirical performance evaluation for every configuration in $\mathcal{A}'$, by running each configuration in $\mathcal{A}'$ on every instance in $I$ and recording the value of the performance metric $m$ thus obtained. The configuration in $\mathcal{A}'$ with the best metric value is selected as the best and returned by determine_best.

Given that one parameter is eliminated from consideration in every round, ablation on instance set $I$ with $p$ differing parameters between $\theta_{\mathsf{source}}$ and $\theta_{\mathsf{target}}$ using this brute-force approach will require up to $|I| \cdot p \cdot (p+1)/2$ individual runs of algorithm $\mathcal{A}$. Therefore, this procedure can be extremely time-consuming in the presence of high runtime cutoffs or large instance sets. Consider a typical case of ablation between a source and target configuration with $25$ differing parameters, and an instance set $I$ with $1\,000$ benchmark instances. Over the course of ablation using the brute-force method, $325\,000$ algorithm runs will be performed. Even with a mean CPU time of only $30$ seconds per run of $\mathcal{A}$ for any instance from $I$ for all configurations considered in the analysis, this implies an overall runtime requirement of $9\,750\,000$ CPU seconds or $112$ CPU days. We note that, by parallelizing runs across a cluster of machines (as we do in our experiments), this does not necessarily render ablation using this method completely impractical, it represents a formidable computational burden. Clearly, a more efficient ablation procedure would be highly desirable.

### 3.2 Acceleration via Racing

Based on early work for solving the model selection problem in memory-based supervised learning [17], F-Race is a prominent racing method for algorithm selection [3]. Given a benchmark instance set and performance metric, F-Race takes a set of candidate algorithms (or configurations of a parameterised algorithm) and iterates between gathering performance data by running the candidate algorithms on benchmark instances, and eliminating candidates once there is enough statistical evidence to justify removing them. The algorithms remaining at the end of the procedure are the winners of the race.

We apply F-Race to ablation analysis round winner determination, adhering very closely to the statistical framework described by Birattari *et al.* [3]. In this context, F-Race starts with a set of candidate configurations containing all configurations in $\mathcal{A}'$ and subsequently performs a sequence of stages. In stage $k$, the remaining candidate configurations $C = (c_1, c_2, \ldots, c_n)$ are evaluated on a new instance $i_k$ from $I$, and the results are then combined with the results of the previous stages for each configuration.

These results are then organised into $k$ *blocks*, with the $j^{\text{th}}$ block containing the $n$ performance metric values resulting from running the configurations in $C$ on $i_j$.

On these blocks, a Friedman two-way analysis of variance by ranks, also known as the Friedman test, is performed [5]. If the null hypothesis of this test is rejected, we can conclude that at least one configuration in $C$ has statistically significantly better performance than at least one other configuration. In this case, we proceed to pairwise testing to identify which configurations should be removed from the candidate list $C$. We use the same pairwise test here as described by Birattari *et al.* for F-Race, by comparing the configuration with the best sum of ranks across all blocks with the other $n - 1$ configurations in $C$ using a modified $t$-test with $n - 1$ degrees of freedom [3]. After culling any configurations deemed to be statistically significantly worse, we proceed to the next phase. The race terminates when only one configuration remains, or when a specified maximum number of rounds have been performed. In the latter case, the configuration with the best mean metric score across all rounds is selected as the winner. In the case of further ties, tie-breaking is performed uniformly at random. (Further details on the statistical tests used in this procedure can be found in [3, 5]).

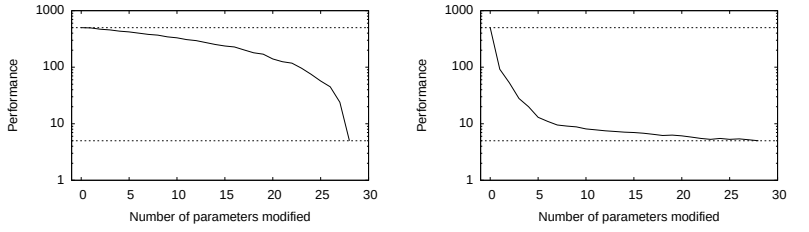
To measure the speed-ups of this racing approach to ablation analysis, compared to the brute-force approach described earlier, we performed experiments using two of the scenarios described in Section 4. We examined two alternatives for our racing approach, first with the maximum number of rounds set to the number of instances in the benchmark instance set (302 for SPEAR and 1 000 for CPLEX), and second with the maximum number of rounds set to 200. Determining the optimal value for this parameter is not straightforward, but in general it can be expected to depend on the homogeneity of the given set of benchmark instances. We used our conservative choice of 200 after subsampling runs from the full instance sets, and choosing the lowest value that did not change the distribution of runtime over the resulting set in any substantial way for any of our scenarios. In these two examples, racing with the maximum number of rounds set to 200 reduced the runtime required for our SPEAR and CPLEX scenarios to 23% and 14% of the brute-force runtime, respectively. We also note that every algorithm run used by our racing approach inside each call of determine_best can be performed in parallel, which in our case resulted in wall-clock times of merely a few hours, a further 25% and 11% of the total CPU time.

While it is possible that racing will return different ablation paths than the brute-force approach, we do not consider this to be a problem, since the brute-force approach is also not guaranteed to compute the optimal path between two configurations. Furthermore, in all of our experiments, the brute-force and racing results were closely aligned.

### 3.3 Ablation Paths

We call the path of configurations $(\theta_0, \ldots, \theta_l)$ obtained between $\theta_{\mathsf{source}}$ and $\theta_{\mathsf{target}}$ computed by our ablation procedure along with the respective performance values on set $I$ (or an independent test set of instances similar to those in $I$) an *ablation path*. These paths can take several qualitatively different forms, depending on the relative performance of $\theta_{\mathsf{source}}$ and $\theta_{\mathsf{target}}$ and characteristics of the response surface that captures the functional dependency of the performance of $\mathcal{A}$ on its parameter settings. Figure 1 illustrates these cases; each point represents the performance of one of the configurations $\theta_i$, from $\theta_{\mathsf{source}}$ on the left-hand side to $\theta_{\mathsf{target}}$ on the right. Figure 1(a) illustrates one extreme case, where $\theta_{\mathsf{source}}$ and $\theta_{\mathsf{target}}$ differ in performance on $I$ and all parameters are of equal importance. A case at the opposite extreme (not shown) would be if the performance difference between $\theta_{\mathsf{source}}$ and $\theta_{\mathsf{target}}$ was fully explained by the modification of a single parameter.

A more realistic case lies between these extremes, with the modification of a small number of parameters explaining most of the difference in performance between $\theta_{\mathsf{source}}$ and $\theta_{\mathsf{target}}$ (Figure 1(b)). Figures 1(c) and 1(d) show two additional cases that may occur when the source and target configurations have roughly equal performance on $I$. In 1(c), $\theta_{\mathsf{source}}$ and $\theta_{\mathsf{target}}$ are connected by a path of configurations that all have the same performance; this could arise in a situation where both lie on a large plateau of the response surface. However, it is also possible that the two configurations lie in separate basins of the response surface, such that a "saddle" of worse performance must be surmounted along the ablation path from $\theta_{\mathsf{source}}$ to $\theta_{\mathsf{target}}$, as illustrated in 1(d). We note that the results from all of our experiments




(a) parameters contribute equally (b) small number of parameters dominate

(c) region of equal performance (d) saddle of worse performance

Figure 1: Ablation paths can take several qualitatively different forms, illustrated by these (idealised) runtime examples. Lower values indicate better performance (for details, see text).

performing ablation from algorithm defaults to configurations obtained from automated configurators fall into case 1(b).

# 4 Experiment design

In order to empirically evaluate our ablation methods, we performed experiments on five scenarios using state-of-the-art solvers for SAT, MIP and AI planning. We implemented the brute-force and racing-based ablation methods as plugins for HAL, a Java-based platform for distributed experiment execution and data management [19]. All runs were performed using machines in the Compute-Calcul Canada / Westgrid Orcinus cluster, equipped with two Intel Xeon X5650 6-core 2.66Ghz processors, 12MB of cache and 24GB of RAM, running CentOS 5 Linux. For each target algorithm run, we used a single core and enforced a maximum of 2GB (SPEAR and CPLEX) or 6GB (LPG) of RAM.

Using the existing PARAMILS plugin for HAL 1.1.6, we performed 10 independent runs of PARAMILS for each scenario, with each configurator run allocated 48 CPU hours of total runtime. In each case, we minimised penalised average runtime (PAR10), a standard performance metric for configuration and empirical analysis; under PAR10, each run that terminates successfully is assigned a score equal to the CPU time used, and runs that crash or do not produce a valid solution on a given instance are assigned a score of 10 times the runtime cutoff (this heavily penalises these cases to attempt to enforce good instance set coverage.) Of the 10 configurations produced in our PARAMILS runs, we selected the one with the best PAR10 performance on the full training set for that scenario. (This corresponds to one of the standard protocols for using PARAMILS.)

We then performed two ablation experiments using our racing variant on the training set for each scenario, with the maximum number of rounds set to 200. Ablation was performed in two directions, first from the default to the optimised configuration obtained through automated configuration using maximum gain, and then from the optimised configuration to the default using minimum loss. Each configuration on the resulting ablation paths for each scenario was subsequently evaluated using the independent test set for that scenario. (Performing ablation directly on test sets produced very similar results.)

**SAT using SPEAR.** The propositional satisfiability problem, or SAT, is the prototypical $NP$-hard problem with important real-world application, including circuit design as well as hardware and software verifica-

tion. SAT has also been widely studied in the context of automated algorithm configuration [11, 14, 15]. We chose to analyze the industrial SAT solver SPEAR 1.2.1, winner of one category of the 2007 Satisfiability Modulo Theories Competition [9]; SPEAR has 26 configurable parameters, creating a space of $8.34 \times 10^{17}$ configurations. SPEAR has also been used in two recent investigations of parameter importance using forward selection [13] and functional ANOVA [12]. We analyzed the performance of SPEAR on the SWV software verification instance set used in several previous investigations. This set, consisting of 604 software verification conditions produced by an automated static checker, is partitioned into a training set (used for configuration and ablation analysis) and test set (used for evaluation of the ablation paths) consisting of 302 instances each. Following previous work, we used a 300 CPU-second runtime cutoff for automated configuration and all analysis runs.

**MIP using CPLEX.** Mixed integer programming (MIP) is another widely-studied problem with many prominent real-world applications. IBM ILOG CPLEX is one of the most widely used MIP solvers, both in academia and industry, and has a highly-parameterised configuration space containing 76 configurable parameters that directly impact solver performance (a total of $1.90 \times 10^{47}$ configurations). Automated configuration of CPLEX has proven successful in past work [10, 11], and CPLEX has also been used in the same parameter importance investigations as mentioned for SPEAR. We chose to use CPLEX 12.1 and the CORLAT instance set for this scenario [6]; CORLAT is a set of computational sustainability MIP instances based on real data used for wildlife corridor construction for grizzly bears in the Northern Rockies region. This set has been used both in previous work on algorithm configuration and on parameter importance, and is partitioned into a training and test set containing 1 000 instances each. A 300 CPU-second runtime cutoff was used for all runs.

**AI Planning using LPG.** The design and configuration of highly-parameterised solvers has recently proven successful in the AI planning community, contributing to both the winner and runner-up in the Learning Track of the 7th International Planning Competition (IPC-2011) [20]. Highly-parameterised general-purpose planners represent ideal candidate scenarios for studying parameter importance, because intuitively, the benefits to be gained by exploiting the structure and differences between various planning domains suggest that high-performance configurations will vary widely between such domains. We chose to investigate the configuration space of LPG td-1.0, a state-of-the-art local search based planner, and a key component in the winner of the IPC-2011 Learning Track. LPG has 66 configurable parameters, with a total of $9.11 \times 10^{36}$ possible configurations. We analyzed LPG's performance on three planning domains: depots, satellite, and zenotravel. These three domains have been used in previous planning competitions, as well as in previous work on automated configuration for planning. Each instance set contains disjoint 2 000-instance training and test sets generated using the same parameter settings of a randomised instance generator. Consistent with previous work, a 60 CPU-second runtime cutoff was used for configuration, while a 300 CPU-second cutoff was used for all test-set evaluation and ablation analysis runs.

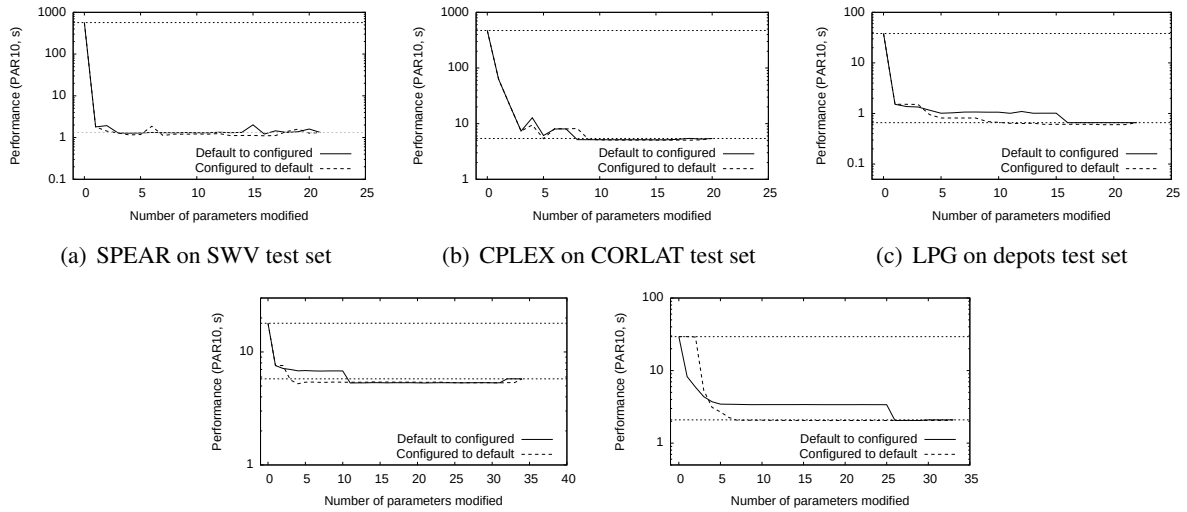
# 5 Results

Table 1 shows the training and test set performance for the default configurations and automatically optimised configurations in all five scenarios considered; as expected, and consistent with previously published results for these solvers, we observed 3- to 422-fold speedups after configuration. Interestingly, nearly every SPEAR parameter was changed from the default, while for the CPLEX and LPG scenarios, approximately one-third to one-half of the parameters were modified.

Figure 2(a) illustrates the mean PAR10 score on the SWV test set for every configuration along the path found through racing-accelerated ablation analysis of SPEAR on the SWV training set. Expressing the performance gain from a single ablation round as a percentage of the total gain between $\theta_{\mathsf{source}}$ and $\theta_{\mathsf{target}}$, 99.92% of the performance gain between the default configuration and the chosen configuration can be achieved by modifying the value of a single parameter, *sp-var-dec-heur*. This parameter controls the choice of variable decision heuristic in SPEAR, which is known to be an important parameter in most state-of-the-art SAT solvers. Furthermore, if we modify only four parameters (*sp-var-dec-heur*,

| | | Training set performance (PAR10, s) | | | | Test set performance (PAR10, s) | | | |
|---|---|---|---|---|---|---|---|---|---|
| solver | instance set | q25 | q50 | q75 | mean | q25 | q50 | q75 | mean |
| SPEAR default | SWV | 0.122 | 0.528 | 23.649 | 573.649 | 0.102 | 0.499 | 11.392 | 569.645 |
| SPEAR configured | SWV | 0.122 | 0.592 | 1.279 | 1.359 | 0.079 | 0.531 | 1.114 | 1.321 |
| CPLEX default | CORLAT | 0.101 | 3.563 | 90.596 | 556.531 | 0.097 | 3.551 | 70.602 | 471.722 |
| CPLEX configured | CORLAT | 0.110 | 1.220 | 5.812 | 5.511 | 0.112 | 1.238 | 5.650 | 5.411 |
| LPG default | depots | 0.551 | 1.086 | 8.182 | 43.245 | 0.535 | 1.055 | 7.194 | 38.097 |
| LPG configured | depots | 0.220 | 0.318 | 0.510 | 0.671 | 0.220 | 0.324 | 0.511 | 0.658 |
| LPG default | satellite | 15.232 | 17.580 | 20.595 | 17.962 | 15.173 | 17.575 | 20.514 | 17.940 |
| LPG configured | satellite | 4.827 | 5.645 | 6.404 | 5.662 | 4.943 | 5.760 | 6.529 | 5.783 |
| LPG default | zenotravel | 20.092 | 26.377 | 34.642 | 29.671 | 19.792 | 26.026 | 34.929 | 29.361 |
| LPG configured | zenotravel | 1.414 | 1.826 | 2.490 | 2.065 | 1.407 | 1.841 | 2.556 | 2.092 |

Table 1: Training and test set performance results for all 5 of our scenarios, for both the default configurations and those selected by PARAMILS. Runtime cutoffs in all cases were 300 CPU seconds.




(a) SPEAR on SWV test set (b) CPLEX on CORLAT test set (c) LPG on depots test set

(d) LPG on satellite test set (e) LPG on zenotravel test set

Figure 2: Ablation paths determined using racing with up to 200 rounds on the training sets of the five configuration scenarios, with each configuration on the ablation path evaluated using the corresponding test set. The horizontal lines indicate the PAR10 scores of the default (source) and automatically-configured (target) configurations on the test set for each scenario.

*sp-rand-var-dec-scaling*, *sp-res-cutoff-cls*, and *sp-first-restart*) from their default values, we obtain a configuration with slightly *better* performance on the test set than the target configuration obtained with PARAMILS. In contrast, Hutter *et al.* noted in their functional ANOVA work [12] that *sp-var-dec-heur* was important, but only 83% of the improvement over the default could be attributed to single-parameter effects in their model. We hypothesize that *sp-var-dec-heur* is much more important in high-performance parts of the SPEAR configuration space, a bias that is not taken into account by the Hutter *et al.* models.

Similarly, Figure 2(b) shows the performance of configurations on the path found through racing-accelerated ablation analysis of CPLEX on the CORLAT training set, evaluated on the test set. Here, 87.64% of the performance gain resulted from modifying the value of a single parameter, *mip_cuts_covers*, which controls whether or not to generate cover cuts. 99.58% of the gain can be achieved by modifying just three CPLEX parameters (*mip_cuts_covers*, *mip_strategy_heuristicfreq* and *simplex_dgradient*). We note that *simplex_dgradient* was not in the top 10 important parameters found in the CPLEX CORLAT scenario in [12], although 6 of the 10 most important CPLEX parameters as identified in that work were not changed from their default values in our experiments (this effect was also noted by Hutter *et al.*).

Finally, Figures 2(c), 2(d) and 2(e) illustrate the performance along the ablation paths for each of the three LPG scenarios: depots, satellite and zenotravel. For the depots and satellite scenarios, the top three parameters were the same. Modifying the value of *cri_intermediate_levels* resulted in 97.7% and 84.55%

of the target configuration performance over the default, respectively. Furthermore, modifying the values of three parameters (*cri_intermediate_levels*, *vicinato* (the neighbourhood choice), and *hpar_cut_neighb*) resulted in 99.22% of the performance gain for the depots scenario.

For the zenotravel scenario (Figure 2e), we observed different choices for the two most important parameters, depending on the direction in which ablation was performed. Modifying *triomemory* and *fast_best_action_evaluation* from their default values resulted in 85.99% of the overall performance gain over the default, while modifying *vicinato* and *hpar_cut_neighb* (similar to the other two LPG scenarios) resulted in 88.09% of the total performance gain. Four parameter modifications (*vicinato*, *hpar_cut_neighb*, *triomemory*, and *noise*) accounted for 97.8% of the total performance gain.

It is interesting to note that there is a conditional parameter interaction between *hpar_cut_neighb* and *vicinato*, as *hpar_cut_neighb* is only active when *vicinato* takes certain values. In the results, modifying *vicinato* often does not produce large gains in performance by itself, but allows for modification of *hpar_cut_neighb*, which in turn results in large performance improvements. The effect of conditional parameters can also be seen in the "late" performance improvements in the three LPG scenarios in the direction of maximum gain.

Due to space restrictions, we have included the full set of experimental data in an online appendix [2].

## 6 Conclusions and future work

In this work, we have introduced a new procedure, ablation analysis, which allows developers of highly-parameterised algorithms to ascertain which of their parameters contribute most to performance differences between two algorithm configurations. Using ablation analysis, it is possible to determine which modifications of a given default configuration were truly necessary to achieve improved performance, and which modifications can essentially be considered spurious side effects of an automated (or manual) configuration process.

We validated our approach in an experimental study using five well-studied configuration scenarios from propositional satisfiability, mixed-integer programming and AI planning, with 26 to 76 configurable parameters. We showed that a variant of our approach accelerated by a racing method required 25% of the CPU time needed by the brute-force variant, while achieving qualitatively similar results. In all of these scenarios, we found that 95–99% of the performance improvements achieved by automated configuration of the given, highly-parametric solver could be obtained with the modification of only 1–4 parameters, a small fraction of total number of parameters for each algorithm. In two cases, we found that modification of a single parameter could achieve 99.92% and 87.64% of the performance gain between the default configuration and one found by PARAMILS. Similar results have been reported for the global impact of parameters previously, but we show that this is true locally for high-performance configurations, and in some cases the locally-important parameters are different from those that are important globally.

This work can be extended in various directions, for example improved handling of conditional effects and interdependencies between parameters. We believe that ablation analysis can also be used for slightly different purposes – to investigate performance generalisation near a given configuration or to obtain information regarding the region of configuration space between several configurations of similarly high-performance. Furthermore, we believe that our approach and the model-based techniques discussed in Section 2 are complementary and can be combined, *e.g.*, by building functional ANOVA models using configurations sampled along ablation paths or from the localised region between the two input configurations.

**Acknowledgements.** The authors would like to thank Frank Hutter for providing us with the SPEAR and CPLEX scenario data, as well as for useful feedback on an earlier draft of this work. We also thank Compute Canada for providing the computing resources for our experiments and acknowledge funding provided by NSERC to HH under the Discovery Grant and Discovery Accelerator Supplement Programs. This work was supported in part by the Institute for Computing, Information and Cognitive Systems (ICICS) at UBC.

[2] http://www.cs.ubc.ca/labs/beta/Projects/Ablation/mic2013/appendix.pdf

## References


[1] Carlos Ansótegui, Meinolf Sellmann, and Kevin Tierney. A gender-based genetic algorithm for the automatic configuration of algorithms. In *Proceedings of CP 2009*, pages 142–157, 2009.

[2] Thomas Bartz-Beielstein. *Experimental Research in Evolutionary Computation—The New Experimentalism*. Natural Computing Series. Springer, Berlin, Heidelberg, New York, 2006.

[3] Mauro Birattari, Thomas Stützle, Luis Paquete, and Klaus Varrentrapp. A racing algorithm for configuring metaheuristics. In *Proceedings of GECCO'02*, pages 11–18, 2002.

[4] Marco Chiarandini and Yuri Goegebeur. Mixed models for the analysis of optimization algorithms. *Experimental Methods for the Analysis of Optimization Algorithms*, pages 225–264, 2010.

[5] W.J. Conover. *Practical Nonparametric Statistics*. John Wiley and Sons, New York, NY, USA, 1999. Third edition.

[6] Carla P. Gomes, Willem-Jan van Hoeve, and Ashish Sabharwal. Connections in networks: a hybrid approach. In *Proceedings of CPAIOR 2008*, pages 303 – 307, 2008.

[7] Giles Hooker. Generalized functional ANOVA diagnostics for high dimensional functions of dependent variables. *Journal of Computational and Graphical Statistics*, 16:709–732, 2007.

[8] Holger H. Hoos. Programming by optimization. *Communications of the ACM*, 55(2):70–80, February 2012.

[9] Frank Hutter, Domagoj Babić, Holger H. Hoos, and Alan J. Hu. Boosting verification by automatic tuning of decision procedures. In *Formal Methods in Computer-Aided Design*, pages 27–34, 2007.

[10] Frank Hutter, Holger H. Hoos, and Kevin Leyton-Brown. Automated configuration of mixed integer programming solvers. In *Proceedings of CPAIOR 2010*, pages 186–202, 2010.

[11] Frank Hutter, Holger H. Hoos, and Kevin Leyton-Brown. Sequential model-based optimization for general algorithm configuration. In *Proceedings of LION-5*, pages 507–523, 2011.

[12] Frank Hutter, Holger H. Hoos, and Kevin Leyton-Brown. An efficient approach for assessing algorithm parameter importance. 2013. Under review.

[13] Frank Hutter, Holger H. Hoos, and Kevin Leyton-Brown. Identifying key algorithm parameters and instance features using forward selection. In *Proceedings of LION-7*, 2013. To appear.

[14] Frank Hutter, Holger H. Hoos, Kevin Leyton-Brown, and Thomas Stützle. ParamILS: An automatic algorithm configuration framework. *Journal of Artificial Intelligence Research*, 36:267–306, 2009.

[15] Frank Hutter, Holger H. Hoos, and Thomas Stützle. Automatic algorithm configuration based on local search. In *AAAI 07*, pages 1152–1157, 2007.

[16] Manuel López-Ibáñez, Jérémie Dubois-Lacoste, Thomas Stützle, and Mauro Birattari. The irace package, iterated race for automatic algorithm configuration. Technical Report TR/IRIDIA/2011-004, IRIDIA, Université Libre de Bruxelles, Belgium, 2011.

[17] Oded Maron and Andrew Moore. Hoeffding races: Accelerating model selection search for classification and function approximation. In *Advances in Neural Information Processing Systems*, volume 6, pages 59–66, 1994.

[18] Volker Nannen and A.E. Eiben. Relevance estimation and value calibration of evolutionary algorithm parameters. In *Proceedings of IJCAI 2007*, pages 975–980, 2007.

[19] Christopher Nell, Chris Fawcett, Holger H. Hoos, and Kevin Leyton-Brown. HAL: A framework for the automated analysis and design of high-performance algorithms. In *Proceedings of LION-5*, pages 600 – 615, 2011.

[20] Mauro Vallati, Chris Fawcett, Alfonso E. Gerevini, Holger H. Hoos, and Alessandro Saetti. Automatic generation of efficient domain-optimized planners from generic parametrized planners. In *Proceedings of IJCAI RCRA Workshop 2011*, pages 111–123, 2011.