J Heuristics (2016) 22:431–458
DOI 10.1007/s10732-014-9275-9

Analysing differences between algorithm configurations through ablation

Chris Fawcett · Holger H. Hoos

Received: 20 January 2014 / Revised: 21 November 2014 / Accepted: 10 December 2014 /
Published online: 3 January 2015

Abstract Developers of high-performance algorithms for hard computational problems increasingly take advantage of automated parameter tuning and 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 optimised configurations obtained from automated configuration tools can be explained by modifying the values of a small number of parameters (1–4 in the scenarios we studied). We also investigate the use of our ablation analysis procedure for producing configurations that generalise well to previously-unseen problem domains, as well as for analysing the structure of the algorithm parameter response surface near and between high-performance configurations.

Electronic supplementary material The online version of this article (doi:10.1007/s10732-014-9275-9) contains supplementary material, which is available to authorized users.

C. Fawcett (✉) · H. H. Hoos
Computer Science Department, University of British Columbia,
201-2366 Main Mall, Vancouver, BC V6T 1Z4, Canada
e-mail: fawcettc@cs.ubc.ca

H. H. Hoos
e-mail: hoos@cs.ubc.ca

Keywords Ablation analysis · Parameter importance · Automated algorithm configuration · Empirical analysis

1 Introduction

High-performance solvers for hard computational problems, such as propositional satisfiability (SAT) or mixed-integer programming (MIP), are typically run by users on classes of problem instances from different application domains; for example, a SAT solver might be run on sets of random 3-SAT, hardware verification or software verification instances, and a MIP solver might be used to solve MIP-formulations of logistics, routing or production planning problems. The existence of such varied domains provides an incentive to the developers of such solvers to parameterise aspects of their implementation in order to be able to obtain good performance 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., Birattari et al. (2002)), the introduction of advanced procedures capable of dealing with dozens of parameters, such as ParamILS (Hutter et al. 2007b, 2009), GGA (Ansótegui et al. 2009), irace (López-Ibáñez et al. 2011) and SMAC (Hutter et al. 2011), 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) (Hoos 2012), 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 perfor-

mance is retained[1]. 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 SAT, MIP 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. This observation has the potential to improve the design of automated configuration approaches, by searching intelligently in the local region around an algorithm's default configuration rather than rapidly (and randomly) diversifying to other parts of the configuration space.

We also present the results of several additional experiments using our ablation analysis approach to produce algorithm configurations that generalise well to unseen problem instances from different domains. Finally, we carried out experiments to analyse the structure of the algorithm parameter response surface near and between configurations with high performance, by performing ablation analysis with these high-performance configurations as the source and target configurations.

The remainder of this paper is structured as follows: In Sect. 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 Sect. 3. Section 4 presents the details of the experimental study that we performed, with the results of that study shown and discussed in Sects. 5 and 6. We discuss opportunities for future work in this area in Sect. 8 and then conclude in Sect. 9.

2 Background and related work

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 (Hutter et al. 2007a), as well as the automated design of general-purpose frameworks for AI planning (Vallati et al. 2011, 2013). However, there has been relatively little work on systematic techniques for assessing 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 (Hooker 2007) approaches to decomposing model or function response variance into low-order components. Furthermore, related work on interactive parameter exploration using contour plot visualization (Bartz-Beielstein 2006), on evolutionary algorithms for parameter relevance estimation (Nannen and Eiben 2007) and on experimental design for analysing optimization algorithms (Chiarandini and Goegebeur 2010) can be found in the literature.

[1] Our use of the term *ablation* follows that of Aghaeepour and Hoos (2013) and loosely echoes its meaning in medicine, where it refers to the surgical removal of organs, organ parts or tissues. We ablate (i.e., remove) changes in the settings of algorithm parameters to better understand the contribution of those changes to observed differences in algorithm performance.

Chiarandini and Goegebeur (2010) present a thorough investigation of experimental design for analysing optimization algorithms, using linear mixed-effects models. Their analysis only includes parameter configuration spaces of low size and dimension, with fewer than 50 possible configurations. It is unclear whether this approach scales to larger configuration spaces (the scenarios considered in our study comprise up to 1.90×10^{47} possible configurations). Nannen and Eiben (2007) uses an entropy-based approach with evolutionary algorithms to perform parameter relevance estimation, but this approach also makes a smoothness assumption about the parameter response surface which eliminate categorical parameters from consideration, as well as an assumption that there will be few parameters (on the order of 10) in the problem.

Gunawan et al. (2011) use a design of experiments (DOE) approach to automated algorithm configuration, a component of which is an initial factorial experiment design which attempts to rank parameters by importance in order to screen out "unimportant" parameters before performing configuration on the reduced configuration space. For scenarios with k parameters and n instances, this initial 2^k experiment design requires $n \cdot 2^k$ observations, which is unfortunately infeasible for algorithms with large configuration spaces (involving often more than 50 parameters). Finally, Bartz-Beielstein (2006) uses sequential parameter optimization (SPO) to estimate 1- and 2-parameter effects, as well as providing interactive contour plots in the configuration space. This approach also has difficulties with the discrete configuration spaces induced by the categorical parameters encountered in many algorithm configuration scenarios. As these techniques each have difficulties with the high dimensionality and/or the discrete nature of the configuration spaces of typical highly-parameterised algorithms, an approach overcoming both has high utility for algorithm developers.

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 (Hutter et al. 2013) and functional ANOVA (Hutter et al. 2014). Both approaches require an initial data-gathering step to obtain algorithm performance data, which is then partitioned into training and test sets. In (Hutter et al. 2013), this data was obtained by sampling 1,000 –10,000 pairs of configurations and instances uniformly at random, while in (Hutter et al. 2014) several experiments were performed using 10,000 randomly sampled runs, the algorithm runs performed during executions of the SMAC configurator (13,452–454,336 additional runs), as well as a combination of both.

In the forward selection approach by Hutter et al. (2013), 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. Hutter et al. (2014), on the other hand, introduced 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.

The current version of the forward selection approach constructs 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 or, for functional ANOVA, to broad sets of samples restricted to: (1) configurations with better performance than the default configuration, or (2) configurations in the top 25 % in terms of performance. 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. (2014), 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 aim to explain 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 Sect. 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 see this approach as complementary to the recent model-based techniques of Hutter et al. (2013, 2014), 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 Sect. 9).

There is an interesting conceptual connection between our ablation approach and that of *path relinking*, a general-purpose method for combining the diversification and intensification stages in heuristic search (Glover 1994, 1997; Glover and Laguna 1997). Search strategies based on path relinking typically maintain a population of solutions and create new candidate solutions by constructing paths in the search space between existing solutions in the population. Each point in the search space lying on these paths is a potential solution, often with properties similar to the start or end points of the path. For further information on the history and application of path relinking, we refer the interested reader to two surveys by Glover et al. (2000, 2003).

Our ablation analysis approach can be seen as an application of path relinking in the configuration space of a given parameterised algorithm, as we are constructing specific paths between a source and target configuration (as such, to the best of our

Algorithm 1: ablationAnalysis $(\mathcal{A}, \theta_{\text{source}}, \theta_{\text{target}}, I, m)$

Input: Parameterised algorithm $\mathcal{A}$, two parameter configurations of $\mathcal{A}$, θ_{source} and θ_{target}, benchmark instance set I, performance metric m (to be minimised)
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. $\theta_0 = \theta_{\text{source}}$ and $\theta_l = \theta_{\text{target}}$

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

knowledge, we are describing the first application of ideas from path relinking in the context of algorithm configuration). However, ablation analysis pursues a different goal from path relinking: While the latter is used to identify candidate configurations during search, the former aims to identify individual parameter settings responsible for the performance of a given target algorithm. As we demonstrate in Sect. 6, ablation analysis can also be used to find configurations that generalise well to different types of problem instances solvable by a given target algorithm, and this application of ablation analysis is even more closely related to path relinking.

3 Ablation analysis

Given a parameterised algorithm $\mathcal{A}$ with d parameters and configuration space Θ, along with a source and target configuration ($\theta_{\text{source}}, \theta_{\text{target}} \in \Theta$) of that algorithm; a set of training benchmark instances I, ideally sampled from a distribution as close as possible to the typical instances that $\mathcal{A}$ is executed on in practice; and a performance metric m (e.g., penalised average runtime or mean solution quality, although any scalar measure of performance is supported), our ablation analysis procedure proceeds as follows. We first compute the set of parameters whose values differ between θ_{source} and θ_{target}. Then, beginning from θ_{source}, we proceed through a series of rounds: in each round, we use a subprocedure determineBest to choose a configuration from the set of all configurations obtained by flipping one parameter in the current configuration to its value in θ_{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 determineBest $(\mathcal{A}', I, m)$ 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 determineBest $(\mathcal{A}', I, m)$ will be the one with *minimum loss* compared to the con-

figuration θ from the previous round. Conversely, when θ_{target} has better performance than θ_{source} on I, the configuration selected by determineBest will be that with *maximum gain* over the previous θ. 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. This means that modification of conditional parameters may be delayed until late in the ablation process if the corresponding parent parameters have little effect on algorithm performance. Similar delayed effects can occur in the presence of other complex interactions between subsets of parameters, for example if parameters $p_1, p_2, \ldots, p_k$ must all be modified before performance will improve. However, the presence of these interactions will often be identifiable after performing ablation analysis, as a section of "flat" performance followed by a significant performance improvement. We discuss these issues further in Sect. 5 and explore a potential solution in Sect. 7.

In the experiments we present in Sect. 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 θ_{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 θ_{target}. As a greedy approach (not unlike forward selection), ablation in either direction may produce suboptimal results at any distance except 1 from θ_{source}. In light of this, performing the analysis in two directions provides additional robustness. In the following, we describe two variants of determineBest $(\mathcal{A}', I, m)$: a naïve brute-force method (Algorithm 2), which is easy to implement but slow, and a greatly accelerated version based on a racing method (Algorithm 3).

3.1 Brute-force ablation

Our brute-force implementation of determineBest $(\mathcal{A}', I, m)$, detailed in Algorithm 2, 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 performance as determined by the metric m. These scalar performance values are then suitably aggregated; in our experiments we use the mean, but the median and other statistics are also valid and supported choices. We keep track of the configuration $\theta' \in \mathcal{A}'$ with the best measured performance, and return it as the result of determineBestBruteForce.

Given that one parameter is eliminated from consideration in every round, ablation on instance set I with p differing parameters between θ_{source} and θ_{target} using this brute-force approach will require up to $|I| \cdot p \cdot (p+1)/2$ individual runs of algorithm $\mathcal{A}$. (In the presence of conditional parameters, more than one parameter can be eliminated in one round of ablation if an inactive parameter is set to its default value in the target configuration.) Therefore, this procedure can be extremely time-consuming in

Algorithm 2: determineBestBruteForce $(\mathcal{A}', I, m)$

Input: The set of candidate algorithm configurations $\mathcal{A}'$, benchmark instance set I, performance metric m (to be minimised)
Output: $\theta' \in \mathcal{A}'$, the configuration in $\mathcal{A}'$ deemed to have the best performance

```
θ' ← nil
bestPerformance ← ∞
for θ ∈ A' do
    runData ← ∅
    for i ∈ I do
        runData ← runData ∪ evaluateConfiguration (θ, i, m)
    end
    performance ← aggregateRunData (runData)
    if performance < bestPerformance then
        θ' ← θ
        bestPerformance ← performance
    end
end
return θ'
```

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 s 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 while, 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 (Maron and Moore 1994), F-Race is a prominent racing method for algorithm selection (Birattari et al. 2002). 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.

As outlined in Algorithm 3, we apply F-Race to determine the best configurations in each round of ablation analysis, adhering very closely to the statistical framework described by Birattari et al. (2002). 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 candidateConfigurations $= \{\theta_1, \theta_2, \ldots, \theta_n\}$ are evaluated on a new instance $i_k \in I$,

Algorithm 3: determineBestRacing $(\mathcal{A}', I, m)$

Input: The set of candidate algorithm configurations $\mathcal{A}'$, benchmark instance set I, performance metric m (to be minimised)
Parameters: MIN_STAGES and MAX_STAGES, configurable constants containing the minimum and maximum number of racing stages
Output: $\theta' \in \mathcal{A}'$, the configuration in $\mathcal{A}'$ deemed to have the best performance

```
MAX_STAGES ← min (MAX_STAGES, |I|)
candidateConfigurations ← A'
instanceList ← (i | i ∈ I)
results ← [ ]
stage ← 1
while |candidateConfigurations| > 1 and stage ≤ MAX_STAGES do
    i ← instanceList [stage]
    results [stage] ← [ ]
    for θ ∈ candidateConfigurations do
        results [stage] [θ] ← evaluateConfiguration (θ, i, m)
    end
    if stage ≥ MIN_STAGES then
        (F, p, ranks, sumsOfSquaredRanks) ← FriedmanTest (results)
        /* F is the Friedman test statistic, p the test p-value, ranks
           contains the ranks of configurations in each block, and
           sumsOfSquaredRanks is sum of squared ranks for each
           θ ∈ candidateConfigurations                                  */

        if p < 0.05 then
            bestRankSum ← min_θ sumsOfSquaredRanks [θ]
            θ_best ← arg min_θ sumsOfSquaredRanks [θ]
            toEliminate ← ∅
            for θ ∈ candidateConfigurations \ {θ_best} do
                if modifiedTTest (θ, ranks, bestRankSum) then
                    toEliminate ← toEliminate ∪ {θ}
                end
            end
            candidateConfigurations ← candidateConfigurations \ toEliminate
        end
    end
    stage ← stage + 1
end
θ' ← nil
bestPerformance ← ∞
for θ ∈ candidateConfigurations do
    runData ← ∅
    for s = 1 ... stage do
        runData ← runData ∪ results [s] [θ]
    end
    performance ← aggregateRunData (runData)
    if performance < bestPerformance then
        θ' ← θ
        bestPerformance ← performance
    end
end
return θ'
```

and the results are then combined with the results of the previous stages for each configuration. These results are organised into k *blocks*, with the j^{th} block containing the n performance metric values resulting from running the configurations in candidateConfigurations on instance i_j.

On these blocks, a Friedman two-way analysis of variance by ranks, also known as the Friedman test, is performed (Conover 1999). In order to prevent early rejection of configurations, this test is only performed after a given number of stages (MIN_STAGES) have been performed. We follow the convention of the F-Race authours and set this parameter to five stages. If the null hypothesis of this test is rejected, we can conclude that at least one configuration in candidateConfigurations 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 candidateConfigurations. 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 candidateConfigurations using a modified t-test with $n - 1$ degrees of freedom (Birattari et al. 2002). After culling any configurations deemed to be statistically significantly worse, we proceed to the next stage of the race. The race terminates when only one configuration remains, or when a specified maximum number of stages have been performed (MAX_STAGES). In the latter case, the configuration with the best aggregate performance (according to m) across all stages is selected as the winner. (Further details on the statistical tests used in this procedure can be found in Birattari et al. (2002); Conover (1999).)

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 Sect. 4. Using our SPEAR and CPLEX scenarios, we performed ablation analysis from the solver default to the configuration obtained from PARAMILS using three ablation analysis approaches. The brute-force approach required approximately 5 CPU days for the SPEAR scenario, and 32 CPU days for the CPLEXscenario. Using our racing approach with MAX_STAGES set to the size of the benchmark set I reduced these requirements to just over 1 CPU day for SPEAR, and 10.7 CPU days for CPLEX. By limiting MAX_STAGES to to 200 (rather than $|I| = 302$ for SPEAR and 1,000 for CPLEX), we achieved further reductions to 4.6 CPU days for CPLEX. We also note that every algorithm run inside a single stage of our racing approach can be performed in parallel, resulting in further reductions in the wall-clock time required. Our implementation approach can take advantage of the now-ubiquitous multiple cores on a single machine, and can also be distributed across a compute cluster for even greater wall-clock performance. The result is a reasonable computing expense on modern hardware for typical configuration scenarios. Table 1 gives a summary of these performance gains.

Determining the optimal value for the MAX_STAGES parameter is not straightforward, but in general it can be expected to depend on the number of instances in the given set of benchmark instances for which algorithm performance is highly correlated. We determined our conservative choice of 200 by 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.

Table 1 Results from performing ablation analysis on two scenarios with three ablation variants: the brute force approach, racing with the maximum number of rounds equal to the number of instances, and racing with the maximum number of rounds equal to 200

Scenario	SPEAR SWV		CPLEX CORLAT	
	Runs	CPU time (s)	Runs	CPU time (s)
Brute force	70,366	472,686.17	212,000	2,801,674.25
Racing (max #rounds = $\|I\|$)	43,982	107,418.36	115,665	921,487.05
Racing (max #rounds = 200)	28,544	108,036.14	51,349	399,092.70

The SPEAR and CPLEX scenarios themselves are identical to those described in detail in Sect. 4. For both racing rows, we note that a full empirical analysis was performed on the chosen benchmark instances after each racing round in order to calculate the performance of every configuration on the ablation path. Run counts and CPU times for racing alone are approximately half of the reported values

While it is possible that racing will return different ablation results 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. We additionally tested how close our racing approach comes to the optimal parameter choice, by performing complete evaluations of the 1-neighbourhood of configurations around the default for each of our five scenarios. These neighbourhoods consist of those configurations obtained by modifying a single parameter from its value in the source configuration to its value in the target configuration. In all cases, the most important parameter selected by our racing approach for each scenario is the same as the best parameter in the 1-neighbourhood around the source configuration. The results for the LPG solver on our depots scenario are given in Table 2.

3.3 Ablation paths

We call the path of configurations $(\theta_0, \ldots, \theta_l)$ obtained between θ_{source} and θ_{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 θ_{source} and θ_{target}, and on 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 θ_i, from θ_{source} on the left-hand side to θ_{target} on the right. Figure 1a illustrates one extreme case, where θ_{source} and θ_{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 θ_{source} and θ_{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 θ_{source} and θ_{target} (Fig. 1b). Fig. 1c, d show two additional cases that may occur when the source and target configurations have roughly equal performance on I. In 1c,

Table 2 Depots training and test set performance obtained when evaluating every parameter configuration in the 1-neighbourhood of the LPG default configuration

Parameter modified	Training set performance	Test set performance
default	30.499	22.140
cri_intermediate_levels	1.548	1.574
walkplan	14.923	15.633
triomemory	29.124	26.216
tabu_fct	29.225	23.607
tabu_length	29.293	24.958
donot_try_suspected_actions	30.497	24.798
dynoiseTabLen	30.587	24.937
dynoisecoefnum	30.622	23.588
tabu_act	30.624	24.959
no_cut	30.642	26.323
vicinato	31.964	22.238
maxnoise	33.387	24.963
inc_re	38.475	31.776
numrestart	39.073	29.187
static_noise	45.005	39.559
extended_effects_evaluation	49.412	51.042
numtry	70.004	51.778
weight_mutex_in_relaxed_plan	71.346	61.178
noise	132.844	121.984
noise_incr	175.963	148.953
ri_list	402.606	371.674
depots configured	0.642	0.637

θ_{source} and θ_{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 θ_{source} to θ_{target}, as illustrated in 1d. [2]. We note that the results from all of our experiments performing ablation from algorithm defaults to configurations obtained from automated configurators fall into case 1b.

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,

[2] The ablation path illustrated in Fig. 1d resembles the folding pathways observed in biopolymers like RNA that have to overcome a thermodynamic barrier in order to change from one low-energy structure to another (Reidys 2011).

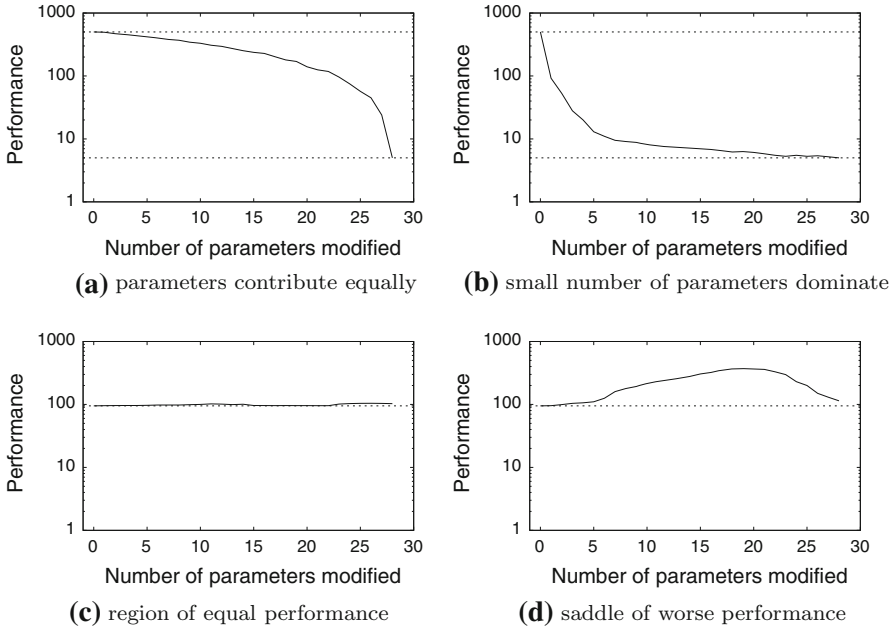


Fig. 1 Ablation paths can take several qualitatively different forms, illustrated by these (idealised) runtime examples. *Lower values* indicate better performance (for details, see text)

a Java-based platform for distributed experiment execution and data management (Nell et al. 2011). All runs were performed using machines in the Compute-Calcul Canada / Westgrid Orcinus cluster, each equipped with two Intel Xeon X5650 6-core 2.66 Ghz processors, 12 MB of cache and 24 GB 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.

Following standard machine learning practice, each of our benchmark instance sets was partitioned (by the creators of those sets) into mutually exclusive training and test sets. This partition is typically achieved through the use of a randomized instance generator to sample a desired instance distribution, uniformly at random, to create a set of unique instances which are subsequently divided into the training and test set (again uniformly at random, or using stratified sampling in the case of multiple generator settings).

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 was assigned a score equal to the CPU time used, and runs that crash or do not produce a valid solution on a given instance were assigned a score of 10 times the runtime cutoff (this heavily penalised 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-based ablation procedure 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 SAT problem, or SAT, is the prototypical NP-hard problem with important real-world application, including circuit design as well as hardware and software verification. SAT has also been widely studied in the context of automated algorithm configuration Hutter et al. (2007b, 2009, 2011). We chose to analyze the industrial SAT solver SPEAR 1.2.1, winner of one category of the 2007 Satisfiability Modulo Theories Competition (Hutter et al. 2007a); SPEAR has 26 configurable parameters, creating a space of 8.34×10^{17} configurations. SPEAR has also been used in two recent investigations of parameter importance using forward selection (Hutter et al. 2013) and functional ANOVA (Hutter et al. 2014). 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. 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 1.90×10^{47} configurations). Automated configuration of CPLEX has proven successful in past work (Hutter et al. 2010, 2011), 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 (Gomes et al. 2008); 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 in previous work on algorithm configuration and on parameter importance; it 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 automated 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) (Vallati et al. 2011, 2013). 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 9.11×10^{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 Ablation between default and optimized configurations

Tables 3 and 4 show the training and test set performance, respectively, 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.

Table 5 details the design space size for each of the three solvers used in our empirical analysis, as well as the number of parameters that were changed from the default in each of our five scenarios. 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 details the full ablation paths using up to 200 rounds of racing, for the SPEAR, CPLEX and LPG scenarios. Table 6 contains the training and test set performance of CPLEX at each point of the ablation paths in Fig. 2b, along with [10, 90 %] bootstrap confidence intervals for PAR10 score on the test set. Corresponding tables for our other four scenarios are given in Online Resource 1. In all cases, we present the ablation

Table 3 Training set performance results for all 5 of our scenarios, for both the default configurations and those obtained from PARAMILS

Solver	Instance set	Training set performance (PAR10, s)			
		q25	q50	q75	Mean
SPEAR default	SWV	0.122	0.528	23.649	573.649
SPEAR configured	SWV	0.122	0.592	1.279	1.359
CPLEX default	CORLAT	0.101	3.563	90.596	556.531
CPLEX configured	CORLAT	0.110	1.220	5.812	5.511
LPG default	depots	0.551	1.086	8.182	43.245
LPG configured	depots	0.220	0.318	0.510	0.671
LPG default	satellite	15.232	17.580	20.595	17.962
LPG configured	satellite	4.827	5.645	6.404	5.662
LPG default	zenotravel	20.092	26.377	34.642	29.671
LPG configured	zenotravel	1.414	1.826	2.490	2.065

Runtime cutoffs in all cases were 300 CPU seconds

Table 4 Test set performance results for all 5 of our scenarios, for both the default configurations and those obtained from PARAMILS

Solver	Instance set	Test set performance (PAR10, s)			
		q25	q50	q75	Mean
SPEAR default	SWV	0.102	0.499	11.392	569.645
SPEAR configured	SWV	0.079	0.531	1.114	1.321
CPLEX default	CORLAT	0.097	3.551	70.602	471.722
CPLEX configured	CORLAT	0.112	1.238	5.650	5.411
LPG default	depots	0.535	1.055	7.194	38.097
LPG configured	depots	0.220	0.324	0.511	0.658
LPG default	satellite	15.173	17.575	20.514	17.940
LPG configured	satellite	4.943	5.760	6.529	5.783
LPG default	zenotravel	19.792	26.026	34.929	29.361
LPG configured	zenotravel	1.407	1.841	2.556	2.092

Runtime cutoffs in all cases were 300 CPU seconds

Table 5 Design space size and number of parameters changed from the default configuration by PARAMILS for all 5 scenarios

Solver	Instance set	Design space size	# parameters	# changed from default
SPEAR	SWV	8.34×10^{17}	26	21
CPLEX	CORLAT	1.90×10^{47}	76	20
LPG	depots	9.11×10^{36}	66	22
LPG	satellite	9.11×10^{36}	66	35
LPG	zenotravel	9.11×10^{36}	66	33

Note that the number of parameters changed from the default represents an upper bound on the number of ablation rounds, as conditional parameters can cause fewer rounds to be required if they are inactive in the source configuration and active with their default values in the target configuration

paths using the default configuration as the source and the PARAMILS configuration as the target. The ablation paths in the reverse direction are qualitatively similar, with small deviations due to parameter conditionality in the case of our CPLEX and LPG scenarios.

Figure 2a 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 θ_{source} and θ_{target}, 99.92 % of the performance gain between the default configuration and the optimised configuration can be achieved by modifying the value of a single parameter, *sp-var-dec-heur*. This parameter controls the choice of the variable decision heuristic used by 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*, *sp-rand-var-dec-scaling*, *sp-res-cutoff-cls*, and *sp-first-restart*) from their default values, we obtain a configuration

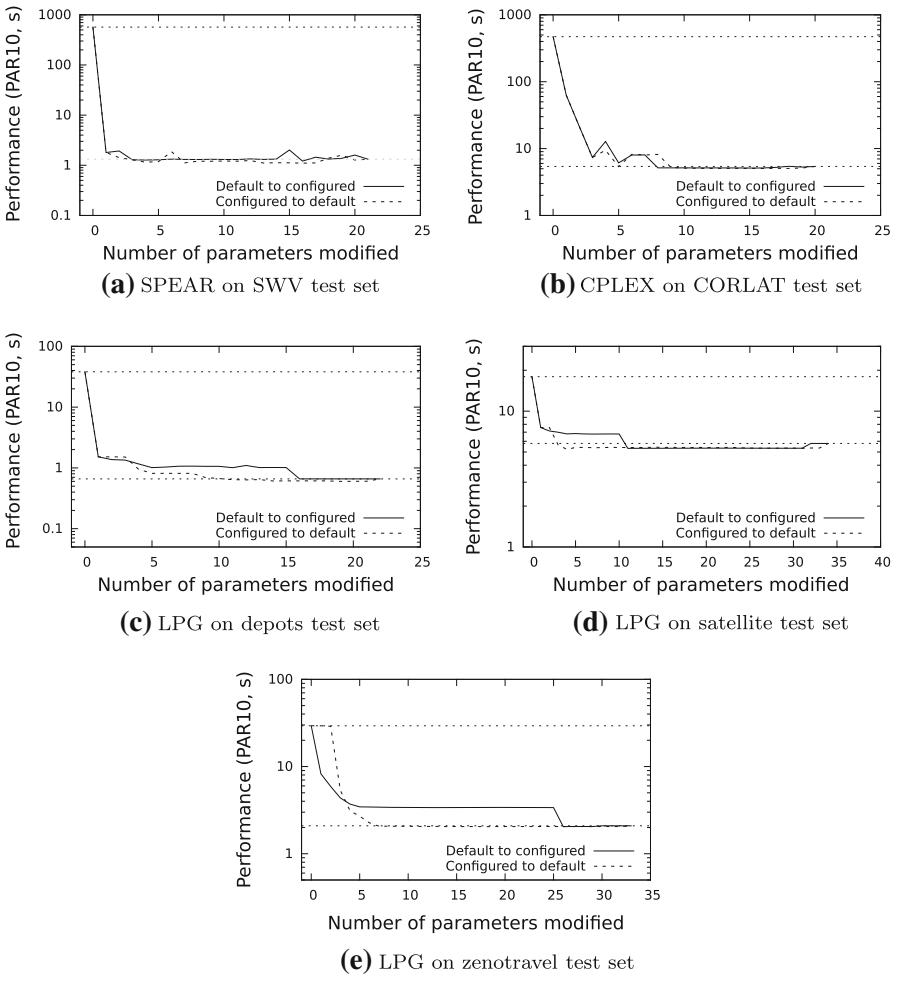


Fig. 2 Ablation paths determined using racing with up to 200 rounds on the training sets of the five configuration scenarios, with each configuration on the ablation path evaluated using the corresponding test set. The *horizontal lines* indicate the PAR10 scores of the default (source) and automatically-configured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

with slightly *better* performance on the test set than the target configuration obtained with ParamILS. In contrast, Hutter et al. (2014) noted in their functional ANOVA work that *sp-var-dec-heur* was important, but only 76 % 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 empirical performance models used by Hutter et al.

Similarly, Fig. 2b 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

Table 6 Parameters chosen and resulting PAR10 performance on both the training and test sets for ablation analysis accelerated by racing (maximum of 200 rounds) for CPLEX on the CORLAT training set

Round	Parameter	Performance		Test bootstrap CI
		Training	Test	[10, 90 %]
default	-	556.531	471.722	[429.845, 515.609]
1	mip_cuts_covers	90.776	63.067	[47.885, 78.801]
2	mip_strategy_heuristicfreq	16.599	21.385	[14.206, 29.822]
3	simplex_dgradient	8.080	7.368	[6.563, 8.205]
4	simplex_tolerances_markowitz	10.345	12.763	[7.103, 18.718]
5	mip_limits_aggforcut	6.900	6.120	[5.475, 6.798]
6	mip_strategy_variableselect	8.974	8.012	[4.800, 11.432]
7	simplex_pgradient	5.799	8.037	[4.815, 11.477]
8	mip_strategy_fpheur	8.699	5.155	[4.647, 5.687]
9	lpmethod	8.700	5.151	[4.654, 5.679]
10	barrier_crossover	8.662	5.146	[4.654, 5.671]
11	preprocessing_symmetry	8.696	5.140	[4.641, 5.657]
12	sifting_algorithm	8.695	5.145	[4.635, 5.672]
13	barrier_limits_growth	8.678	5.137	[4.638, 5.660]
14	mip_limits_gomorycand	8.704	5.143	[4.641, 5.670]
15	mip_limits_cutsfactor	5.639	5.115	[4.635, 5.614]
16	mip_cuts_gubcovers	5.672	5.132	[4.650, 5.633]
17	mip_cuts_gomory	5.447	5.236	[4.727, 5.760]
18	preprocessing_repeatpresolve	5.608	5.433	[4.948, 5.938]
19	mip_strategy_presolvenode	5.766	5.288	[4.830, 5.769]
20	mip_cuts_mircut	5.511	5.411	[4.923, 5.922]
configured	-	5.511	5.411	[4.923, 5.922]

Ablation was performed using the CPLEX default configuration as the source, and the configuration produced by PARAMILS as the target. The "Test bootstrap CI" column indicates the [10, 90 %] confidence interval of PAR10 performance, using 10,000 bootstrap samples of our test set data for each configuration

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 the parameter *simplex_dgradient* was not in the top 10 important parameters identified for this scenario by Hutter et al. (2014), although 6 of the 10 most important CPLEX parameters as identified by their functional ANOVA approach were not changed from their default values in our experiments (this effect was also noted by Hutter et al.).

Finally, Fig. 2c, d and 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 (Fig. 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 our experiments, modifying *vicinato* often did not produce large gains in performance by itself, but allowed for modification of *hpar_cut_neighb*, which in turn resulted 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.

6 Extended uses of ablation

So far, we have focussed on results from straightforward applications of ablation analysis. However, ablation analysis can also be used to find configurations that generalise better than the default configuration to new domains (i.e., problem instance classes) of interest, or to obtain information about the topology of the algorithm configuration space around and between high-performance configurations. As mentioned in Sect. 2, these extended uses of ablation analysis strongly echo ideas from path relinking.

In several of these experiments, we focussed on our three LPG scenarios, since these allowed us to investigate generalisation of performance between different classes of instances (here: planning domains).

Generalisation to other domains. After spending hours of CPU time to produce an algorithm configuration that is highly-optimised for a specific problem instance set, it would be ideal if a user could somehow use that configuration to find a different configuration with better performance than the default on a different, previously unseen problem instance set. To investigate the potential for ablation analysis to produce such configurations, we performed three sets of experiments, in order to investigate how configurations "near" the LPG default (along ablation paths toward a chosen PARAMILS configuration) perform on scenarios other than the one used to determine the original ablation paths. Table 7 illustrates the results of these experiments. For depots and satellite, configurations that are a small number of ablation rounds away from the default perform substantially better than the default configuration itself (2.5- to 6.1-fold speedups), before the ablation path hits more specialised configurations that do not generalise as well to new domains. We do not see the same effect in the zenotravel domain, where the ablation path does improve in quality before getting worse, but fails to exceed at any point the performance of the default configuration.

Table 7 Performance analysis of the LPG default, configurations obtained from PARAMILS, and the first 4 flips of ablation, for all 3 LPG scenarios, on the depots, satellite and zenotravel test sets

Configuration	Test set performance (PAR10)		
	Depots	Satellite	Zenotravel
default	38.097	17.940	29.361
depots ablation flip 1	1.520	7.275	1114.835
depots ablation flip 2	1.390	7.260	908.578
depots ablation flip 3	1.261	7.279	896.241
depots ablation flip 4	1.037	7.234	894.766
depots configured	0.658	7.617	1028.118
satellite ablation flip 1	10.413	7.577	1467.274
satellite ablation flip 2	6.194	7.141	1468.614
satellite ablation flip 3	224.976	6.997	870.385
satellite ablation flip 4	223.590	6.997	867.646
satellite configured	2643.840	5.783	2478.914
zenotravel ablation flip 1	11.983	17.358	8.258
zenotravel ablation flip 2	11.790	17.381	5.912
zenotravel ablation flip 3	368.500	16.323	4.369
zenotravel ablation flip 4	1720.010	10.102	3.713
zenotravel configured	2807.531	7.952	2.092

Ablation paths between high-quality configurations. Next, we investigated whether ablation analysis could be used to better understand the topology of the search space between high-quality configurations for each of our five scenarios. If the configurations found on ablation paths between two high-quality configurations do not differ in performance from the source and target, we might expect the existence of a large plateau of high-quality configurations in the search space. Conversely, if the configurations lying on such ablation paths have worse performance than the source and target, we have reason to believe that the source and target may lie in different basins of the parameter response surface.

We examined the ablation paths between two high-quality configurations for each of our five scenarios, defined as the best and second-best configurations found by PARAMILS, evaluated on the training set. We used our racing approach, with the maximum number of rounds again set to 200. Figure 3 gives a visual summary of these paths, while Table 8 contains the test set PAR10 performance and [10, 90 %] bootstrap confidence intervals of each configuration along the ablation path for the SPEAR scenario. Tables containing the data for our other four scenarios can be found in Online Resource 1.

We see that for SPEAR, there appears to be at least one ridge of poor-quality configurations that must be surmounted in order to move from the source to the target configuration. In all of the other four scenarios, no such ridge is present, and every configuration along the ablation paths share the quality of the source/target configurations. We note that as ablation analysis is a greedy process, the occurrence of such ridges on the ablation paths does not strictly imply the existence of barriers that must be overcome by all paths between the source and target configurations. However, in

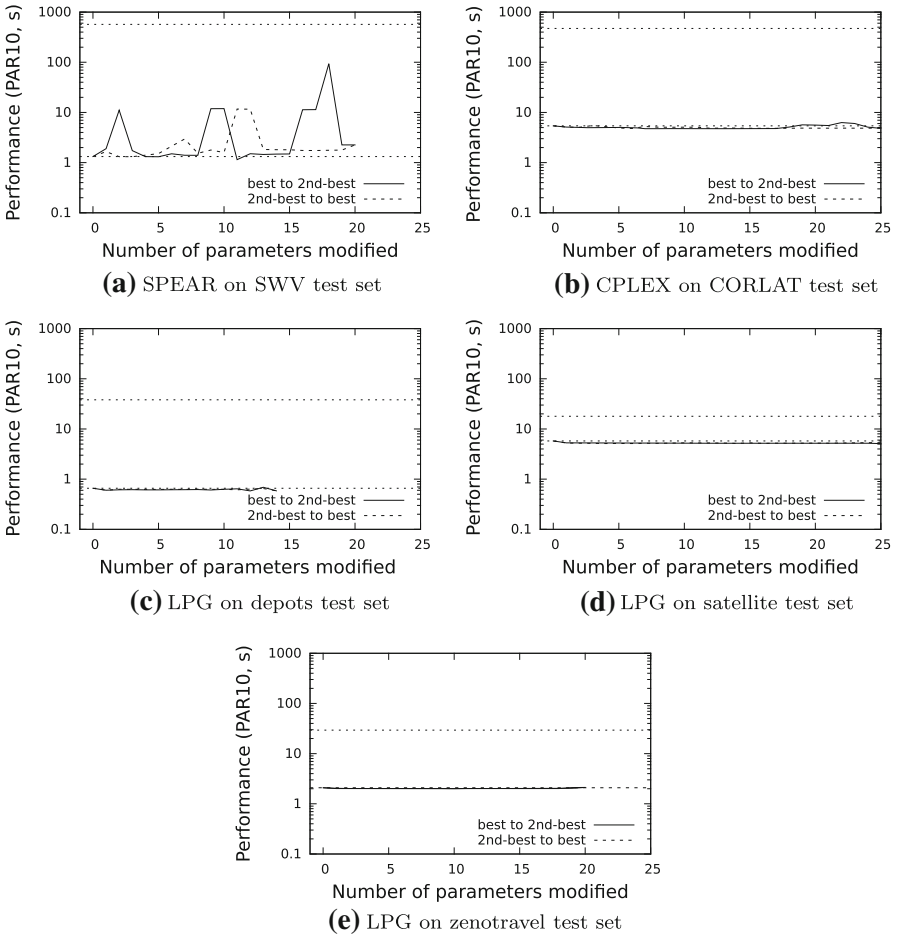


Fig. 3 Ablation paths determined using racing with up to 200 rounds on the test sets of the five configuration scenarios. Ablation was performed from the best ParamILS incumbent to the second-best incumbent on each scenario. The *horizontal lines* indicate the PAR10 scores of the default (source) and automatically-configured (target) configurations

the cases where no such ridges are present in the ablation paths, at least one path of equal performance between the source and target configurations is guaranteed to exist.

Ablation between high-quality configurations for different domains. Next, we investigated the use of ablation paths between high-performance configurations for different domains, to see how different configurations "needed" to be before high performance was achieved on a domain different from that for which they were originally optimised. (We note that this question is closely related to the problem of transfer learning.) We performed ablation analysis experiments between the chosen ParamILS configurations for two of the LPG scenarios, evaluated using the test set from one of those scenarios. Fig. 4a, b show the ablation paths between the depots and satellite configurations, evaluated on the depots and satellite test sets respectively. Similarly, Fig. 4c,

Table 8 Parameters chosen and resulting test set performance for ablation analysis accelerated by racing (maximum of 200 rounds) for SPEAR on SWV test set, using the best PARAMILS configuration as the source and the second-best PARAMILS configuration as the target

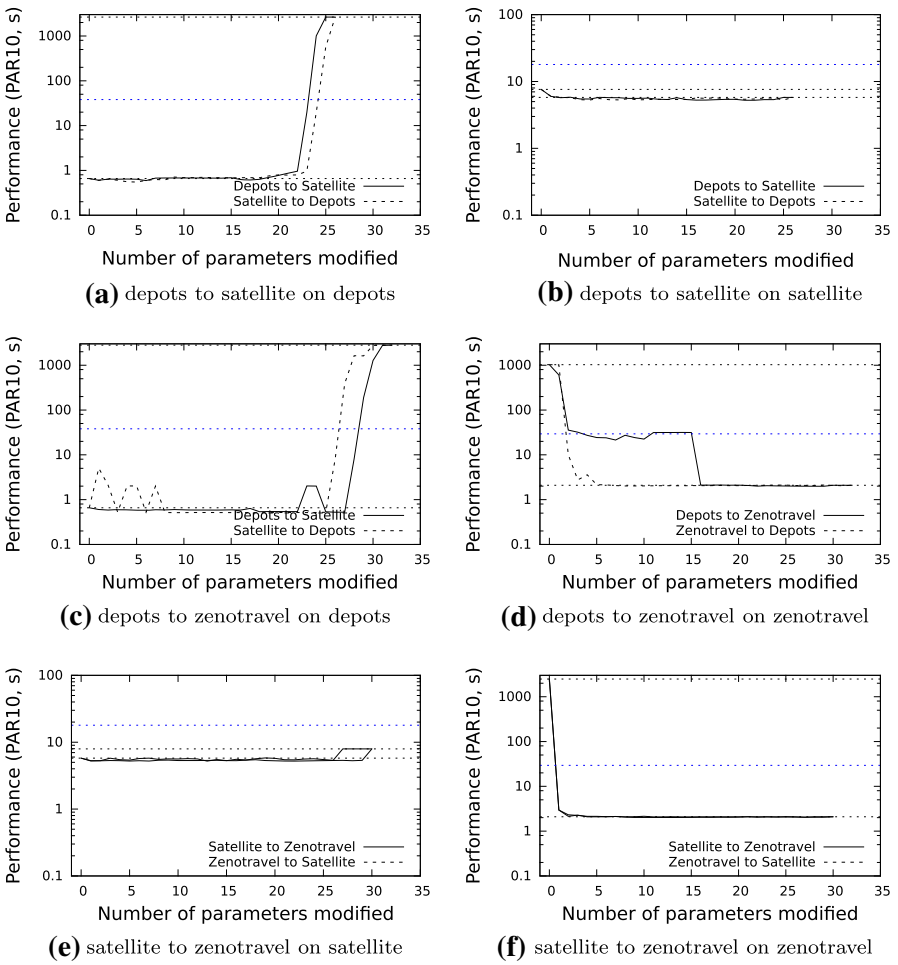
Round	Parameter	Performance	Test bootstrap CI [10, 90 %]
best	-	1.321	[1.055, 1.617]
1	sp-res-cutoff-lits	1.899	[1.058, 2.824]
2	sp-learned-clauses-inc	11.104	[1.056, 21.274]
3	sp-max-res-lit-inc	1.734	[1.391, 2.096]
4	sp-phase-dec-heur	1.316	[1.084, 1.564]
5	sp-max-res-runs	1.310	[1.060, 1.579]
6	sp-restart-inc	1.501	[1.083, 1.971]
7	sp-variable-decay	1.404	[1.137, 1.683]
8	sp-clause-decay	1.397	[1.134, 1.667]
9	sp-rand-var-dec-freq	11.892	[1.641, 22.305]
10	sp-learned-size-factor	11.934	[1.709, 22.464]
11	sp-rand-phase-dec-freq	1.1474	[0.906, 1.412]
12	sp-use-pure-literal-rule	1.504	[1.101, 1.937]
13	sp-clause-activity-inc	1.455	[1.072, 1.867]
14	sp-clause-del-heur	1.481	[1.090, 1.913]
15	sp-update-dec-queue	1.488	[1.087, 1.920]
16	sp-learned-clause-sort-heur	11.327	[1.173, 21.641]
17	sp-res-cutoff-cls	11.390	[1.239, 21.752]
18	sp-rand-var-dec-scaling	93.552	[54.900,132.696]
19	sp-first-restart	2.256	[1.499, 3.113]
second-best	-	2.256	[1.499, 3.113]

The "Test bootstrap CI" column indicates the [10, 90 %] confidence interval of PAR10 performance, using 10 000 bootstrap samples of our test set data for each configuration

d show the resulting paths for ablation between the depots and zenotravel configurations, and Fig. 4e, f show the paths for ablation between the satellite and zenotravel configurations. Tables containing the performance of each configuration along these paths, for each scenario, can be found in Online Resource 1.

We also performed additional ablation experiments between the configurations obtained from PARAMILS for each pair of LPG scenarios, evaluated using the test set of the third scenario. This set of experiments further tests the use of ablation paths to find configurations that generalise well to previously unseen problem instance sets. The results of these experiments are shown visually in Fig. 5, and tables detailing the performance of each configuration along these ablation paths are given in Online Resource 1.

For the ablation from the depots configuration to the satellite configuration, both the source and target configurations are significantly worse than the default on the zenotravel test set. However, there are configurations along the resulting ablation path with better performance than the default.



(a) depots to satellite on depots **(b)** depots to satellite on satellite

(c) depots to zenotravel on depots **(d)** depots to zenotravel on zenotravel

(e) satellite to zenotravel on satellite **(f)** satellite to zenotravel on zenotravel

Fig. 4 Ablation paths determined using racing with up to 200 rounds on the test sets of the three LPG scenarios. Ablation was performed from the best PARAMILS incumbent on domain A to the best incumbent on domain B on each scenario, evaluated on domain A and B. The *horizontal lines* indicate the PAR10 scores of the default (*blue*) and the domain A/B incumbents (*grey*) (Color figure online)

For the ablation from the depots configuration to the zenotravel configuration, both the source and target configurations are better than the default, but we again see improvement along the ablation path. In fact, many of the configurations along the ablation path are better than the satellite PARAMILS configuration, evaluated on the satellite test set.

Finally, for the ablation from the satellite configuration to the zenotravel configuration evaluated on depots, again we see that the source and target configurations are orders of magnitude worse than the default configuration. As with the other two experiments, the configurations along the ablation path have better performance on the depots test set than either the source or target configuration, although none reach the performance of the default configuration.

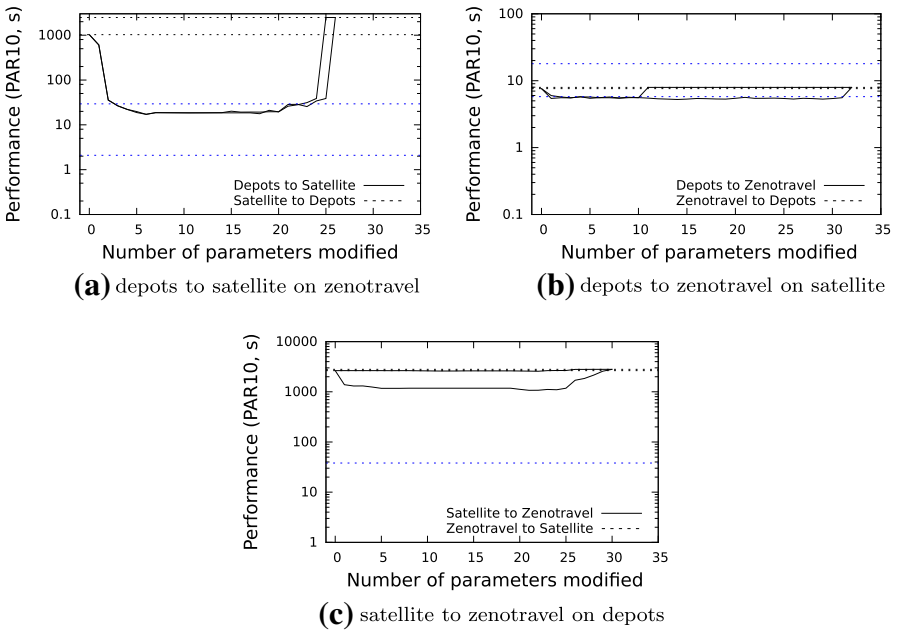


Fig. 5 Ablation paths determined using racing with up to 200 rounds on the test sets of the three LPG scenarios. Ablation was performed from the best PARAMILS incumbent on domain A to the best incumbent on domain B on each scenario, evaluated on domain C. The *horizontal lines* indicate the PAR10 scores of the default (*blue*), the domain A/B incumbents (*grey*), and the performance of the PARAMILS configuration for domain C (*lowest blue*) (Color figure online)

7 Support for conditional parameters

As we outlined in Sect. 3, algorithm configuration spaces often contain conditional parameters, i.e., those parameters whose value only have an effect on algorithm performance if one or more *parent* parameters are set to specific values. If the parent parameters have been set to the required values, we say that the conditional parameter is *active*. In each round of the ablation analysis approach given in Algorithm 1, the set of candidate configurations $\mathcal{A}'$ is constructed by only considering configurations obtained by changing the value of one active parameter to its value in the target configuration. This active parameter requirement can cause an important conditional parameter to appear much later in the ablation path than it should, as all of its parent parameters must be modified in earlier ablation rounds, and those parent parameters may have minimal or no effect on algorithm performance. We observed this behaviour in all three of our LPG scenarios, where the conditional *hpar_cut_neighb* parameter was inactive until its parent *vicinato* parameter was modified, often later in the ablation process.

In order to improve the performance of ablation analysis on such scenarios, we extended our approach to allow the modification of more than one parameter in each ablation round. During the construction of $\mathcal{A}'$, rather than ignoring each inactive parameter we include the configuration obtained by modifying the value of that parameter

Algorithm 4: ablationWithConditionalSupport $(\mathcal{A}, \theta_{\text{source}}, \theta_{\text{target}}, I, m)$

Input: Parameterised algorithm $\mathcal{A}$, two parameter configurations of $\mathcal{A}$, θ_{source} and θ_{target}, benchmark instance set I, performance metric m (to be minimised)

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. $\theta_0 = \theta_{\text{source}}$ and $\theta_l = \theta_{\text{target}}$

$\theta \leftarrow \theta_{\text{source}}$
remParameters $\leftarrow$ set of parameters of $\mathcal{A}$ with different values in θ_{source} and θ_{target}
ablationRoundsBest $\leftarrow (\theta_{\text{source}})$
while remParameters $\neq \emptyset$ **do**
  $\mathcal{A}' \leftarrow$ set of algorithms with configurations obtained from θ via flipping 1 parameter in remParameters, **along with all ancestor parameters whose modification is required to make that parameter active**, to the value in θ_{target}, ignoring configurations that are prohibited in the configuration space.
  $\theta' \leftarrow$ determineBest $(\mathcal{A}', I, m)$
  remParameters $\leftarrow$ set of parameters of $\mathcal{A}$ with different values in θ' and θ_{target}
  ablationRoundsBest $\leftarrow$ ablationRoundsBest $+ (\theta')$
  $\theta \leftarrow \theta'$
end
return ablationRoundsBest

together with all of the ancestor parameters whose modification is required to make that parameter active. Pseudocode for this modification is given in Algorithm 4, and we note that the required change from Algorithm 1 only affects the construction of $\mathcal{A}'$. In particular, no modifications were necessary to either of our determineBest () implementations.

To evaluate this modified approach, we repeated the experiments on our three LPG scenarios detailed in Sects. 4 and 5. The performance of the configurations along the resulting ablation paths are illustrated in Fig. 6a, b and c, while the corresponding data tables are contained in Online Resource 1.

For the depots and satellite scenarios, the conditional parameter *hpar_cut_neighb* and its parent *vicinato* now appear as the second configuration on the ablation path, rather than the last and middle positions seen in our previous experiments. For the zenotravel scenario, this effect is even greater as *vicinato* and *hpar_cut_neighb* are now selected as the very first parameters on the ablation path, rather than their former position close to the end, where they appeared in our earlier results. In all three scenarios we now see that all of the performance gains appear in the first several configurations along the ablation path, rather than the long plateau of performance before a late improvement seen in Fig. 2c, d and e. This allows for a more straightforward estimate of parameter importance, and users only interested in the first k important parameters (or with limited CPU budgets) have the option of truncating the ablation analysis process after k rounds.

8 Future work

This work can be extended in various directions, and we believe the primary opportunity for improvement and extension is the investigation of alternatives and improve-

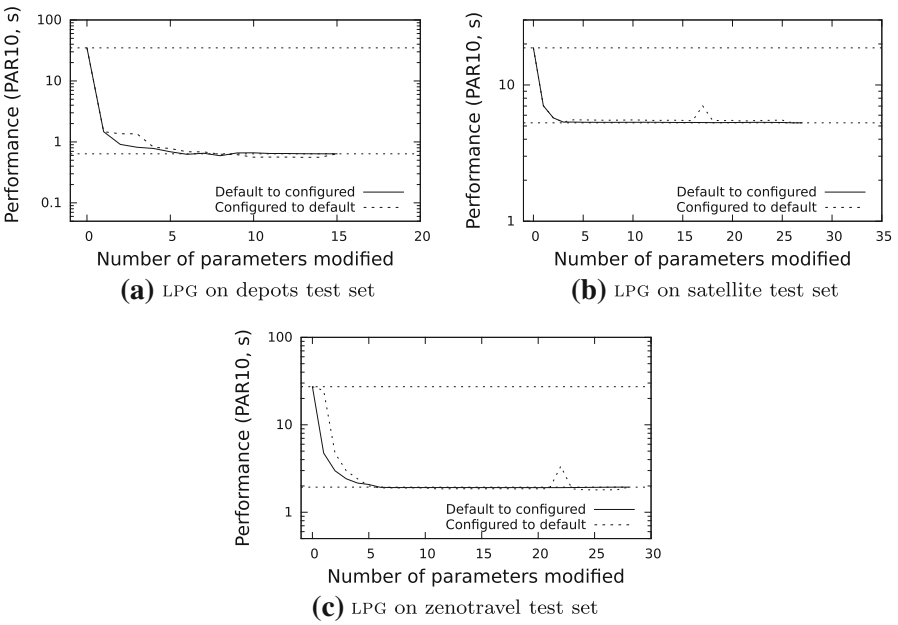


Fig. 6 Ablation paths determined using our extended support for conditional parameters and the racing variant with up to 200 rounds, on the training sets of our three LPG scenarios. Each configuration on the ablation paths was evaluated using the corresponding test set for each scenario. The *horizontal lines* indicate the PAR10 scores of the default (source) and automatically-configured (target) configurations on those test sets

ments to our racing-based ablation approach. For example, we have observed that in later ablation rounds, F-Race quickly eliminates all candidate configurations except for a small number of configurations with very similar performance. In these cases, the maximum number of racing rounds are used for these remaining configurations. We also observed a related case where the F-Race was quickly reduced to two candidate configurations, where the p-value of the Friedman test was sufficient to proceed to pairwise elimination, but the p-value of the subsequent t-test was insufficient to eliminate the configuration with worse performance. Although this is consistent with what is known in terms of the impact of violated normality assumptions on the power of tests involving the t-statistic, it would be beneficial to investigate further as a solution to this problem could substantially reduce the number of algorithm runs required. Possible candidates would be to use a non-parametric test instead, possibly in combination with multiple-testing correction (see Styles and Hoos (2013)).

Another avenue for further work is to make support for complex parameter interdependencies more flexible, for example by extending the approach in Sect. 7 to allow sets of parameters without conditional relationships to be modified in the same ablation round, as indicated by the end-user or by automated methods during the ablation process.

Finally, we believe that our approach and the model-based techniques discussed in Sect. 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.

9 Conclusions

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 SAT, MIP and AI planning, with 26–76 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 (e.g., by Hutter et al. (2014)), 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. Overall, we believe that our ablation analysis approach can be of great use to help algorithm developers and end users understand more about which algorithm parameters (and therefore algorithm subsystems and behaviours) are most responsible for high performance on problem instances of interest. The implementation of our approach has additional been made available for other researchers to use, please see the project page [3] for more details.

Acknowledgments The authors would like to thank our anonymous reviewers, the anonymous reviewers of the earlier version of this paper presented at MIC 2013, and members of the UBC BETA lab for the helpful suggestions and fruitful discussion regarding our approach.

References

Aghaeepour, N., Hoos, H.H.: Ensemble-based prediction of RNA secondary structures. BMC Bioinform. **14**, 139 (2013)

Ansótegui, C., Sellmann, M., Tierney, K.: A gender-based genetic algorithm for the automatic configuration of algorithms. Proceedings of CP, pp. 142–157. (2009)

Bartz-Beielstein, T.: Experimental Research in Evolutionary Computation-The New Experimentalism. Natural Computing Series. Springer, Berlin (2006)

[3] http://www.cs.ubc.ca/labs/beta/Projects/Ablation/.

Birattari, M., Stützle, T., Paquete, L., Varrentrapp, K.: A racing algorithm for configuring metaheuristics. In: Proceedings of GECCO'02, pp. 11–18. (2002)
Chiarandini, M., Goegebeur, Y.: Mixed models for the analysis of optimization algorithms. Experimental Methods for the Analysis of Optimization Algorithms, pp. 225–264. (2010)
Conover, W.: Practical Nonparametric Statistics, 3rd edn. Wiley, New York (1999)
Glover, F.: Genetic algorithms and scatter search: unsuspected potentials. Stat. Comput. **4**, 131–140 (1994)
Glover, F.: Tabu search and adaptive memory programing—advances, applications and challenges. In: Barr, R.S., Helgason, R.V., Kennington, J.L. (eds.) Interfaces in Computer Science and Operations Research, vol. 7, pp. 1–75. Springer, Nwe York (1997)
Glover, F., Laguna, M.: Tabu Search. Kluwer Academic Publishers, Boston (1997)
Glover, F., Laguna, M., Martí, R.: Fundamentals of scatter search and path relinking. Control Cybern. **39**, 653–684 (2000)
Glover, F., Laguna, M., Martí, R.: Scatter search and path relinking: advances and applications. In: Glover, F., Kochenberger, G.A. (eds.) Handbook of Metaheuristics, International Series in Operations Research & Management Science, vol. 57, pp. 1–35. Springer, New York (2003)
Gomes, C.P., van Hoeve, W.J., Sabharwal, A.: Connections in networks: a hybrid approach. Proceedings of CPAIOR , pp. 303–307. (2008)
Gunawan, A., Lau, H.C., Lindawati.: Fine-tuning algorithm parameters using the design of experiments approach. In: Proceedings of LION-5, pp. 278–292. (2011)
Hooker, G.: Generalized functional ANOVA diagnostics for high dimensional functions of dependent variables. J. Comput. Graph. Stat. **16**, 709–732 (2007)
Hoos, H.H.: Programming by optimization. Commun. ACM **55**(2), 70–80 (2012)
Hutter, F., Babić, D., Hoos, H.H., Hu, A.J.: Boosting verification by automatic tuning of decision procedures. In: Formal Methods in Computer-Aided Design, pp. 27–34. (2007a)
Hutter, F., Hoos, H.H., Stützle, T.: Automatic algorithm configuration based on local search. In: AAAI 07, pp. 1152–1157. (2007b)
Hutter, F., Hoos, H.H., Leyton-Brown, K., Stützle, T.: ParamILS: an automatic algorithm configuration framework. J. Artif. Intel. Res. **36**, 267–306 (2009)
Hutter, F., Hoos, H.H., Leyton-Brown, K.: Automated configuration of mixed integer programming solvers. Proceedings of CPAIOR, pp. 186–202. (2010)
Hutter, F., Hoos, H.H., Leyton-Brown, K. :Sequential model-based optimization for general algorithm configuration. In: Proceedings of LION-5, pp. 507–523. (2011)
Hutter, F., Hoos, H.H., Leyton-Brown, K.: Identifying key algorithm parameters and instance features using forward selection. In: Proceedings of LION-7, pp. 364–381. (2013)
Hutter, F., Hoos, H.H., Leyton-Brown, K.: An efficient approach for assessing hyperparameter importance. In: Proceedings of the 31st International Conference on Machine Learning (ICML 2014), pp. 754–762. (2014)
López-Ibáñez, M., Dubois-Lacoste, J., Stützle, T., Birattari, M.: The irace package, iterated race for automatic algorithm configuration. Tech. Rep. TR/IRIDIA/2011-004, IRIDIA, Université Libre de Bruxelles, Belgium (2011)
Maron, O., Moore, A.: Hoeffding races: accelerating model selection search for classification and function approximation. Adv. Neural Inf. Process. Syst. **6**, 59–66 (1994)
Nannen, V., Eiben, A.: Relevance estimation and value calibration of evolutionary algorithm parameters. Proceedings of IJCAI, pp. 975–980. (2007)
Nell, C., Fawcett, C., Hoos, H.H., Leyton-Brown, K.: HAL: A framework for the automated analysis and design of high-performance algorithms. In: Proceedings of LION-5, pp. 600–615. (2011)
Reidys, C.M.: Combinatorial Computational Biology of RNA: Pseudoknots and Neutral Networks. Springer, New York (2011)
Styles, J., Hoos, H.H.: Ordered racing protocols for automatically configuring algorithms for scaling performance. In: Proceedings of the Genetic and Evolutionary Computation Conference (GECCO) 2013, pp. 551–558. (2013)
Vallati, M., Fawcett, C., Gerevini, A.E., Hoos, H.H., Saetti, A.: Automatic generation of efficient domain-optimized planners from generic parametrized planners. Proceedings of IJCAI RCRA Workshop 2011, pp. 111–123. (2011)
Vallati, M., Fawcett, C., Gerevini, A.E., Hoos, H.H., Saetti, A.: Automatic generation of efficient domain-specific planners from generic parametrized planners. In: Proceedings of the 6th Annual Symposium on Combinatorial Search (SoCS-13), pp. 184–192. (2013)