

Per instance algorithm configuration of CMA-ES with limited budget

Nacim Belkhir, Johann Dreo, Pierre Savéant, Marc Schoenauer

▶ **To cite this version:**

Nacim Belkhir, Johann Dreo, Pierre Savéant, Marc Schoenauer. Per instance algorithm configuration of CMA-ES with limited budget. GECCO 2017 - Proceedings of the Genetic and Evolutionary Computation Conference , Jul 2017, Berlin, Germany. pp. 681-688. hal-01613753

HAL Id: hal-01613753
https://inria.hal.science/hal-01613753

Submitted on 11 Oct 2017

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Per Instance Algorithm Configuration of CMA-ES with Limited Budget

Nacim Belkhir
Inria Saclay
Orsay, France
nacim.belkhir@inria.fr

Johann Dréo
Thales Research&Technology
Palaiseau, France
johann.dreo@thalesgroup.com

Pierre Savéant
Thales Research&Technology
Palaiseau, France
pierre.saveant@thalesgroup.com

Marc Schoenauer
Inria Saclay
Orsay, France
marc.schoenauer@inria.fr

ABSTRACT

Per Instance Algorithm Configuration (PIAC) relies on features that describe problem instances. It builds an Empirical Performance Model (EPM) from a training set made of (instance, parameter configuration) pairs together with the corresponding performance of the algorithm at hand. This paper presents a case study in the continuous black-box optimization domain, using features proposed in the literature. The target algorithm is CMA-ES, and three of its hyper-parameters. Special care is taken to the computational cost of the features. The EPM is learned on the BBOB benchmark, but tested on independent test functions gathered from the optimization literature.The results demonstrate that the proposed approach can outperform the default setting of CMA-ES with as few as 30 or 50 time the problem dimension additional function evaluations for feature computation.

KEYWORDS

Algorithm Configuration, problem features, empirical study, numerical black box optimization

ACM Reference format:
Nacim Belkhir, Johann Dréo, Pierre Savéant, and Marc Schoenauer. 2017. Per Instance Algorithm Configuration of CMA-ES with Limited Budget. In *Proceedings of ACM Conference, Washington, DC, USA, July 2017 (Conference'17),* 8 pages.
https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

It is widely acknowledged today in the optimization community at large that the quest for a general optimization algorithm, that would quasi-optimally solve any optimization problem, is vain (see the numerous works after the seminal *No Free Lunch theorem* [30]). Hence, tackling an unknown optimization problem first amounts to choose the 'best' algorithm (also known as the *Algorithm Selection* problem), and/or to choose the 'best' parameter setting for a given algorithm (the *Algorithm Configuration* problem) – where 'best' is related to some user-defined performance measure, usually balancing between computational cost and domain-specific measure of solution quality: precision (and constraint violation) for continuous optimization, number of unsatisfied constraints in SAT domains, number of solved instances within a given large testbench for decision problems, ...). The Algorithm Selection and Configuration problems thus amount to meta-optimization problems [14], whose "meta-objective" is the performance measure of the algorithm at hand in a given parameter configuration. However, in the Black-Box scenario, such performance can only be empirically assessed at the cost of running this algorithm on some problem instances, making the task of direct meta-optimization immensely costly, be it for a single or a whole class of problem instances.

On the other hand, for some domains, there exist *features* describing more or less accurately the problem instances: given sufficiently many examples of (instance,'best' algorithm/configuration) pairs (where the instance is represented by its features), one can learn a mapping from instance space to algorithm or algorithm configuration space. However, this requires the knowledge of the 'best' for many sample instances, and an alternative is to learn an empirical performance model (EPM) that gives the performance of the algorithm for sample (instance, algorithm/configuration) pairs. It is then straightforward to find the 'empirical best' algorithm or configuration for a new instance provided its features can be computed easily. This approach is know as the Per Instance Algorithm Configuration (PIAC) [15]. However, most existing PIAC works address combinatorial domains (more in Section 3), even if relevant features have been proposed for continuous optimization (see Section 4). This paper tries to fill this gap, presenting an empirical validation of the PIAC approach for Black-Box continuous optimization.

It is organized as follows: Section 2 surveys previous work in AC, and motivates the use of PIAC by focusing on the computational cost (in terms of number of function evaluations) required to solve the AC problem. The proposed methodology is then detailed in Section 3. Note that both these sections apply to any black-box search algorithm on any search space. From there on, the paper focuses on continuous black-box optimization. Different features proposed in the literature are first reviewed in Section 4, and discussed according to the computational cost of their actual computation. The CMA-ES algorithm is briefly introduced in Section 5, with a focus on the three hyper-parameters that will be subject to Automatic Configuration. Section 6 introduces the training and test sets of functions used for the validation of the proposed approach, one of the originalities of this work: Whereas the EPM is trained on the

Conference'17, July 2017, Washington, DC, USA
2017. ACM ISBN 978-x-xxxx-xxxx-x/YY/MM...$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

now classical BBOB benchmark suite, exhibiting known difficulties [10], it is validated on an completely different set of objective functions, carefully gathered from the continuous optimization literature. Section 7 presents the practicalities of the experiments, whose results are presented in Section 8,followed by a discussion in Section 9, and our conclusion in Section 10.

2 BACKGROUND

2.1 Hypotheses and Notations

From thereon, only the Algorithm Configuration problem will be considered, in the context of Black Box Optimization. An algorithm $\mathcal{A}$ is given together with its control parameters $\theta \in \mathbf{\Theta}$. In its simplest form, for a given problem instance, identified by abuse of notation to its *objective function* $f : \Omega \mapsto \mathbb{R}$ (no hypothesis is made on Ω at this point), the *instance-based AC problem* aims at finding the parameter setting $\theta^*_\varphi(f)$ of $\mathcal{A}$ such that it best optimizes f, for some given performance measure $\varphi : \mathcal{F} \times \mathbf{\Theta} \mapsto \mathbb{R}$, where $\mathcal{F}$ is the set of all possible objective functions.

Additionally, we will assume when so mentioned that all objective functions f can be described by their *features* $\boldsymbol{\psi}(f) \in \mathbf{\Psi}$. Furthermore, we will assume that φ is well defined on $\mathbf{\Psi} \times \mathbf{\Theta}$, implicitly assuming that two different objective functions f and g with same features share the same best configuration (i.e., $\boldsymbol{\psi}(f) = \boldsymbol{\psi}(g) \Rightarrow \theta^*_\varphi(f) = \theta^*_\varphi(g)$), an hypothesis that will be discussed in Section 4.

2.2 Algorithm Configuration

Several methods have been proposed in the literature that directly solve this single-instance AC problem, considering it as a meta-optimization problem in $\mathbf{\Theta}$ [4, 7, 16, 17, 27]. The goal of this meta-optimization problem is clear: optimize the chosen performance measure. The result is a single parameter setting $\theta^*(f)$, that hardly helps for any other problem instance. And its cost is huge, involving at least several dozens of actual runs of $\mathcal{A}$ on f, and thus intractable in practice.

A possible workaround is to address the *class-based AC problem*: for a given function class $\mathcal{C} \subset \mathcal{F}$, find the parameter setting $\theta^*(\mathcal{C})$ of $\mathcal{A}$ such that it best optimizes simultaneously all functions $f \in \mathcal{C}$, for φ. The underlying idea is to use a training set $f_1, \ldots, f_n$ of functions of $\mathcal{C}$ and to solve the meta-optimization problem that minimizes some aggregation of $\varphi(f_1), \ldots, \varphi(f_n)$. This approach suffers from some severe drawbacks: First, it is clear that $\varphi(f, \theta^*_\varphi(\mathcal{C}))$ will be worse than $\varphi(f, \theta^*_\varphi(f))$ for all $f \in \mathcal{C}$, even more so as the size of $\mathcal{C}$ increases. Second, class-based AC faces the critical issue of generalization, even though standard ML approach involving training and validation sets can be used to avoid overfitting the training set. Furthermore, the definition of the class of instances is there left to the user prior knowledge (see e.g., the numerous possible definitions of SAT instances tracks in the different SAT competitions), thus possibly involving some fuzziness.

The situation is different when the classes can be defined automatically, with minimal user intervention. This is the case when there exist some intrinsic description of the problem instances using some characteristics of the search space, aka *features*. These features can be used to cluster the training set, be it for Algorithm Selection [1] or Algorithm Configuration [19]. Class-based AC is applied to each cluster in turn, and further unknown instances are assigned to one cluster based on their feature values, and assigned corresponding parameter configuration. This approach is termed instance-specific algorithm configuration (ISAC) by its authors. However, relying on class-based AC, it suffers from similar issues regarding generalization.

The PIAC approach [15, 21] takes another step toward learning the mapping between features and best parameter configuration. However, acquiring examples of such mapping would require applying some instance-based AC solver to functions $f_1, \ldots, f_n$ of the training set, and would result in only n examples of that target mapping. The PIAC approach, on the other hand, uses an indirect approach: it learns an approximation $\widehat{\varphi} : \mathcal{F} \times \mathbf{\Theta} \mapsto \mathbb{R}$ of φ, using as examples all available results of applying $\mathcal{A}$ with some configuration θ on some function $f \in \mathcal{F}$, as will be detailed in next Section.

A critical issue for both ISAC and PIAC approaches is the existence of features with sufficient discriminative power (can different instances, with different optimal parameter setting, have the same features). In the SAT [15, 32] or MIP [18] domains, features had been designed during several decades of research, and the PIAC approach was demonstrated successful. Nevertheless, for continuous black box optimization problems (with a search subspace of $\mathbb{R}^d$ for some $d \in \mathbb{N}$), feature-based approaches to AC remain under investigated. Different features have been proposed [23–25], but these features have mainly been used to characterize the fitness landscape – though some promising results have been obtained for the class-based Algorithm Configuration and Algorithm Selection problems [8, 26]. However, these works rely on some cross-validation procedures, where only one numerical black box test bench is used for their empirical study. Despite their promising results, their works may be limited by the cost of features [26] or the limited number of features [8]. Therefore, this paper aim at empirically investigating the PIAC in a real-world conditions, where the objective function may be computationally costly and only a limited budget of evaluations is available, and in particular with a new test bench in order to assess the performance of the PIAC.

3 PIAC

Initially proposed in the combinatorial domain of auction problems [21], the PIAC approach was successfully used in several other combinatorial and MIP domains [15, 18, 33], using the runtime to reach a solution of the instance at hand as performance measure.

3.1 General Overview

The PIAC approach is a two phases process, as described in Figure 1:

The **learning phase** consists in learning an *Empirical Performance Model* (EPM) $\widehat{\varphi}$ that approximates φ on $\mathbf{\Psi} \times \mathbf{\Theta}$. $\mathcal{A}$ is run to optimize different functions f_i described by their features $\psi(f_i)$, using different parameter configurations θ_j, thus computing $\varphi\left(\psi(f_i), \theta_j\right)$[1]. The set of all $((\psi(f_i), \theta_j), \varphi(\psi(f_i), \theta_j))$ is a training set that can be used as input to any standard regression method to learn the approximate model $\widehat{\varphi}$. Note that building such a model is done once, and hence its computational cost is not a critical issue.

In the **decision phase**, a new objective function g is to be optimized with $\mathcal{A}$. Its features $\psi(g)$ are computed, and the optimization

[1] In particular, the same θ_j need not have been tried for all f_i.

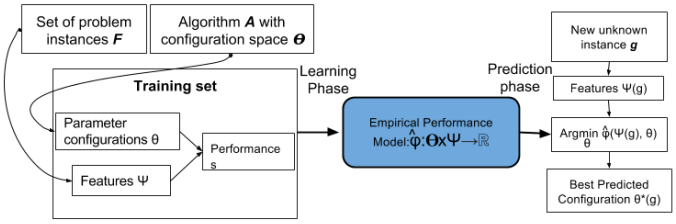


Figure 1: The PIAC approach

of $\widehat{\varphi}(\psi(g), \theta)$ on parameter space $\mathbf{\Theta}$ leads to the empirical optimal parameters $\theta^*_{\varphi}(g)$ of $\mathcal{A}$ for g. The cost of computing the features $\psi(g)$, in terms of number of calls to g, is here of utter importance. In particular, it should be compared to the cost of running a full instance-based meta-optimization of $\mathcal{A}$ parameters for g (using e.g., SMAC [16]).

3.2 PIAC in Continuous Domain

As of today, the PIAC approach remains under-investigated for the continuous domain. Different works directly related to the continuous domain and based on EPM for Algorithm Configuration were proposed in [1, 8, 26]. Abell et al. [1] proposed to use the ISAC method [19] for an algorithm portfolio based on the features in order to find the best solver, among 21 different solvers, for problem instances of the BBOB test bench [10]. In [26], the Per Instance Algorithm Configuration is tackled, by using a set of features based on samples to predict the best parameter setting of CMA-ES on each function of the BBOB test bench. In contrast to [18], Muñoz et al. [26], the empirical performance model was learned by using a multi-layer neural network method. More recently, based on *aforementionned* features in Section 4, Belkhir et al. [5] investigated the PIAC on Differential Evolution [29], by considering different sets of features and a discretization of the parameter space ($\approx$ 8000 parameter configurations. According to [18], an empirical performance model is learned with a Random Forest regression, and based on a cross-validation procedure where each test function of the BBOB test were removed one at a time, Belkhir et al. [5] empirically demonstrated that predicted parameter setting found with PIAC can outperform a robust parameter setting, and approach the best parameter setting, whereas remains outperformed by a specialized parameter setting found with an AC method like SMAC.

However, the features computed on these empirical studies requires a large sample of the objective function, making the PIAC methodology unfeasible for real-world conditions where the objective function might be costly or when the overall budget of function is limited. Therefore, the following experiments will investigate the PIAC methodology on CMA-ES, when a limited budget for the optimization process is allowed, such that only a low budget of sample is available for computing the features.

4 FEATURES FOR CONTINUOUS DOMAIN

In the black-box context, only samples of function values can be used to compute the features[2]. We will hence assume the existence of a sample set $\mathcal{S}$ i.e. a set of pairs $\{(x_i, f(x_i)) \in \Omega \times \mathbb{R} | i \in [1, n]\}$. The set $\{x_i | i \in [1, n]\}$ (resp. $\{f(x_i) | i \in [1, n]\}$) is denoted $\mathcal{X}$ (resp. $\mathcal{Y}$). The computational cost of acquiring $\mathcal{S}$ from scratch is hence n function evaluations.

A first set of features was proposed in [24] and validated on the classification of the BBOB test functions. These features are grouped into six classes: the *y−Distribution* features are related to the distribution of the values of $\mathcal{Y}$, the *Levelset* features refers to the relative position of $\mathcal{Y}$ w.r.t. given thresholds, the *Meta-Model* features are based on meta-modeling of $\mathcal{S}$ set w.r.t. simple regression models (linear and quadratic), the *Curvature* features rely on some numerical estimations of the gradient and the Hessian of f from $\mathcal{S}$, the *Convexity* features are based on some empirical probability of convexity, and the *Local Search* features on the ratio between local and global optima, estimated using some iterated local search procedure starting from points in $\mathcal{X}$. *Dispersion* features were proposed in [23], based on comparisons of the distances between the best samples from different percentiles of $\mathcal{X}$ (in term of solution quality), to the mean and median distance between all samples. Finally, Munoz et al. [25] proposed a set of *Information Content* features, giving information about the global structure of the landscape.

In terms of computational costs, the *y−Distribution*, the *Levelset* and the *Meta-Model* features can all be evaluated on $\mathcal{S}$ without any additional evaluation. On the other hand, a large number (usually unknown a priori) of additional evaluations are required to compute the other features. The orders of magnitudes are about $10^3 \times d$ for the *Convexity* features, around $10^4 \times d$ for the *Curvature* and the *Local Search* features, following [24].

The discriminative power of these features was assessed in the context of classification of optimization problems [6, 20, 23–25], and of Algorithm Configuration [5, 26].

However, all *aforementioned* works used a large sample size ($\geq 500 \times d$) to compute features. And as expected, the error of computing the features increases the budget decreases, as demonstrated with a limited budget of $\leq 100 \times d$ in [6, 20]. This drawback can be somehow limited by building a surrogate model $\widehat{f}$ from $\mathcal{S}$ and gathering further samples using $\widehat{f}$ without additional evaluations of f [6, 20]. However, there is room for improvement there too – and this is one of the goals of the present work.

5 CMA-ES

Covariance Matrix Adaptation-Evolution Strategy (CMA-ES) [11] or some of its variants (e.g., the BI-POP-CMA-ES [9]) is considered today one of the state of the art optimizer for continuous black box optimization. It evolves a multi-variate Gaussian distribution, adapting online its covariance matrix (and some scaling parameter called the step-size). It is considered quasi-parameter-free, thanks to its invariance properties [12], that allowed to determine default values for almost all its parameters on a small set of functions.

However, as discussed in Section 2, there is still room for improvement in specific situations (e.g., expensive functions [3]). For instance, the on-line adaptation of three parameters of the adaptation mechanism of CMA-ES, the learning rates c_1, c_μ and c_c, was demonstrated to outperform the static version of CMA-ES [22] on some test functions from BBOB benchmark (Section 6): This was one motivation to try improving over BI-POP-CMA-ES using the PIAC approach described in Section 3.

[2] the only external feature is the dimension d of the search space

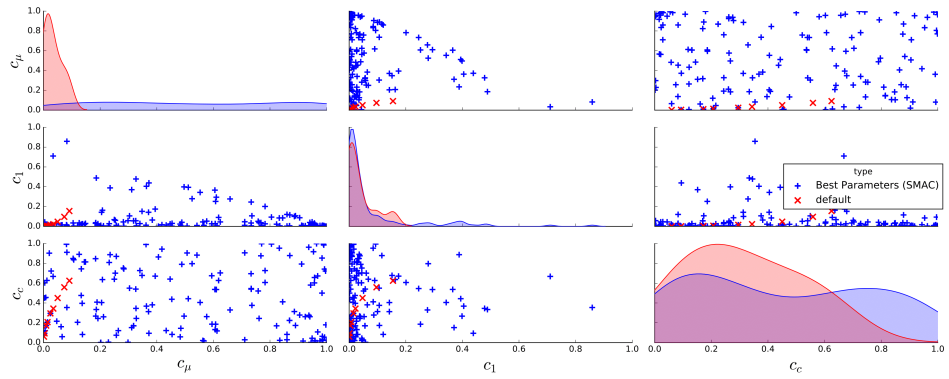


Figure 2: Comparison of the default (×) and best (+) parameter setting for c_1, c_μ and c_c parameters of CMA-ES, for each function from BBOB and each dimension $d \in \{2, 4, 5, 8, 10, 16, 20, 32, 40, 64\}$. Two axes projections are displayed out-diagonal, while density distributions for the corresponding parameter are shown on the diagonal.

A further hint that these parameters might be good candidates for per instance configuration is given in Figure 2. For each function of the BBOB suite (Section 6) and each dimension, a single-instance AC is run for the configuration of c_1, c_μ and c_c, using SMAC framework[3] [16] optimizing the accuracy of the solution found within a fixed (small) budget of $10^3 \times d$ function evaluations. The resulting optimal parameters are displayed in Figure 2 (blue '+') together with the default values (red 'x'), demonstrating that specialized tuning can be quite different from the overall robust seting.

6 TEST BENCHES

Two sets of test functions have been used in this work: the BBOB testbench was used as training set for learning the EPM, while an original set of 21 analytically defined functions gathered from different sources of optimization literature were used as test set to validate the approach.

The **Black Box Optimization Benchmark** (BBOB) [10] is made of 24 functions analytically defined on $[-5, 5]^d$, with known global optima. They have been manually classified in five classes of problems with respect to their global properties (e.g. separability, multimodality,…) and have been used in many of the works cited in this paper as testbench. The EPM is learned from a training set made of the 24 test functions in different dimensions $d \in \{2, 4, 5, 8, 10, 16, 20, 32, 40, 64\}$.

As advocated in the original framework, each of the 15 independent runs that are run on each function actually use a variant of the original function, obtained by a translation of the optimum and, for the non-separable functions, a rotation of the coordinate system.

The **Validation Test Functions** are completely independent of the BBOB testbench, in contrast with [5, 26], where cross-validation over the BBOB test functions was used to assess the performance of the proposed approaches. We selected 21 test functions from the literature [2, 13, 28], such that they are d-dimensional, the global optimum and bounds are known, and they are not included into the BBOB test functions.. In agreement with the BBOB methodology, all 15 independent experiments for each function are run on variants of the original function, as described above. Furthermore, the validation runs are performed for dimensions $d \in \{2, 4, 8, 10, 16, 20, 32, 40, 50, 66, 100\}$, i.e., for some additional dimensions that were not used on the BBOB testbench during the training phase.

The analytical definitions of these functions are not included here due to space constraints, but can be found at URL https://goo.gl/CrcFcn together with the contour plots in 2D, or, for some of them, at URLs http://al-roomi.org/benchmarks/unconstrained/n-dimensions and http://www.sfu.ca/~ssurjano/calibrat.html with some discussions of their properties and the optimum value.

7 EXPERIMENTAL SETTING

The Features: Following the discussion in Section 4, and in agreement with [5]; in addition to the dimension d, that is given 'for free', the following experiments only consider features that are computed from the same fixed size sample $\mathcal{S}$ of objective function values, without any specific additional function evaluations: 9 *Meta-Model*, 5 *Information Content*, 18 *Levelset*, 3 *y-Distribution* and 16 *Dispersion* features: the feature space $\mathbf{\Psi}$ is of size 51[4].

The sample size n is set to $k \times d$ for some $k \in \mathbb{N}$, and different values of k are be investigated ($k \in \{10, 30, 50\}$). For each k, two different approaches to the $k \times d$ budget for feature computations are also be investigated: the features can be computed directly from $\mathcal{S}$ (denoted $\boldsymbol{\psi}_k$), or, following [6], a surrogate model $\widehat{f}$ of f can be built from $\mathcal{S}$ and used as a proxi to compute some "surrogate-assisted features" denoted $\widehat{\boldsymbol{\psi}}_k$. In the latter case, as in [6], a Random Forest regressor is used to learn $\widehat{f}$ (using the Scikit-Learn library implementation with 20 trees and maximal a depth of 500).

The Empirical Performance Model: Following [5, 18], the empirical performance model is learned with a Random Forest regressor, using the same meta-parameters than [5], i.e., 10 trees and a maximal depth of 200 – here again, using the Scikit-Learn library

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

[4] All features were computed using the R package gracefully provided by Pascal Kerschke at http://github.com/flacco

implementation.

The Experimental Protocol follows the approach described in Section 3.

First, for each of the 24 test functions from BBOB testbench, and for each dimensions listed in Section 6, 15 independent sample objective functions are generated and their respective features computed (some features can slightly differ for different sample of the same function, in accordance with Belkhir et al. [6] that observed a higher in the features values when they are computed from small samples), and the best parameter setting of CMA-ES, as discovered by SMAC, are computed and their performance kept as the upper bound. The performance of the default CMA-Es setting is also computed, to be used as baseline results. From this sample, the EPM is learned.

The validation of the EPM is done using the 21 Validation Test Functions and the 11 dimensions described in Section 6. Their features $\boldsymbol{\psi}_k$ and $\widehat{\boldsymbol{\psi}}_{\boldsymbol{k}}$ are computed using $k \times d$ uniformly drawn samples, and the empirical optimal parameters are computed accordingly (Section 3). The versions of CMA-ES using these empirical optimal parameters are denoted EPM-CMA-ES-$\boldsymbol{\psi}_k$ and EPM-CMA-ES-$\widehat{\boldsymbol{\psi}}_{\boldsymbol{k}}$ respectively, and they are computed to the default CMA-ES[5].

For all optimization runs, the maximum evaluation budget is set to $10^3 \times d$, and the target precision to be reached is $\Delta_f = 10^{-6}$.

Empirical Cumulative Distribution Functions (ECDF) plots are used in next Section to present all the results: they display the proportion of solved instances (to the target precision) vs the number of function evaluations (normalized by the dimension). Such plots, proposed in the BBOB workshops [10] allow to aggregate many results into a single plots (e.g., all runs for a given function, all functions for a given dimension, ...).

8 RESULTS

Figure 3 compares the ECDF, aggregated per dimensions, of different sample sizes k and computation methods $\boldsymbol{\psi}$ and $\widehat{\boldsymbol{\psi}}$ as well as those of the default CMA-ES.

First, regarding the computation of features $\boldsymbol{\psi}$, we observe a clear preference of a larger sample size $k > 30$. As it could be expected, when k decreases, the performance of EPM-CMA-ES decreases, such that $k = 50$ is the best setting to compute features.

As could be expected, the quality of the empirical performance model tends to greatly improve when k increases. The results are more visible for $d > 8$, and some improvement of almost 20% over default CMA-ES is observed with $\boldsymbol{\psi}_{\mathbf{50}}$.

About the use of surrogate assisted features $\widehat{\boldsymbol{\psi}}$, the results are rather mixed regarding the sample size and dimensions. When $k = 50$, the choice between $\widehat{\boldsymbol{\psi}_{\mathbf{50}}}$ and $\boldsymbol{\psi}_{\mathbf{50}}$ is unclear, whereas when d is small, surrogate assisted features tend to slightly improve the performance. But to our surprise, the surrogate assisted features are clearly worse when $d > 8$, in contrast to the results for small k.

Figure 4 displays typical ECDF comparing the simple $\boldsymbol{\psi}_{\mathbf{50}}$ approach with the $\boldsymbol{\psi}_{\mathbf{50}} - \theta_{new}$ restart strategy (and the baseline default CMA-ES). The results are similar for all the dimensions. While we can observe a faster convergence on some dimensions for EPM-CMA-ES without θ_{new}, no strategy significantly and repeatedly outperforms the other.

Figure 5 displays some typical results of EPM-CMA-ES and default CMA-ES when run beyond the $10^3 \times d$ budget that was used for training: the advantage of EPM-CMA-ES remains clear, even though the surrogate assisted features tend to stay at the same level.

9 DISCUSSION

In agreement with [5], the results presented here first show that the PIAC approach of learning an EPM from a large sample of results of runs of different parameter settings on different instances can be successful, outperforming the default settings of CMA-ES, an algorithm which is known for its robustness parameterwise.

The computation of features is the core element of the empirical performance model and its predictive power. Belkhir et al. [6] had empirically shown that the accuracy of the feature critically depends on their computational costs. The present study was focused on low budgets to compute features. From the results, it is clear that reducing the computational cost of the feature indeed reduces the efficiency of the approach (e.g., for $k < 50$), even more so when the dimension of problems increases. For very low sample sizes ($10 \times d$), the improvement over the default CMA-ES becomes insignificant, hence making the PIAC methodology useless.

The surrogate-assisted features use a surrogate model of the objective function to artificially add samples (of the surrogate model) to the training set for the computation of the features. However, the efficiency of this approach for PIAC is not as clear as for the task of retrieving the BBOB classification, as in [6]. On the one hand, surrogate assisted features seems beneficial when the dimension is low ($d < 16$), or when the initial sample size is large enough to correctly learn the surrogate model – and $50 \times d$ might still be a little small, in particular in large dimensions. As a matter of fact, surrogate assisted features are completely misleading when the dimension increases, and in particular with even lower sample sizes (10 or $30 \times d$). The final conclusion at the light of these experiments seems to be that directly computing the features from the initial sample, however small, remains the best option. However, this opens a path for research around the improvement of features computation from small samples, hence suggesting to explore other method to approximate an objective function, ... or to come up with new features that are less sensitive to the sample size.

Even if EPM-CMA-ES is globally better than CMA-ES, some of the test functions must be distinguished from the others. In particular, when run on the Needle Eye, or the Happycat test functions, all variants fails to come even close to the optimum. These test functions were intentionally included with the hope that the PIAC based tuning would allow the algorithm to more quickly reach the surrounding of the needle optimum, and hence have more iterations to eventually find it, but this did not happen. Indeed, Needle Eye has a plateau shape, on which CMA-ES is known to perform poorly, and BBOB is known to have a few test functions with plateaus. However, the EPM was not able to learn more successful setting for plateaus, and additional experiments should be made including more examples of test functions with plateaus.

[5] According to [5], a specialized tuning is the clear winner, but the predicted setting outperforms a robust setting for a set of problem instances

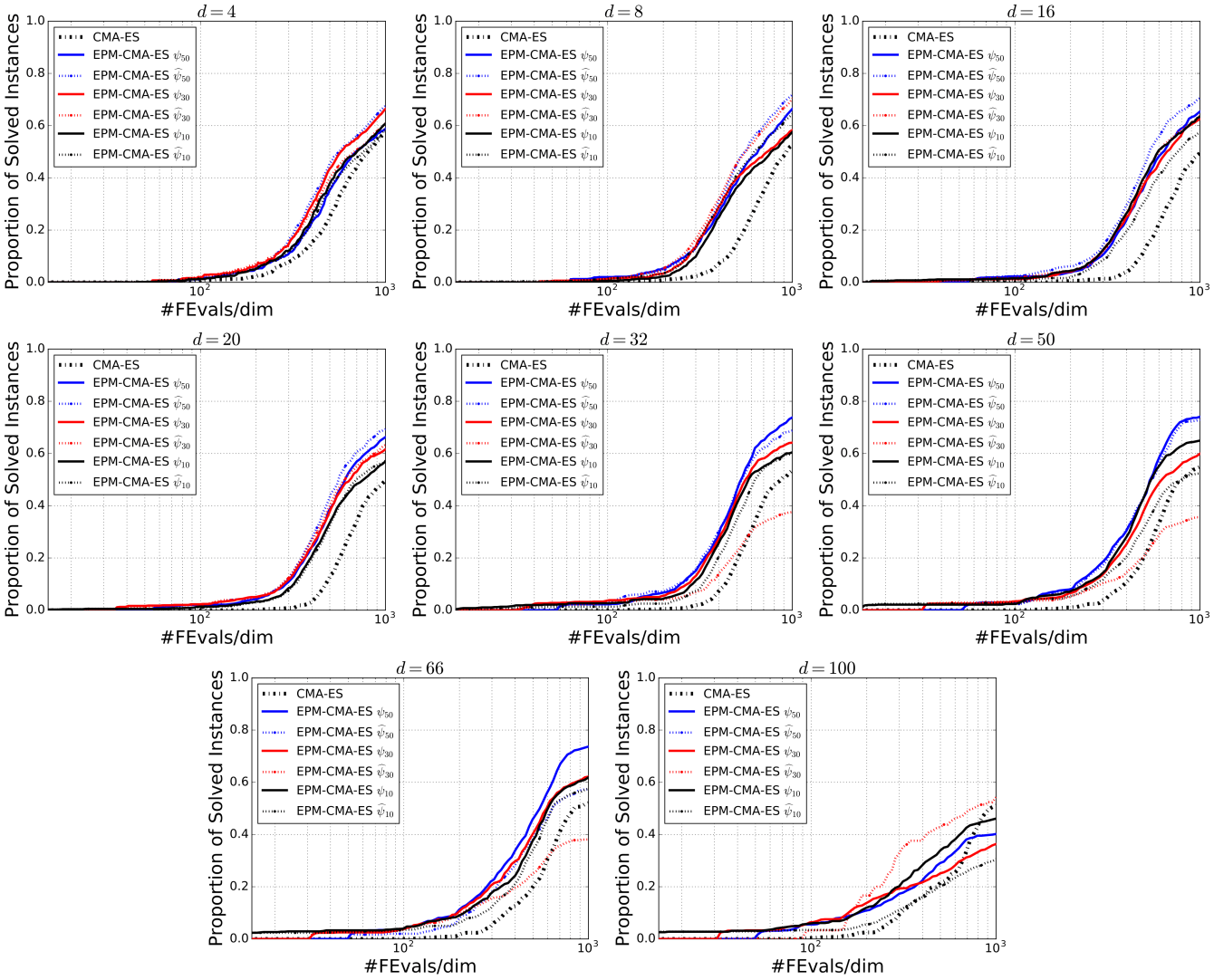


Figure 3: ECDF comparing EPM-CMA-ES with ψ_k or $\widehat{\psi_k}$ ($k \in \{10, 30, 50\}$), and default CMA-ES.

After detailed observations, we proposed and investigated an alternative strategy after a stopping criterion of BIPOP-CMA-ES is met, such that at each restart the features are recomputed and a the corresponding parameter setting is applied to CMA-ES. Because of their poor performances, surrogate assisted features have not been tested together with this alternative strategy. However, while few improvements can be observed on some functions and dimensions, the additional cost of re-computing the features tend to hinder the performance too much, as can be observed on $d = 8$ for instance. Also, in such experimental conditions where the maximum function evaluations is rather small ($10^3 \times d$), the effect of such alternative strategy might not be visible. Nevertheless, additional experiments with a larger budget ($10^4 \times d$ as in Figure 5), did not exhibit any improvement, with small plateaus representing restart with no improvements

Finally, we empirically investigated the PIAC methodology such that an empirical performance model is learned from a large set of test functions with different level of difficulty, and taking into consideration different dimensions of problems (up to 64). The experimental setup aimed at empirically investigating EPM-CMA-ES for test functions with dimensions considered in the learning phase or dimensions in the same range, or close to them, e.g. $d = 50$ or $d = 66$. We observed that the empirical performance model was able to generalize to dimensions close to those of the learning set. By contrast, when test functions and dimensions are very different, e.g. $d = 100$ in Figure 4, the results of EPM-CMA-ES are worse than those of the default CMA-ES, strongly suggesting that predicted parameter setting are poor. Such results questions the generalization power of EPM-CMA-ES to larger dimensions, hence requiring that the EPM is learned on all dimensions that are likely to be considered during the later decision phase.

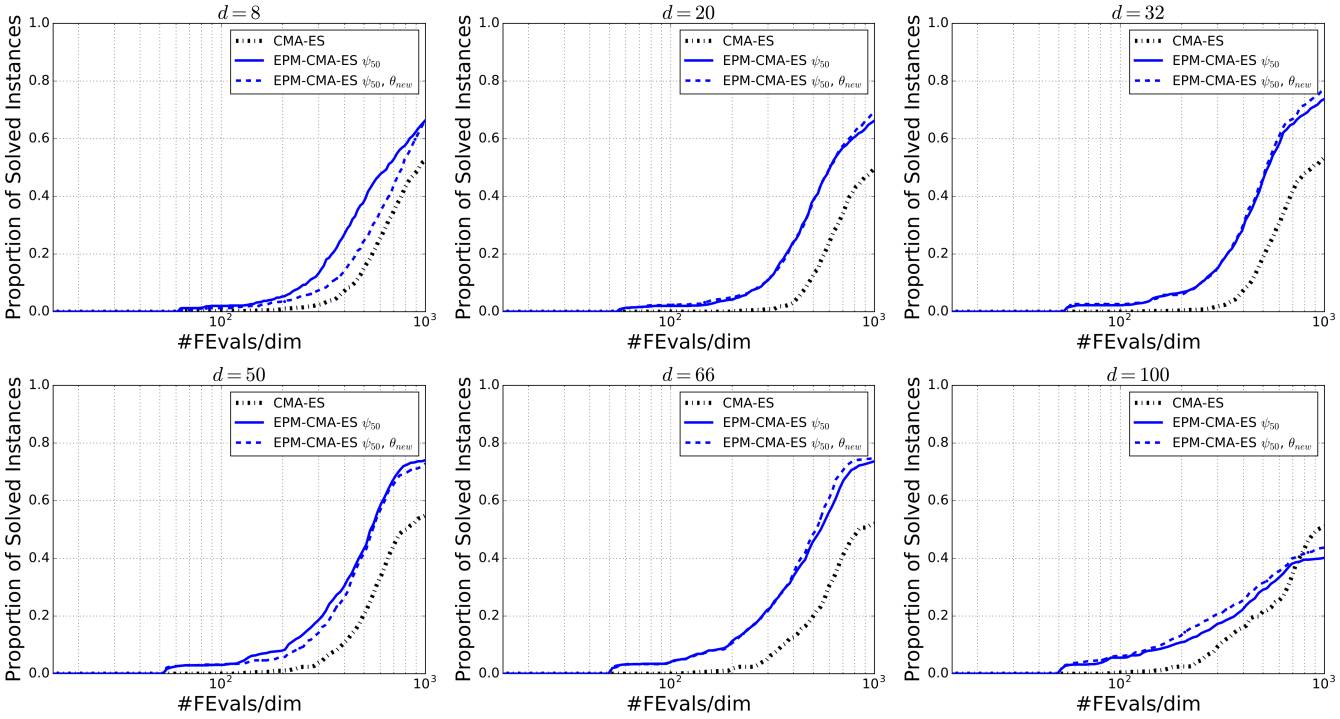


Figure 4: ECDF comparing ψ_50 without and with the alternative restart strategy (θ_{new}), and the default CMA-ES.

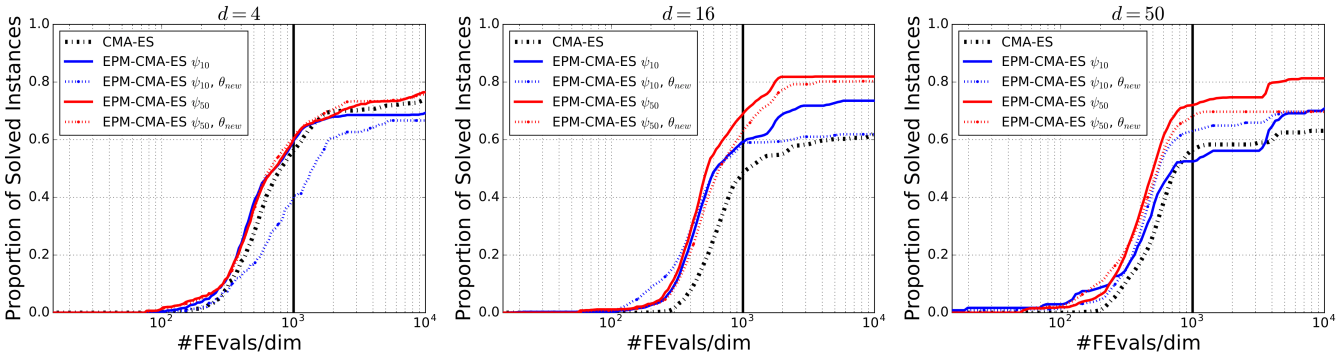


Figure 5: Typical Results of ECDF of EPM-CMA-ES compared to CMA-ES beyond the $10^3 \times d$ initial budget limit

Finally, when a new problem instance g is processed, the features $\psi(g)$ are computed and the empirical best setting for g is found by solving the auxiliary optimization problem. At the moment, for the three parameters involved in our experiments, this is done by a brute grid search. However, one advantage of the PIAC approach over cluster-based methods such as ISAC [1, 19] is that any optimization algorithm could be used on the EPM thus being able to handle any (reasonnable) number of distinct parameters, whereas the cluster approach would suffer a much more intense curse of dimensionality. Another drawback of cluster-based approach is that they rely on some class-based meta-optimization (see Section 2.2) that will give poor results as the size of the clusters increases, hindering their homogeneity with respect to parameter setting. On the other hand, the PIAC approach will always take into account the specific features of the instance at hand.

10 CONCLUSION AND FURTHER WORKS

This paper has presented the validation of the Per Instance Algorithm Configuration methodology on BIPOP-CMA-ES, one of the state of the art optimizers for continuous black box optimization. We empirically investigated the computation and the use of an Empirical Performance Model (EPM) with a small overall budget of $10^3 \times d$, de facto implying some low upper bound on the computation cost of the feature themselves. As in many previous works, the EPM was trained on the famous BBOB benchmark. However, it was

tested on independent test functions from the optimization literature. The good results (the EPM-CMA-ES consistently outperforms the default CMA-ES) also demonstrate that the BBOB benchmark is a good testbench for continuous optimization, at least for the class of artificial analytically defined test functions.

An incremental strategy that recomputes the instance features at each restart of CMA-ES using some additional samples did not show significant improvement, raising the issue of dynamic feature (re)computation. Additional samples are de facto gathered by the optimization algorithm, and could be used to compute the features more accurately as the fitness landscape changes. This on-line parameter control is the subject of on-going work.

Other directions of further research concern the different levels of the PIAC methodology. The choice of the learning method for EPM should include more detailed investigation of rank based methods, and the use of Deep Networks). Similarly, other approaches for learning the surrogate $\widehat{f}$ in the surrogate assisted feature approach should be investigated.

Finally, a promising research path is to extend the use of the Empirical Performance Model to the full Algorithm Selection and Configuration problem, as in [31, 34]: the EPM is used to predict the best algorithm with the best parameter setting for a new and unseen problem.

REFERENCES

[1] Tinus Abell, Yuri Malitsky, and Kevin Tierney. 2012. *Fitness landscape based features for exploiting black-box optimization problem structure.* Technical Report. Technical Report TR-2012-163, IT University of Copenhagen.

[2] Ernesto P Adorio and U Diliman. 2005. Mvf-multivariate test functions library in c for unconstrained global optimization. *Quezon City, Metro Manila, Philippines* (2005).

[3] M. Andersson, S. Bandaru, A. H. C. Ng, and A. Syberfeldt. 2015. Parameter tuned CMA-ES on the CEC'15 expensive problems. In *Proc. CEC'15)*. 1950–1957.

[4] Thomas Bartz-Beielstein, Christian WG Lasarczyk, and Mike Preuß. 2005. Sequential Parameter Optimization. In *CEC'05*, Vol. 1. IEEE, 773–780.

[5] Nacim Belkhir, Johann Dréo, Pierre Savéant, and Marc Schoenauer. 2016. Feature Based Algorithm Configuration: A Case Study with Differential Evolution. In *International Conference on Parallel Problem Solving from Nature*. Springer, 156–166.

[6] Nacim Belkhir, Johann Dréo, Pierre Savéant, and Marc Schoenauer. 2016. Surrogate assisted feature computation for continuous problems. In *International Conference on Learning and Intelligent Optimization*. Springer, 17–31.

[7] Mauro Birattari, Thomas Stützle, Luis Paquete, Klaus Varrentrapp, and others. 2002. A Racing Algorithm for Configuring Metaheuristics.. In *GECCO*, Vol. 2. 11–18.

[8] Jakob Bossek, Bernd Bischl, Tobias Wagner, and Günter Rudolph. 2015. Learning Feature-Parameter Mappings for Parameter Tuning via the Profile Expected Improvement. In *Proceedings of the 2015 on Genetic and Evolutionary Computation Conference*. ACM, 1319–1326.

[9] Nikolaus Hansen. 2009. Benchmarking a BI-population CMA-ES on the BBOB-2009 Function Testbed. In *GECCO Companion*, Franz Rothlauf (Ed.). ACM, 2389–2396.

[10] Nikolaus Hansen, Anne Auger, Steffen Finck, and Raymond Ros. 2010. *Real-Parameter Black-Box Optimization Benchmarking 2010: Experimental Setup.* Technical Report RR-7215. INRIA.

[11] Nikolaus Hansen and Andreas Ostermeier. 2001. Completely derandomized self-adaptation in evolution strategies. *Evolutionary computation* 9, 2 (2001), 159–195.

[12] Nikolaus Hansen, Raymond Ros, Nikolas Mauny, Marc Schoenauer, and Anne Auger. 2011. Impacts of Invariance in Search: When CMA-ES and PSO Face Ill-Conditioned and Non-Separable Problems. *Applied Soft Computing* 11 (2011), 5755–5769. https://doi.org/10.1016/j.asoc.2011.03.001

[13] Willi Hock and Klaus Schittkowski. 1980. Test examples for nonlinear programming codes. *Journal of Optimization Theory and Applications* 30, 1 (1980), 127–129.

[14] Holger H Hoos. 2012. Programming by optimization. *Comm. of the ACM* 55, 2 (2012), 70–80.

[15] Frank Hutter, Youssef Hamadi, Holger H Hoos, and Kevin Leyton-Brown. 2006. Performance prediction and automated tuning of randomized and parametric algorithms. In *Intl Conf. on Principles and Practice of Constraint Programming*. Springer, 213–228.

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

[17] Frank Hutter, Holger H Hoos, Kevin Leyton-Brown, and Thomas Stützle. 2009. ParamILS: an automatic algorithm configuration framework. *Journal of Artificial Intelligence Research* 36, 1 (2009), 267–306.

[18] F. Hutter, L. Xu, H. H. Hoos, and K. Leyton-Brown. 2014. Algorithm runtime prediction: Methods & evaluation. *Artificial Intelligence* 206 (2014), 79–111.

[19] Serdar Kadioglu, Yuri Malitsky, Meinolf Sellmann, and Kevin Tierney. 2010. ISAC - Instance-Specific Algorithm Configuration. In *ECAI*, Vol. 215. 751–756.

[20] Pascal Kerschke, Mike Preuss, Simon Wessing, and Heike Trautmann. 2016. Low-budget exploratory landscape analysis on multiple peaks models. In *Proceedings of the 2016 on Genetic and Evolutionary Computation Conference*. ACM, 229–236.

[21] Kevin Leyton-Brown, Eugene Nudelman, and Yoav Shoham. 2002. Learning the empirical hardness of optimization problems: The case of combinatorial auctions. In *Principles and Practice of Constraint Programming-CP 2002*. Springer, 556–572.

[22] Ilya Loshchilov, Marc Schoenauer, Michele Sebag, and Nikolaus Hansen. 2014. Maximum Likelihood-based Online Adaptation of Hyper-parameters in CMA-ES. In *PPSN XIII*, Th. Bartz-Beielstein et al. (Ed.). LNCS 8672, Springer Verlag, 70–79.

[23] Monte Lunacek and Darrell Whitley. 2006. The dispersion metric and the CMA evolution strategy. In *Proc. 8th GECCO*. ACM, 477–484.

[24] Olaf Mersmann, Bernd Bischl, Heike Trautmann, Mike Preuss, Claus Weihs, and Günter Rudolph. 2011. Exploratory landscape analysis. In *Proc. 13th GECCO*. ACM, 829–836.

[25] Mario Munoz, Michael Kirley, Saman K Halgamuge, and others. 2015. Exploratory landscape analysis of continuous space optimization problems using information content. *Evolutionary Computation, IEEE Transactions on* 19, 1 (2015), 74–87.

[26] Mario A Muñoz, Michael Kirley, and Saman K Halgamuge. 2012. A meta-learning prediction model of algorithm performance for continuous optimization problems. In *PPSN XII*. Springer, 226–235.

[27] Volker Nannen and Agoston E Eiben. 2007. Relevance Estimation and Value Calibration of Evolutionary Algorithm Parameters.. In *IJCAI*, Vol. 7. 6–12.

[28] K Shittowski. 1987. More test examples for nonlinear programming codes. *More test examples for nonlinear programming codes* (1987).

[29] Rainer Storn and Kenneth Price. 1997. Differential evolution–a simple and efficient heuristic for global optimization over continuous spaces. *Journal of global optimization* 11, 4 (1997), 341–359.

[30] D.H. Wolpert and W.G. Macready. 1997. No free lunch theorems for optimization. *Evolutionary Computation, IEEE Trans. on* 1, 1 (1997), 67–82.

[31] Lin Xu, Holger Hoos, and Kevin Leyton-Brown. 2010. Hydra: Automatically Configuring Algorithms for Portfolio-Based Selection.. In *AAAI*, Vol. 10. 210–216.

[32] Lin Xu, Frank Hutter, Holger H Hoos, and Kevin Leyton-Brown. 2007. SATzilla-07: the design and analysis of an algorithm portfolio for SAT. In *Intl Conf. on Principles and Practice of Constraint Programming*. Springer, 712–727.

[33] Lin Xu, Frank Hutter, Holger H Hoos, and Kevin Leyton-Brown. 2008. SATzilla: portfolio-based algorithm selection for SAT. *Journal of Artificial Intelligence Research* (2008), 565–606.

[34] Lin Xu, Frank Hutter, Holger H Hoos, and Kevin Leyton-Brown. 2011. Hydra-MIP: Automated algorithm configuration and selection for mixed integer programming. In *RCRA workshop on experimental evaluation of algorithms for solving problems with combinatorial explosion at the international joint conference on artificial intelligence (IJCAI)*. 16–30.