Accelerating Evolutionary Algorithms with Gaussian Process Fitness Function Models

Dirk Büche, Nicol N. Schraudolph, and Petros Koumoutsakos

***Abstract*— We present an overview of evolutionary algorithms that use empirical models of the fitness function to accelerate convergence, distinguishing between evolution control and the surrogate approach. We describe the Gaussian process model and propose using it as an inexpensive fitness function surrogate. Implementation issues such as efficient and numerically stable computation, exploration versus exploitation, local modeling, multiple objectives and constraints, and failed evaluations are addressed. Our resulting Gaussian process optimization procedure clearly outperforms other evolutionary strategies on standard test functions as well as on a real-world problem: the optimization of stationary gas turbine compressor profiles.**

***Index Terms*— evolutionary algorithms, fitness function modeling, evolution control, surrogate approach, Gaussian process, gas turbine compressor design**

I. Introduction

THE cost of optimizing expensive problems is dominated by the number of fitness function evaluations required to reach an acceptable solution. For evolutionary algorithms, various approaches exist to reduce this cost by exploiting knowledge of the history of evaluated points. This knowledge can for instance be used to adapt the recombination and mutation operators in order to sample offspring in a promising areas. Thus the covariance matrix adaptation (CMA) algorithm [1], [2] uses the path of successful mutations to build up a covariance matrix; the new population is then sampled with this covariance.

Knowledge of past evaluations can also be used to build an empirical model that approximates the fitness function to optimize. The approximation is then used to predict promising new solutions at a smaller evaluation cost than the original problem. The prediction quality generally improves with a growing number of evaluated points in the optimization process. Such models are also referred to as surrogates [3], [4], response surfaces (especially for polynomial approaches), or metamodels [5], [6]. A prerequisite for using them is that the expense of model construction and prediction is lower than evaluating the fitness function; they are thus used primarily for expensive optimization problems.

Manuscript received August 31, 2003; revised January 31, 2003. This work was supported in part by Alstom Power Technology Center Dättwil, Switzerland, and the Swiss Commission for Technology and Innnovation CTI project 4817.1. This paper was recommended by Guest Editor Y. Jin.

Dirk Büche and Petros Koumoutsakos are with the Institute of Computational Sciences, Swiss Federal Institute of Technology (ETH), Zürich, Switzerland (e-mail: bueche@inf.ethz.ch, petros@inf.ethz.ch).

Nicol N. Schraudolph is an independent consultant in Zürich, Switzerland (e-mail: gpop@schraudolph.org).

Digital Object Identifier 10.1109/TSMCC.2004.841917

II. Models in Evolutionary Algorithms

There are two main ways to integrate models into an evolutionary optimization. In the first, a fraction of individuals is evaluated on the fitness function itself, the remainder merely on its model. Jin *et al.* [7] refer to individuals that are evaluated on the fitness function as *controlled*, and call this technique *evolution control*. In the second approach, the optimum of the model is determined, then evaluated on the fitness function. The new evaluation is used to update the model, and the process is repeated with the improved model. This *surrogate approach* evaluates only predicted optima on the fitness function, otherwise using the model as a surrogate.

A. Evolution Control

In evolution control [7], a *controlled* fraction of individuals are evaluated on the fitness function, the remainder only on the model. Assuming perfect approximation of the fitness function by the model, and computational cost dominated by the fitness function evaluation, this produces a relative reduction in cost equal to the fraction of uncontrolled individuals. Various implementations can be distinguished according to their selection of controlled individuals. Jin *et al.* [7] define two main classes of control rule: in *individual-based* evolution control a fraction of each population is controlled, while in *generation-based* evolution control the entire population is either controlled or uncontrolled.

In individual-based evolution control it is still an open question which of the individuals should be controlled, and the fraction of controlled individuals varies between 10% [5] and 50% [8]. Jin *et al.* [8] state that when randomly controlling individuals, about 50% of the offspring need to be controlled. When pre-evaluating all solutions on the model, then evaluating the best individuals on the fitness function, Giotis *et al.* [9] and Jin *et al.* [8] control about 40% of the population. Emmerich *et al.* [5] show that for simple problems such as, *e.g.*, a symmetric quadratic function, controlling the best 10% of pre-evaluated individuals is sufficient.

For more complex problems such as multimodal or correlated functions, however, their algorithm easily gets stuck. They address this problem by using a merit function as proposed by Torczon and Trosset [3]. Merit functions are a weighted sum of the model prediction and a negative density measure. The density measure promotes unexplored regions, *i.e.*, areas where no points have yet been evaluated. Thus, merit functions balance the goal of finding promising solutions (exploitation) with improving the model by obtaining information about new regions (exploration), thus decreasing the risk of premature convergence [3], [10].

Beltagy and Keane [11] employ Gaussian process models [12], which provide an uncertainty measure (in terms of a standard deviation) along with the predicted fitness function value. They only control individuals whose predicted standard deviation exceeds a certain limit, assuming the model prediction to be accurate otherwise. The limit is decreased linearly over the course of the evolution.

In generation-based evolution control, the entire population is evaluated on either the model or the fitness function; this allows for better parallelization. Again, various rules have been developed to determine which generations to control. Ratle [13] controls the first generation of his GA; subsequent generations are evaluated only on the model until the model predictions do not improve for a given number of generations. The next generation is again controlled. Compared to the original GA, this approach accelerated convergence for uni- and multimodal functions. Jin *et al.* [14] control generations until the error between model prediction and fitness function drops below a certain threshold. The population then remains uncontrolled for a given number of generations before control resumes.

For both control methods several questions remain open. First, there is disagreement about which and how many individuals of a population need to be controlled. Then it is not clear whether all or just a fraction of evaluated points should be used for model construction, so as to perform a global [11], respectively local (*i.e.*, recent) approximation [5]. The model's complexity must also be appropriate for the amount of data used in order to avoid overfitting the data.

Finally, inexact model predictions may mislead the selection operator to propagate inferior individuals. This may be especially detrimental to optimization algorithms that are themselves adaptive. We hypothesize:

> The more information from the population is exploited by the evolutionary algorithm (*e.g.*, to adapt the mutation distribution), the higher the fraction of controlled individuals has to be in order to provide sufficient information for the adaptation process.

In other words, evolution control is based on the assumption that the evolutionary algorithm needs very little information, which can be provided by evaluating (controlling) just a fraction of the population. However, this argumentation holds only for those inefficient algorithms that indeed use little information — for "smarter" algorithms such as CMA [1], [2] that pull much more information out of a population, we expect that virtually all individuals must be controlled.

The success of evolution control is thus highly dependent on the fraction of controlled individuals, which is difficult to determine as it depends on both the fitness function complexity and the optimization algorithm. It is always a compromise between avoiding the computational cost of fitness function evaluation and the danger of a poor model misleading the optimization [14].

B. Surrogate Approach

Def

In the surrogate approach, a fitness function model is constructed for an initial training set of evaluated points. An optimization algorithm then searches for the optimum of the model's fitness prediction. The predicted optimum constitutes an ideal candidate for an improved solution to the problem, and is therefore evaluated on the fitness function. The result of the evaluation is added to the model's training data, facilitating an improved approximation of the fitness function by the model. The procedure then iterates by searching for the optimum of the improved model. This surrogate approach has been discussed by Torczon *et al.* [3]; similar methods can be found in [15], [13], [16], [10].

Similar to evolution control, the surrogate approach is in danger of getting stuck in a local minimum unless points in unexplored regions of the search space are added to the model's training set. Thus, merit functions (or similar techniques) are also in use here [3], [10]. Open issues again include the question of whether global or local modeling should be preferred. For local models, the search for the optimum must be limited to the region that is well-approximated by the model. The potential reduction in computational cost is higher for the surrogate approach than for the evolution control, especially once enough data is available to allow for construction of a model that is accurate near the true optimum. Since the surrogate approach proceeds sequentially from one predicted optimum to the next, however, it is more difficult to parallelize.

C. Empirical Models

A wide variety of empirical models are used in the literature as fitness function models for an optimization procedure. The most prominent among them are polynomial models [17], artificial neural networks [8], radial basis function networks [18], [10], and Gaussian processes [13], [7], [11], [5].

Polynomial Models: Polynomial fitness function models — also referred to as *response surfaces* — can easily be fitted to data with a least squares approach (see, *e.g.*, [17]). The order of the polynomial is important: quadratic or cubic polynomials are mainly used, with quadratic polynomials best suited for continuous, unimodal problems. Since higher-order polynomial fits tend to oscillate too much, multimodal shapes are better approximated by splines, *i.e.*, piecewise polynomial fits with continuity constraints at the boundaries.

Artificial Neural Networks: Artificial Neural Networks (ANNs) [19] consist of a large number of highly interconnected processing units, each aggregating information from a large number of connected peers. Given a sufficient number of units, an ANN can approximate any function. An ANN can be trained by adapting the weights that specify the amplification of signals between connected units. Among the most popular training algorithms for ANNs is *error back-propagation* [20].

A major disadvantage of ANNs is that the resulting weights of the trained network are difficult to interpret. Another problem is the difficulty of finding an appropriate network topology and size. In order to avoid both underfitting (*i.e.*, bad approximation of the training data) resulting from too small networks, and overfitting (*i.e.*, bad generalization) due to too large networks, often several networks of different size and/or topology must be trained [21], and their performance compared on a separate set of test data to estimate their generalization properties.

Radial Basis Functions Networks: The output of a Radial Basis Function Network (RBFN) [19] is a weighted sum of radial basis functions, each characterized by its center $\mu \in \mathbb{R}^n$ in the design space, and a function which declines with increasing distance from the center. With Gaussian radial basis functions, for instance, the output $\hat{y}$ of an RBFN is given by

$$\hat{y}(\mathbf{x}) = \sum_{i=1}^{N_R} w_i \exp\left(-\frac{\left\|\mu_i - \mathbf{x}\right\|^2}{r^2}\right), \quad (1)$$

where N_R is the number of radial basis functions, w_i their weights, μ_i their centers, and r their rate of decay. Often each given data point is used as the center of a radial basis function. The weights are obtained by a least squares fit similar to the polynomial approach [19], [10]. A major difficulty in RBFNs is to set the decay parameter r, which has a large impact on the approximation. Only limited theory and some empirical formulæ exist to address this problem. In [10], the decay parameter is set as:

$$r = d_{\max}\left(n\ N_R\right)^{-1/n}, \quad (2)$$

where $d_{\max}$ is the maximal distance between the data.

Gaussian Processes: Gaussian processes (GPs) [12] specify a probabilistic model over a given set of data points, constructed such that the likelihood of the function value given the decision variable values is maximized for all data points. This model can then be extended to predict the mean and standard deviation of the function value at new data points. GPs have a small number of *hyperparameters* which can be set by the user, or optimized via a maximum likelihood approach.

Like ANNs, GPs can approximate any function. Their main advantage over ANNs is their simplicity: no network size or topology must be chosen. In contrast to the weights of an ANN, the hyperparameters of a GP have intrinsic meaning—specifying, *e.g.*, typical length scales—and can therefore be set using prior knowledge of the problem, such as noise levels, location of discontinuities, *etc*. One drawback of GPs is their computational cost: for N data points, it takes $\mathcal{O}(N^3)$ steps to construct the GP, $\mathcal{O}(N)$ to predict the mean function value at a new point, and $\mathcal{O}(N^2)$ to predict the standard deviation.

III. Gaussian Process Model

Among the above empirical models, Gaussian processes (GPs) appear the most promising to us for fitness function modeling, as they are the only approach to combine the following properties: a GP does not require a predefined structure, can approximate arbitrary function landscapes including discontinuities and multimodality, has meaningful hyperparameters, and includes a theoretical framework for optimizing these hyperparameters. A further advantage of GPs is that an uncertainty measure in form of a standard deviation is provided for predicted function values; we will make use of this in our proposed optimization procedure.

We define a GP using the notation of MacKay [12]: let $f(\mathbf{x})$ be an unknown scalar function and $\mathbf{x} \in \mathbb{R}^n$ a point in an n-dimensional decision space. Evaluating f at N data points $\mathbf{X}_N = \{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_N\}$ yields the function values $\mathbf{t}_N = \{t_1, t_2, \ldots, t_N\}$, where $(\forall i)\ t_i = f(\mathbf{x}_i)$. Note that subscripts are used here to indicate vector and matrix sizes (for $\mathbf{t}_N$ and $\mathbf{X}_N$) as well as to index elements of those vectors and matrices (*e.g.*, t_i, $\mathbf{x}_i$). The modeling task is to predict the function value t_{N+1} at a new point $\mathbf{x}_{N+1}$. In the following we present the main equations for GPs; for additional details refer to MacKay [12].

The GP imposes a probabilistic model on the given data, namely that the vector of known function values $\mathbf{t}_N$ is one sample of a multivariate Gaussian distribution with joint probability density $p(\mathbf{t}_N|\mathbf{X}_N)$. Similarly, when adding a new point $\mathbf{x}_{N+1}$, the resulting vector of function values $\mathbf{t}_{N+1}$ is assumed to be a sample of the Gaussian joint probability density $p(\mathbf{t}_{N+1}|\mathbf{X}_{N+1}) \equiv p(\mathbf{t}_N, t_{N+1}|\mathbf{X}_N, \mathbf{x}_{N+1})$. Note that the dimensionality of each probability density here equals the number of data points, and is independent of the dimensionality of the decision space n.

Using the rule of conditional probabilities, $p(A|B) = p(A,B)/p(B)$, we can write the probability density for t_{N+1} given the known data points as

$$p(t_{N+1}|\mathbf{X}_{N+1}, \mathbf{t}_N) = \frac{p(\mathbf{t}_{N+1}|\mathbf{X}_{N+1})}{p(\mathbf{t}_N|\mathbf{X}_N)} \quad (3)$$

This gives the probability density for the function value t_{N+1} at a new data point $\mathbf{x}_{N+1}$ as a univariate Gaussian, given the N known data points, their associated function values, and the location of the new data point. In the following we transform Eq. 3 so as to express the distribution of t_{N+1} in terms of its mean $\hat{t}_{N+1}$ and standard deviation $\sigma_{t_{N+1}}$. The multivariate Gaussian in the denominator on the right-hand side of Eq. 3 is

$$p(\mathbf{t}_N|\mathbf{X}_N) = \frac{\exp\left(-\frac{1}{2}\mathbf{t}_N^T\mathbf{C}_N^{-1}\mathbf{t}_N\right)}{\sqrt{(2\pi)^N \det(\mathbf{C}_N)}} \quad (4)$$

where $\mathbf{C}_N$ is the covariance matrix of the Gaussian distribution, and its mean has been set to zero. Similarly we obtain for $N+1$ data points

$$p(\mathbf{t}_{N+1}|\mathbf{X}_{N+1}) = \frac{\exp\left(-\frac{1}{2}\mathbf{t}_{N+1}^T\mathbf{C}_{N+1}^{-1}\mathbf{t}_{N+1}\right)}{\sqrt{(2\pi)^{N+1} \det(\mathbf{C}_{N+1})}} \quad (5)$$

The covariance matrix for the $N+1$ points can be written as

$$\mathbf{C}_{N+1} = \begin{pmatrix} \mathbf{C}_N & \mathbf{k} \\ \mathbf{k}^T & \kappa \end{pmatrix}, \quad (6)$$

where $\mathbf{k}$ is a vector containing the covariances between the N known points and the new point, and κ is the variance of the new point. We will determine $\mathbf{k}$ and κ later. Also $\mathbf{C}_{N+1}^{-1}$ can be expressed in terms of $\mathbf{C}_N^{-1}$ [22]:

$$\mathbf{C}_{N+1}^{-1} = \begin{pmatrix} \mathbf{M} & \mathbf{m} \\ \mathbf{m}^T & \mu \end{pmatrix}, \quad (7)$$

$$\text{where} \quad \begin{aligned} \mathbf{M} &= \mathbf{C}_N^{-1} + \mu^{-1}\mathbf{m}\mathbf{m}^T, \\ \mathbf{m} &= -\mu\,\mathbf{C}_N^{-1}\mathbf{k}, \quad \text{and} \\ \mu &= \left(\kappa - \mathbf{k}^T\mathbf{C}_N^{-1}\mathbf{k}\right)^{-1}. \end{aligned}$$

Inserting Eqs. 4 and 5 into Eq. 3 gives

$$p(t_{N+1}|\mathbf{X}_{N+1}, \mathbf{t}_N) \propto \exp\left(\frac{1}{2}\left(\mathbf{t}_N^T\mathbf{C}_N^{-1}\mathbf{t}_N - \mathbf{t}_{N+1}^T\mathbf{C}_{N+1}^{-1}\mathbf{t}_{N+1}\right)\right), \quad (8)$$

which by the use of Eq. 7 simplifies to

$$p(t_{N+1}|\mathbf{X}_{N+1},\mathbf{t}_N) \propto \exp\left(-\frac{1}{2}\frac{(t_{N+1}-\hat{t}_{N+1})^2}{\sigma^2_{t_{N+1}}}\right), \quad (9)$$

a univariate Gaussian with mean and variance given by

$$\hat{t}_{N+1} = \mathbf{k}^T\mathbf{C}_N^{-1}\mathbf{t}_N\,, \quad (10)$$
$$\sigma^2_{t_{N+1}} = \kappa - \mathbf{k}^T\mathbf{C}_N^{-1}\mathbf{k}\,. \quad (11)$$

The covariance matrices $\mathbf{C}_N$ and $\mathbf{C}_{N+1}$ are defined by way of a covariance function C which embodies our prior assumptions about the function to be modeled. Specifically, we define the covariance between function values at two data points $\mathbf{x}_p$ and $\mathbf{x}_q$ to be given by the smooth covariance function [12]

$$C(\mathbf{x}_p,\mathbf{x}_q) = \theta_1\exp\left(-\frac{1}{2}\sum_{i=1}^{n}\frac{(x_{p,i}-x_{q,i})^2}{r_i^2}\right)+\theta_2+\delta_{pq}\theta_3\,. \quad (12)$$

Here, the first term reflects a distance-dependent correlation between two data points: if their distance is small compared to the length scales r_i, the exponential term is close to one; with increasing distance it exponentially decays to zero. The hyperparameter θ_1 scales this correlation. In the second term, θ_2 specifies a certain offset of the function values from zero. Finally, the third term adds white noise to the model, scaled by θ_3 and applied only to the diagonal elements of the covariance matrix. The covariance vector $\mathbf{k}$ and variance κ from Eq. 6 can be expressed in terms of the covariance function as

$$k_i = C(\mathbf{x}_i,\mathbf{x}_{N+1}),\quad i=1,\ldots,N, \quad (13)$$
$$\kappa = C(\mathbf{x}_{N+1},\mathbf{x}_{N+1}) = \theta_1+\theta_2+\theta_3\,. \quad (14)$$

A. Optimizing the Hyperparameters

The GP employs a set of hyperparameters $\theta = \{\theta_1,\theta_2,\theta_3, r_1,r_2,\ldots r_n\}$ which can be set by the user or optimized such that the log-likelihood of the given function values $\mathbf{t}_N$ under the multivariate Gaussian with zero mean and covariance $\mathbf{C}_N = C(\mathbf{X}_N,\theta)$ is maximal. This log-likelihood and its derivative with respect to θ can be expressed as

$$\mathcal{L} = \log\left(p(\mathbf{t}_N|\mathbf{X}_N,\theta)\right) \quad (15)$$
$$= -\frac{1}{2}\left(\log\det\mathbf{C}_N + \mathbf{t}_N^T\mathbf{C}_N^{-1}\mathbf{t}_N + N\log 2\pi\right)$$
$$\frac{\partial\mathcal{L}}{\partial\theta} = \frac{1}{2}\left(\mathbf{t}_N^T\,\Gamma_N\,\mathbf{C}_N^{-1}\mathbf{t}_N - \mathrm{trace}(\Gamma_N)\right), \quad (16)$$
$$\text{where}\quad \Gamma_N \equiv \mathbf{C}_N^{-1}\frac{\partial\mathbf{C}_N}{\partial\theta}$$

Gradient-based or evolutionary algorithms can be used to optimize the hyperparameters, each with their own advantages. Gradient methods are fast local optimizers of smooth functions for which the analytic gradient is available, as is the case here. However, MacKay [12] shows that the hyperparameter optimization landscape is multimodal. This suggests that a slower but more robust global optimizer, such as an evolutionary algorithm, may yield better results.

We therefore use both an evolutionary algorithm (CMA [1]) and a quasi-Newton gradient method (BFGS [23]) in combination. CMA is always used for the first optimization of the likelihood in order to identify the global minimum. In our experience it suffices to limit CMA to 3 000 likelihood evaluations. To track adjustments to the optimal hyperparameters, *e.g.*, after additional data points have been added, we employ BFGS as a fast optimizer, coupled with a line search by golden section (see, *e.g.*, [24]). The initial step size for each line search is set to 10% of the search space in order to escape local minima. For BFGS, we found that computing 20 gradients and evaluating 20 points in the line search is sufficient. After optimizing the likelihood 10 times with BFGS, CMA is used again to avoid getting trapped in a local minimum.

To simplify the setting, we normalize the training data such that function values lie within $[0,1]$ and decision variables within $[-1,1]$. We then enforce the following bounds on the hyperparameters:

$$\theta_1 \in [10^{-3},1]$$
$$\theta_2 \in [10^{-3},1]$$
$$\theta_3 \in [10^{-9},10^{-2}]$$
$$r_i \in [10^{-2},10],\quad i=1,\ldots,n$$

Since the ratios of upper to lower bounds are very large, we operate with the log of the hyperparameter values, as proposed by Williams [25]. If the computation of the inverse of $\mathbf{C}_N$ fails too often during optimization, we can increase numerical stability by raising the lower bound on θ_3.

B. Computational Cost

The main equations of the GP are Eqs. 10 and 11 for predicting the mean and standard deviation of a new data point, and Eqs. 15 and 16 for optimizing the hyperparameters. Although all four equations contain the inverse of the covariance matrix $\mathbf{C}_N^{-1}$, the explicit inverse is only needed to compute the gradient of the log-likelihood in Eq. 16, required only for gradient-based optimization algorithms. The other equations contain the product of the inverse with a vector, which amounts to solving a linear system of equations. We can avoid computing the explicit inverse by performing an LU decomposition of $\mathbf{C}_N$:

$$\mathbf{C}_N = \mathbf{L}\,\mathbf{U}, \quad (17)$$

where $\mathbf{L}$ and $\mathbf{U}$ are a lower and upper triangular matrix, respectively. The LU decomposition can be computed in order $\mathcal{O}(N^3)$, and is numerically more robust than operating with the explicit inverse. Then, after calculating $\mathbf{C}_N^{-1}\mathbf{t}_N$ via the LU decomposition in $\mathcal{O}(N^2)$, predicting mean and standard deviation for a new point $\mathbf{x}_{N+1}$ are of order $\mathcal{O}(N)$ and $\mathcal{O}(N^2)$, respectively. The log-likelihood for a given LU decomposition and $\mathbf{C}_N^{-1}\mathbf{t}_N$ can also be obtained in $\mathcal{O}(N)$, provided the determinant is computed as proposed in Eq. 18 below.

The gradient of the log-likelihood requires explicit computation of $\mathbf{C}_N^{-1}$, which is about 3 times as expensive as the LU decomposition. Evolutionary algorithms typically require more evaluations but do not need gradient information, so each evaluation is cheaper. For large N several methods for computing sparse covariance matrices have been proposed (*e.g.*, [26]) which we do not pursue here.

C. Numerical Difficulties

We encountered numerical problems in computing the log-likelihood according to Eq. 15. Specifically, for $N > 30$ the determinant of the covariance matrix can underflow IEEE 754 double precision floating-point arithmetic, especially if the covariance matrix is ill-conditioned.

Gibbs and MacKay [27] avoid computing the log-likelihood by using only gradient information in their conjugate gradient implementation, including their line search. By contrast, we prefer the likelihood over its gradient since it is computationally cheaper, and allows us to use direct search methods such as evolutionary algorithms. We rewrite the computation of the determinant in the following numerically more stable form:

$$\log\det \mathbf{C}_N \;=\; \sum_{i=1}^{N} (\log \mathbf{L}_{ii} \;+\; \log \mathbf{U}_{ii}) \qquad (18)$$

In words, we compute the log-determinant as a sum of logs of the elements in the trace of the $\mathbf{L}$ and $\mathbf{U}$ matrices. All quantities in this computation remain within machine precision even where the determinant itself would underflow.

IV. Gaussian Process Optimization Procedure

We propose an optimization procedure based on the surrogate approach that uses a Gaussian process as an inexpensive model of the objective function. This Gaussian Process Optimization Procedure (GPOP) starts from an initial set of points, obtained, *e.g.*, from previous optimization runs, by random sampling, or a short run of a conventional optimization algorithm. Our optimization loop then proceeds as follows:

A GP is constructed for the given data set, and the hyperparameters are optimized. An Evolutionary Algorithm (EA) is then used to search for minima of the GP prediction, using several *merit functions* as fitness functions for the EA. Finally, the resulting minima are evaluated on the objective function and added to the data set. These three steps constitute one iteration of GPOP, and are repeated until a termination criterion is reached.

In the following, we discuss merit functions, describe techniques to limit the training effort of a GP to the local neighborhood of the best solution found so far, and show how real world problems with more than one objective and constraints can be handled.

A. Merit Functions

Searching the GP for the minimal predicted value $\hat{t}$ *exploits* the knowledge of the GP to find the most promising candidate for reducing the objective function value. However, this carries the risk of premature convergence to non-optimal solutions [3], [5]. To improve the prediction quality of the GP and promote a more thorough global search, there is also a need to *explore* new regions of decision space.

To balance exploration with exploitation, Torczon and Trosset [3] propose a merit function f_{M} which adds a density measure to the predicted function value $\hat{t}$ so as to promote unexplored regions of the decision space. One possible density measure is the maximin distance [28], *i.e.*, the distance to the closest evaluated point. Another possibility is the predicted standard deviation σ_t of the GP, as proposed in [15]. Both measures are minimal at known data points and increase with distance to evaluated solutions. In contrast to the standard deviation, however, the maximin distance is not bounded for unbounded search spaces, and its derivative is discontinuous at all positions equidistant from the two closest evaluated points. We therefore prefer the standard deviation, and define the merit function f_{M} as

$$f_{\mathrm{M}}(\mathbf{x}) \;=\; \hat{t}(\mathbf{x}) \;-\; \alpha\,\sigma_t(\mathbf{x}), \qquad (19)$$

where $\alpha \geq 0$ balances the two terms by scaling the density measure. (For maximization problems α must be negative.)

We optimize 4 merit functions, using $\alpha = 0, 1, 2, 4$, respectively. Setting $\alpha = 0$ exploits the information of the model by searching for the predicted minimum. By contrast, $\alpha = 4$ strongly pushes the optimization into unexplored regions. The resulting minima can be evaluated in parallel, and updating the GP with several evaluations at a time also serves to reduce the number of GPOP iterations. In our experience, using more than 4 merit functions is not beneficial for the convergence of the optimization.

B. Local Modeling and Local Search

Both global and local fitness function models are considered in the literature. While global models use all evaluated points, local models only take into account points from a certain region of decision space. Although they thus throw away information, local models have a number of advantages: the precision of any model is limited, and is mainly determined by the difference in the objective function values among the points being modeled. For complicated (*e.g.*, multimodal, discontinuous) function landscapes it may not be feasible to build an accurate global model at all. Finally, since they use less training data, local models typically carry a far smaller computational cost.

Since we want to converge towards the optimum with arbitrary precision, we restrict our model to the local neighborhood of the current best solution $\mathbf{x}^{\mathrm{best}}$. We use as training data for the GP the union of the N_C points closest to $\mathbf{x}^{\mathrm{best}}$ in the decision space, and the N_{R} most recently evaluated points. The closest points serve to model the neighborhood of $\mathbf{x}^{\mathrm{best}}$, while the most recent points are included to allow the data set (and hence the model) to evolve even when the N_C closest points remain the same.

To avoid searching areas that may be poorly modeled, we also restrict the search to a hypercube around the current best solution:

$$\mathbf{x}^{\mathrm{best}} - \mathbf{d}/2 \;\leq\; \mathbf{x} \;\leq\; \mathbf{x}^{\mathrm{best}} + \mathbf{d}/2\,, \qquad (20)$$

with the hypercube's diagonal $\mathbf{d}$ set to reflect the spread of the N_C closest points:

$$d_i \;=\; \max_c(x_{c,i}) \;-\; \min_c(x_{c,i}), \qquad (21)$$

where i indexes the dimensions of the search space, and c the N_C closest points. Thus the search space moves around with the current best solution.

C. Enhancements for Real-World Applications

GPOP was designed to model a single fitness function. For real-world applications this fitness function will typically be composed of several objectives as well as penalties for constraint violations. To model such a complex aggregate of functions with a single GP may prove difficult. We therefore model each objective and penalty by its individual GP, and construct the overall fitness function model as an aggregate of all GPs. While this is computationally somewhat more expensive, it leads to a more accurate model of composite fitness functions.

In real-world applications, the objective or constraint evaluation might fail for some points, due to an infeasible design or inadequacies of the evaluation code. Such failed evaluations should not be used for training the GP [4]. We initially require $N_C/2$ points for training the GP, which we generate with CMA. If some of the evaluations fail, CMA is used to produce an additional $N_C/2$ points at a time, until at least $N_C/2$ points have been evaluated successfully. After this initialization, the iterative optimization procedure is started. In each iteration, the local GP models are constructed for the current data set. Then the minimum for each merit function is located, evaluated on the expensive fitness function, and added to the training data.

If in one iteration the evaluation of all points fails, the GP models remain unchanged. To avoid stagnation in this situation, we add a small Gaussian perturbation $\mathbf{x}^R$ of the current best solution $\mathbf{x}^{\text{best}}$ to the training data:

$$x_i^R \;=\; x_i^{\text{best}} + z_i\, d_i/100\,, \quad z_i \sim \mathcal{N}(0,1) \tag{22}$$

With these enhancements, our GPOP can be summarized as:

while less than $N_C/2$ points evaluated successfully:

- use (2,10)-CMA to generate $N_C/2$ points
- evaluate them on expensive fitness function

while termination criterion not reached:

- find $\mathbf{x}^{\text{best}}$, the best of all points evaluated so far
- training set = N_C points closest to $\mathbf{x}^{\text{best}}$
  + N_{R} most recent successful evaluations
- **for** each objective and constraint:
  - optimize GP hyperparameters (Eqs. 15–18)
  - **for** each merit function (Eq. 19):
    find predicted optimum (Eqs. 10–14)
- remove optima that have already been evaluated
- evaluate new optima on expensive fitness function
- **while** no evaluation successful:
  - generate and evaluate perturbation (Eq. 22)

V. Performance Analysis

We analyze the performance of GPOP on three unimodal test functions [1] with different properties (a simple quadratic function, Schwefel's function, and Rosenbrock's function) and on a multimodal function (Rastrigin's function):

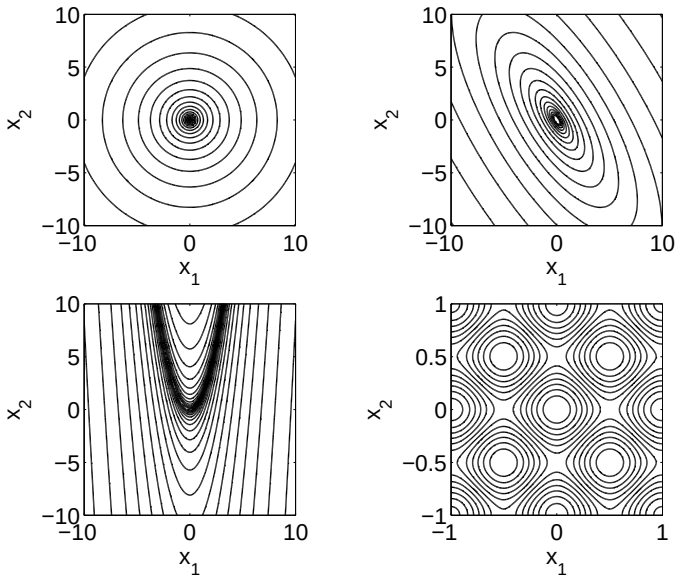


Fig. 1. Contour plots of the sphere function (upper left), Schwefel's function (upper right), Rosenbrock's function (lower left), and Rastrigin's function (lower right); all in two dimensions.

$$f_{\text{sphere}}(\mathbf{x}) \;=\; \sum_{i=1}^{n} x_i^2, \tag{23}$$

$$f_{\text{Schwefel}}(\mathbf{x}) \;=\; \sum_{i=1}^{n} \left(\sum_{j=1}^{i} x_j \right)^2, \tag{24}$$

$$f_{\text{Rosen}}(\mathbf{x}) \;=\; \sum_{i=1}^{n-1} \left(100(x_{i+1} - x_i^2)^2 + (1 - x_i)^2 \right) \tag{25}$$

$$f_{\text{Rastrigin}}(\mathbf{x}) \;=\; 10n + \sum_{i=1}^{n} \left(x_i^2 - 10\cos(2\pi x_i) \right) \tag{26}$$

For all functions, the minimal function value is $f = 0$ and the search space is restricted to $x_i \in [-10, 10]$. For the three unimodal functions, we measure the number of function evaluations required to reach a function value smaller than $f = 10^{-10}$. For the multimodal function, we analyze the capability of the algorithm to converge to the global optimum. Thus, we measure the final function value as soon as the algorithm is caught in a local minima. GPOP is assumed to be caught, if the size of the local search area is smaller than $f = 10^{-6}$. Four different training set sizes with: $N_{\text{R}} = N_{\text{C}} = 15, 30, 60, 120$ are analyzed. The results are compared to CMA, an evolutionary algorithm known to perform well on all three unimodal functions [1]. The results represent always the mean of 5 independent optimization runs. Different problem dimensions were analysed. If for a certain dimension and training set size, no result is given, then this is due to a too poor convergence compared to CMA.

A. Sphere function

The minimum of the sphere function is at $\mathbf{x} = 0$. Its contours (Fig. 1, left) are hyperspheres in decision space,

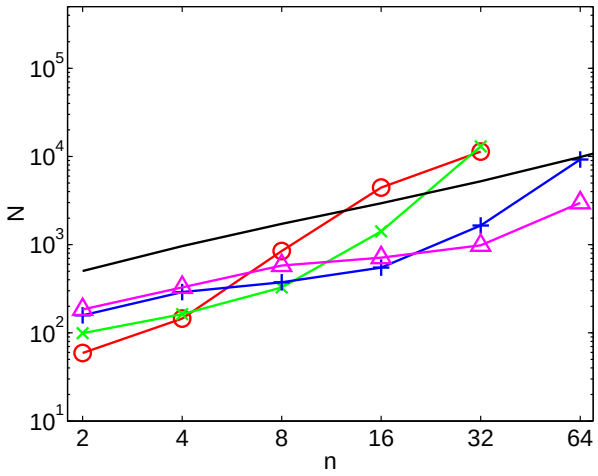


Fig. 2. Mean number N of function evaluations against problem dimensionality n to reach $f_{\text{sphere}} = 10^{-10}$ for GPOP with training set size $N_{\text{R}} = N_{\text{C}}$ = 15 (o), 30 (x), 60 (+), and 120 (Δ), compared to CMA (unmarked).

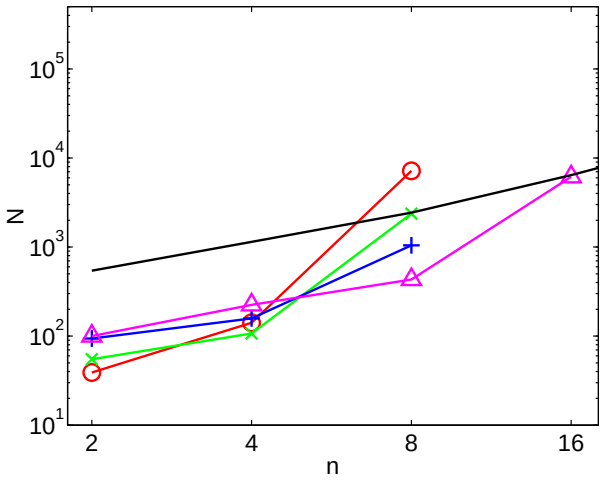


Fig. 3. Mean number N of function evaluations against problem dimensionality n to reach $f_{\text{Schwefel}} = 10^{-10}$ for GPOP with training set size $N_{\text{R}} = N_{\text{C}}$ = 15 (o), 30 (x), 60 (+), and 120 (Δ), compared to CMA (unmarked).

making this function trivial to optimize. Results for GPOP and CMA are shown in Fig. 2, plotted against the problem dimensionality n on a log-log scale.

The number N of function evaluations required increases with n for both algorithms. The increase gets steeper for GPOP past a certain threshold dimensionality, beyond which the training data no longer suffices to model the function well, and the algorithm gets inefficient. As a rule of thumb, this threshold is located at about $n = N_{\text{C}}/2$. In other words, for a sphere function in n dimensions we should have a training set of $N_{\text{C}} \geq 2n$ points, with $N_{\text{R}} = N_{\text{C}}$. As long as this rule is obeyed, GPOP requires about 4 to 5 times fewer function evaluations than CMA.

B. Schwefel's function

The minimum of Schwefel's function is also at $\mathbf{x} = 0$; its contours are hyperellipsoids (Fig. 1, center). Note that the principal axes are not parallel but rotated relative to the coordinate axes of the decision space: Schwefel's function is non-separable, *i.e.*, the decision variables are highly correlated. In contrast to CMA, which was designed to be invariant to such rotations [1], our GP scales the covariance function by a separate hyperparameter along each dimension, and is thus sensitive to them.

Nevertheless, our results for Schwefel's function (Fig. 3) indicate that our GP is able to model non-separable functions, and GPOP can optimize them efficiently. The number N of function evaluations required to converge is about the same as for the sphere function, again outperforming CMA by a large factor. For efficient optimization, however, more training data should be used, due to the more complex function topology and the correlation of the decision variables. While the precise threshold for the training set size is less clear here, as a rule of thumb we might require $N_{\text{C}} \geq 8n$ points, with $N_{\text{R}} = N_{\text{C}}$.

C. Rosenbrock's function

Rosenbrock's function is also non-separable, with highly correlated decision variables. As an added difficulty the minimum is located (at $\mathbf{x} = 1$) in a long, flat-bottomed, curved, and narrow valley (Fig. 1, right). Both algorithms consequently require more function evaluations, as shown in Fig. 4. The performance gap between GPOP and CMA has narrowed but is still evident, provided that GPOP is given $N_{\text{C}} \geq 5n$ training data points, with $N_{\text{R}} = N_{\text{C}}$.

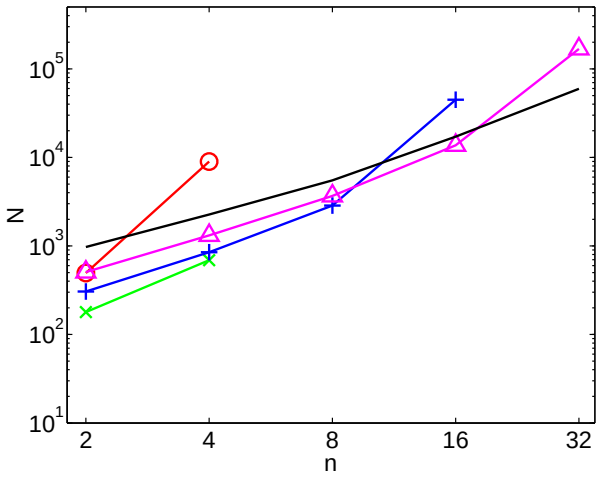


Fig. 4. Mean number N of function evaluations against problem dimensionality n to reach $f_{\text{Rosen}} = 10^{-10}$ for GPOP with training set size $N_{\text{R}} = N_{\text{C}}$ = 15 (o), 30 (x), 60 (+), and 120 (Δ), compared to CMA (unmarked).

D. Rastrigin's function

Rastrigin's function is a superposition of the sphere function and a cosine function with a high oscillation frequency and amplitude. The global minimum is located at (at $\mathbf{x} = 0$). In Fig. 5 and 6, CMA is compared with GPOP. For Rastrigin's function, we are interested in the global convergence and not the convergence speed. Thus, each optimization runs until the algorithm converged to a local or the global minimum. An algorithm is considered converged as soon as the step size (CMA) or the local search area (GPOP) is 10^{-4} times smaller than the size of the surrounding valley, which is ≈ 1. In the figures, the final function value f_f of the optimization is plotted. The square root of f_f is equal to the distance of

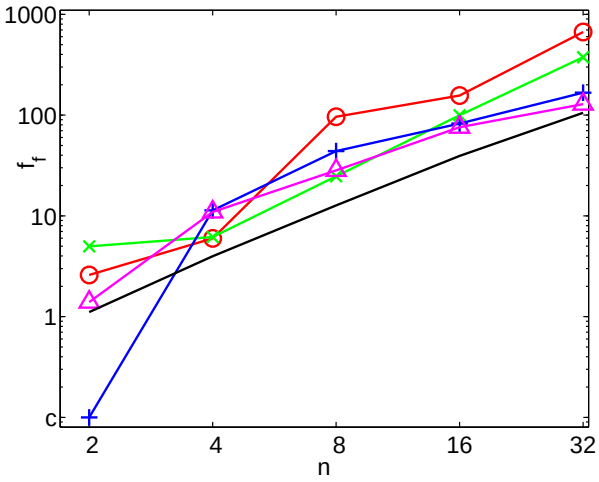


Fig. 5. Final function value f_f of the converged optimization run on $f_{\text{Rastrigin}}$ against problem dimensionality n for GPOP with training set size $N_{\text{R}} = N_{\text{C}}$ = 15 (o), 30 (x), 60 (+), and 120 (Δ), compared to CMA (unmarked).

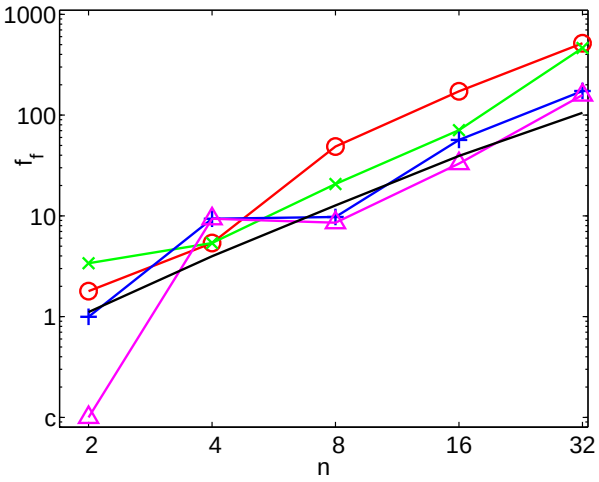


Fig. 6. Final function value f_f of the converged optimization run on $f_{\text{Rastrigin}}$ against problem dimensionality n for GPOP with training set size $N_{\text{R}} = N_{\text{C}}$ = 15 (o), 30 (x), 60 (+), and 120 (Δ), and a fixed hyperparameter value $\theta_3 = 0.01$, compared to CMA (unmarked).

the final point to the optimum.

In Fig. 5, the same settings for GPOP are used as for the three unimodal functions. The figure shows that GPOP and CMA are not able to determine the global minima for $n > 2$. This is due to the large number of local minima and the high oscillation amplitude of the function between the minima. For the standard settings, CMA performs better then GPOP. However, with increasing training set for GPOP and problem dimension the difference between CMA and GPOP decreases.

There are several possibilities to improve the performance of GPOP on multimodal functions:

- larger values of α in the merit function would promote better exploration of the search space.
- multiple local search spaces (*e.g.*, around the best N_{best} solutions) would facilitate recovering multiple minima.
- increasing the noise level θ_3 of the GP model leads to a smoother approximation of the fitness function; this may remove the local minima from the GP model.

In Fig. 6, the third possibility was chosen and the noise level was fixed at $\theta_3 = 0.01$. With this setting and training set size of 60 to 120, GPOP performs similar to CMA.

E. Summary of the Performance Analysis

GPOP has been shown to be capable of optimizing difficult test problems. Furthermore GPOP has proven to be an efficient optimization procedure as it requires less function evaluations than CMA to find the minimum of three unimodal functions within given precision.

The performance of GPOP is dependent on the training set size as sufficient training data is required for adequately modeling the fitness function. For all considered test problems, a rule of thumb for setting the training set size is given as a function of the problem dimensionality. Following this rule, GPOP requires on average about 3 to 6 times less function evaluations than CMA.

Compared to CMA, a main drawback of GPOP is the higher computational cost of the optimization procedure. The cost scales $\mathcal{O}(N^3)$ with the training set size and is only $\mathcal{O}(N)$ with the problem dimension. Thus, GPOP should mainly be applied to expensive optimization problems, requiring at least several seconds of CPU time per function evaluation. The computational expense is also the reason why the training set size was limited to 120. With this training set size GPOP is more efficient then CMA on problems with a problem dimension of up to 16-64 decision variables.

For the multimodal Rastrigin's function, CMA performs better than GPOP when comparing the final function values. However, several ways to improve the performance of GPOP exist, and by adding white noise with variance $\theta_3 = 0.01$ to the GP model we have obtained performance similar to that of CMA.

It is subject to future studies to analyze the performance of GPOP on a wider set of test functions with different optimization difficulties. This has already been done for CMA. In [1], CMA was analyzed on a set of unimodal test functions. Recently, CMA was further developed such that it can operate with large populations [2], while the presented GPOP evaluates 4 solutions in parallel. Large populations improve global search capabilities making CMA competitive with algorithms designed for multimodal problems [29].

VI. Optimization of Compressor Profiles

We now compare the performance of GPOP to evolution strategies from literature for the design optimization of subsonic compressor profiles for stationary gas turbines. The aerodynamic properties of a profile are analyzed by CFD on a 2D grid, which takes into account the streamline thickness and radius variation along the compressor axis. The objective of the profile design is to find shapes that fulfill various aerodynamic and mechanical constraints. In addition, low aerodynamic losses at design and off-design conditions are desired.

The compressor profile is parameterized by four Bezier spline segments: suction surface, leading edge, pressure surface, and trailing edge. Each segment is defined by 6 control

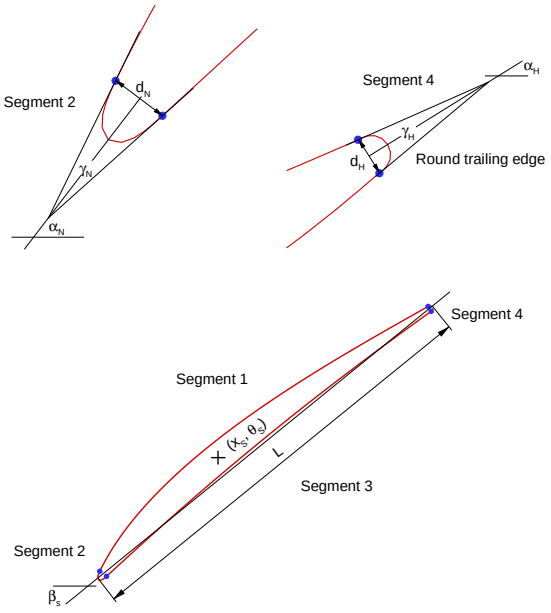


Fig. 7. Some engineering parameters of a compressor profile.

points with two coordinates, giving 48 parameters in total. 16 parameters are determined by enforcing $\mathcal{C}^2$-continuity between the segments. Introducing simplifications for the trailing edge (circular) and leading edge (elliptical), the remaining parameters are translated into 19 engineering parameters, such as metal angles, wedge angles, profile length, *etc.*, as shown in Fig. 7. Compared to the control points, engineering parameters simplify the comparison and illustration of different profile parameter sets.

The aerodynamic properties of compressor profiles are analyzed with the Q3D solver MISES [30]. MISES solves an Euler equation for a single profile and flow path, assuming periodic boundary conditions between two adjacent compressor blades in a row. Viscous effects are modeled by a fully coupled integral boundary layer formulation. The stream tube contraction as well as radius variations along the compressor axis are considered (Q3D). We limit the design optimization to subsonic flow as shocks are three-dimensional flows and pose various optimization difficulties [31].

For the profile optimization, the fitness function aggregates several objectives and constraints as proposed by Büche *et al.* [32]. The objective of the profile optimization is to minimize the aerodynamic losses l for the design inlet flow angle α_{D} and two inlet flow angle variations $\alpha_{\mathrm{D}} \pm \Delta\alpha$, so as to promote high efficiency under both design and off-design conditions. The losses at the off-design angles of incidence are dependent on the profile shape as well as the inlet flow angle variation $\Delta\alpha$; the latter is therefore declared a second objective, which is to be maximized. Furthermore, Büche *et al.* propose to use $\Delta\alpha$ as an additional decision variable, *i.e.*, the inlet flow angle variation is directly modified by the optimization algorithm.

Aerodynamic constraints are enforced for the flow properties at design inlet flow angle: the deviation of the turning angle $\Delta\beta$ from its target value is constrained; the Mach number Ma and the non-dimensional shape factor H_{12} at the suction side trailing edge are bounded above so as to avoid transonic effects and flow separation, respectively. For every flow calculation that fails, an additional penalty (greater than the largest possible aerodynamic loss) is added. We also enforce a mechanical constraint by setting a lower bound on the thickness of the profile. We combine objectives and constraints into the single fitness function

$$\begin{aligned} f \;=\; & l_{\alpha_{\mathrm{D}}} + l_{\alpha_{\mathrm{D}}-\Delta\alpha} + l_{\alpha_{\mathrm{D}}+\Delta\alpha} \\ & + a_0\,\Delta\alpha \\ & + a_1 \max(0, |\Delta\beta| - \Delta\beta^{\mathrm{limit}}) \\ & + a_2 \max(0, H_{12} - H_{12}^{\mathrm{limit}}) \\ & + a_3 \max(0, \mathrm{Ma}_{\max} - \mathrm{Ma}_{\max}^{\mathrm{limit}}) \\ & + a_4 \max(0, d^{\,\mathrm{limit}} - d)\,, \end{aligned} \tag{27}$$

where $a_{\{0,1,2,3,4\}} > 0$ are user-defined weights scaling the different penalty terms in the fitness function.

A. Optimization Setup

GPOP is compared with evolution strategies, represented by Rechenberg's Success Rule [33], Rotating Angle Mutation (ROT-ES) [24], and Covariance Matrix Adaptation (CMA) [1]. The Success Rule is defined for a $(1{+}1)$ evolution strategy and has two free parameters: a multiplicative factor c for adapting the step size, and the success rate p_{s}. We set $c = 0.817$ as proposed by Schwefel [24]. The success rate is usually set to $p_{\mathrm{s}} = 1/5$, but Rechenberg [33] and Schwefel [34] recommend smaller values for noisy and constrained problems. Setting $p_{\mathrm{s}} = 1/20$ gave us the best results. For ROT-ES we use the standard settings [35] with a $(15, 100)$ selection scheme, while CMA uses $(2, 10)$ and $(3, 12)$ selection schemes.

For GPOP we model each of the three profile losses and four constraints with a separate Gaussian process (GP). The predicted fitness function is then constructed from the 7 GPs according to Eq. 27. Only converged MISES calculations (computed at three different incidences) are used to train the GPs. Since each incidence computation may fail for different points, the training sets for the 7 GPs may differ from each other. The standard deviation used in the merit function is based solely on the predicted standard deviations for the losses at the 3 incidences. As before, we analyze training set sizes of $N_{\mathrm{C}} = N_{\mathrm{R}} = 15$, 30, 60, and 120.

We bound the decision variables for all optimization runs. Using knowledge from previous optimizations, the bounds are set such that the search space is limited as much as possible, and to mainly physical solutions — we never generate profiles with zero or negative thickness, for instance. Since the decision variables have different orders of magnitude, the bounded design space is then scaled such that the optimization algorithms operate in a unit hypercube. The initial step size for all evolution strategies is set to 0.1.

B. Optimization Results

Due to the computational expense of the profile optimization, only a single optimization run is performed per algorithm.

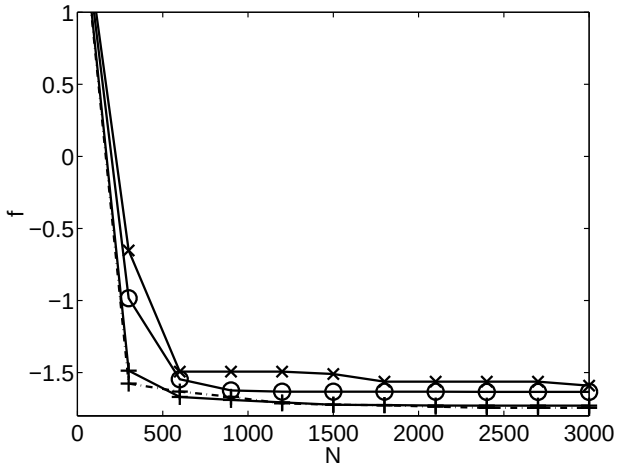


Fig. 8. Convergence of the Success Rule with 1/20th success rate (o), ROT-ES (x), (2,10)-CMA (+,solid line), and (3,12)-CMA (+, dash-dotted line).

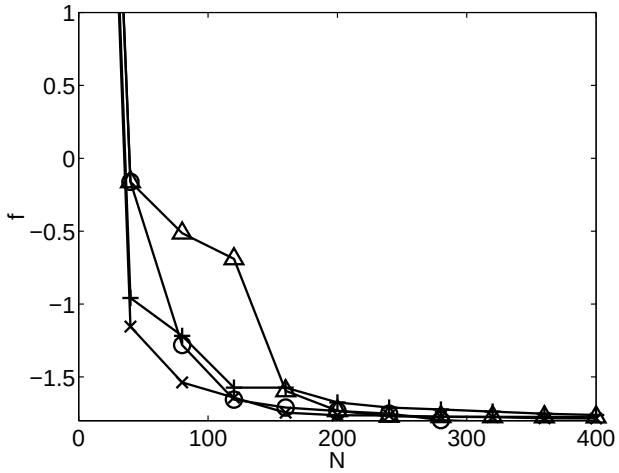


Fig. 9. Convergence of the GPOP algorithm with a training set size of $N_{\mathrm{R}} = N_{\mathrm{C}} = 15$ (x), 30 (+), 60 (o), and 120 (Δ).

However, these results are representative as similar results were obtained for different weighting of the objectives in the fitness function and for different profiles. Due to limited space, we refer to [36]. The convergence of the evolution strategies is shown in Fig. 8. In the beginning (up to about $N = 200$ evaluations) all converge about equally fast. This may result from the equal initial step size, which is not changed much by the different adaptation schemes. Small differences in convergence are due to differences in population size and other properties.

Convergence of the Success Rule depends on the success rate p_{s}. Best results were obtained for $p_{\mathrm{s}} = 1/20$, but even so the step size becomes inefficiently small after about $N = 1\,250$ evaluations. ROT-ES converges far more slowly than the Success Rule, due to a much slower decrease of the step sizes. After $N = 3\,000$ evaluations, the fitness function value is still improving, lagging far behind the other algorithms. Covariance Matrix Adaptation (CMA) has the fastest convergence and also produces the best fitness function values of all considered evolution strategies.

Our results for the different evolution strategies permit some conclusions about the optimization problem. The Success Rule operates with an isotropic mutation distribution, which converges efficiently only for well-scaled problems with nearly circular contours. Since the Success Rule works reasonably well on the profile design problem, it can be assumed to be not very ill-conditioned. Nonetheless the correlated mutations of CMA, which can adapt to scaled and correlated decision variables, are still beneficial here. ROT-ES also adapts correlated mutations but by a less efficient scheme, resulting in poor convergence. Comparing the results, the deterministic adaptation schemes for the mutation distribution in the Success Rule and CMA outperform the stochastic self-adaptation of ROT-ES.

Fig. 9 shows the convergence of GPOP for training set sizes $N_{\mathrm{C}} = N_{\mathrm{R}} = 15$, 30, 60, and 120. For all sizes the algorithm converged to similar final fitness function values. The results obtained by GPOP are superior to the evolution strategies even though we gave GPOP only about 1/7th the number of function evaluations, as documented in Table I.

TABLE I

PERFORMANCE COMPARISON OF ALGORITHMS FOR COMPRESSOR PROFILE OPTIMIZATION.

Optimization algorithm	$\min(f)$	N
Evolution Strategy		
1/20th success rule	-1.63	3000
$(15, 100)$-ROT-ES	-1.59	3000
$(2, 10)$-CMA	-1.73	3000
$(3, 13)$-CMA	-1.74	3000
GPOP with different training set sizes		
$N_{\mathrm{C}} = N_{\mathrm{R}} = 15$	-1.78	400
$N_{\mathrm{C}} = N_{\mathrm{R}} = 30$	-1.76	400
$N_{\mathrm{C}} = N_{\mathrm{R}} = 60$	-1.82	400
$N_{\mathrm{C}} = N_{\mathrm{R}} = 120$	-1.77	400

The good convergence of GPOP for the small training set size $N_{\mathrm{C}} = N_{\mathrm{R}} = 15$, together with our results of Section V, supports our above conclusion that our profile design problem must be reasonably well-conditioned. The aerodynamic properties of the optimized profiles are discussed in [32].

VII. CONCLUSIONS

When optimizing expensive fitness functions, it is important to reduce the number of function evaluations required as much as possible. This can be achieved by exploiting knowledge of past evaluated points to train an empirical model that can be used as an inexpensive surrogate of the fitness function.

We have shown how Gaussian processes (GPs) can be used as fitness function surrogates. The resulting Gaussian Process Optimization Procedure (GPOP) is capable of efficiently optimizing even very ill-conditioned problems such as Rosenbrock's function. Here GPOP clearly outperformed CMA, an algorithm known to be efficient in such functions [1].

A scaling analysis over the number of decision variables showed that sufficient training data points must be supplied for GPOP to converge efficiently. The computational expense of the optimization algorithm itself is consequently much higher for GPOP than for CMA. While CMA can be efficiently

applied to problems with more than 300 decision variables [1], we encountered computational limits for our test functions as early as 16 to 64 variables. These could be addressed by a sparse GP implementation [26].

GPOP also showed global search capabilities: adding some white noise to the GP model allowed GPOP to perform similar to CMA. GPOP's 4 merit functions constitute different compromises between exploration and exploitation, thus balancing the inherent conflict between converging to and escaping from local minima.

Finally we applied GPOP to a real-world problem: the optimization of stationary gas turbine compressor profiles. GPOP converged much faster than a range of alternative evolution strategies, and to significantly better results.

REFERENCES

[1] N. Hansen and A. Ostermeier, "Completely derandomized self-adaptation in evolution strategies," *Evolutionary Computation*, vol. 9, no. 2, pp. 159–195, 2001.

[2] N. Hansen, S. D. Müller, and P. Koumoutsakos, "Reducing the time complexity of the derandomized evolution strategy with covariance matrix adaptation (CMA-ES)," *Evolutionary Computation*, vol. 11, no. 1, pp. 1–18, 2003.

[3] V. Torczon and M. W. Trosset, "Using approximations to accelerate engineering design optimization, ICASE Report No. 98-33," NASA Langley Research Center Hampton, VA 23681-2199, Tech. Rep., 1998.

[4] A. J. Booker, J. E. Dennis Jr., P. D. Frank, D. B. Serafini, V. Torczon, and M. W. Trosset, "A rigorous framework for optimization of expensive functions by surrogates," *Structural Optimization*, vol. 17, no. 1, pp. 1–13, 1999.

[5] M. Emmerich, A. Giotis, M. Özdemir, T. Bäck, and K. Giannakoglou, "Metamodel-assisted evolution strategies," in *Parallel Problem Solving from Nature - PPSN VII*. Springer-Verlag, 2002, pp. 361–370.

[6] J. D. Martin and T. W. Simpson, "Use of adaptive metamodeling for design optimization," in *AIAA/ISSMO Symposium on Multidisciplinary Analysis and Optimization*, 2002.

[7] Y. Jin, M. Olhofer, and B. Sendhoff, "Managing approximate models in evolutionary aerodynamic design optimization," in *Proceedings of IEEE Congress on Evolutionary Computation, Vol. 1, Seoul, Korea*, 2001, pp. 592–599.

[8] ——, "On evolutionary optimisation with approximate fitness functions," in *Proceedings of the Genetic and Evolutionary Computation Conference GECCO, Las Vegas, Nevada*, 2000, pp. 786–793.

[9] A. P. Giotis and K. C. Giannakoglou, "Single- and Multi-Objective Airfoil Design Using Genetic Algorithms and Artificial Intelligence," in *Proceedings of EUROGEN'99*, 1999.

[10] H. Nakayama, M. Arakawa, and R. Sasaki, "A comptutational intelligence approach to optimization with unknown objective functions," in *Artificial Neural Networks-ICANN 2001*, 2001, pp. 73–80.

[11] M. A. El-Beltagy and A. J. Keane, "Evolutionary optimization for computationally expensive problems using gaussian processes," in *Proc. Int. Conf. on Artificial Intelligence IC-AI'2001, Las Vegas*, H. Arabnia, Ed. CSREA Press, 2001, pp. 708–714.

[12] D. J. C. MacKay, "Gaussian processes - a replacement for supervised neural networks?" in *Lecture notes for a tutorial at NIPS*, 1997.

[13] A. Ratle, "Accelerating the convergence of evolutionary algorithms by fitness landscape approximation," in *Parallel Problem Solving from Nature - PPSN V*. Springer-Verlag, 1998, pp. 87–96.

[14] Y. Jin, T. Okabe, and B. Sendhoff, "Adapting Weighted Aggregation for Multiobjective Evolution Strategies," in *First International Conference on Evolutionary Multi-Criterion Optimization*, E. Zitzler, K. Deb, L. Thiele, C. A. Coello Coello, and D. Corne, Eds. Springer-Verlag. Lecture Notes in Computer Science No. 1993, 2001, pp. 96–110.

[15] M. N. Gibbs and D. J. C. MacKay, "Manual for tpros, v2.0," 1997.

[16] K. C. Giannakoglou, "Acceleration of genetic algorithms using artificial neural networks - theoretical background," in *von Karman Institute Lectures Series on Genetic Algorithms for Optimization in Aeronautics and Turbomachinery*, 2000.

[17] T. W. Simpson, T. M. Mauery, J. J. Korte, and F. Mistree, "Comparison of response surface and kriging models for multidisciplinary design optimization," in *AIAA paper 98-4758*. 7 th AIAA/USAF/NASA/ISSMO Symposium on Multidisciplinary Analysis and Optimization, St. Louis, USA, 1998.

[18] K. C. Giannakoglou, "Optimization and inverse design in aeronautics: how to couple genetic algorithms with radial basis function networks," in *Innovative Tools for Scientific Computation in Aeronautical Engineering*, J. Periaux, P. Joly, O. Pironneau, and E. Onate, Eds., 2001.

[19] S. Haykin, *Neural Networks - A Comprehensive Foundation*. Prentice-Hall, 1994.

[20] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning internal representations by error propagation," *Parallel distributed processing: explorations in the microstructures of cognition, Bradford Books/MIT Press, Cambridge, MA*, vol. 1, pp. 318–362, 1986.

[21] C. A. L. Bailer-Jones, H. K. D. H. Bhadeshia, and D. J. C. MacKay, "Gaussian process modelling of austenite formation in steel," *Materials Science and Technology*, vol. 15, no. 3, pp. 287–294, 1999.

[22] S. Barnett, *Matrix Methods for Engineers and Scientists*. McGraw Hill, Inc., 1979.

[23] J. Dennis and R. B. Schnabel, *Numerical Methods for Unconstrained Optimization and Nonlinear Equations, Series in Computational Mathematics*. Prentice-Hall, 1993.

[24] H.-P. Schwefel, *Numerical Optimization of Computer Models*. Wiley, Chichester, 1981.

[25] C. K. I. Williams, "Prediction with gaussian processes: From linear regression to linear prediction and beyond," in *Learning and Inference in Graphical Models*, M. I. Jordan, Ed., 1998.

[26] M. Seeger, C. Williams, and N. Lawrence, "Fast forward selection to speed up sparse gaussian process regression," in *9th International Workshop on Artificial Intelligence and Statistics*. Key West, Florida, 2003.

[27] M. N. Gibbs, "Bayesian gaussian processes for regression and classification," Ph.D. thesis, University of Cambridge, England, 1997.

[28] M. E. Johnson, L. M. Moore, and D. Ylvisaker, "Minimax and maximin distance designs," *Journal of Statistical Inference and Planning*, vol. 26, pp. 131–148, 1990.

[29] S. Kern, S. Müller, D. Büche, N. Hansen, and P. Koumoutsakos, "Learning probability distributions in continuous evolutionary algorithms," in *Workshop on Fundamentals in Evolutionary Algorithms, Thirteenth International Colloquium on Automata, Languages and Programming, Eindhoven*, 2003.

[30] M. Drela and H. Youngren, "A user s guide to MISES 2.1." Massachusetts Institute of Technology, Tech. Rep., 1995.

[31] L. Huyse and R. M. Lewis, "Aerodynamic shape optimization of two-dimensional airfoils under uncertain operating conditions, ICASE No. 2001-1," NASA Langley Research Center Hampton, VA, Tech. Rep., 2001.

[32] D. Büche, G. Guidati, and P. Stoll, "Automated design optimization of compressor blades for stationary, large-scale turbomachinery," in *Proceedings of the ASME/IGTI Turbo Expo 2003*. ASME, 2003.

[33] I. Rechenberg, *Evolutionsstrategie: Optimierung technischer Systeme nach Prinzipien der biologischen Evolution*. Fromman-Holzboog, 1973.

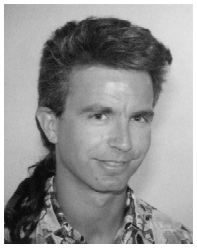
[34] H.-P. Schwefel, *Evolution and Optimum Seeking*. John Wiley & Sons, 1995.

[35] H.-P. Schwefel and G. Rudolph, "Contemporary evolution strategies," in *Proc. of the Third European Conference on Artificial Life (ECAL'95)*, Granada, Spain, 1995, pp. 893–907.

[36] D. Büche, *Multi-Objective Evolutionary Optimization of Gas Turbine Components*. Dissertation ETH Zürich, No. 15240, to appear, 2004.

Dirk Büche received the Dipl. degree in aerospace engineering from the University of Stuttgart, Germany in 1999. Sine 2000, he has been a PhD student at the Institute of Computational Sciences (ICOS), Swiss Federal Institute of Technologies (ETH), Switzerland. He is involved in a cooperation of ICOS with the Alstom Power Technology Center, Dättwil, Switzerland for the optimization of gas turbine components.

From September 1997 to June 1998, he was a graduate student at the Northwestern University, Evanston, Illinois. From November 1999 to April 2000, he worked as a researcher at the ABB-ALSTOM research center in Dättwil, Switzerland. His current research interests include the development of evolutionary algorithms with a focus on multi-objective problems and fitness function models for accelerated convergence.



Nicol N. Schraudolph received the B.Sc. (Hons) degree in Computer Science from the University of Essex, England, in 1988, and the M.S. degree in Computer Science from the University of California, San Diego, in 1990. From 1991 to 1995 he was a fellow of the MacDonnell-Pew Center for Cognitive Neuroscience at the Computational Neurobiology Lab of the Salk Institute, earning a Ph.D. degree in Cognitive Science and Computer Science from the University of California, San Diego, in 1995.

From 1996 to 2001 he was senior research associate at the Istituto Dalle Molle di Studi sull'Intelligenza Artificiale (IDSIA) in Lugano, Switzerland. From 2001 to 2003 he has led the machine learning group at the Institute of Computational Sciences (ICOS) of the Federal Institute of Technology (ETH) in Zürich, Switzerland. His research in machine learning currently focuses on developing rapid stochastic gradient methods, and applying them to adaptive signal processing problems.

Petros Koumoutsakos received the Dipl. degree in naval architecture and mechanical engineering from the National Technical University of Athens, Greece, in 1986, the M.Sc. degree in naval architecture from the University of Michigan, Ann Arbor, in 1987, and the M.Sc. degree in aeronautics and the Ph.D. degree in aeronautics and applied mathematics from the California Institute of Technology, Pasadena, in 1988 and 1992, respectively.

From 1992 to 1994, he was a National Science Foundation Postdoctoral Fellow in parallel supercomputing with the California Institute of Technology. Since 1994, he has been a Senior Research Associate with the Center for Turbulence Research, NASA Ames/Stanford University. From September 1997 to June 2000, he was an Assistant Professor of Computational Fluid Dynamics, Swiss Federal Institute of Technology (ETH), Zürich, Switzerland. Since July 2000 he has been a Full Professor of Computational Sciences at ETHZ. His current research interests are in the areas of multiscale computation and biologically inspired optimization and the application of these techniques to problems of interest in the areas of engineering and life sciences.