Parallel Budgeted Optimization Applied to the Design of an Air Duct

Ramunas Girdziusas, Rodolphe Le Riche, Fabien Viale, David Ginsbourger

Parallel Budgeted Optimization Applied to the Design of an Air Duct

ANR OMD2 Project Deliverable No. WP3.2.1

Ramūnas Girdziušas[1]
[1]DEMO, Centre Fayol, EMSE,
158 cours Fauriel,
Saint-Étienne, France
girdziusas[at]emse.fr

Rodolphe Le Riche[1,2]
[2]CNRS, UMR 5146 Cl. Goux,
France
leriche[at]emse.fr

Fabien Viale[3]
[3]INRIA, 2004 rue des Lucioles,
BP 93 06902, Sophia Antipolis,
France
fabien.viale[at]inria.fr

David Ginsbourger[4]
[4] IMSV, University of Bern,
Alpeneggstrasse 22, CH-3012
Bern, Switzerland
david.ginsbourger[at]stat.unibe.ch

Abstract

This work explores the benefits of cloud computing in the development of kriging-based parallel optimization algorithms dedicated to expensive-to-evaluate functions. We first show how the application of a multi-point expected improvement criterion allows to gain insights into the problem of shape optimization in a turbulent fluid flow, which arises in the automobile industry. Our work then proceeds with a variety of experiments conducted on the ProActive PACA Grid cloud. Due to a multiplicative increase in search space dimensionality, the multi-point criterion cannot exploit a large number of computing nodes. Therefore, we employ the criterion with an asynchronous access to the simulation resources, when the available nodes are immediately updated while accounting for the remaining running simulations. Comparisons are made with domain decomposition which is applied here as an alternative parallelization technique. Our experiments indicate weaknesses in the use of the multi-point criterion with a synchronous node access, and benefits when working in the asynchronous mode. Finally, a relatively fast and accurate method is developed for the estimation of the expected improvement at multiple points.

Contents

1 Introduction

We shall study optimization of expensive-to-evaluate functions (budgeted optimization) with a particular application in the design of the shape of an air duct. The latter demands time-consuming numerical simulations of a turbulent fluid flow. Our aim is to implement and parallelize the algorithms known as Bayesian optimization [11], and in particular, the *expected improvement* (EI) algorithm [27, 28]. More specifically, our work relies on a multi-point EI criterion studied in [35, 20, 18], and the goal is to test the algorithms with synchronous and asynchronous node access.

1.1 Expected Improvement

The sequential algorithms that we aim to parallelize have first been developed independently by J. Mockus and H. Kushner in the early 60s [27, 22]. Both authors considered Gaussian process models for an expensive-to-evaluate function and suggested maximization of the auxiliary quantities for the generation of new candidate locations. H. Kushner advocated maximization of the probability of an improvement (PI), J. Mockus studied both, the probability and expectation of an improvement.

The third prominent direction of a budgeted optimization utilizes the upper confidence bound (UCB) of an improvement [14]. Recent publications have abbreviated the algorithm as GP–UCB and supplied it with a wealth of analyses about Gaussian processes in the setting of the so called *multi-armed bandit problem* [5, 38, 6]. The key difference from the previous algorithms here is that the law between exploitation (sampling at the regions with a low preditive mean) and exploration (sampling where a predictive variance is high) changes as the optimization proceeds in time. In addition, the focus here is on sharper bounds of the so called cumulative regret function which can be a temporal integral of the absolute differences between the ideal sought cost function value and the value obtained at a particular time. The aim is to minimize or bound the cumulative regret by temporally changing the deviation weight in the UCB expressions.

A recent survey of the use of the three criteria can be found in [11] where they are also referred to as *acquisition functions*. Preferences over them remain rather subjective, and we shall focus on the expectation-based algorithms because they have less parameters to adjust.

The authors of [33] emphasize the lack of convergence proofs related to the EI algorithms. This problem has been investigated more thoroughly only recently [40, 12, 42]. The exists a proof for two continuity classes of objective functions, albeit for algorithms that use Gaussian processes with fixed covariance function parameters [40]. In this regard, it could be worth citing the following text [12]:

"*...For practitioners, however, these results are somewhat misleading. In typical applications, the prior is not held fixed, but depends on parameters estimated sequentially from the data. This process ensures the choice of observations is invariant under translation and scaling of f, and is believed to be more efficient (Jones et al., 1998, §2). It has a profound effect on convergence, however: Locatelli (1997, §3.2) shows that, for a Brownian motion prior with estimated parameters, expected improvement may not converge at all.*"

It is possible to develop better parameter estimators [12], but our algorithms in general do not re-estimate the covariance function, as they can easily be fixed before each optimization, clf. Eq. (9). Another example of a simple rule of thumb for setting up the covariance function parameters before the optimization can be found in [6]. The ability to use fixed parameters is hardly a practical limitation of the EI algorithms.

A more relevant problem is that the so called NEB assumption stated in [40] does not provide convergence results for the Gaussian processes whose covariance function is the Gaussian kernel. Recently, it has been established in [42] that there exist a class of univariate analytic (infinitely differentiable) objective functions which cannot be optimized with the EI algorithm that relies on the Gaussian kernel. One should bear in mind, however, that "*realistic optimization budgets may be too low in many problems for the indicated asymptotic behavior to be relevant*" [42].

The mismatch between the theory and practice is also evident as often the smoothness class of an objective function is neither known nor even relevant. In addition, hardly any existing algorithm can be implemented so that the global maximum of an acquisition function is always reached. This displaces the actual programs further away from their theoretical counterparts discussed in [40, 12, 42].

1.2 Early Ideas of Parallelization

Once the cloud computing became widespread, it has been realized that most of the algorithms are sequential, and their parallelization demands a separate research. A parallel EI algorithm [37] may utilize a gradient-based maximization of the single point EI criterion, applied with multiple starting points. Parallelization can thus be achieved by enriching the standard EI algorithm with local maxima of an acquisition function.

Another early practical attempt to parallelize relevant algorithms is reported in [33]. Instead of the improvement-based criteria, the authors utilize a variety of other "acquisition functions" and compare their algorithms with the one developed in [37]. Notably, parallelization is achieved by using multiple reference cost function values $f_{\min}$ in the EI-related criteria. The generation is created by adding one point at a time, and each point is obtained by maximizing the EI criterion with different reference values. Uniqueness of candidate points is achieved by imposing distance constraints.

Considering the parallelization performed in [33, 34], one can draw a useful warning that the speed-ups over sequential algorithms can be quite small. For example, with four computing nodes, the speed-ups are generally less than four, and for the modified Rosenbrock and Ackley functions, each with five variables, the reported speed-ups are 1.83, and 1.44. Our results will indicate a problem where speed-ups can be lower. This difficulty could be avoided by designing algorithms which can leverage a larger number of computing nodes. However, one should note that various stochastic sampling methods have already been studied with large generation sizes, and the speed-up values have often turned out to be bounded by $\mathcal{O}(1)$ [39].

1.3 Dynamic Parallelization

Many existing parallelization ideas somewhat blindly generate multiple candidate points at a time by capitalizing on the fact that budgeted optimization algorithms have a lot of free parameters. Dynamic parallelization tries to predict the outcome of a sequential algorithm without the use of expensive function evaluations. It may also switch off parallelization at the times when the prediction is not possible, and thus adaptively request additional evaluations of an expensive function.

Most of the presently known dynamic parallelization algorithms, see e.g. [10, 15] rely on a heuristic sequential technique, first introduced by M. Schonlau [35]. The core insight utilizes the fact that the variance of any Gaussian process conditioned on the observations does not depend on the actual observation value, but only on its spatial location. This property can be exploited to create a batch (generation) of distinct candidate locations bypassing their expensive evaluation sequantially, thus, allegedly speeding up the optimization. The candidate points are generated one at a time by maximizing an acquisition function and updating the predictive variance (and possibly, but not necessarily, predictive mean).

This technique is applied in [6, 8], where the generation of new locations is built in a sequential manner described above, by maximizing one-point EI criterion at a time, and simply replacing the corresponding expensive function values with the ones sampled from its posterior density conditioned on the current design of experiments (DOE). After obtaining a sample of candidate points, clustering is then performed to decrease redundancy and size of the generation. The clustering criterion is simply the sum of weighted Euclidean distances between the generation points and its cluster centers. The weights are probabilities that a certain cluster point provides a better cost function value than the rest of the cluster centers. There are no known explicit expressions for such probabilities even in the case of normal variables, and thus the assumption of independence is made and the standard formulae of the Gaussian order statistics is employed. The experiments have been performed with generation sizes fixed to 5 and 10.

In their more recent research [10], the authors drop out the clustering-based method entirely, and they build the generation directly (without any postprocessing) by maximizing one-point EI criterion in the spirit of their previous method. However, the generation size is made adaptive and it increases only if the bound on the deviation of the predictive mean from its true value (that would, in theory, be obtained with a sequential one-point EI algorithm) does not exceed a specified threshold. A newly added location in the generation must be associated not with an arbitrary cost function value (mean, random sample from posterior), but its globally optimal value which is assumed to be known. Often, this is indeed the case when only the globally optimal argument of an expensive function is unknown, but the cost function value itself can be determined

with a satisfactory precision.

A very similar in spirit parallelization, albeit of the GP–UCB algorithm, called GP–BUCB, is presented in [15]. One difference is that the process mean function is employed to model the expensive-to-evaluate values during the construction of the generation, but the mean values of the generation points may not even be updated. Instead, the UCB deviation weight is adjusted when building the generation whose size also changes dynamically. The latter is controlled by an available cumulative regret bound. The authors of [15] also suggest replacing the exact variance updates with certain bounds in order to speed up the creation of a new generation of candidate locations. This trick is also employed in [10], but the latter work uses different bounds. An interesting byproduct of both of these methods is that they provide indicators of when an expensive function evaluation should be performed, and when it is good enough to use the regression model to generate a new candidate location.

However, in addition to the difficulties of setting up newly introduced threshold parameters, the problem with these methods is that they cannot effectively explore all the available computational nodes as the size of the generation is determined algorithmically and changes with time, while parallel resources are often fixed and limited. Another drawback is that the sizes can be nonuniform, which may yield suboptimal total optimization times.

The latter aspect is addressed in [7, 9]. The authors assume that there exist a specific distribution for the duration of an expensive evaluation, and the total optimization time is limited by a fixed known value. The number of total function evaluations is also fixed, and so is the maximal size of the generation of candidate locations. Assuming this information exists, the authors develop a general model which aims to distribute generation sizes and determine the corresponding durations for their parallel evaluations. They introduce the so called CPE criterion, which is a cumulative temporal sum of the number of jobs completed at a time. Its maximization is shown to prefer uniform schedules (distributions of the generation sizes) and can thus be used to limit the parallelization so that the algorithm utilizes more expensive function values and is still able to meet a specified time horizon.

One difficulty with this general setting is that parallel execution times are stochastic (and often the exact distributions are unknown or changing), but the model imposes the upper limit on the duration for the evaluation of the generation. Thus, the evaluation may actually fail to complete, and the authors further address this difficulty by introducing the notion of a probabilistic safety of an algorithm. Therefore, the aim is to maximize the probability of a safe completion which is not guaranteed to be unity.

1.4 Our Preferences

Instead of applying sequential heuristic techniques discussed above, we shall directly maximize *the multi-point EI criterion*, which seems to have been introduced by M. Schonlau, see §5.3 in [35], and whose practical relevance has been justified only recently, see e.g. [18, 20]. It has been demonstrated that a multi-point EI will be large where, simultaneously, the corresponding one-point EI values are large, and the generation points are not correlated. Thus, the multi-point EI criterion gives preference to distinct multiple candidate points automatically, without any additional parameters, heuristic distance constraints, or additive penalty functions.

The criterion demands fewer adjustable parameters, but its maximization is only possible when the generation sizes λ are small, typically $\mathcal{O}(1)$. It should be understood that a small value of λ does not limit the parallelization. In particular, we shall advocate *an asynchronous node access* where one first submits a large number of expensive function evaluations to the cloud, and then updates only λ nodes at a time (the algorithm remains parallel even when $\lambda = 1$).

The multi-point EI criterion has already been applied to select parameters of various statistical models in order to further increase their performance on some known machine learning benchmarks [36]. We shall report deviations in the optimization evolutions w.r.t. the initial DOE, which turn out to be higher than the error bars that can be seen in [36]. This indicates that certain parameters, such as an initial DOE, can affect the outcome of the optimization results more than a better regression model. High performance variability w.r.t. the initial DOE is also reported in [32].

An attempt to improve maximization of the multi-point EI criterion is presented in [13], where it is shown how to compute the gradient of this acquisition function analytically. This is a research direction which could be very important for the asynchronous node access where the time it takes to generate and communicate new points (blocking time) should be minimal. Maximization of the multi-point EI criterion is also a computational bottleneck during the testing of any of the relevant algorithms, and a faster maximization would provide an appreciable aid here. However, one should bear in mind that the multi-point EI criterion is multimodal, and there is no easy way to reach its global maximum with local optimization techniques.

One could emphasize that the framework introduced in [7, 9] is a very general formalization of a budgeted optimization problem. Our asynchronous optimization study that will be presented in Section 4 corresponds to a particular case which the authors call *Online Fastest Completion Policy* (OFCP). This policy is just a strategy to calculate and evaluate new λ candidate locations immediately as λ computational nodes become available. Their main critique, and quite a profound insight, is that "*it does not use the full time horizon, even when doing so would allow for much less concurrency*" [7]. The works in [7, 9] introduce a new perspective to Bayesian optimization because they explicitly quantify and minimize the actual optimization time instead of relying on a prevalent statement that Bayesian optimization is "known to be efficient".

We do not necessarily advocate the use of this policy over others and our results, provided in Section 4, could be seen as a further analysis and numerical evidence that only better characterizes this policy. However, the OFCP policy is a natural choice when the overall time horizon is not given, or when the exact timing characteristics of the expensive evaluations are not known (but we shall provide analysis when such information is available). The OFCP policy simply works with an assumption of a fixed number of total function evaluations, it maximizes the node occupation time, does not need any sophisticated scheduling, and there is obviously no need to consider a probabilistic safety in this case. For the sake of simplicity, we shall bypass the decision theoretic vocabulary and instead of the OFCP policy shall frequently employ a less informative description of the asynchronous node access.

1.5 Structure of the Report

The report first provides the results of the application of a synchronous four-point EI algorithm to the industrial problem of shape design, which are summarized in Section 2. The optimization operates in such a way that one first submits four points for their evaluation, and then waits until all of them are completed. The regression model is then updated, the multi-point EI criterion is maximized, and the process is continued until the budget of expensive evaluations is exceeded. The evaluation of a cost function takes about twenty minutes. We improve a recently reported result in [31], and provide insights into physical, and statistical aspects of the problem.

Section 3 states performance results of various parallelization techniques dedicated to a synchronous node access. Our results indicate that a simple strategy such as the domain decomposition is competitive with more advanced methods, but there are problems where none of the methods is suitable for parallel optimization and a single point EI algorithm performs equally well. One should note that the tests are structured in such a way that parallel algorithms are executed on a single machine, and independent simulations pertaining to different initial DOEs are then sent to the cloud to assess how an initial DOE affects the results. The reason for this particular way of utilizing the cloud is that timing characteristics of the parallel algorithms can be rather obvious, and in the testing phase the cost functions are not expensive to evaluate.

Optimization with an asynchronous node access is discussed in Section 4. We state a particular model for the execution time of an expensive-to-evaluate function, simulate the asynchronous point generation scenarios based on the proposed timing model, and test the performance of the multi-point algorithms by submitting independent optimizations, each with a different initial DOE, to the cloud. Here the focus is on the average time for a new generation to *actually* be sent to the cloud, which will be referred to as a *wall clock time*. A wall clock time depends not only on the time it takes to maximize the improvement and to communicate the results to the remote nodes, but also on *when and where a particular node becomes available while other nodes are active with the evaluation*. This is one difference with the previous work on budgeted optimization considered in the literature.

Section 5 focuses on the possibilities to further speed-up evaluations of the integral for the multi-point EI criterion. We have observed in our numerical experiments that the integral has a peculiar property that its upper bound lies extremely close to the actual value, especially (but not necessarily) when the examined expected improvements are further away from the locations where they are maximal. In essence, we choose to work within the framework of *systematic sampling* [16] (as opposed to *importance sampling*) and show that one can considerably improve symmetric monomial rules (unscented transforms) by replacing monomials with integrated one-point improvements. However, one must also mention that a standard Monte Carlo sampling proves to be a very reliable integration technique, especially at the locations where the expected improvements are maximal.

As will be seen in the results provided in this report, a significant benefit of using a computing cloud is that it allows large scale testing of the algorithms with different parameter settings. For example, parameters, such as an initial DOE, greatly affect the optimization results and are very hard to "integrate out". The ability to utilize cloud resources allows one to actually send replicas of the original simulation with parameter changes and then see the effects. This is very hard to achieve when running things locally on a single computer (in a serial manner) because a budgeted optimization of inexpensive-to-evaluate functions is itself a very time-consuming process. In our work, a single cost function evaluation in the rank-one matrix approximation problem may take microseconds to evaluate, but a single complete optimization may easily reach ten hours (when the CPU rate is 2.5GHz). Our ability to run the codes on the ProActive PACA Grid cloud [3] allows to obtain about one hundred such independent optimizations in a day, which is a remarkable asset in testing.

2 Shape Optimization

2.1 Expensive-to-Evaluate Function

Our goal is to optimize the geometry of a cooling duct, which has already been studied in [31]. The criterion is the normalized pressure difference of the flow at the inlet and outlet of a duct, which is indicated in Fig. 1a. The optimization parameters are shown in Figs. 1b–d.

It will suffice to emphasize that the criterion is a positive quantity whose computation is a demanding numerical solution of the k–ϵ model of a fluid flow. The flow is *linear*, viscous ($\nu = 1.6 \cdot 10^{-4}\ m^2/s.$), incompressible, and turbulent ($Re = 4000$).

The k–ϵ model is a mixed system of *nonlinear* partial differential and algebraic equations [17]:

$$\frac{\partial \bar{u}_i}{\partial t} + \bar{u}_j \frac{\partial \bar{u}_i}{\partial x_j} = \frac{\partial}{\partial x_j}\left((\nu + \nu_T)\left(\frac{\partial \bar{u}_j}{\partial x_i} + \frac{\partial \bar{u}_i}{\partial x_j}\right)\right) - \frac{1}{\rho}\frac{\partial}{x_i}\left(\bar{p} + \frac{2}{3}\rho k\right), \tag{1}$$

$$\frac{\partial \bar{u}_j}{\partial x_j} = 0, \tag{2}$$

$$\frac{\partial k}{\partial t} + \bar{u}_j \frac{\partial k}{\partial x_j} = \frac{\partial}{\partial x_j}\left(\left(\nu + \frac{\nu_T}{\sigma_k}\right)\frac{\partial k}{\partial x_j}\right) + \nu_T\left(\frac{\partial \bar{u}_i}{\partial x_j} + \frac{\partial \bar{u}_j}{\partial x_i}\right)\frac{\partial \bar{u}_i}{\partial x_j} - \epsilon, \tag{3}$$

$$\frac{\partial \epsilon}{\partial t} + \bar{u}_j \frac{\partial \epsilon}{\partial x_j} = \frac{\partial}{\partial x_j}\left(\frac{\nu_T}{\sigma_\epsilon}\frac{\partial \epsilon}{\partial x_j}\right) + C_{\epsilon 1}\frac{\epsilon}{k}\nu_T\left(\frac{\partial \bar{u}_i}{\partial x_j} + \frac{\partial \bar{u}_j}{\partial x_i}\right)\frac{\partial \bar{u}_i}{\partial x_j} - C_{\epsilon 2}\frac{\epsilon^2}{k}, \tag{4}$$

$$\nu_T = C_\mu k^2/\epsilon. \tag{5}$$

It describes the time averages of the pressure field p and the flow velocity field $\mathbf{u}$:

$$\bar{p} \equiv \lim_{T\to 0}\frac{1}{T}\int_0^T p(\mathbf{x}, t)dt,\ \ \bar{u}_i \equiv \lim_{T\to\infty}\frac{1}{T}\int_0^T u_i(\mathbf{x}, t)dt. \tag{6}$$

The auxilliary fields k, ϵ, and ν_T are the turbulent kinetic energy k, the spatial dissipation rate of k, called ϵ, and the turbulent viscosity ν_T, resp. One should notice that the kinematic viscosity ν is a constant, while ν_T is a field.

The initial and boundary conditions are indicated in Table 1. The implementation uses the open source library called OpenFOAM [2]. The wall functions "kw", "ϵw", and "ν_Tw" are the OpenFOAM functions "kqRWallFunction", "epsilonWallFunction", and "nutWallFunction", resp. The latter two override their default parameter values with $C_\mu = 0.09$, $\kappa = 0.41$, $E = 9.8$. The initial values of the quantities computed by the wall "functions" correspond to the initial values of the fields shown in the last column of Table 1.

In addition to OpenFOAM, a complete software stack of this fluid dynamics simulation includes CATIA [1] (a 3D model of a duct), STAR–CCM+ [4] (computational mesh generation), and ParaView [19] (visualization).

2.2 Algorithm

It is not transparent how the pressure difference depends on the parameters which specify the geometry of a duct. Various admissable changes of the geometry are not visually discernable, and the model is a massive nonlinear dynamical system. This motivates the application of a budgeted optimization. This type optimization estimates the kriging model of an expensive-to-evaluate function, and generates new candidate locations by maximizing the multi-point expected improvement. In particular, given μ active points $\mathbf{x}_{1:\mu}$ and λ free nodes, the algorithm finds λ new points by solving the following problem:

$$\max_{\mathbf{x}\in\mathbb{R}^{d\lambda}} \mathbb{E}\Big(\max\left(0, \min\left(f_{\min}, Y(\mathbf{x}_{1:\mu})\right) - \min Y(\mathbf{x})\right)|A\Big), \tag{7}$$

Table 1: Initial and boundary conditions for key quantities of the k–ϵ model.

Name	Field	Units	Boundary conditions			Initial conditions
			Inlet	Outlet	Wall	
$\tilde{p} = \bar{p}/\rho$	Normalized pressure	$\frac{m^2}{s^2}$	$\nabla\tilde{p} = \mathbf{0}$	$\tilde{p} = 0$	$\nabla\tilde{p} = \mathbf{0}$	0
$\mathbf{u}$	Flow velocity	$\frac{m}{s}$	$-\mathbf{n}$	$\mathbf{0}$ if $\mathbf{u}\cdot\mathbf{n} \leq 0$	$\mathbf{0}$	$\mathbf{0}$
k	Turb. kin. energy	$\frac{m^2}{s^2}$	10^{-3}	$\nabla k = \mathbf{0}$	"kw"	10^{-3}
ϵ	Dissipation Rate of k	$\frac{m^2}{s^3}$	10^{-1}	$\nabla\epsilon = \mathbf{0}$	"ϵw"	10^{-1}
ν_T	Turbulent Viscosity	$\frac{m^2}{s^3}$	0	$\nabla\nu_T = \mathbf{0}$	"ν_Tw"	0

where $f_{\min}$ is the current minimum, $Y(\mathbf{x}_{1:\mu}) = (Y(\mathbf{x}_1), \ldots, Y(\mathbf{x}_\mu))$ and $Y(\mathbf{x}) = (Y(\mathbf{x}_{\mu+1}), \ldots, Y(\mathbf{x}_{\mu+\lambda}))$ are random surrogates (kriging model). A denotes the event when Y values equal to all known expensive-to-evaluate functions at all the known locations. Methods to compute the expectation in Eq. (7) are discussed in Section 5.

Considering the use of kriging in the optimization, one may refer to [18, 20] for more details. In addition, we have applied a few changes to what seems to be a standard practice. They are not conceptually interesting, but are worth mentioning:

1. The expected improvement is maximized by using the CMA-ES algorithm [29, 39]. Box constraints are handled by projecting the coordinates on the bounds and adding the penalty term to an expected improvement. The penalty is proportional to the Euclidean distance from the optimization point to the boundary if the point is out of bounds, and is zero otherwise.

2. Conditional expectations are calculated by using the pseudoinverse of the DOE covariance matrix. This method overestimates the conditional variances, but it does not demand any additional parameters, and it also reduces to the standard inverse in the absence of singularities.

3. When the conditional covariance matrix of the kriging responses is singular, the value of the expected improvement is simply set to zero. Here by "singularity" it is meant anything that breaks the Cholesky decomposition. The latter is placed inside the "try block" of the "try and catch" exception handling.

4. Multi-point expected improvements are calculated by using the Monte Carlo sampling with one thousand points. This standard method is simple, computationally inexpensive, and reliable w.r.t. increasing dimensions of an integration domain. The seed of the random generator is set to the current generation number, so that the integration routine uses the same random points when evaluating the expected improvement at different spatial locations.

5. Kriging is applied with Gaussian kernels whose variances vary with each coordinate. The variances are determined by squaring the median of the absolute deviations from the median of a particular coordinate. This is simpler and faster than any iterative estimation and, more importantly, it guarantees that the appearance of close points in DOE does not change kernel variances unexpectedly.

We shall apply what is known as the synchronous multi-point algorithm [18] with $\lambda = 4$ points, which is briefly abbreviated as $\mathrm{EI}^{0,4}$. The choice of generating four points at a time demands the optimization with $8 \times 4 = 32$ variables. Asking for more points at a time, or using DOEs with more points than $\mathcal{O}(10^3)$ would introduce severe numerical difficulties.

2.3 Results

The minimization of the pressure difference is shown in Fig. 2. The first 320 observations are generated by using the Latin Hypercube Sampling (LHS) algorithm, so that the actual optimization starts at the

Table 2: Main Results

	LOBS	UPBS	Worst	[31]	Our result
x_1	0.0036	0.0166	0.0036	0.0149	0.0132
x_2	0.3	0.8	0.3760	0.4202	0.4756
x_3	0.0027	0.0207	0.0207	0.0102	0.0207
x_4	0.0405	0.0595	0.0595	0.0479	0.0450
x_5	1.25	1.5707	1.2525	1.5582	1.5707
x_6	0.21	0.42	0.2254	0.3849	0.3914
x_7	0.047	0.055	0.055	0.0541	0.0547
x_8	0.0008	0.0088	0.00081	0.0014	0.0016
pd	nan	nan	1.28	0.59 ± 0.01	0.56 ± 0.01

observation number 320. The optimization then proceeds via a synchronous generation of four candidate points. They are obtained by maximizing the expected improvement with the CMA-ES algorithm [29] which uses its default parameters, except that the initial coordinate deviation is chosen to be 0.1, and the number of iterations is set to 3000.

Optimization results are presented in Table 2. One can see the bounds of the variables, the worst observed point which gives the maximal pressure difference value 1.28, previously available best result [31], along with our improvement. The presence of "nan" values indicates that the pressure values are not available at the points whose all coordinates are simultaneously equal to either the lower or upper bound. The geometry cannot be meshed in these two extreme cases.

2.4 Analysis of Results

The optimization results can also be highlighted by comparing the optimal fields with the worst observed cases. The worst observed geometry is shown in Fig. 3. It only serves the purpose of displaying the slicing plane on which the field values will be displayed, and in setting up the range for the pressure values, which is $[0, 4]$. The surface of the duct is colored with the ParaView [19] scheme "hsv-blue-to-red" whose range of colors is also displayed in the color bar.

The values of the pressure field on the surface and its slice are shown in Fig. 4. Both shapes are hard to discern visually, but the differences can still be noticed without any special tools. In the optimized case, the pressure values are smaller on the walls at the inflection of the duct.

The components of the velocity field are shown in Fig. 5. For comparison purposes, the ranges of the field values are kept the same in both the worst and optimal cases, and the color space is the one used with the pressure fields, clf. Fig. 3. The ranges for the x, y, and z-components are $[-0.3, 1]$, $[-0.4, 1.4]$, and $-1.6, 0.2$, resp. One can see that the velocity field of a flow in the optimized case is generally smoother, and effectively uses a larger volume of a duct.

The optimized geometries are very hard to discern visually, and the pressure fields are nearly optically identical, which is also accompanied by rather small differences in the numerical values of the pressure fields. However, the results are not identical, and the differences become most pronounced when looking at velocity fields shown in Fig. 6. One can see that our result is slightly smoother, which can be seen in the upper left areas of the slices in the x and z-components (a,d,c,f), and at the inflection point of a duct in the case of the y component (b,e).

In order to see if our result differs from the one in [31] statistically, we have performed the principal component analysis on the data *correlation* (not covariance) matrix. The data is the matrix of size 8×788 whose columns are the candidate locations generated during the optimization (the data correlation matrix is of size 8×8). The results are shown in Fig. 7. They indicate the projections of the data vectors on the chosen eigenvectors of the correlation matrix. In addition to the data, several important locations are

Table 3: Eigenvectors of the correlation matrix of all the geometries.

Coord.	$\mathbf{v}_1$	$\mathbf{v}_2$	$\mathbf{v}_3$	$\mathbf{v}_4$	$\mathbf{v}_5$	$\mathbf{v}_6$	$\mathbf{v}_7$	$\mathbf{v}_8$
1	0.47061	−0.01957	0.00218	−0.34455	−0.09434	−0.65004	−0.26550	−0.39683
2	−0.24119	−0.39696	0.20201	−0.27114	−0.80995	0.01527	0.04344	0.10855
3	−0.50275	−0.03497	0.19678	0.10206	0.19948	−0.48401	−0.50937	0.40419
4	0.04089	0.33272	0.87867	0.14932	−0.04204	0.13262	−0.04856	−0.26749
5	−0.49912	0.22259	−0.30076	−0.06057	−0.11561	0.22181	−0.39944	−0.62056
6	−0.33944	0.20969	0.11621	−0.74648	0.29979	−0.07762	0.41716	0.01265
7	−0.16022	0.56533	−0.18227	0.33491	−0.36915	−0.44528	0.41711	0.02842
8	−0.27554	−0.56302	0.10589	0.31937	0.23239	−0.26807	0.39781	−0.45799

indicated with different markers, and they are: the present optimal solution (Opt), lower and upperb bounds (LB, UP, resp.), previous result [31] (PrevBest), the average value of the bounds (Midpoint), and the worst observed point during the optimization (Worst).

As the concentration of variance by the first principal components is not very pronounced, one finds out that data does not live in a subspace of $\mathbb{R}^8$ and all the coordinates are valuable. Therefore, the parameterization of the problem is not redundant. However, the dimension of the problem could have been reduced down to $\mathbb{R}^5$ because the second, fifth, and eight pricipal components do not discriminate the optimal location from the middle point or the worst point.

When compared to the previously available result [31], our solution is situated further away from the worst case scenario when looking at things along the principal directions 1, 6, and 7, but is closer to it in the direction 8. Interestingly, in the four-dimensional subpsace spanned by the eigenvectors 2, 3, 4, and 5, the result in [31] is almost identical to ours.

The first principal component allows to separate the optimized points from the initial DOE. It turns out that the third coordinate of the first eigenvector has the largest magnitude, which, incidentally, is the only coordinate which makes our solution significantly different from the previous result (in our case x_3 is roughly doubled). For the sake of completeness, the coordinates of all of the eigenvectors are shown in Table 3.

2.5 Conclusions

When a vast majority of admissible fluid domains are optically indistinguishable, the optimization of a geometry can be hard to perform manually. Kriging-based optimization proves to be handy when making a progress with a small budget of the cost function evaluations which is typically less than $\mathcal{O}(10^3)$. We have made an improvement to the previous solution obtained in [31] and have identified its relation to our result. Interestingly, the previous optimization is almost identical to ours in the subspace of $\mathbb{R}^8$ spanned by four eigenvectors of the correlation matrix of all the points gathered during the search. The principal component analysis suggests that x_3 is an imporant parameter, and the intrinsic dimension of the problem, i. e. the number of independent parameters which could differentiate the optimal geometry from the suboptimal one, is at least five.

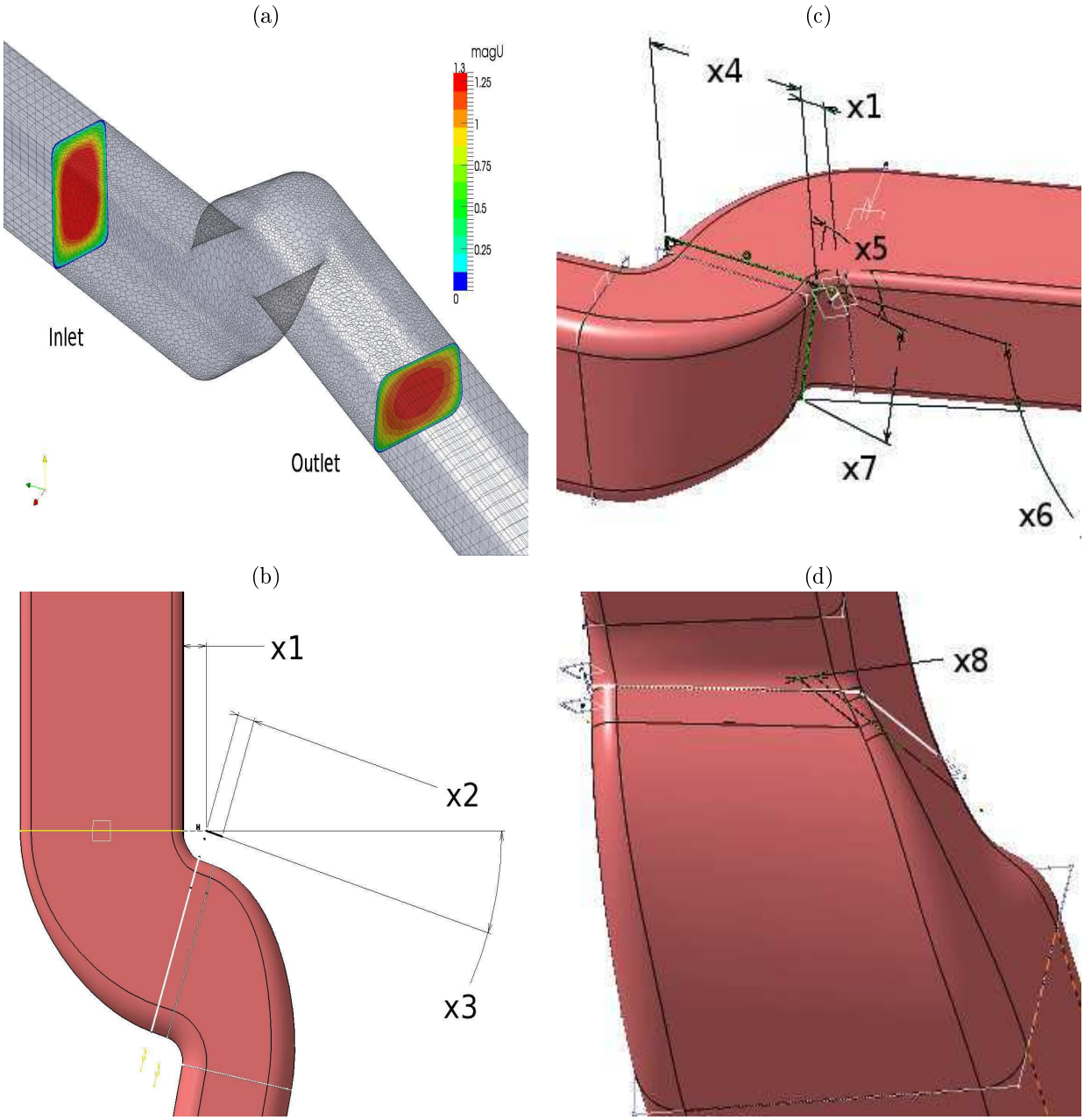


Figure 1: Optimization criterion is the difference between the average (normalized) pressure field at the inlet and outlet (a). The optimization parameters are x_1–x_3 (b), x_1, x_4–x_7 (c), and x_8 (d).

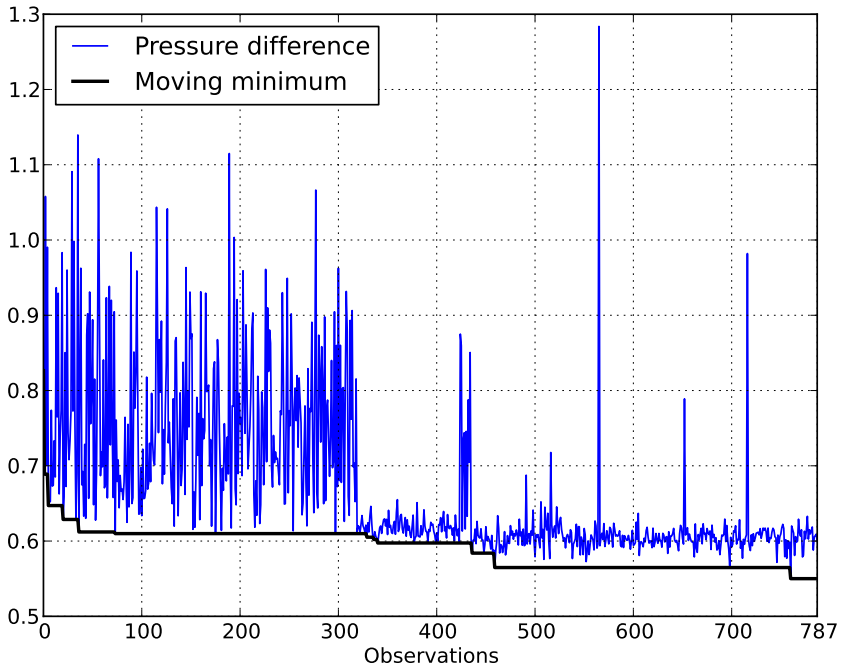


Figure 2: Normalized pressure difference $[\frac{m^2}{s^2}]$ w.r.t. increasing number of observations during the optimization. The first 320 observations are generated via the LHS algorithm.

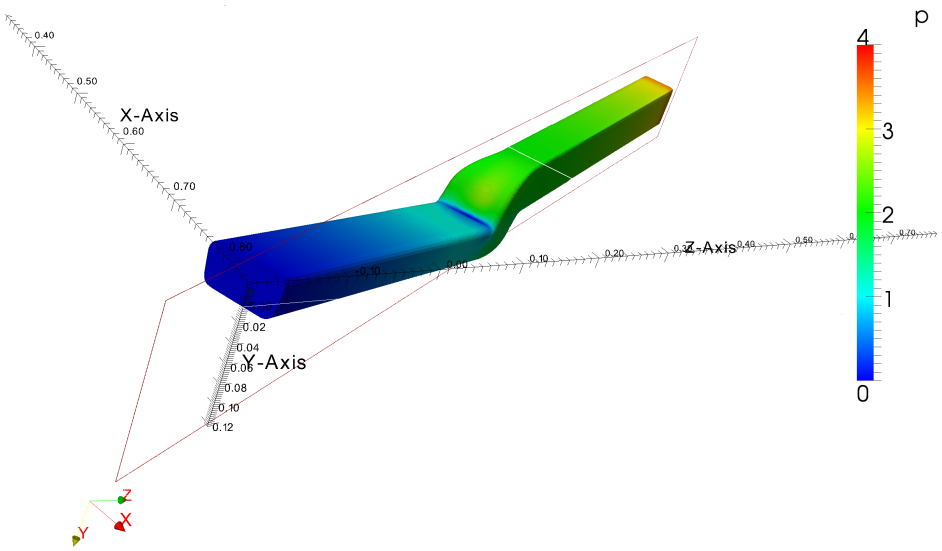


Figure 3: The example of a duct geometry, the observation plane, and the chosen color scheme for the pressure field values.

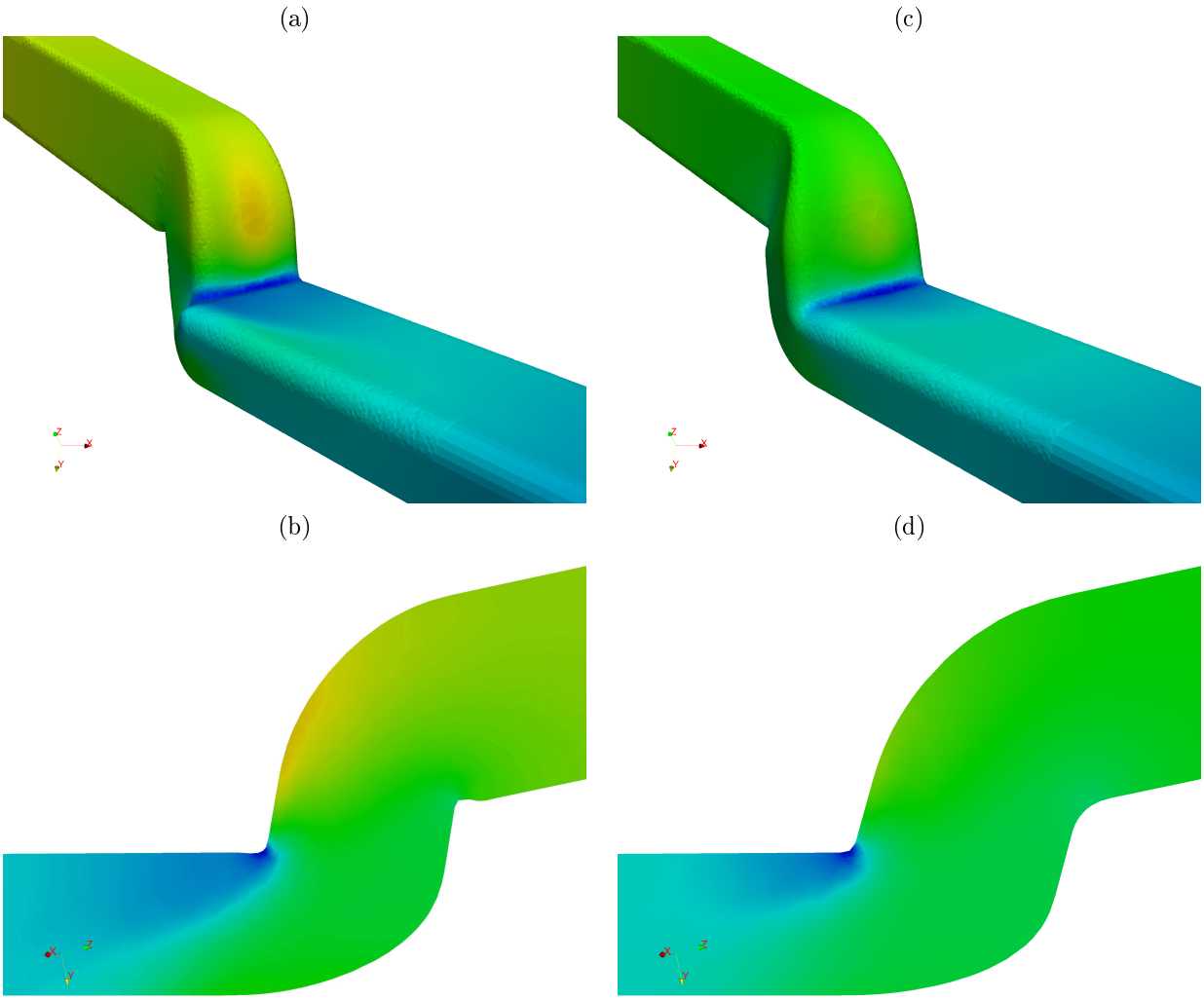


Figure 4: Pressure fields: worst observed case (a,b), and optimal solution (c,d).

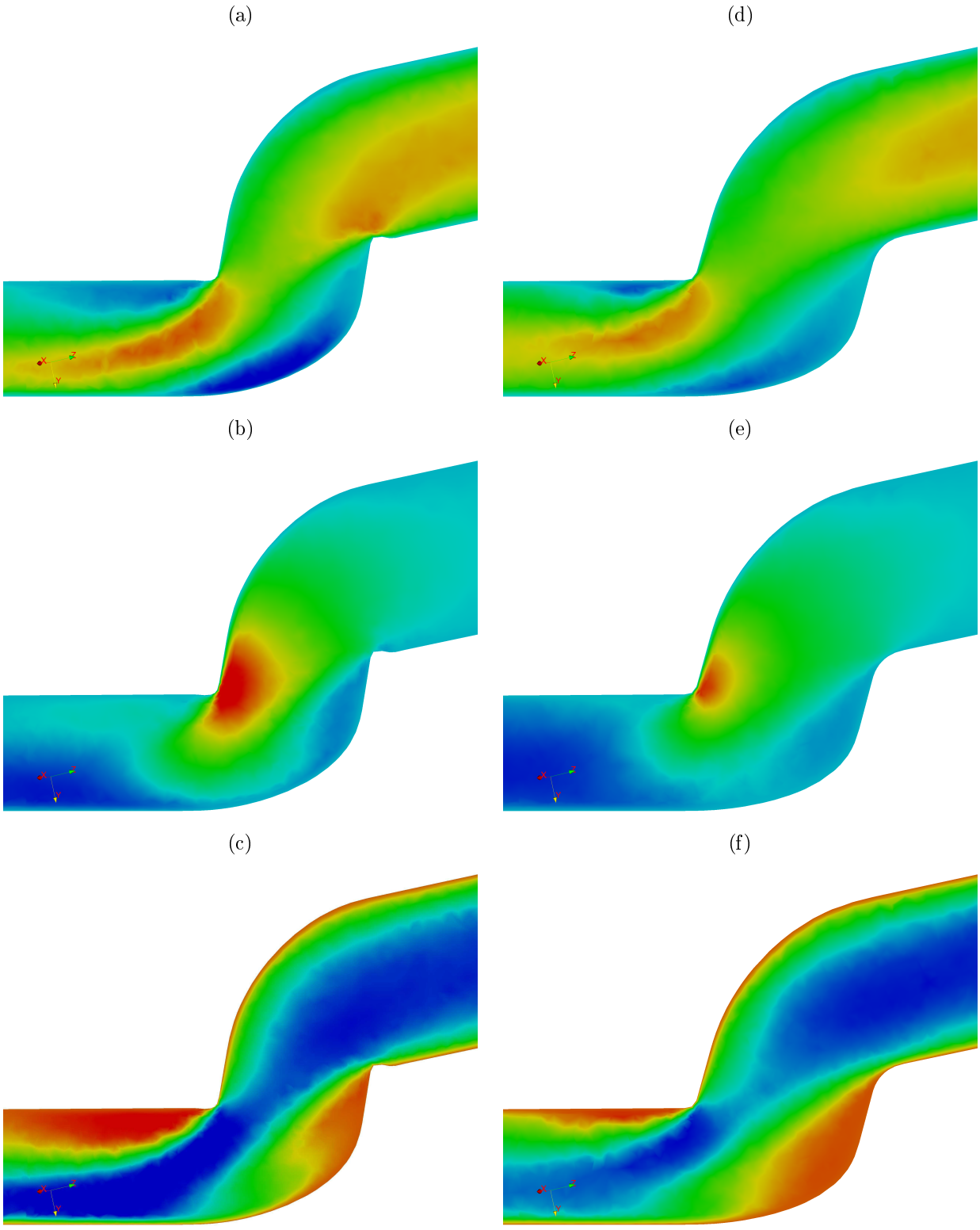


Figure 5: Velocity fields: x-components (a,d), y-components (b,e), and z-components (c,f). The first column corresponds to the worst observed scenario; the second column shows the optimized fields.

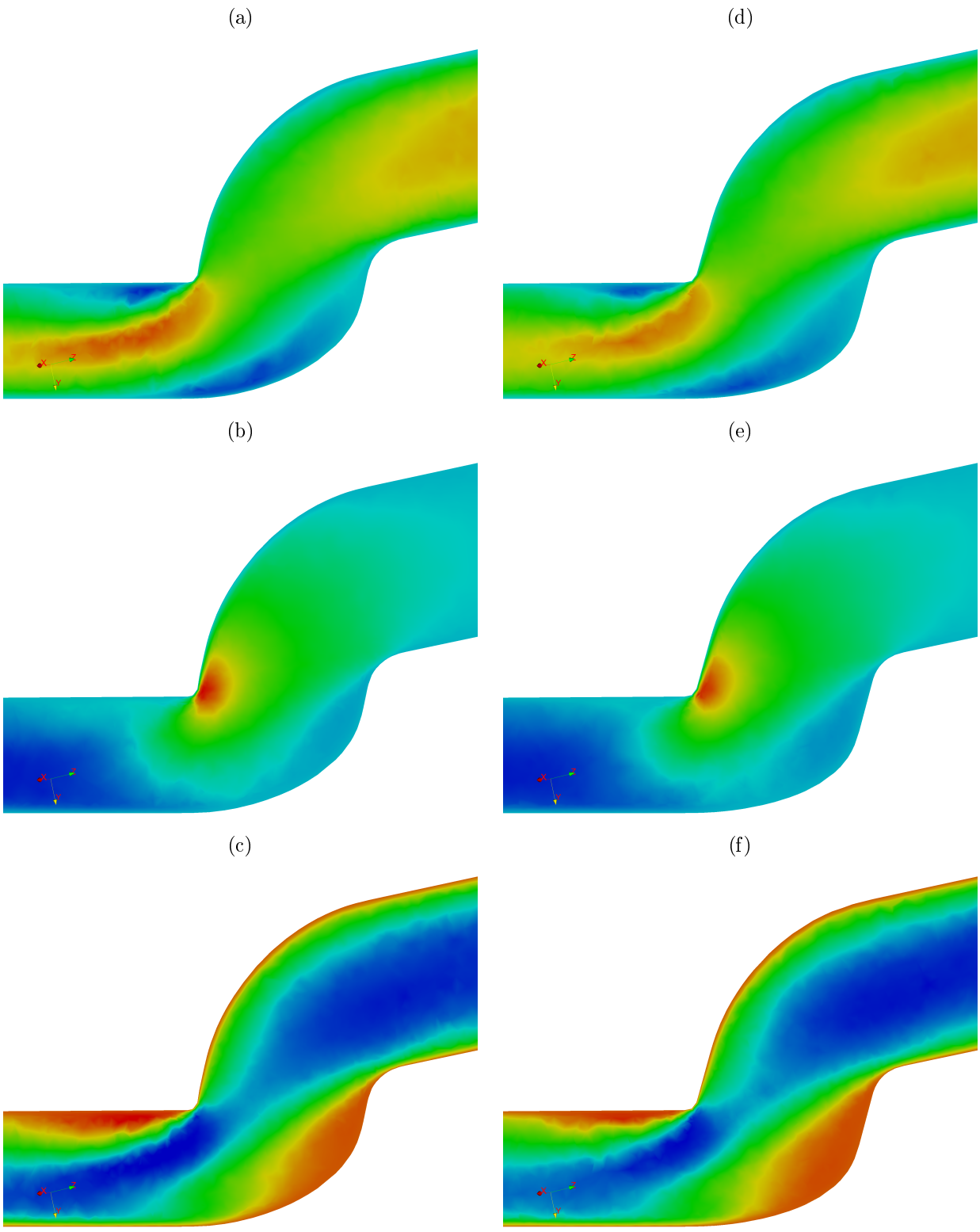


Figure 6: The components of the velocity field of a flow: x-direction (a,d), y-direction (b,e), and z-direction (c,f). The first column corresponds to the result in [31]; the second column is our result which is the replica of the second column in Fig. 5.

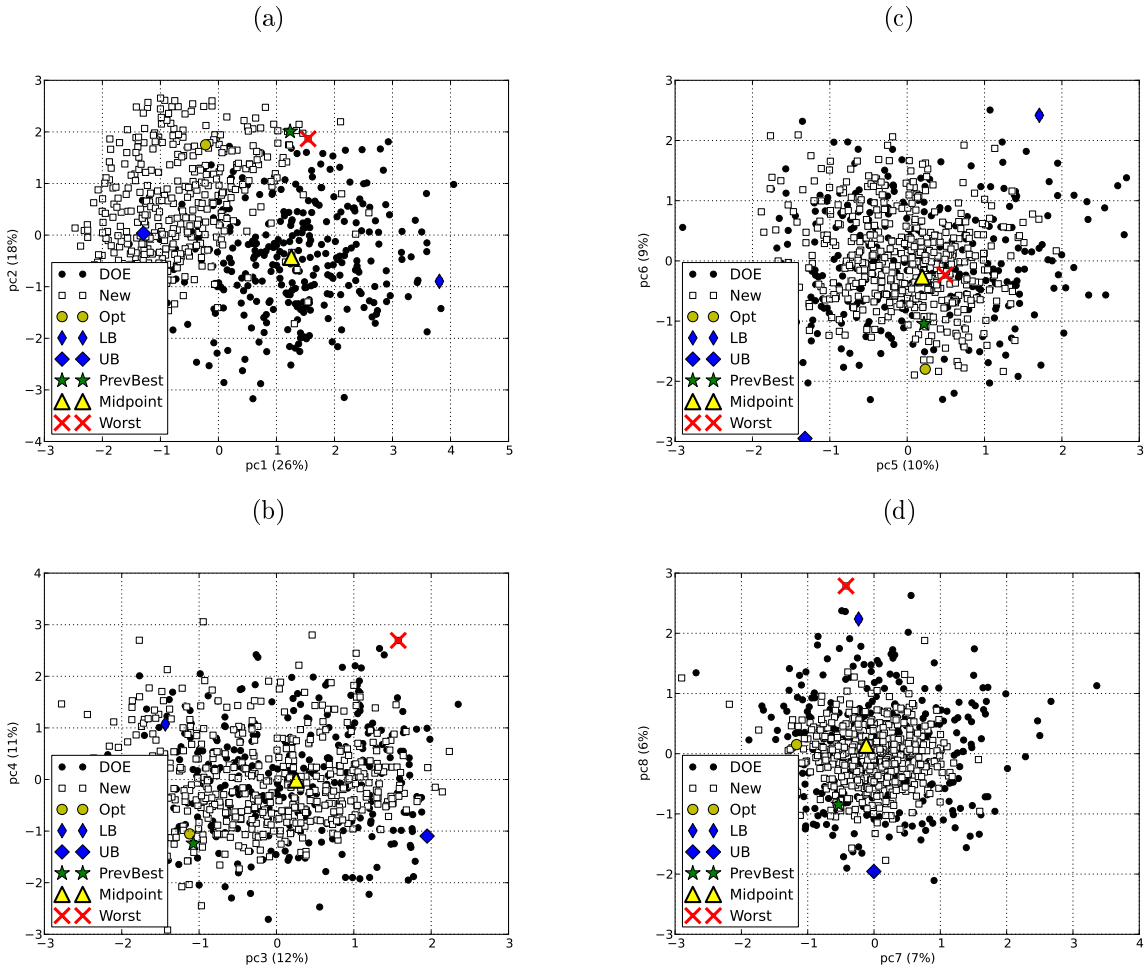


Figure 7: The principal components (scores) of candidate locations generated during the optimization.

3 Experiments with Synchronous Node Access

3.1 Introduction

Section 2 has focused on the application of a particular kriging-based optimization algorithm to the industrial problem. On average, it takes twenty minutes to evaluate a cost function in such a problem. A single optimization then demands days to complete. Considering the down-times of the cloud, a single optimization may demand weeks to complete.

Thus, one may ask whether our results rightfully reflect what can be achieved with a whole family of multi-point improvement-based algorithms described in [20]. One should note that so far we have applied only one such algorithm, which generates $\lambda = 4$ points at a time, synchronously. It was applied once, and only with a single cost function.

We shall report our tests with artificial functions, which will further indicate some limitations and unexplored possibilities of the kriging-based algorithms. In this section, we will focus on the asynchronous node access and will try to measure whether multi-point improvements help. The algorithms will be tested along with the strategy of the domain decomposition.

3.2 Algorithms

3.2.1 Multi-Point Improvements

The use of multi-point improvements [20, 18] is a theoretically appealing direct extension of the kriging algorithms with one-point improvements. The problem with this approach is that it does not scale well as the maximization of the expected λ-point improvements demands the optimization in $\lambda \times d$ dimensions. In addition, λ cannot be very large in principle because the minimum over an increasing number of random variables is pushed down independently of the demands of a problem, and thus the expected improvements become severely overestimated. They are typically overestimated anyway, but one suspects that when the generation sizes are not big, such as $\lambda = 4$, the algorithm can be implemented correctly and one may achieve a faster optimization.

How fast an optimization can be? Let us introduce the quantity called *wall clock time* (WCT), which is the average time between two consecutive updates of the nodes in the cloud. It determines the rate at which the points are sent to (received from) a remote cloud. Figure 8 presents timing analysis of the synchronous optimization with multi-point improvements.

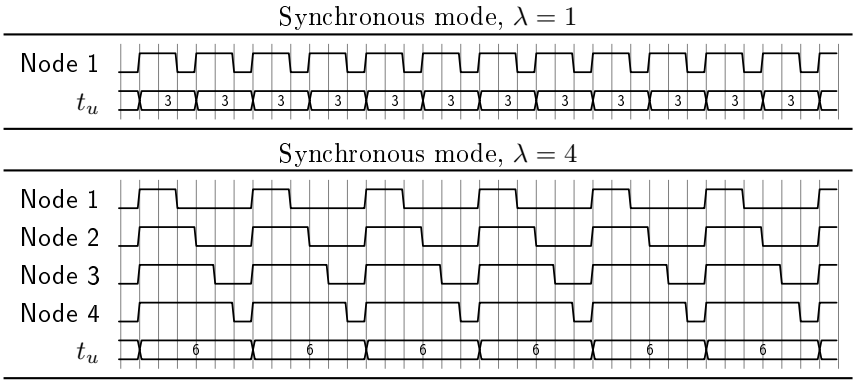


Figure 8: In the synchronous case, it is the slowest node that determines the node update time. The time costs of updating $\lambda > 1$ points will typically be greater than in the single point case, unless every one out of λ computational nodes is faster than the one applied in the optimization with one-point improvements. This example shows the case when t_b is one time unit, independently of an algorithm. The wall clock time increases twice when λ changes from 1 to 4.

The high level of the node signal indicates the time when the node is busy while calculating an expensive-to-evaluate function, and the low level spans the time when the node is idle. One can see that the use of multiple points increases the wall clock time (WCT), and the latter will be solely determined by the slowest computational node.

Let us introduce the blocking time t_b, which is the time it takes to: (i) receive λ function evaluations, (ii) generate λ candidate points, and (iii) send them to the λ free nodes. One can then perform a more precise analysis by assuming that the time it takes to evaluate an expensive function is uniformly distributed. The node update time will then be a random variable defined as

$$T_u = t_b + \max\{T_1, T_2, \ldots, T_\lambda\}, \;\; T_i \sim U(t_{\min}, t_{\max}). \tag{8}$$

For example, let $t_{\min} = 10$, $t_{\max} = 30$ and $t_b = 2$ time units. Then, WCT $\equiv E(T_u) = 22$ when $\lambda = 1$, but it increases up to WCT $= 28$ when $\lambda = 4$. Therefore, the synchronous multi-point optimization algorithm needs to approach the optimum at least $28/22 = 1.27$ times as fast in order to save time.

3.2.2 Domain Decomposition

This is one of the simplest strategies to employ when making any optimization algorithm parallel. The domain is divided into s parts (subdomains) and the optimization is performed in each subdomain independently, preferrably in parallel. In what follows, we shall perform the optimization in $d = 2$, 6, and 9 dimensions, and the number of subdomains will be 32. In the case of two dimensions, we divide the first coordinate into eight equal parts, and the second into four. In the case of a larger number dimensions, we simply halve the first five coordinates and obtain in this way $2^5 = 32$ subdomains.

It is important to emphasize that the domain decomposition is a *strategy*. It can be applied to make any algorithm parallel. We shall use it with both, one-point improvements, and multi-point improvements as well. The interesting question is whether the use of the domain decomposition with the one-point improvements can be as good as the use of multi-point improvements alone. In that case, one would definitely prefer the former, as it achieves a perfect isolation between the parallel flows of the program whereas the multi-point algorithm is more demanding regarding its implementation.

3.3 Performance Criteria

All of the optimization algorithms are made to be deterministic in order to remove the unnecessary degrees of variance. Firstly, we switch off the maximum likelihood estimation of the Gaussian kernel variances in the kriging. Instead of estimating them, the following simple rule is applied

$$\text{kernel variance}_i = \left(\frac{|\text{upb}_i - \text{lob}_i|}{2^{1+\frac{8}{d}}} \right)^2, i = 1, \ldots, d. \tag{9}$$

Here d is the number of dimensions of the optimization space.

The main idea behind this formula is that we shall typically generate 500 points during the entire optimization (including the points of the initial DOE). This is a realistic budget for an expensive-to-evaluate function on one hand, and the limit after which working with dense matrices becomes very inefficient (at best). Thus, in all of the simulations, on average, the number of observations used in kriging is 250. We then "round" this number up to $2^8 = 256$, and then $2^{8/d}$ becomes the number of "ticks" that can be placed on each coordinate axis when assuming that the points are distributed uniformly in space. The addition of unity is somewhat arbitrary and not really crucial, but it serves one purpose. When $d = 8$, the variance becomes equal to a squared "median of the median of absolute differences coordinate-wise".

In addition, the Monte Carlo (MC) integration of the expected improvement is always initialized to the current generation number. Thus, the only "degree of freedom" is the initial DOE, and each family of the algorithms can now be tested with a number of optimizations. Each optimization will then correspond to a different initial DOE. This number will be set to one hundred, but it may actually become smaller if some nodes fail to complete the optimization.

The performance of the algorithms is assessed by using three artificial functions as the optimization criteria. The details are given in Table 4.

Table 4: Optimization Criteria

Label	Cost function	Domain	Minimal value	Modality
"michalewicz2d"	$\sum_{i=1}^{2} \sin(x_i) \sin^2(ix_i^2/\pi)$	$[0,5]^2$	-1.841	multimodal
"rosenbrock6d"	$\sum_{i=1}^{5} 100(x_{i+1} - x_i^2)^2 + (1-x_i)^2$	$[0,5]^6$	0	unimodal
"rank1approx9d"	$\|\mathbf{A}_{4\times5} - \mathbf{x}_{4\times1}\mathbf{y}_{1\times5}\|_2$, $a_{ij} \sim U(0,1)$[1]	$[-1,1]^9$	0.7119	bimodal

These functions are simple to state and to implement. They are also fast to evaluate. The latter feature still does not let to perform testing on a single machine easily as a kriging-based optimization may take hours even when applied to create only one hundred generations. However, the use of the ProActive PACA Grid cloud [3] provides the possibility to test the algorithms with different initial conditions at once.

The optimization quality will be assessed by using the normalized real improvement (NRI) defined as

$$\mathrm{NRI}(\mathrm{generation}) = \frac{f_0 - f_{\min}(\mathrm{generation})}{f_0 - f_{\mathrm{true}}}. \tag{10}$$

Here f_0 is the smallest value of the cost function achieved on the initial DOE, which is created by using the Latin Hyper-Cube Sampling (LHS), $f_{\min}$ denotes the value achieved after a particular generation of points is evaluated, and f_{true} is the true ideal minimal value, which is given in Table 4.

Also, it is useful to summarize the performance of various algorithms by defining their speed-up, such as

$$S_0(\mathrm{NRI}) \equiv \frac{\text{time to reach NRI by } \mathrm{EI}^{0,1}}{\text{time to reach NRI by } \mathrm{EI}^{0,\lambda}}. \tag{11}$$

Here the reference algorithm is kriging with one-point improvements, and the speed-up is defined for the kriging-based optimization with $\lambda \geq 1$ points.

In order to take into account the blocking time, one defines the real-time speed-up of the multi-point algorithm over its single point counterpart according to

$$S_1(\mathrm{NRI}) = \frac{S_0(\mathrm{NRI})}{\mathrm{RTF}} = S_0 \times \frac{\text{WCT for the algorithm } \mathrm{EI}^{0,1}}{\text{WCT for the algorithm } \mathrm{EI}^{0,\lambda}}. \tag{12}$$

Here RTF is *a real time factor* which is the ratio of the corresponding wall clock times. The corresponding criteria for the domain decomposition are defined similarly. The WCT values of all the algorithms that are tested with the synchronous node access are given in Table 5.

3.4 Results

The optimization results are shown in Figs. 9 and 10. One can see that the optimization paths vary a lot w.r.t. the initial DOE, but this effect is less pronounced in the problem "rank1approx9d". In the space with a large number of dimensions it is harder to generate an initial DOE which contains points close to the global optimum. The problem "rosenbrock6d" seems to be easy and its solution is closer to the problem "michalewicz2d" than the "rank1approx9d" case. In the former two cases the approach to the optimum is much faster.

The values of the speed-up S_0 are compared in Table 5. One can see that parallelization brings notable improvements when solving the problems "michalewicz2d" and "rosenbrock6d", but the gain is very small for the problem "rank1approx9d". The latter point becomes especially strong if we consider the speed-up S_1 which takes into account the real time factor.

[1] The actual matrix is generated with the Scilab 5.3.3 "grand" function. The Mersenne Twister is applied with an initial seed set to the number 29.

Table 5: Wall clock times, real time factors, and speed-ups of synchronous optimization, NRI = 0.8. Parameters: $t_{\min} = 10$, $t_{\max} = 30$, $t_b = 2$.

	WCT	RTF	"michalewicz2d"		"rosenbrock6d"		"rank1approx9d"	
			S_0	S_1	S_0	S_1	S_0	S_1
$EI^{0,1}$	22	1	1	1	1	1	1	1
$EI^{0,4}$	28	1.3	3.7	2.9	2.7	2.1	1.3	1.0
$EI^{0,1}$+decom	30	1.4	2.4	1.8	2.1	1.5	0.70	0.50
$EI^{0,4}$+decom	30	1.4	4.6	3.4	4.5	3.2	1.2	0.86

One finds out that domain decomposition is about as good as the use of multi-point improvements. Both parallel optimization methods can also be combined to yield an even greater performance. However, none of the parallelization methods are worth the effort considering the "rank1approx9d" problem. Considering the domain decomposition, perhaps this is not very hard to explain. In a high-dimensional space, i.e., $d = 9$, halving the first five coordinates can make the kriging algorithms less explorative (global).

A good use of domain decomposition seems to be a quick assessment of multimodality of the cost function. Figure 11 indicates one out of one hundred optimizations in full detail. One can see that some of the optimizations reach very high NRI values indicating that the corresponding subdomain may contain the global optimum.

3.5 Conclusions

The use of multi-point improvements ($\lambda = 4$) brings notable speed-ups to the problems "michalewicz2d" and "rosenbrock6d". However, the algorithm is not as efficient as the classical EGO (i.e. $EI^{0,1}$) method in the case of "rank1approx9d". This is most likely due to the increased dimensionallity of the problem although additional tests would be necessary to study if this could also be an effect of the functional landscape. The same applies to the domain decomposition. Considering the "michalewicz2d" and "rosenbrock6d" problems, running the $EI^{0,1}$ algorithm with 32 subdomains is better than using $EI^{0,4}$ without any domain decomposition. Both methods have been combined to gain an additive effect on the overall speed-up. However, neither domain decomposition nor multi-point improvements provide an advantage with respect to EGO in the "rank1approx9d" problem.

The algorithms with multi-point improvements fundamentally cannot scale well because they internally involve a maximization of the joint improvement in $d \times \lambda$ dimensions.

In order to make a further progress, it seems that one could: (i) either increase the λ value dramatically (to tens and hundreds of points) by making substantial sacrifices in the quality of the improvement maximization, or (ii) resort to the asynchronous node access which may reduce wall clock times. The second way seems to be more viable and will further be investigated.

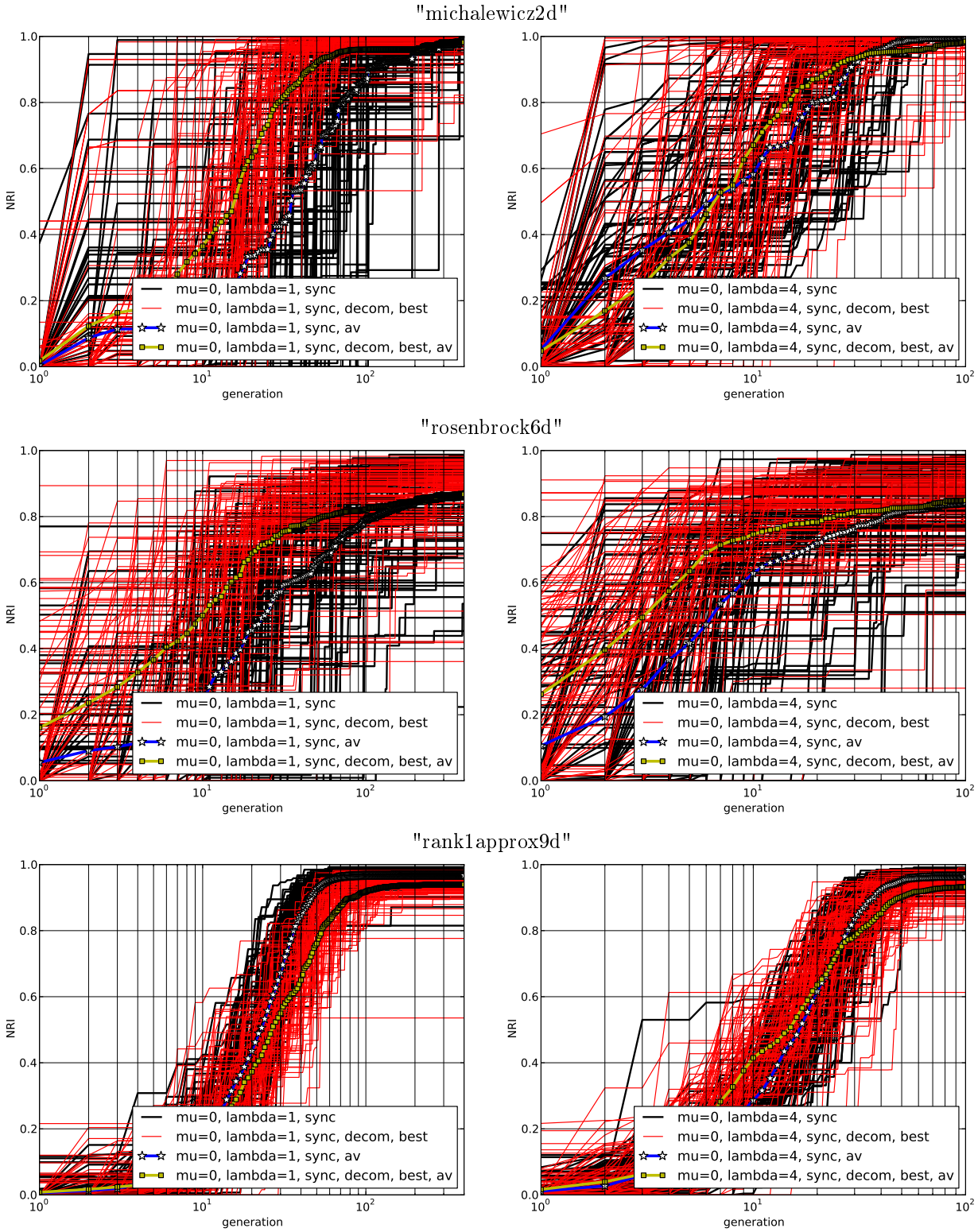


Figure 9: Optimization with a synchronous selection of points: Details.

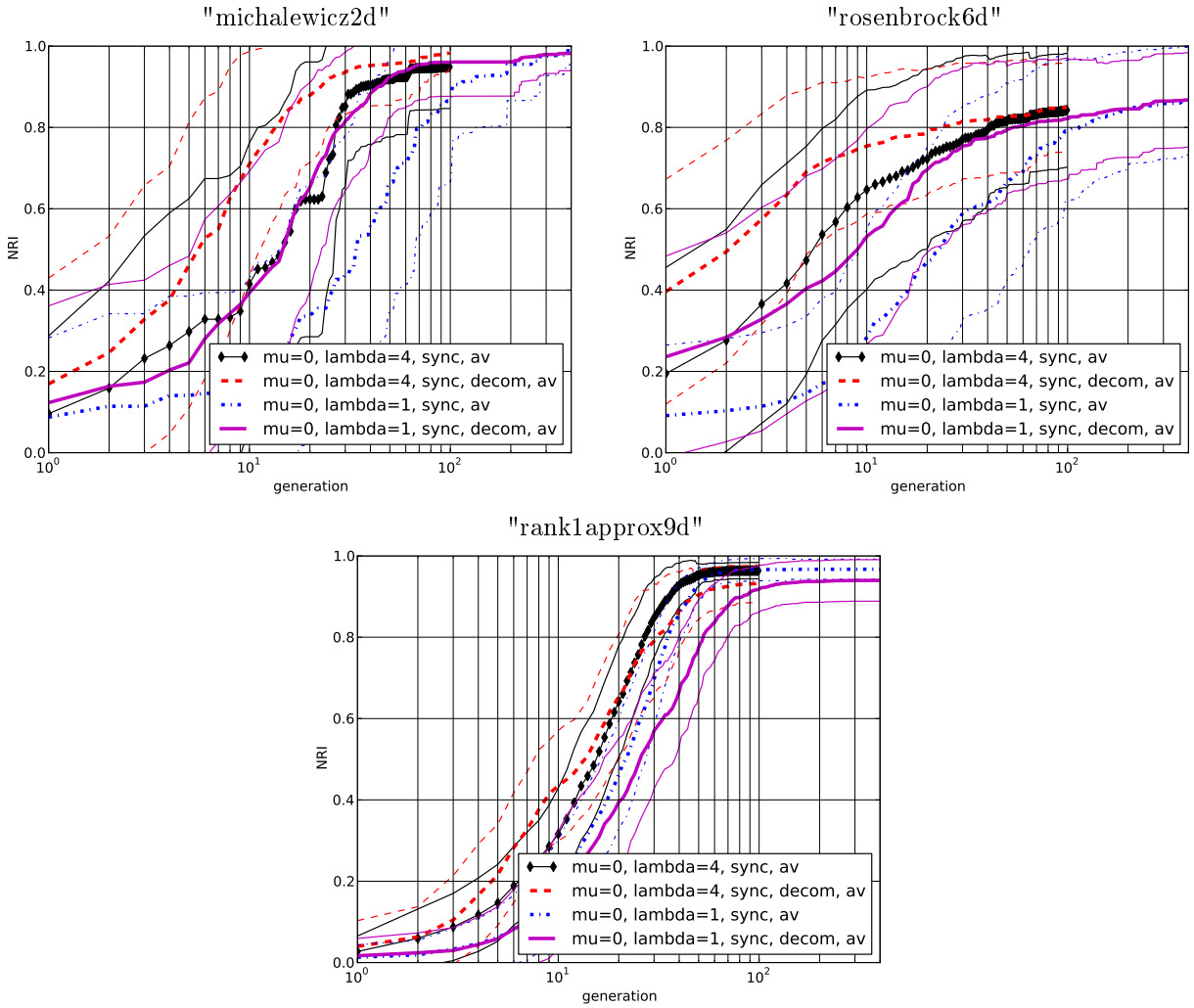


Figure 10: Optimization with a synchronous selection of points: Averages.

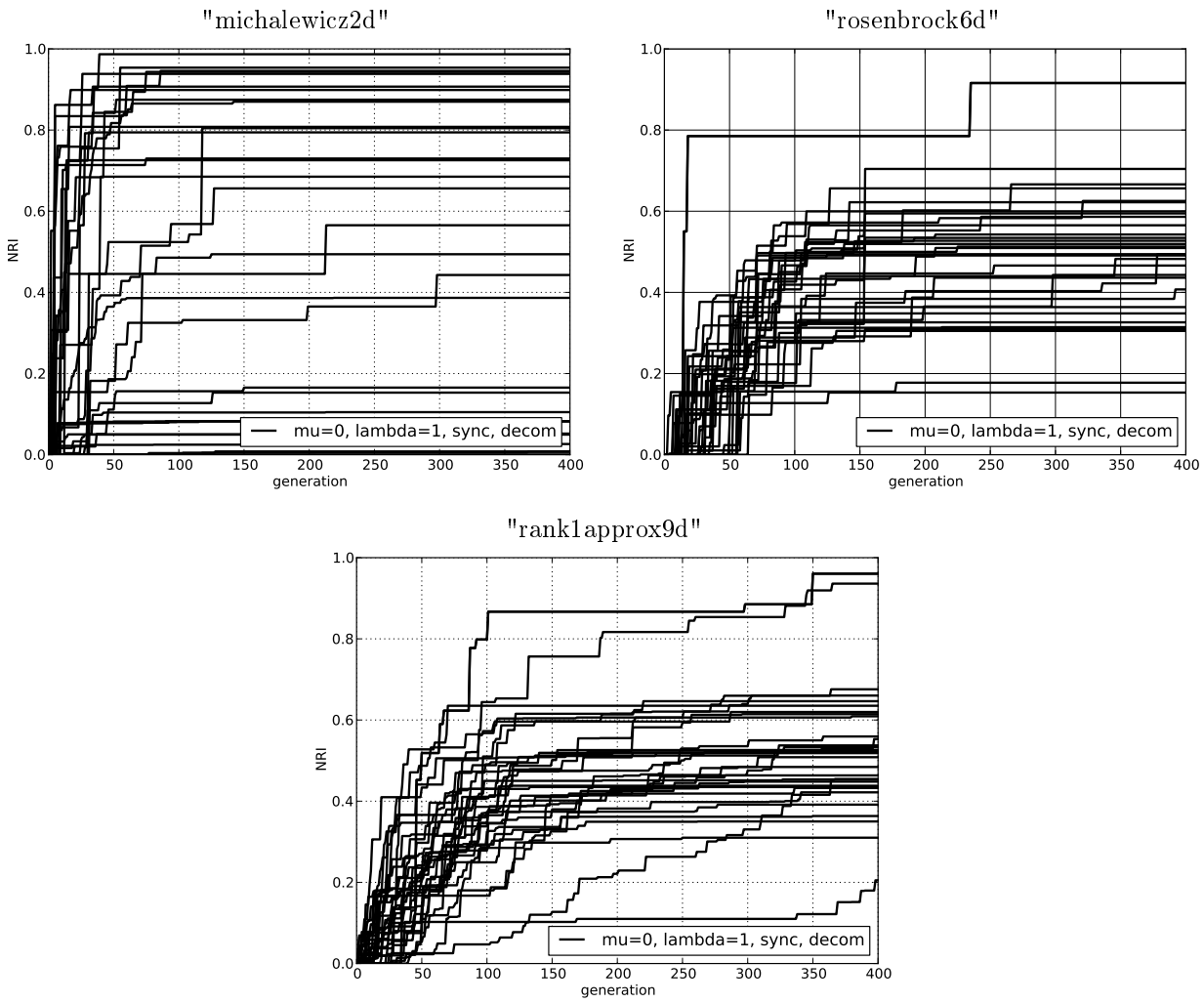


Figure 11: Optimization with domain decomposition allows to detect the presence of multimodality. Here each single optimization path corresponds to the optimization in a different subdomain. There are 32 subdomains which completely cover the original optimization domain. One infers that the "michalewicz2d" criterion is multimodal, while the "rosenbrock6d" and "rank1approx9d" problems are clearly unimodal and bimodal, resp.

4 Experiments with Asynchronous Node Access

4.1 Asynchronous Model

Let m be the number of nodes, i.e. the number of virtual machines (computers) available on a remote cloud to evaluate an expensive function. Let the average time of the function evaluation be distributed uniformly in the interval $(t_{\min}, t_{\max})$, and suppose that the access to the cloud is possible every time λ nodes provide a result. Typically, $\lambda \ll m$, such as $\lambda = 1, 2, 3, 4$ while $m = 32$. Let t_b be the blocking time which the time it takes to calculate and send λ new arguments to update the free nodes.

We will show that the wall clock time can be reduced to the blocking time by simply increasing the number of nodes m. Moreover, it turns out that the decrease of the WCT value w.r.t. m is hyperbolic, and its variance becomes negligible with an increasing value of m.

In order to show that this is possible, let us introduce an asynchronous access model. Let T be the set of m elements t_i which are the real numbers indicating the time it takes to evaluate an expensive function. The node update time can then be computed by using these steps:

1. Find λ smallest elements of T (not necessarily distinct), and create the set S out of them:

$$S = \{t_{i_1}, t_{i_2}, \ldots, t_{i_\lambda}\}. \tag{13}$$

2. Find the largest element in S, and call it the computation time t_c:

$$t_c = \max S. \tag{14}$$

3. Compute the update time

$$t_u = t_b + t_c. \tag{15}$$

4. Form the set $M = T \setminus S$, and map every element t of M according to:

$$t \mapsto \max(0, t - t_u). \tag{16}$$

5. Update the set T:

$$T = M \cup S. \tag{17}$$

The process of the node update with the asynchronous buffer model is shown in Fig. 12. The following three rules are enforced here:

1. The falling front indicates that the node becomes available.
2. It takes one time unit to update the node.
3. In case more than one node is available at the access time, the faster node is preferred.

The initial set T models the actual computational times of expensive evaluations. The simplest adequate model so far seems to be the uniform distribution with a finite support given by $t_{\min}$ and $t_{\max}$. The motivation behind this choice is the analysis of the data which we have gathered during the simulation of the expensive to evaluate functions. The latter have been chosen to be the kriging-based optimization processes themselves.

Figure 13 indicates the distributions of times that nodes demand to evaluate an expensive function on the ProActive PACA Grid cloud [3]. Here expensive-to-evaluate functions are complete budgeted optimizations of inexpensive functions whose evaluation takes only microseconds to complete. One can see that the heterogeneous nature of the cloud is such that $t_{\max} = \mathcal{O}(t_{\min})$.

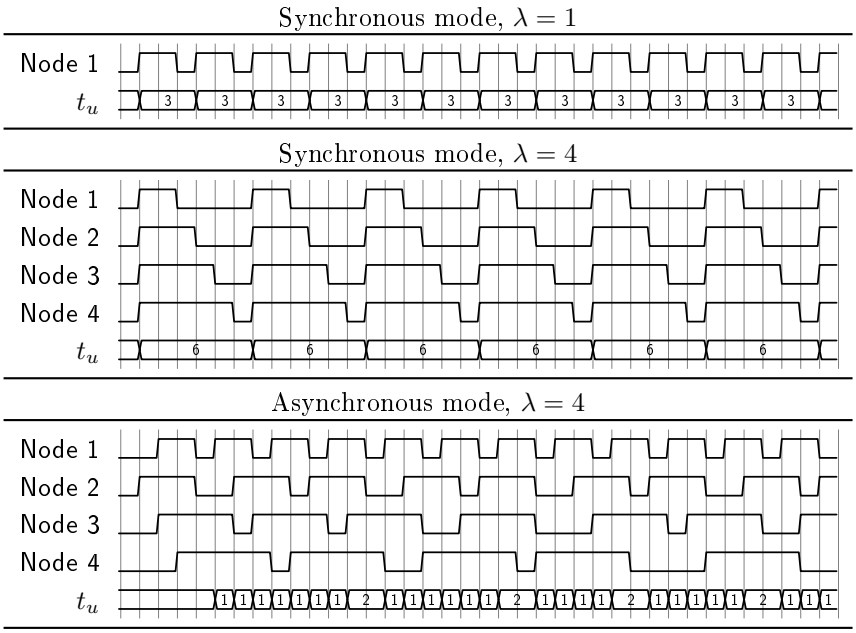


Figure 12: Advantages of the asynchronous node access. In the synchronous case with $\lambda = 1$, WCT = 3. Adding three slower Nodes 2–4 allows to have four simultaneous evaluations, but the wall clock time will be determined by the worst node. However, the asynchronous access reduces the t_u values to t_b for the majority of expensive function evaluations.

4.2 Computational Analysis of Wall Clock Time

The wall clock time could be computed by performing the five steps indicated above. They need to be repeated as many times as the number of λ-generations demands, and also repeating the runs with different initial sets T. The Scilab code of a single run is provided in Appendix sect:listingwctasync, where "buffsz" stands for m, and "lamb" for λ.

Fig. 14 indicates how the WCT value decreases w.r.t. an increasing value of m. One can see that when m is large enough, the WCT values become sharply concentrated at the t_b value.

The WCT values decrease roughly as $\mathcal{O}(m^{-1})$. A more precise rule that fits the data presented in Fig. 14 is $\mathcal{O}(m^{-1-\alpha})$, where

$$\alpha \approx \frac{t_b}{3t_{\min}}(\lambda - 1). \tag{18}$$

Notice that $t_{\max}$ is not present in the equation.

The setting that matches the ProActive PACA Grid cloud best is the one with $t_{\min} = 10$, and $t_{\max} = 30$. When $m = 32$, this allows to update $\lambda = 4$ nodes with the wall clock time approaching t_b. The relevant WCT values are shown in Table 6.

For comparison, here we have also presented the corresponding statistics with a synchronous simulation. As one can see in Table 6, the reduction of the WCT value due to the asynchronous simulation seems to be impressive. *So what exactly is optimization of an expensive-to-evaluate function? The practical function evaluation time is a function of t_{min}, m, and t_b.*

Table 6: Mean and deviation of the node update time t_u for different algorithms. Parameters: $t_{\min} = 10$, $t_{\max} = 30$, $t_b = 2$. Averaging is performed with $25 \cdot 10^4$ points.

Asynchronous	m	λ	Mean (WCT)	Deviation
True	32	1	2.04	0.0024
True	32	4	2.77	0.13
False	0	1	22.0	5.77
False	0	4	28.0	3.27

Table 7: Wall clock times, real time factors, and speed-ups of asynchronous optimization compared to the synchronous case, NRI = 0.8. Parameters: $t_{\min} = 10$, $t_{\max} = 30$, $t_b = 2$.

	WCT	RTF	"rank1approx9d" S_0	"rank1approx9d" S_1
$EI^{0,4}$ sync	28	1	1	1
$EI^{0,4}$ async	2.77	0.099	0.42	4.2
$EI^{28,4}$ async	2.77	0.099	0.56	5.7

4.3 Testing Asynchronous Algorithms

Asynchronous algorithms are expected to reduce the speed of the evolution of the optimization path towards the optimum w.r.t. the number of generations. The reason is that a direct use of the multi-point improvement criterion does not exclude the possibility of a duplicate point generation. One example of the appearance of duplicate points is illustrated in Fig. 15.

As a consequence, the evolution paths of optimization might tend to have more jump discontinuities when the criterion $EI^{0,\lambda}$ is employed in the asynchronous settings. A direct remedy is to utilize a full criterion $EI^{\mu,\lambda}$ where μ points correspond to the candidate locations whose expensive function values are being actively evaluated, but are not known at the time when a request comes to send a new candidate for the evaluation. Eq. (7) states that including active points $x_{1:\mu}$ in the target part of the EI criterion prevents the algorithm from resampling there [20]. It can be seen that if the new λ points form a subset of the μ active points, then $EI^{\mu,\lambda}$ will be zero. More generally, $EI^{\mu,\lambda}$ decreases as some of the new λ search points get closer to active points [20].

The application of the synchrononous algorithm with the $EI^{0,4}$ criterion, as well as the two corresponding asynchronous algorithms, to the "rank1approx9d" problem is summarized in Fig. 16.

One can see that the asynchronous algorithm with the $EI^{0,4}$ criterion is inferior to its synchronous counterpart, but the inclusion of $\mu = 28$ active points improves the algorithm. Still, the $EI^{28,4}$ algorithm makes a slower progress w.r.t. the number of generations. While duplicates are not the major issue anymore, one can notice that a synchronous algorithm always uses a complete information, i.e. both, the location, and the expensive function value, while the asynchronous case only excludes the appearance of duplicates, but it will often do it "blindly" without an available function value.

The examples of the speed-up values are provided in Table 7.

The S_0 values indicate that asynchronous algorithms can make the progress w.r.t. generations slower (2x) than the corresponding synchronous cases, but the real time factor is crucial and may result in an asynchronous algorithm which runs five times faster in a real time.

Optimization paths of asynchronous algorithms are compared with the synchronous cases in Fig. 17. The corresponding means and deviations are shown in Fig. 18. The results indicate that optimization paths increase slower w.r.t. the number of generations when the algorithms are asynchronous. However, one must really calculate the precise values of the speed-ups and then incorporate the real time factors to see a full

Table 8: Wall clock times, real time factors, and speed-ups of asynchronous optimization compared to those of the synchronous case, NRI = 0.75. Parameters: $t_{\min} = 10$, $t_{\max} = 30$, $t_b = 2$.

	WCT	RTF	"michalewicz2d"		"rosenbrock6d"		"rank1approx9d"	
			S_0	S_1	S_0	S_1	S_0	S_1
$EI^{0,1}$ sync	22	1	1	1	1	1	1	1
$EI^{0,4}$ sync	28	1.3	3.8	3.0	2.9	2.3	1.3	1.0
$EI^{0,1}$ async	2.04	0.093	0.86	9.3	0.43	4.6	0.27	2.9
$EI^{0,4}$ async	2.77	0.13	2.0	16	1.2	9.4	0.73	5.8

Table 9: Wall clock times, real time factors, and speed-ups of asynchronous optimization, NRI = 0.75. Parameters: $t_{\min} = 10$, $t_{\max} = 30$, $t_b = 2$.

	WCT	RTF	"michalewicz2d"		"rosenbrock6d"		"rank1approx9d"	
			S_0	S_1	S_0	S_1	S_0	S_1
$EI^{0,1}$ async	2.04	1	1	1	1	1	1	1
$EI^{31,1}$ async	2.04	1	0.89	0.89	0.95	0.95	1.4	1.4
$EI^{0,4}$ async	2.77	1.4	2.3	1.7	2.8	2.0	2.7	2.0
$EI^{28,4}$ async	2.77	1.4	3.0	2.2	2.8	2.0	2.9	2.2

picture. The summary is presented in Table 8. One finds out that the asynchronous node access with the $EI^{0,1}$ criterion may yield $1/S_0 \approx 1/0.27 \approx 3.7$ slower approach to NRI = 0.75 ("rank1approx9d"), but the real time speed up $S_1 = 2.9$ is notable. With the asynchronous access to $\lambda = 4$ nodes, the slow-down of the NRI value increase w.r.t. the number of generations becomes less pronounced. Considering the "rank1approx9d" problem, $1/S_0 \approx 1/0.73 \approx 1.4$, and the real time speed-up $S_1 = 5.8$ exceeds four.

Fig. 19 indicates the optimization evolutions for the asynchronous algorithms with or without the use of μ points. Fig. 20 provides the corresponding summary with NRI averages and their deviations.

Integrating out μ points improves the optimization speed-ups w.r.t. the number of generations, but the actual figures are not very significant. When $\lambda = 1$, the "rank1approx9d" problem yields the value of $S_0 = 1.4$ which is close to our results obtained with very few optimization runs. In particular, $S_0 \approx 0.56/0.42 \approx 1.3$, clf. Table 7. However, the speed-up can be less than unity in the other two considered problems. When $\lambda = 4$, the use of the EI criterion with μ points yields small improvements over the corresponding algorithms with $\mu = 0$.

Fig. 19 reveals that in most cases, the consideration of μ active locations improves the results.

4.4 Conclusions

In the synchronous node access, the wall clock time is given by the slowest node WCT $= \mathcal{O}(t_{\max})$, while the asynchronous mode is paced by the blocking (i.e., optimization and communication) time, WCT $\approx \mathcal{O}(t_{\mathrm{b}})$, $t_{\mathrm{b}} \ll t_{\max}$. Asynchrony slows down the optimization progress w.r.t. the number of generations, but it reduces the WCT values dramatically. The inclusion of active points may improve the asynchronous algorithms, but one should note that the improvements are small. The use of active points may increase the blocking time t_b as the multi-point improvement criterion demands evaluating integrals over domains with higher dimensions. In turn, the generation of new candidate points becomes more costly.

We have neglected this aspect of the problem because one can further parallelize the evaluation of Gaussian conditional expectations [44, 30], and even apply better improvement maximization algorithms [13]. However, one should note that without such adjustments, considering the problem "rank1approx9d", t_b may

increase 10 times when μ increases from 0 to 28.

In the case with the "rank1approx9d" problem, the asynchronous algorithms with the criteria $\mathrm{EI}^{0,1}$ and $\mathrm{EI}^{0,4}$ can be $1/0.27 \approx 3.7$ and $1/0.73 \approx 1.4$ times slower than a synchronous $\mathrm{EI}^{0,1}$ algorithm when making the progress w.r.t. the number of generations. However, the corresponding algorithms will be 2.9 and 5.8 times faster in a real time. Therefore, asynchronous algorithms do not typically achieve best results if the number of function evaluations is a sole performance criterion. Instead, the asynchronous node access provides a faster optimization in a real time.

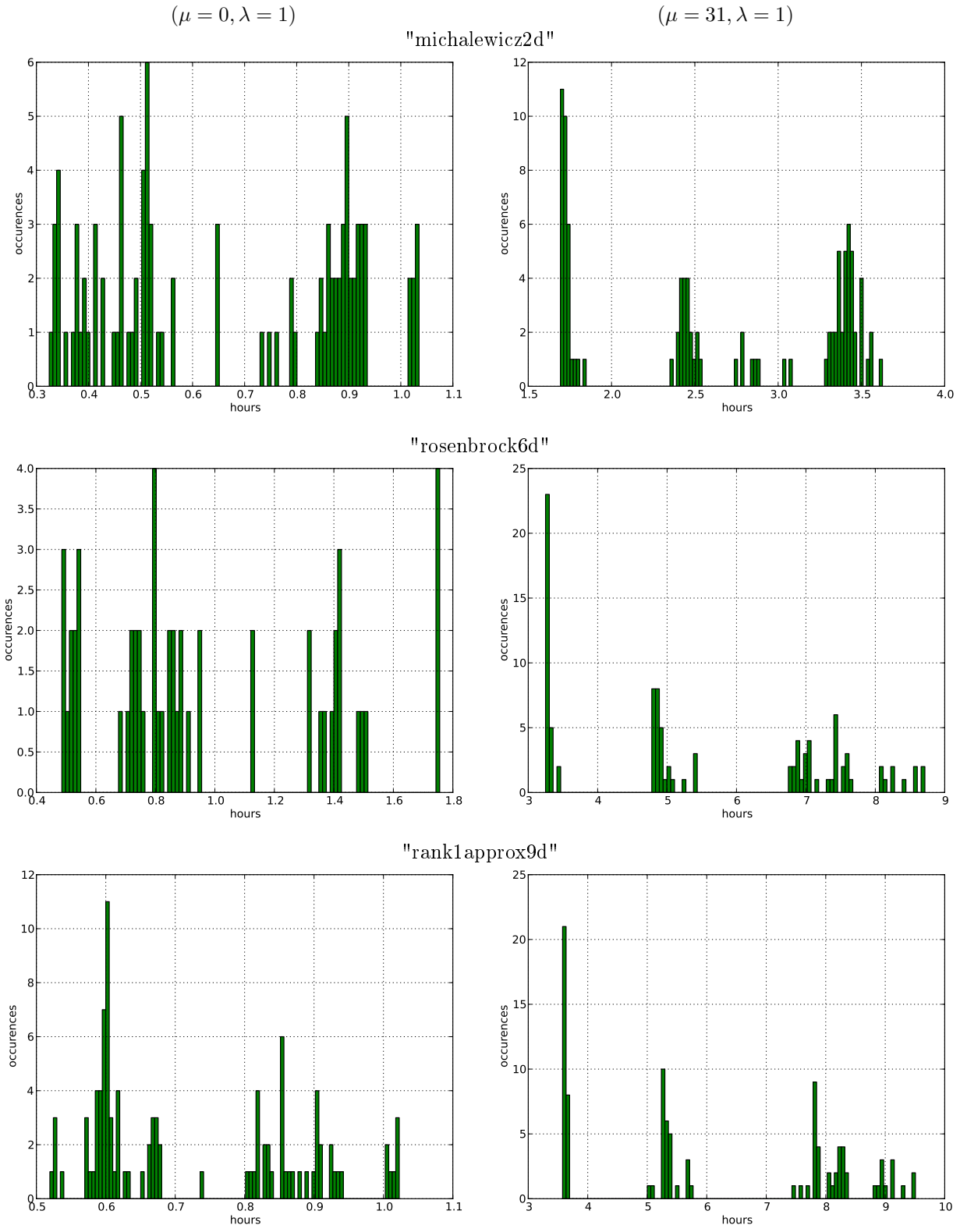


Figure 13: Example times needed to evaluate expensive functions. Each histogram indicates node counts for different time values spent to evaluate expensive functions.

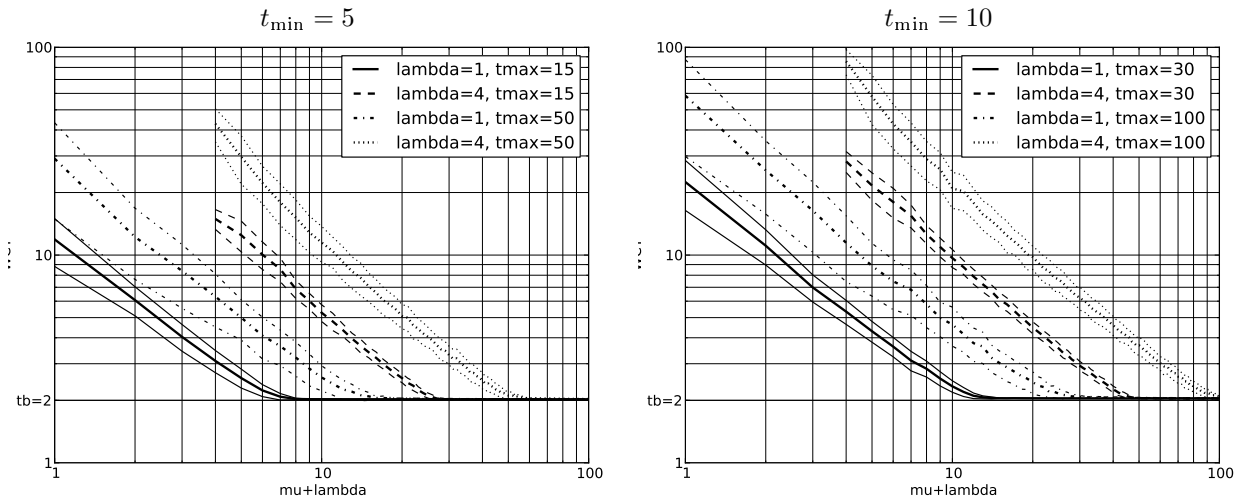


Figure 14: The use of an asynchronous buffer dramatically decreases the WCT value which approaches t_b. In this example $t_b = 2$, averaging is performed over ngenerations $= 250$, and with 100 different runs. Auxiliary thinner lines indicate the estimated deviations.

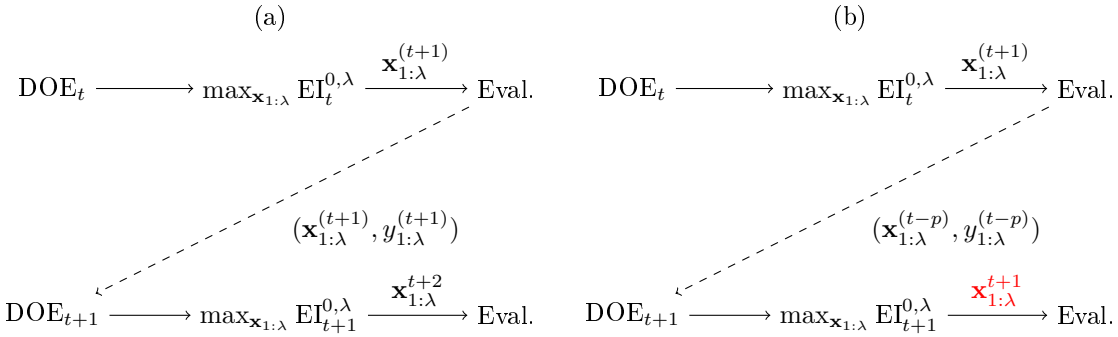


Figure 15: Appearance of duplicate points in the asynchronous algorithms which maximize $\mathrm{EI}^{0,\lambda}$ directly without any algorithmic improvements. In the synchronous case (a), the points $\mathbf{x}_{1:\lambda}$ are sent to the cloud, and one waits for the corresponding expensive function values. DOE_{t+1} is then updated which in turn ensures that the points $\mathbf{x}_{1:\lambda}^{(t+1)}$ cannot be regenerated in the next maximization. In the asynchronous scenario (b), the points $\mathbf{x}_{1:\lambda}$ may be sent to the cloud as in the synchronous case, but one receives the λ points $\mathbf{x}_{1:\lambda}^{(t-p)}$ which have potentially (but not necessarily) been sent to the cloud earlier, $p \geq -1$. If the latter do not affect DOE_t much, then the next maximization produces the points which can be arbitrarily close to $\mathbf{x}_{1:\lambda}^{(t+1)}$. These points are duplicates and are marked in red.

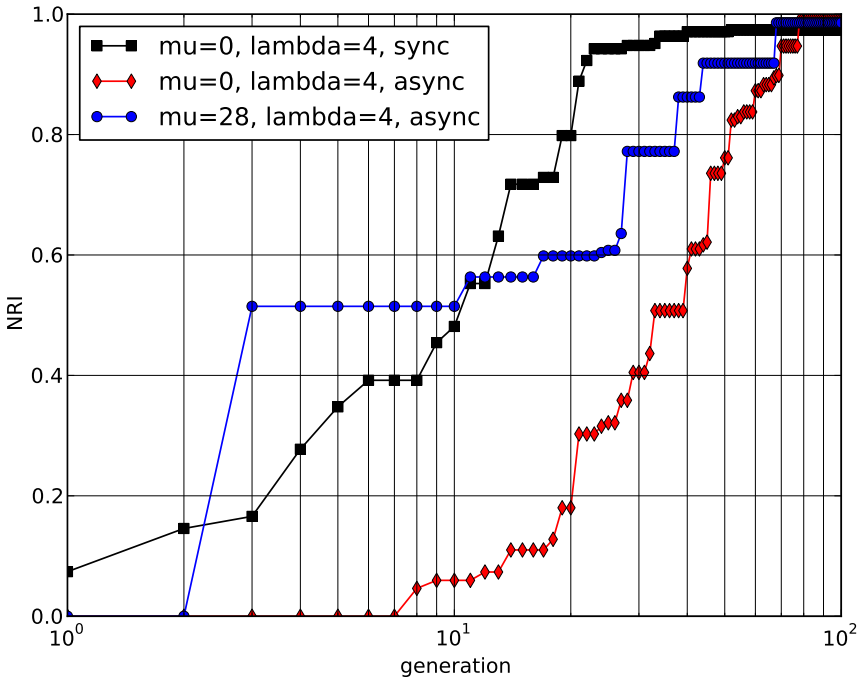


Figure 16: Solution of the "rank1approx9d" problem by using: (i) the synchronous algorithm with the $\mathrm{EI}^{0,4}$ criterion, (ii) and (iii) the asynchronous algorithms with the $\mathrm{EI}^{0,4}$ and $\mathrm{EI}^{28,4}$ criteria.

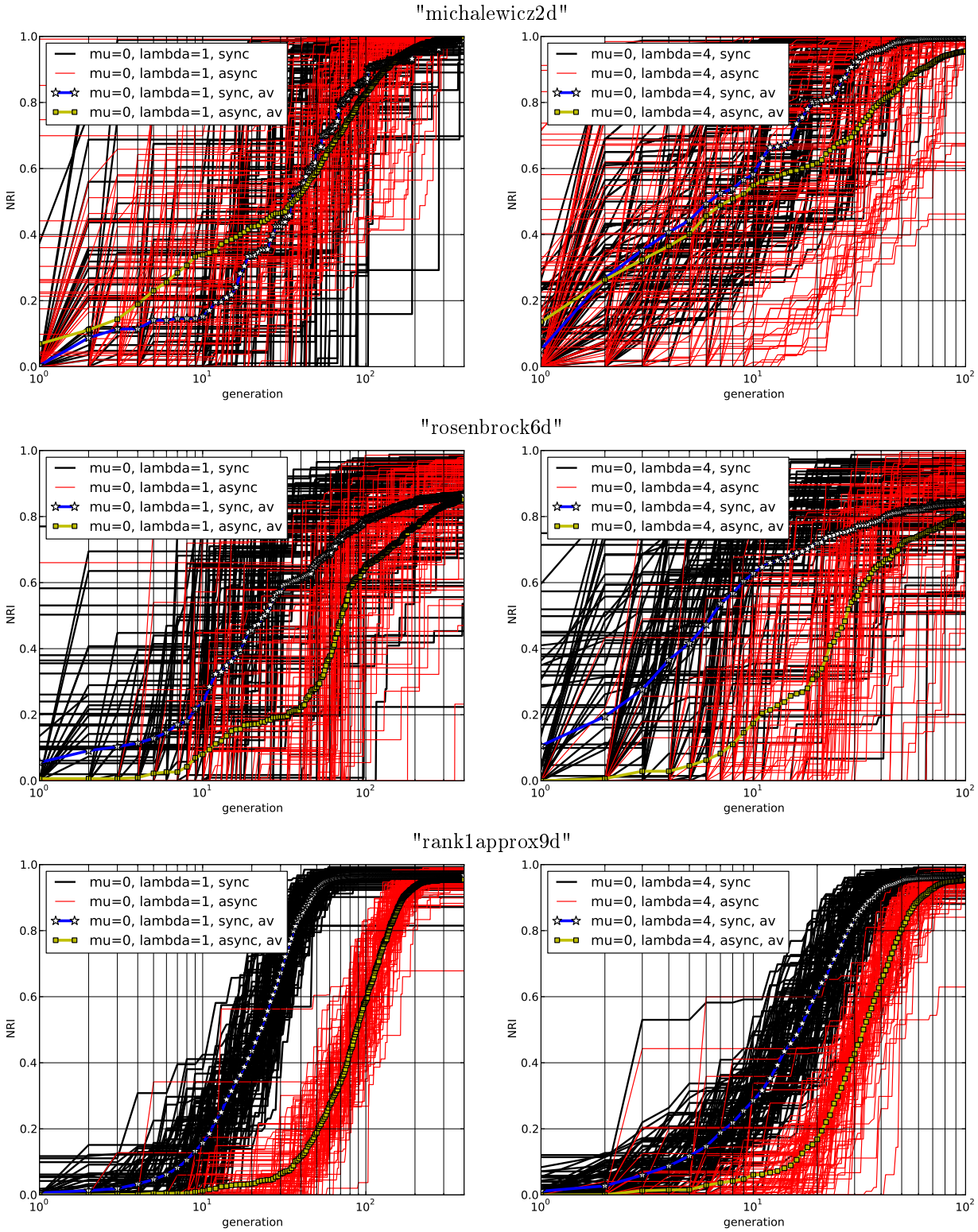


Figure 17: Synchronous vs. asynchronous optimization: Details.

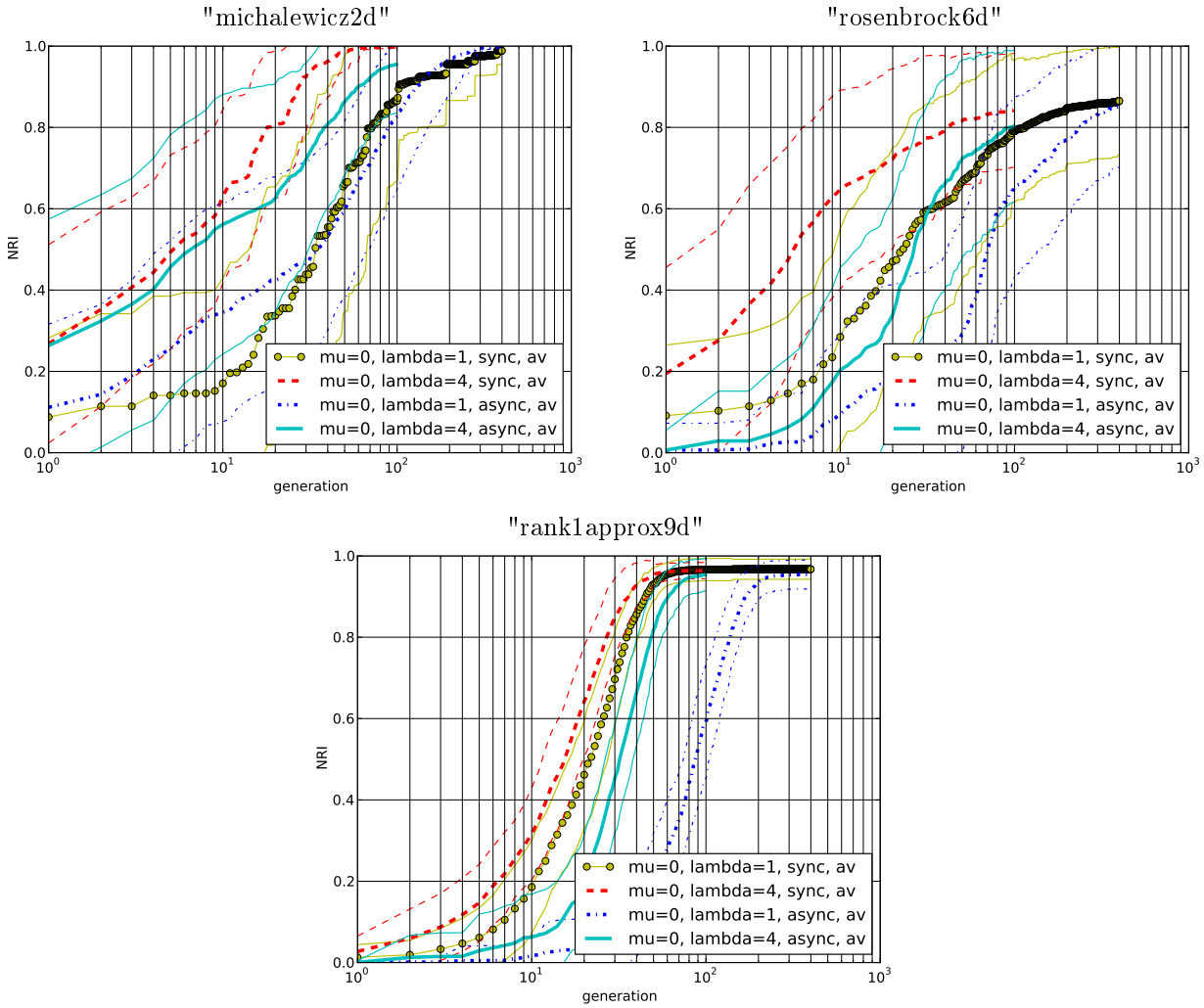


Figure 18: Synchronous vs. asynchronous optimization: Summary.

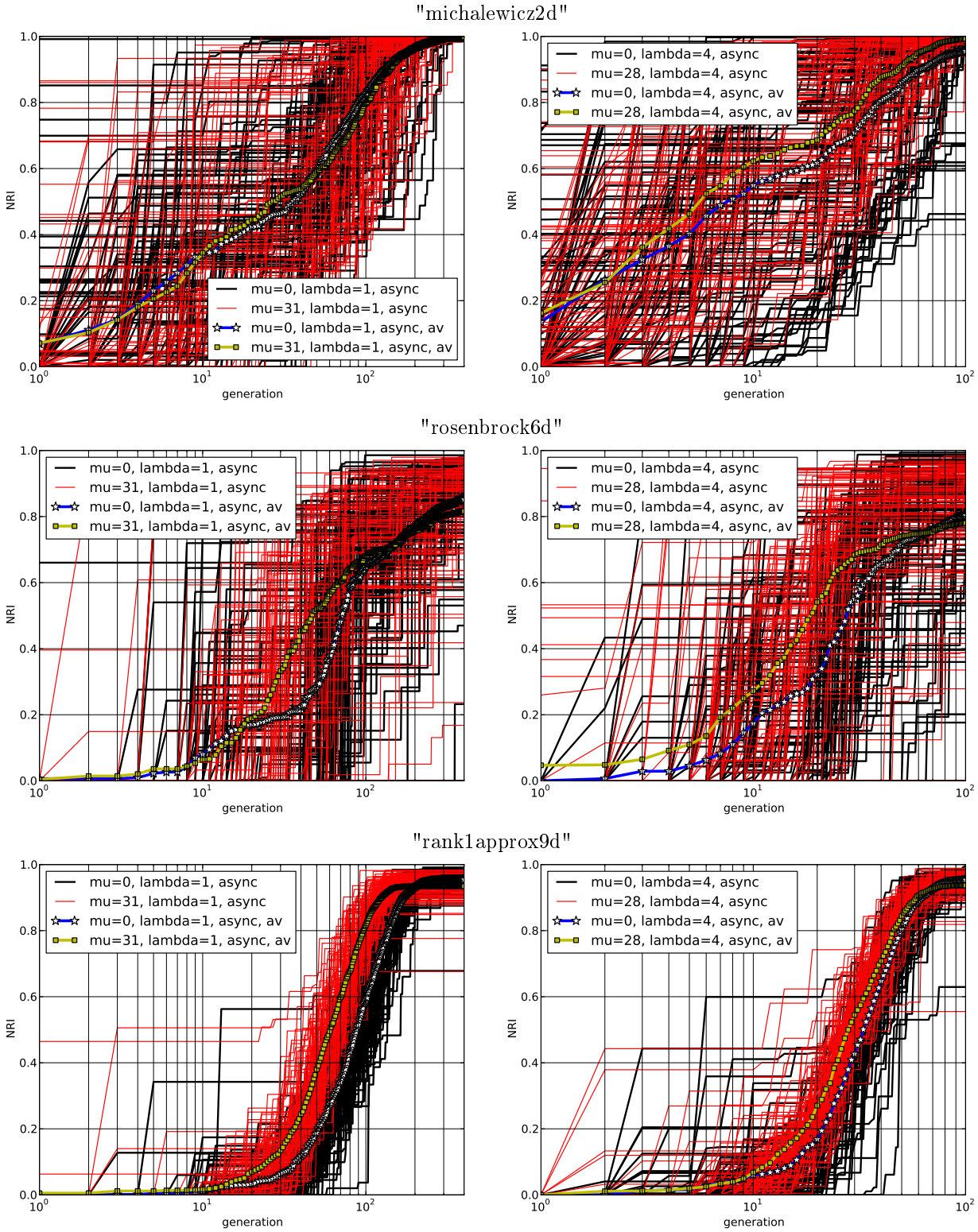


Figure 19: The effect of integrating out μ points in the optimization with the asynchronous node access: Details.

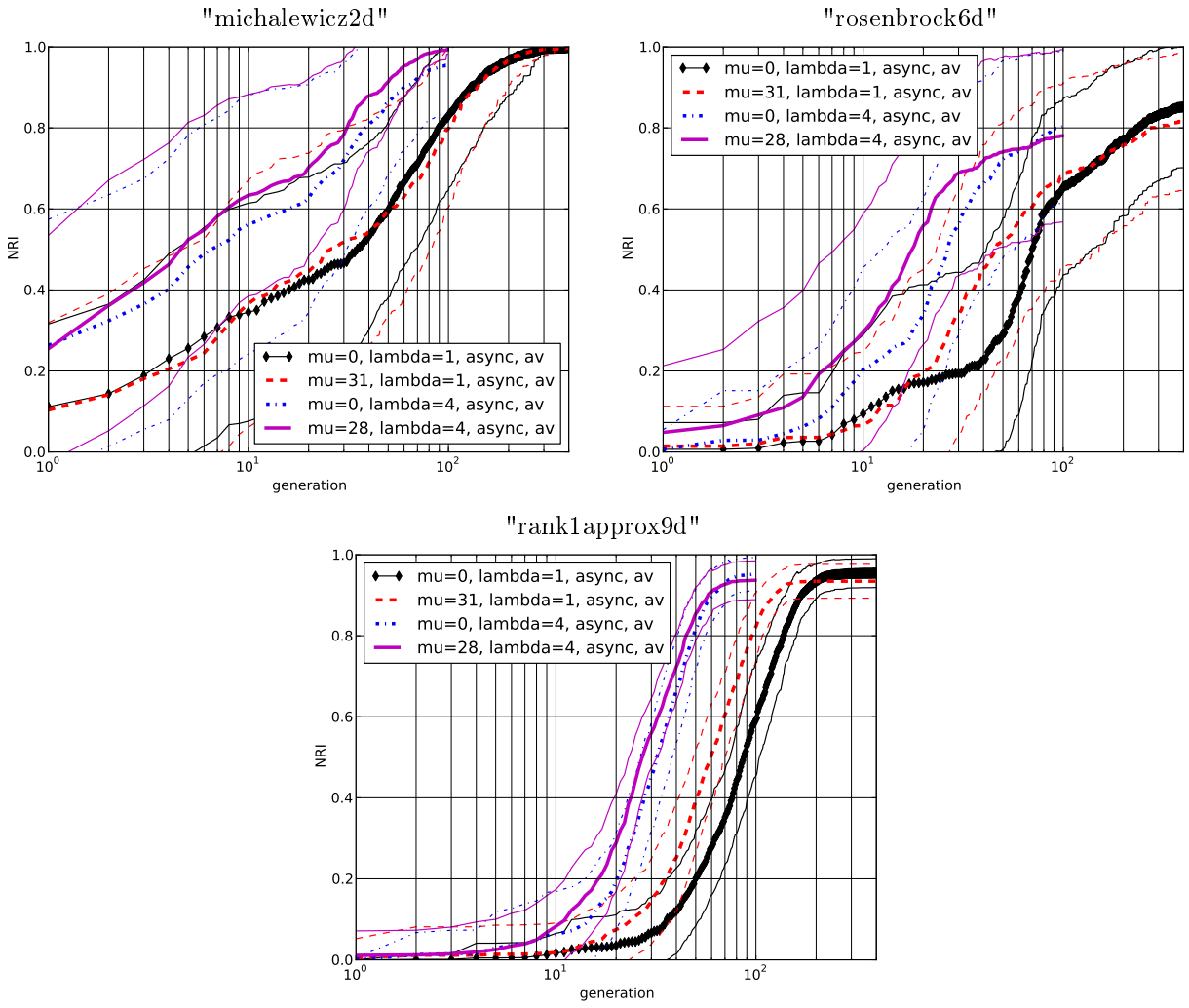


Figure 20: The effect of integrating out μ points in the optimization with the asynchronous node access: Averages.

5 Integral of the Expected Improvement at Multiple Points

5.1 Introduction

Our goal is to approximately evaluate (as fast as possible) the integral for the expected improvement in Eq. (7), which has also been studied previously in [18]:

$$\alpha(\mathbf{m}, \mathbf{C}) = \int_{\mathbb{R}^d} g(\mathbf{y}) p(\mathbf{y}) d\mathbf{y}, \tag{19}$$

$$g(\mathbf{y}) = \max\left(0, f_{\text{best}} - \min \mathbf{y}\right). \tag{20}$$

Here the shortcut $\mathbf{y}$ denotes the collection of d scalar values of the kriging responses which are distributed according to the normal probability density $p(\mathbf{y})$ with the mean $\mathbf{m}$ and covariance $\mathbf{C}$.

The criterion can be applied to sample the candidate points for the function to be minimized, the parameter $f_{\text{best}} \in \mathbb{R}$ is then the minimal presently known value of the function.

The expected improvement will be large where the mean of the kriging responses is small and the responses are uncorrelated. When kriging with the Gaussian covariance function, the uncorrelatedness implies distant events, and the criterion helps to sample multiple points with the small expected cost value and large inter-distances.

Formally, the problem demands an integration w.r.t. the normal density function, and one is tempted to apply fast fully-symmetric rules which exactly integrate the Gaussian moments of the lower order [16, 23, 41]. However, the function $g(\mathbf{y})$ is not a polynomial. Another key difficulty here is that the integration technique should not only approximate the true value of the integral, but it should also ensure that the maximization of the integral w.r.t. $\mathbf{m}$ and $\mathbf{C}$ gives preference to distinct locations of the kriging responses. We shall see that the upper bound of the integral can yield the relative error of the true value as low as 0.3%, but this is useless. The third difficulty is that, typically, the integral needs to be evaluated a million times and more, which excludes a variety of accurate and complex integration procedures.

There have been several attempts to integrate what could be considered as a "typical", or "average case" integrand, by means of the SVD [43], or by applying kriging on the integrand itself, see e.g. [26]. It is very clear, and it is also pointed out in [16], that these frameworks fail to provide practical criteria for the determination of the integration nodes. One could also note that when $d = 2$, the integrand depends on six parameters already.

Our report on integration is organized in the following way. Section 5.2 discusses the properties of the integrands and shows their sample extracted from an optimization process where $\mathbf{m}$ and $\mathbf{C}$ are spatial functions and the expected improvement is maximized w.r.t. the spatial locations. The reader is assumed to be familiar with how the expected improvement criterion is used to sample new locations during the optimization [18]. Section 5.3 introduces a new integration method, Section 5.4 presents the test of the method, while Section 6 concludes our work.

5.2 Integrand

We shall work with the standardized normal density by using the Cholesky decomposition $\mathbf{C} = \mathbf{L}\mathbf{L}^T$:

$$\alpha(\mathbf{m}, \mathbf{C}) = \int_{\mathbb{R}^d} g(\mathbf{m} + \mathbf{L}\mathbf{u}) p(\mathbf{u}) d\mathbf{u}. \tag{21}$$

Hereafter $p(\mathbf{u})$ is the standard d-variate normal probability density function.

The bounds for the expected improvement at multiple points can be expressed as the functions of the one-point improvements [20, 18]:

$$\max_{1 \le i \le d} \alpha(m_i, \sigma_{ii}^2) \le \alpha(\mathbf{m}, \mathbf{C}) \le \sum_{i}^{d} \alpha(m_i, \sigma_{ii}^2), \tag{22}$$

where

$$\alpha(m_i, \sigma_{ii}^2) = (f_{\text{best}} - m_i)\Phi\left(\frac{f_{\text{best}} - m_i}{\sigma_i}\right) + \sigma_i\,\phi\left(\frac{f_{\text{best}} - m_i}{\sigma_i}\right). \tag{23}$$

Here the Φ and ϕ are the distribution and density functions of the standard normal variable, resp. The upper bound is often very close to the true value, but it cannot be applied in the actual optimization when one needs to select the quantities $\mathbf{m}$ and $\mathbf{C}$ in order to determine the locations of the kriging responses which provide the maximal expected improvement. The upper bound simply selects the point with the maximal one-point improvement and replicates it $d-1$ times.

The integrand is not equal to zero only in the region defined as

$$S \equiv \{\mathbf{u} \in \mathbb{R}^d :\ f_{\text{best}} - \min(\mathbf{m} + \mathbf{L}\mathbf{u}) \geq \mathbf{0}\}. \tag{24}$$

The constant f_{best} can be subsummed by the minimum operator when introducing the vector $\mathbf{1}$ with the unity coordinates. The "less than zero" constraint imposed on the minimum over the elements implies that one of the elements is less than zero, which leads to

$$S = \{\mathbf{u} \in \mathbb{R}^d :\ \cup(-f_{\text{best}}\mathbf{1} + \mathbf{m} + \mathbf{L}\mathbf{u} \leq \mathbf{0})\}, \tag{25}$$

where the set union $\cup$ acts on the halfplanes presented as the inequalities (row-wise).

It is good to discuss what the integrands look like in the actual optimization problem. For this reason, consider the problem of approximating a matrix with another one whose rank is one:

$$(\mathbf{p}^*, \mathbf{q}^*) = \arg\min_{\mathbf{p},\mathbf{q}} \|\mathbf{A} - \mathbf{p}\mathbf{q}^T\|, \tag{26}$$

where $\mathbf{A} \in \mathbb{R}^{kl}$, the column-vectors $\mathbf{p} \in \mathbb{R}^k$, and $\mathbf{q} \in \mathbb{R}^l$, and $\|\cdot\|$ is the Frobenius norm.

In particular, a 4×5 matrix $\mathbf{A}$ of uniformly distributed elements in $[0, 1]$ is first generated, and then we seek its 1-rank approximation by determining the vectors $\mathbf{p}$ and $\mathbf{q}$ whose elements are further constrained to be in $[-1, 1]$. This is a continuous nonconvex 9-dimensional box-constrained optimization problem whose solution is given by the SVD transform.

We have applied the expected improvement criterion to generate d new points during each of the 50 iterations of the basic algorithm discussed in [18]. The initial DOE has 100 points and the values of the Frobenius norm lie in $[7.22, 18.26]$. The SVD produces the optimal 1-rank approximation whose Frobenius norm is 0.94. The expected improvement is maximized by using the CMA-ES method [29] with the 10-point population, 500 iterations, and 0.1 initial coordinate standard deviation.

A sample of the integrands extracted from the actual optimization process is shown in Fig. 21. Therein the dashed lines indicate the line equations (hyperplanes) of Eq. (25), and the dash-dotted line is the symmetry axis, which is the set of points where all of the coordinates of the vector $-f_{\text{best}}\mathbf{1} + \mathbf{m} + \mathbf{L}\mathbf{u}$ are equal. The lines intersect at the point where all of the coordinates are equal to zero.

The circles of the radius $\sqrt{2}$ are shown in Fig. 21 in order to emphasize the regions where the normal density concentrates its mass. More generally, it is well known that the region of the maximal concentration is defined by the annulus (shell), whose inner and outer radius is

$$r = \sqrt{d} \pm \frac{1}{\sqrt{2}}. \tag{27}$$

Fig. 21 indicates that the regions where the integrand is substantial depend on the location of the mean values of the kriging responses w.r.t. the f_{best} value. In the bivariate case, the region of substance can only be: either (i) the vicinity of the two edges of a semiinfinite nonconvex polygon, or (ii) the interior of a cropped annulus.

One should also emphasize that the most striking feature of the kriging responses is that they are very weakly correlated. At the random initial points where the expected improvement is small, the responces are virtually uncorrelated, and at the locations of the maximal improvement only roughly $O(ln(d))$ elements of

the row (column) of the covariance matrix $\mathbf{C}$ attain 10% of the value of a diagonal element. The remaining elements are typically either zero, or less than 0.1%. As a consequence, the Cholesky matrix $\mathbf{L}$ is close to the identity matrix $\mathbf{I}$, and the peaks of the integrand appear at the locations whose all but one coordinate are zero. The nonzero coordinate equals to the location of the maximum of the integrand of the one-point expected improvement.

5.3 New Methods for Adaptive Integration

Our main idea is to work with the exact symmetric integration rules [25], but to replace the monomial integrands with d "slices" of the improvement which can be integrated exactly. This leads to a linear system of equations for the integration weights, which depends on the parameters $\mathbf{m}$ and $\mathbf{L}$. The system can be solved quickly for each integrand, and the integration rule becomes adaptive.

Let us expand the subintegrand $h(\mathbf{u}) \equiv g(\mathbf{m} + \mathbf{L}\mathbf{u})$ in Eq. (21):

$$\begin{aligned} h(\mathbf{u}) = \max(0, f_{\text{best}} - \min\big(& m_1 + l_{11}u_1, \\ & m_2 + l_{21}u_1 + l_{22}u_2, \\ & \dots, \\ & m_d + l_{d1}u_1 + l_{d2}u_2 + \dots + l_{dd}u_d\big). \end{aligned} \tag{28}$$

The simplest possible way to get d equations in d unknowns is to apply the following fully-symmetric rule:

$$\int_{\mathbb{R}^d} h(\mathbf{u})p(\mathbf{u})d\mathbf{u} \simeq \sum_{i=1}^{d} w_i\big(h(\mathbf{P}^i\mathbf{v}) + h(-\mathbf{P}^i\mathbf{v})\big), \tag{29}$$

where $\mathbf{P}$ is the circular shift matrix whose only nonzero elements are $p_{i,i+1} = 1$, for $i = 1, \dots d-1$, and $p_{d,1} = 1$. The column-vector $\mathbf{v} = [v, 0, \dots, 0] \in \mathbb{R}^d$, where v is a free parameter.

In order to determine the weights w_i, we demand that each "projection" of $h(\mathbf{u})$ is integrated exactly:

$$\int_{\mathbb{R}^d} \max(0, f_{\text{best}} - m_i - \tilde{\mathbf{l}}_i\mathbf{u})p(\mathbf{u})d\mathbf{u} = m'_i\Phi\left(\frac{m'_i}{\sigma'_i}\right) + \sigma'_i\,\phi\left(\frac{m'_i}{\sigma'_i}\right), \tag{30}$$

where $\tilde{\mathbf{l}}_i$ is the ith row of the matrix $\mathbf{L}$, $m'_i = f_{\text{best}} - m_i$, and $\sigma'_i = \|\tilde{\mathbf{l}}_i\|$.

This results in a linear system of equations $\mathbf{S}\mathbf{w} = \mathbf{s}$ for the unknown vector of the integration weights $\mathbf{w} \in \mathbf{R}^d$. The coordinates of the vector $\mathbf{s}$ are the values of the rhs of Eq. (30). The system matrix $\mathbf{S}$ is not symmetric, and has the elements given by

$$s_{ij} = \max(0, m'_i - vl_{ij}) + \max(0, m'_i + vl_{ij}). \tag{31}$$

In order to guarantee that it is not singular, one can choose the parameter v so that the lower bound for the smallest singular value is greater than zero. In particular, the bound derived in [21] can be applied:

$$\sigma_{\text{smallest}}(\mathbf{S}) \geq \min_{1 \leq i \leq d}\left(|s_{ii}| - \frac{1}{2}\sum_{\substack{j=1 \\ j \neq i}}^{d}|s_{ij}| - \frac{1}{2}\sum_{\substack{j=1 \\ j \neq i}}^{d}|s_{ji}|\right). \tag{32}$$

If the observed values of the function to be optimized are scaled to be of the zero mean and of the unity variance, then, typically, $m'_i = \mathcal{O}(1)$. If we further assume that $m'_i \gg vl_{ij}$ for $i \neq j$, then both of the terms with the off-diagonal elements in Eq. (32) sum to $\mathcal{O}(d)$. This leads to the choice of v dictated by $l_{ii}v \geq O(d)$. In our problem, the value $v = 2d$ is large enough to guarantee the nonsingularity, but we simply set it to 10^4. This effectively erases the information about the mean of the kriging responses in the matrix $\mathbf{S}$, and makes the latter diagonally-dominant if $\mathbf{L}$ is diagonally dominant, which is typically the case. The actual value of v does not affect the accuracy of the integration, provided it is large enough to avoid the singularity of $\mathbf{S}$.

To summarize, the approximation of the integral can be written by emphasizing the difference with the upper bound, clf. Eq. (22):

$$\alpha'(\mathbf{m}, \mathbf{C}) = \sum_{i=1}^{d} b_i c_i \alpha(m'_i, \sigma'_i), \tag{33}$$

$$b_i = g(\mathbf{m} + \mathbf{l}_i) + g(\mathbf{m} - \mathbf{l}_i), \tag{34}$$

$$c_i = \sum_{j=1}^{d} r_{ji}. \tag{35}$$

Here the vector $\mathbf{l}_i$ is the ith column of the matrix $\mathbf{L}$ and r_{ji} are the elements of the matrix $\mathbf{R} \equiv \mathbf{S}^{-1}$.

When $vl_{ii} \gg d$, the diagonal elements of $\mathbf{S}$ become considerably larger than the remaining matrix entries. If we give the latter ones a certain common weight z, then the 0th order Taylor series expansion w.r.t. the parameter z leads to $c_{ii} \approx 1/(l_{ii}v)$. If we further assume that $\mathbf{L} \approx \mathbf{I}$, then $b_i \approx v$, and $\alpha(m'_i, \sigma'_i) \approx \alpha(m, \sigma)$, which leads to the expression for the upper bound, clf. Eq. (22). However, generally, neither $b_i c_i = 1$, nor $\alpha(m'_i, \sigma'_i) = \alpha(m, \sigma)$. At this point, we still do not know whether the approximation will be suitable for the optimization purposes, but it contains enough "anisotropy" w.r.t. the index i to warrant a practical test.

The complexity of the method is bound by the need to solve a linear system of d equations every time a new integrand is presented, which demands $\mathcal{O}(d^3)$ multiplications. The basic Monte Carlo sampling, on the other hand, demands $\mathcal{O}(d^2 n)$ multiplications, where n is the number of samples generated according to the standard normal distribution. Typically, $n \gg d$.

There is one way to modify the method by introducing additional d weights into Eq. (29):

$$\int_{\mathbb{R}^d} h(\mathbf{u}) p(\mathbf{u}) d\mathbf{u} \simeq \sum_{i=1}^{d} w_i \big(h(\mathbf{P}^i \mathbf{v}) + \sum_{i=d+1}^{2d} h(-\mathbf{P}^{d+i} \mathbf{v}) \big), \tag{36}$$

and here one should notice that $\mathbf{P}^{d+i} = \mathbf{P}^i$. The weights can be determined by using d equations for the "mean slices" in Eq. (30), and additional "variance slices" which can also be integrated exactly, clf. Eq. (63):

$$\int_{\mathbb{R}^d} \big(\max(0, f_{\text{best}} - m_i - \tilde{\mathbf{l}}_i \mathbf{u}) - s_i \big)^2 p(\mathbf{u}) d\mathbf{u} = \sigma_i'^2 \Phi^2 \left(\frac{m'_i}{\sigma'_i} \right) + \sigma_i'^2 \phi^2 \left(\frac{m'_i}{\sigma'_i} \right) - m'_i \sigma_i \Phi \left(\frac{m'_i}{\sigma'_i} \right) \phi \left(\frac{m'_i}{\sigma'_i} \right). \tag{37}$$

Here s_i denotes the value of the integral in Eq. (30).

In practice, this modification yields a singular $\mathbf{S}$ matrix, but the use of its pseudoinverse in the determination of the weights $\mathbf{w} \in \mathbb{R}^{2d}$ turns out to be very accurate in some cases. The choice of the value of v, however, greatly affects the integration accuracy. We will report the experiments with the choice $v = \sqrt{d}$ which places the integration nodes at the points of the maximal concentration of the standard normal density. It turns out that this is the best choice when $d = 16$. In lower dimensions, this strategy is very suboptimal, which greatly limits the use of the method ENSEMI2. The details are presented in the next section.

5.4 Results

The test is split into two sets of the integrands extracted from the actual optimization. The set I contains 50 d-variate integrands extracted at the random initial locations generated before the maximization of the expected improvement, and the set II contains the integrands at the optimal locations. The exact values of the expected improvement are not known, but we have applied the Monte Carlo (MC) sampling with 10^7 points to obtain the estimates of the true values. In the actual optimization, this is not possible.

For testing purposes, we have implemented the method developed in [24]. It will simply be referred to as "LDM". This technique exactly integrates all the 5th degree monomials w.r.t. the normal density. Instead of using the orbits of a fully symmetric group, the method maps the vertices and the midpoints of the standard regular simplex onto a sphere. It also incorporates the scaling w.r.t. an increasing dimension,

Table 10: Median Relative Integration Errors, %

	Problem Set I			Problem Set II		
	$d=2$	$d=8$	$d=16$	$d=2$	$d=8$	$d=16$
MC, $n=10^3$	21 ± 6	14 ± 3	10 ± 2	4 ± 1	3 ± 1	3 ± 1
ENSEMI2	100	4.5	11	6.9	10	5.1
ENSEMI1	0.32	1.0	2.7	3.0	6.0	10
Upper Bound [18]	0.37	0.93	1.8	3.0	14	18
Improved LDM	18	34	60	3.6	9.1	45
LDM [24]	100	170	370	10	9.8	68
Lower Bound [18]	34	68	79	25	64	73

which approximately follows the law of the concentration of the normal measure: One integration node is placed at the origin, and all of the remaining nodes lie on the sphere of the radius $\sqrt{d+2}$).

A direct improvement to the LDM method, and practically to any technique in this family, e.g. the unscented transformations [41], is to split the approximate integration into two parts. One first applies the LDM rule to calculate the mean and variance of the minimum over the kriging responses $V = \min(Y)$. The expected improvement can then be obtained by "propagating" V through the deterministic function $\max(0, f_{\text{best}} - V)$ under the assumption of the normality of V, clf. Eq. (49) of Appendix B.

Table 10 indicates the relative median errors obtained by various methods. The lower (upper) bound is abbreviated as LB (UB). The new methods are referred to as "ENSEMI1" and "ENSEMI2", and the recursive acronyms stand for "ENSEMIx is Not Symmetric Exact Monomial Integration".

The results confirm that the evaluation of the expected improvements depends on whether their values are close to maximal, or not. The upper bound is very close to the true value. The standard MC sampling works very well on the problem set II, and the error does not depend on the dimension of the integration space. Considering that the problem set II is more important than the problem set I, an economical MC sampling remains a very tough method to improve. Our faster alternative is worse, but competitive.

The LDM rule is significantly improved, but the resulting method is not accurate enough. Notably, the accuracy of the LDM method and its improvement considerably deteriorates as the dimension d increases. Here one could also suggest the use of the exact known values of the mean and variance over the minimum of two normal variables, followed by the application of Eq. (49). This leads to the median relative error 1.5% on the problem set II, but, surprisingly, it also produces the error of 33% on the problem set I, which clearly indicates that the normal approximation to the minimum over the normal variables is inadequate. Also, this approach does not extend to $d > 2$ as the exact moments of the minimum over the normal variables are not known.

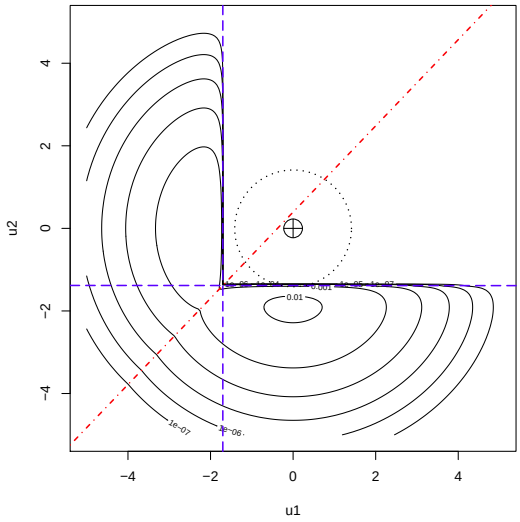
Interestingly, as d increases, the difference between the upper bound and the true expected improvement increases too and it may reach the relative median error of 18% when $d = 16$ (problem set II). Thus, the power of the multi-point improvements is likely to be revealed when d is large. However, a large d introduces severe difficulties in the black-box optimizer, and the true expected improvement may become an overestimated value of the real one.

In order to assess how the integration accuracy affects the optimization performance, we have applied the algorithms to the problem "rank1approx9d" with the $\text{EI}^{0,4}$ criterion in the synchronous node access. The average NRI paths (thicker lines) along with their corresponding deviations (thinner lines) are displayed in Fig. 22. One can see that the ENSEMIx methods outperform the standard MC method with 10^3 samples, and yield similar optimization results when the number of samples is very large ($\text{ns} = 10^4, 10^5$). Notably, the complexity of the ENSEMIx methods is smaller or equal to that of the MC with $\text{ns} = 10^3$ samples. Thus, one obtains memory savings, as well as shorter computational times. The ENSEMIx algorithms run about 1.5 times faster than the MC method with $\text{ns} = 10^3$ samples, and they are roughly eight times faster than the MC method with $\text{ns} = 10^5$ samples. Notably, a further increase of the number of MC samples

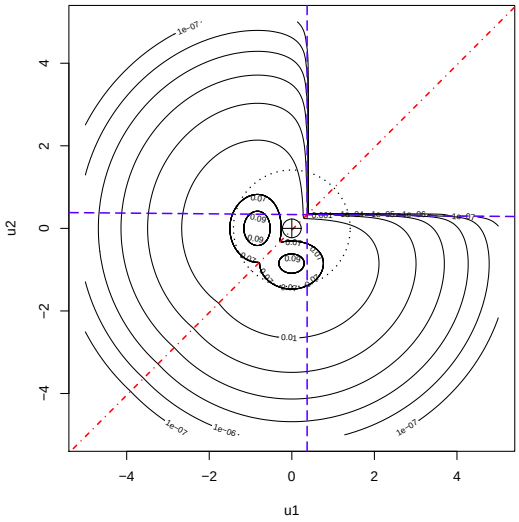
from 10^4 to 10^5 does not improve the overall optimization performance, which indicates that the integration accuracy has a limited impact which is also shadowed by other inadequacies of the model, such as a possibly insufficient number of the CMA-ES iterations set to maximize the multi-point EI criterion.

The algorithms have been implemented in Scilab 5.3.3, and the source code is provided in Appendix C.

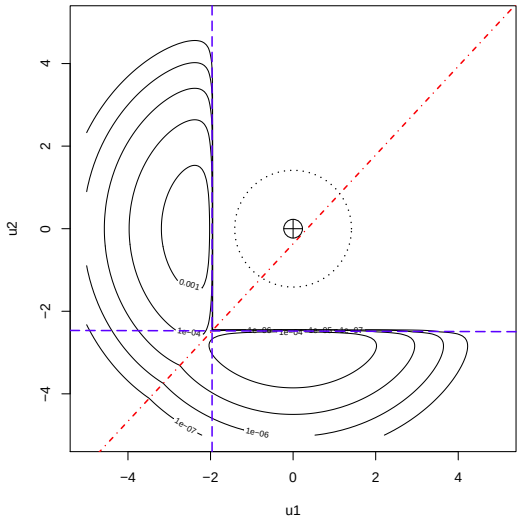
(iter: 1, loc: random)



(iter: 1, loc: optimal)



(iter: 50, loc: random)



(iter: 50, loc: optimal)

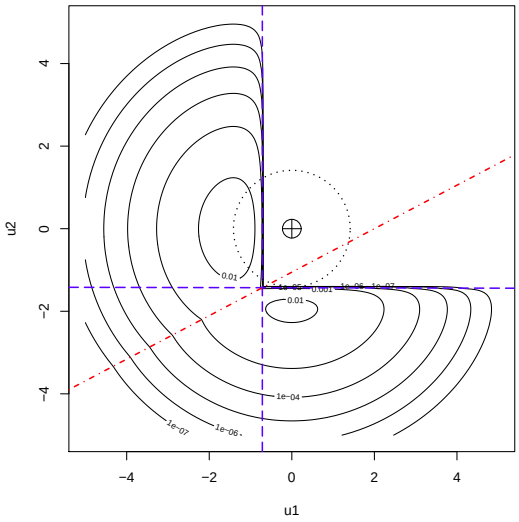


Figure 21: Examples of the bivariate integrand $h(\mathbf{u})p(\mathbf{u})$ which occur during the optimization.

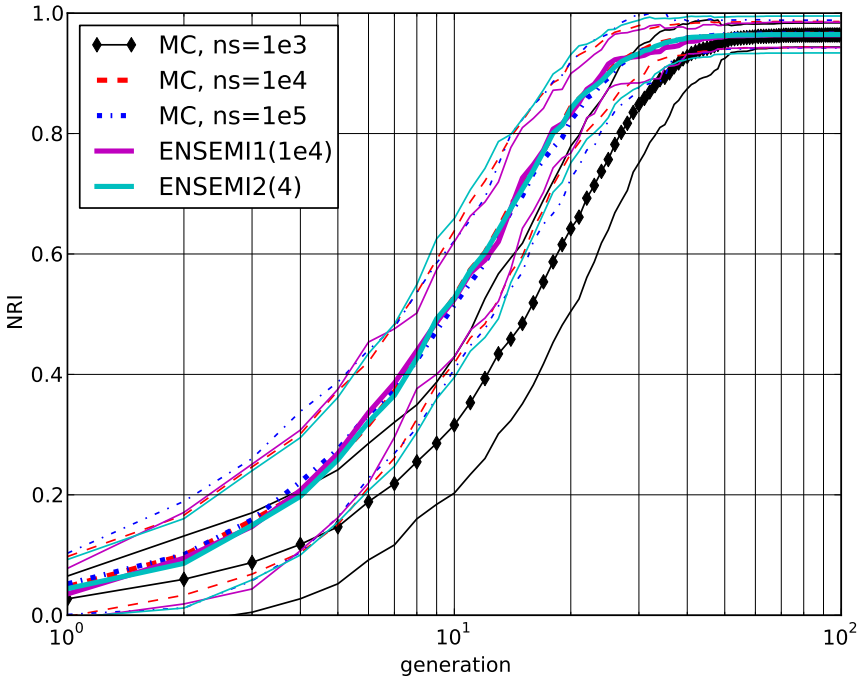


Figure 22: Solution of the "rank1approx9d" problem by using the synchronous algorithm with the $\mathrm{EI}^{0,4}$ criterion and various integration methods applied to estimate the multi-point EI criterion. The ENSEMIx rules outperform the MC method with $\mathrm{ns} = 10^3$, and yield similar results when $\mathrm{ns} = 10^4, 10^5$.

6 Conclusions

We have compared several numerical integration schemes for calculating the multi-point EI criterion. Our experiments indicate that symmetric monomial integration rules are less accurate than a traditional Monte Carlo sampling. The methods can be improved, but they are much less applicable than stated in the present literature on the integration w.r.t. the normal density. The accuracy of the economical MC sampling is consistently better among the methods tried on the problem set II, but simple symmetric integration rules (ENSEMI1 and 2) become competitive if they are adapted to the problem. When $d = 2$, the relative difference between the upper bound and the true value of the expected improvement can be as low as 0.37%. The proposed method (ENSEMI1) can reduce this value further down to 0.32%. The introduced methods demand $\mathcal{O}(d^3)$ scalar multiplications for each integration, while the Monte Carlo estimation consumes $\mathcal{O}(d^2 n)$ scalar multiplications, given the number of samples n. Typically, $d \ll n$, and the loss of the accuracies by the proposed ENSEMI methods is acceptable in our test set (when $d \leq 16$). Moreover, the developed ENSEMI methods have been shown to yield better global optimization results than the MC integration with 10^3 samples.

A Estimation of Wall Clock Time

Listing 1: Scilab 5.3.3 code which estimates the evarage node update time.

```
function [wct] = estimatewct(lamb, buffsz)
tmin = 10;
tmax = 30;
tb = 2;
ngenerations = 250;
grand("setsd", sum((getdate())^2));
tbuff0 = grand(1, buffsz, "unf", tmin, tmax);
tbuff = tbuff0;
tcvec = [];
for i=1:ngenerations
  [vals, inds] = gsort(tbuff, "g", "i");
  tc = max(vals(1:lamb));
  tbuff = tbuff - tc - tb;
  tbuff(inds(1:lamb)) = tbuff0(inds(1:lamb));
  tbuff(find(tbuff<=0)) = 0;
  tcvec($+1) = tc;
end
wct = mean(tcvec)+tb;
endfunction
```

B Moments of the Censored Normal Variable

The lower-order moments of the censored normal variable should not be confused with those that correspond to the normal density on a positive axis. Let

$$Z = \max(0, Y), \tag{38}$$

$$Y \sim N(m_y, \sigma_y^2). \tag{39}$$

As the density is discontinuous at 0, the expectation is the contribution of two discrete events:

$$m_z = P(Z = 0)E(Z|Z = 0) + P(Z > 0)E(Z|Z > 0) \tag{40}$$

$$= P(Y \leq 0) \cdot 0 + P(Y > 0)E(Y|Y > 0) \tag{41}$$

$$= \int_0^\infty y \frac{1}{\sqrt{2\pi\sigma_y^2}} \exp\left(-\frac{(y - m_y)^2}{2\sigma_y^2}\right) dy. \tag{42}$$

The substitution $u_1 \equiv \frac{y-m}{\sigma}$ allows to introduce the standard univariate normal density $\phi(x)$, and its distribution $\Phi(x)$:

$$m_z = \int_{-\frac{m_y}{\sigma_y}}^{\infty} (m_y + \sigma_y u_1)\phi(u_1)du_1, \tag{43}$$

$$= m_y\,\Phi\left(\frac{m_y}{\sigma_y}\right) + \sigma_y \int_{-\frac{m_y}{\sigma_y}}^{\infty} u_1\phi(u_1)du_1. \tag{44}$$

It is easy to integrate the remaining term:

$$\int_{-\frac{m_y}{\sigma_y}}^{\infty} u_1\phi(u_1)du_1 = \int_{-\frac{m_y}{\sigma_y}}^{\infty} u_1 \frac{1}{\sqrt{2\pi}} \exp\left(-\frac{u_1^2}{2}\right) du_1 \tag{45}$$

$$= \frac{1}{\sqrt{2\pi}} \int_{\frac{m_y^2}{2\sigma_y^2}}^{\infty} e^{-u_2} du_2 \tag{46}$$

$$= \frac{1}{\sqrt{2\pi}} \exp\left(-\frac{m_y^2}{2\sigma_y^2}\right) \tag{47}$$

$$= \phi\left(\frac{m_y}{\sigma_y}\right). \tag{48}$$

Therefore,

$$m_z = m_y\,\Phi\left(\frac{m_y}{\sigma_y}\right) + \sigma_y\,\phi\left(\frac{m_y}{\sigma_y}\right). \tag{49}$$

The variance estimation is more tricky:

$$\sigma_z^2 = P(Z=0)E\big((Z-m_z)^2|Z=0\big) + P(Z>0)E\big((Z-m_z)^2|Z>0\big) \tag{50}$$

$$= P(Y\le 0)m_z^2 + P(Y>0)E\big((Y-m_z)^2|Y>0\big) \tag{51}$$

$$= P(Y\le 0)\cdot m_z^2 + P(Y>0)\Big(E(Y^2|Y>0) - 2m_zE(Y|Y>0) + m_z^2\Big) \tag{52}$$

$$= P(Y\le 0)\cdot m_z^2 + P(Y>0)E(Y^2|Y>0) - 2m_zP(Y>0)E(Y|Y>0) + P(Y>0)m_z^2 \tag{53}$$

$$= P(Y>0)E(Y^2|Y>0) - m_z^2 \tag{54}$$

$$= \Phi\left(\frac{m_y}{\sigma_y}\right) \int_0^{\infty} y^2 \frac{1}{\sqrt{2\pi\sigma_y^2}} \exp\left(-\frac{(y-m_y)^2}{2\sigma_y^2}\right) dy - m_z^2. \tag{55}$$

Again, one first reduces the location and scale parameters to their standard values:

$$\int_0^{\infty} y^2 \frac{1}{\sqrt{2\pi\sigma_y^2}} \exp\left(-\frac{(y-m_y)^2}{2\sigma_y^2}\right) dy = \int_{-\frac{\mu}{\sigma}}^{\infty} (m_y + \sigma_y u_1)^2 \frac{1}{\sqrt{2\pi}} \exp\left(-\frac{u_1^2}{2}\right) du_1. \tag{56}$$

The expansion of the quadtratic term produces three integrals. The first one is trivial:

$$\int_{-\frac{m_y}{\sigma_y}}^{\infty} m_y^2 \frac{1}{\sqrt{2\pi}} \exp\left(-\frac{u_1^2}{2}\right) du_1 = m_y^2\,\Phi\left(\frac{m_y}{\sigma_y}\right). \tag{57}$$

The second one has already been evaluated in Eq. (48):

$$2m_y\sigma_y \int_{-\frac{m_y}{\sigma_y}}^{\infty} u_1 \frac{1}{\sqrt{2\pi}} \exp\left(-\frac{u_1^2}{2}\right) du_1 = 2m_y\sigma_y\phi\left(\frac{m_y}{\sigma_y}\right). \tag{58}$$

The remaining integral demands the application of a well-known trick:

$$\sigma_y^2 \int_{-\frac{m_y}{\sigma_y}}^{\infty} u_1^2 \frac{1}{\sqrt{2\pi}} \exp\left(-\frac{u_1^2}{2}\right) du_1 = \sigma_y^2(-2)\frac{d}{d\beta}\int_{-\frac{m_y}{\sigma_y}}^{\infty} \frac{1}{\sqrt{2\pi}} \exp\left(-\beta\frac{u_1^2}{2}\right) du_1\Big|_{\beta=1} \tag{59}$$

$$= \sigma_y^2(-2)\frac{d}{d\beta}\int_{-\sqrt{\beta}\frac{m_y}{\sigma_y}}^{\infty} \frac{1}{\sqrt{2\pi}} \exp\left(-\frac{u_2^2}{2}\right) \frac{du_2}{\sqrt{\beta}}\Big|_{\beta=1} \tag{60}$$

$$= \sigma_y^2(-2)\frac{d}{d\beta}\left(\frac{1}{\sqrt{\beta}}\Phi\left(\sqrt{\beta}\frac{m_y}{\sigma_y}\right)\right)\Big|_{\beta=1} \tag{61}$$

$$= \sigma_y^2\Phi\left(\frac{m_y}{\sigma_y}\right) - m_y\sigma_y\phi\left(\frac{m_y}{\sigma_y}\right). \tag{62}$$

The use of Eqs. (55)–(58) and (62) produces the result:

$$\sigma_z^2 = \sigma_y^2\Phi^2\left(\frac{m_y}{\sigma_y}\right) + \sigma_y^2\phi^2\left(\frac{m_y}{\sigma_y}\right) - m_y\sigma_y\Phi\left(\frac{m_y}{\sigma_y}\right)\phi\left(\frac{m_y}{\sigma_y}\right). \tag{63}$$

C Integration Methods

Listing 2: The ENSEMI1 integration method.

```
function [eimpr] = ensemi1(cmean, L, ybest, v)
Sp = ybest - (kron(ones(1,lamb),cmean)+v*L);
Sm = ybest - (kron(ones(1,lamb),cmean)-v*L);
Z = zeros(lamb,lamb);
S = max(Sp,Z)+max(Sm,Z);
sii = sqrt(sum(L.*L, "c"));
inval = (ybest - cmean)./sii;
cdfu = cdfnor("PQ", inval, zeros(lamb,1), ones(lamb,1));
pdfu = 1.0/sqrt(2*%pi)*exp(-(inval.^2)/2);
cvec = sii.*(inval.*cdfu + pdfu);
w = S\cvec;
w = [w; w];
xvecs = kron(ones(1,2*lamb),cmean)+v*[L, -L];
fws = max([ybest-min(xvecs,   );zeros(1,2*lamb)],    );
eimpr = fws*w;
endfunction
```

Listing 3: The ENSEMI2 integration method.

```
function [eimpr] = ensemi2(cmean, L, ybest, v)
Sp = ybest - (kron(ones(1,lamb),cmean)+v*L);
Sm = ybest - (kron(ones(1,lamb),cmean)-v*L);
Z = zeros(lamb,lamb);
S1 = [max(Sp,Z), max(Sm,Z)];//lambx(2lamb)
sii = sqrt(sum(L.*L, "c"));
inval = (ybest - cmean)./sii;
cdfu = cdfnor("PQ", inval, zeros(lamb,1), ones(lamb,1));
pdfu = 1.0/sqrt(2*%pi)*exp(-(inval.^2)/2);
cvec1 = sii.*(inval.*cdfu + pdfu);
cvec2 = sii.*(sii.*(cdfu.^2+pdfu.^2)-(ybest-cmean).*cdfu.*pdfu);
S2 = (S1-kron(ones(1,2*lamb),cvec1)).^2;
w = [S1;S2]\[cvec1;cvec2];//falls back to least squares if S is sing.
xvecs = kron(ones(1,2*lamb),cmean)+v*[L, -L];
fws = max([ybest-min(xvecs,   );zeros(1,2*lamb)],    );
eimpr = fws*w;
endfunction
```

References

[1] CATIA. http://www.3ds.com/products/catia/, 2012. 8

[2] OpenFOAM: Open Field Operation and Manipulation, v. 2.0.1. http://www.openfoam.org/, 2012. 8

[3] ProActive PACA Grid. http://proactive.inria.fr/, 2012. 7, 21, 26

[4] STAR–CCM+. http://www.cd-adapco.com/products/star_ccm_plus/index.html, 2012. 8

[5] P. Auer. Using Confidence Bounds for Exploitation–Exploration Trade-offs. *J. of Machine Learning Research*, 3:397–422, 2002. 3

[6] J. Azimi, A. Fern, and X. Fern. Batch Bayesian optimization via simulation matching. NIPS, 2010. 3, 4

[7] J. Azimi, A. Fern, and X. Fern. Active function optimization with parallel stochastic-duration experiments. ICML, 2011. 5, 6

[8] J. Azimi, A. Fern, and X. Fern. Budgeted Optimization with Concurrent Stochastic-Duration Experiments. NIPS, 2011. 4

[9] J. Azimi, A. Jalali, and X. Fern. Dynamic Batch Bayesian Optimization. NIPS, 2011. 5, 6

[10] J. Azimi, A. Jalali, and X. Fern. Hybrid Batch Bayesian Optimization. ICML, 2012. 4, 5

[11] E. Brochu, V. Cora, and N. de Freitas. A tutorial on Bayesian optimization of expensive cost functions, with application to active user modeling and hierarchical reinforcement learning. arXiv:1012.2599v1, 2010. 3

[12] A. Bull. Convergence Rates of Efficient Global Optimization Algorithms. *J. of Machine Learning Research*, 12:2879–2904, 2011. 3

[13] S. Clark and P. Frazier. Parallel Global Optimization Using An Improved Multi-points Expected Improvement Criterion. Presentation slides, INFORMS Optimization Society Conference, Miami, 2012. 6, 29

[14] D. Cox and S. John. A statistical method for global optimization. In *IEEE Conf. Sys. Man and Cyber.*, volume 2, pages 1241–1246, 1996. 3

[15] T. Desautels, A. Krause, and J. Burdick. Parallelizing Exploration–Exploitation Tradeoffs with Gaussian Process Bandit Optimization. ICML, 2012. 4, 5

[16] M. Evans and T. Swartz. *Approximating Integrals via Monte Carlo and Deterministic Methods*. Oxford Univ. Press, 2000. 7, 38

[17] E. Furbo. Evaluation of RANS Turbulence Models for Flow Problems with Significant Impact of Boundary Layers. Master's thesis, Uppsala Univ., 2010. 8

[18] D. Ginsbourger, R. L. Riche, and L. Carraro. Kriging is well-suited to parallelize optimization. In *chapter 6 of Computational Intelligence in Expensive Optimization Problems*, pages 131–162. Springer, April 2010. 3, 5, 9, 19, 38, 39, 42

[19] A. Henderson. ParaView Guide, A Parallel Visualization Application. Kitware Inc., 2007. 8, 10

[20] J. Janusevskis, R. Le Riche, and D. Ginsbourger. Parallel expected improvements for global optimization: Summary, bounds and speed-up. Technical report, École Nationale Supérieure des Mines de St-Étienne, 2010. 3, 5, 9, 19, 28, 38

[21] C. Johnson. A Gershgorin-type lower bound for the smallest singular value. *Lin. Alg. Appl.*, 112:1–7, 1989. 40

[22] H. Kushner. A new method of locating the maximum point of an arbitrary multipeak curve in the presence of noise. *J. of Basic Engineering*, 86:97–106, 1964. 3

[23] U. Lerner. *Hybrid Bayesian Networks For Reasoning About Complex Systems.* PhD thesis, Stanford Univ., 2002. 38

[24] J. Lu and D. Darmofal. Higher-dimensional integration with Gaussian weight for applications in probabilistic design. *SIAM J. Sci. Comput.*, 26:613–624, 2004. 41, 42

[25] J. McNamee and F. Stenger. Construction of fully symmetric numerical integration formulas. *Numerische Mathematik*, 10:327–344, 1967. 40

[26] T. Minka. Deriving Quadrature Rules from Gaussian Processes. `http://research.microsoft.com/en-us/um/people/minka/papers/`, 2000. 38

[27] J. Mockus. On a method of distribution of random experiments in the solution of multi-extremal problems. *USSR Comp. Math. and Math. Phys.*, 4(2):246–254, 1964. 3

[28] J. Mockus. The Bayesian approach to global optimization. In *Lec. Notes in Ctrl and Inf. Sc.*, volume 38, pages 473–481, 1982. 3

[29] S. M. N. Hansen and P. Koumoutsakos. Reducing the time complexity of the derandomized evolution strategy with covariance matrix adaptation (CMA-ES). *Evolutionary Comp.*, 11:1–18, 2003. 9, 10, 39

[30] C. Park, J. Huang, and Y. Ding. Domain Decomposition Approach for Fast Gaussian Process Regression of Large Spatial Data Sets. *J. of Machine Learning Research*, 12:1697–1728, 2011. 29

[31] B. Raghavan, P. Breitkopf, Y. Tourbier, and P. Villon. Towards a space reduction approach for efficient structural shape optimization. *Submitted to J. Struct. Multidiscipl. Opt.*, 2012. 6, 8, 10, 11, 17

[32] R. Regis and C. Shoemaker. Improved Strategies for Radial Basis Function Methods for Global Optimization. *J. Glob. Optim.*, 37(1):113–135, 2007. 5

[33] R. Regis and C. Shoemaker. Parallel radial basis function methods for the global optimization of expensive functions. *European J. Oper. Research*, 182:514–535, 2007. 3, 4

[34] R. Regis and C. Shoemaker. Parallel Stochastic Global Optimization Using Radial Basis Functions. *INFORMS J. on Computing*, 21(3):411–426, 2009. 4

[35] M. Schonlau. *Computer Experiments and Global Optimization.* PhD thesis, Univ. of Waterloo, 1997. 3, 4, 5

[36] J. Snoek, H. Larochelle, and R. P. Adams. Practical Bayesian optimization of machine learning algorithms. `arXiv:1206.2944v1`, 2012. 5

[37] A. Sóbester, S. Leary, and A. Keane. A parallel updating scheme for approximating and optimizing high fidelity computer simulations. *J. Struct. Multidisc. Optim.*, 27:371–383, 2004. 4

[38] N. Srinivas, A. Krause, S. Kakade, and M. Seeger. Gaussian Process Optimization in the Bandit Setting: No Regret and Experimental Design. In *ICML*, 2010. 3

[39] F. Teytaud. *Introduction of Statistics in Optimization*. PhD thesis, Univ. Paris Sud, 2011. 4, 9

[40] E. Vazquez and J. Bect. Convergence properties of the expected improvement algorithm with fixed mean and covariance functions. *J. of Stat. Planning and Inference*, 140(11):3088–3095, 2010. 3

[41] Y. Wu, D. Hu, M. Wu, and X. Hu. A numerical-integration perspective on Gaussian filters. *IEEE Trans. Sig. Proc.*, 54:2910–2921, 2006. 38, 42

[42] D. Yarotsky. Examples of inconsistency in optimization by expected improvement. *To appear in J. Glob. Opt.*, 2012. 3

[43] N. Yarvin and V. Rokhlin. Generalized Gaussian quadratures. *SIAM J. Sci. Comp.*, 20:699–718, 1998. 38

[44] W. Zhuo, Prabhat, C. Paciorek, C. Kaufman, and W. Bethel. Parallel kriging analysis for large spatial datasets. In *IEEE Int. Conf. on Data Mining Workshops*, pages 38–44, 2011. 29