Investigation of One-Go Evolution Strategy/Quasi-Newton Hybridizations

Thomas Bartz–Beielstein†, Mike Preuss, Günter Rudolph

Dortmund University, 44221 Dortmund, Germany,
http://ls11-www.cs.uni-dortmund.de/people/

Abstract. It is general knowledge that hybrid approaches can improve the performance of search heurististics. The first phase, exploration, should detect regions of good solutions, whereas the second phase, exploitation, shall tune these solutions locally. Therefore a combination (hybridization) of global and local optimization techniques is recommended. Although plausible at the first sight, it remains unclear how to implement the hybridization, e.g., to distribute the resources, i.e., number of function evaluations or CPU time, to the global and local search optimization algorithm. This budget allocation becomes important if the available resources are very limited. We present an approach to analyze hybridization in this case. An evolution strategy and a quasi-Newton method are combined and tested on standard test functions.

1 Introduction

Hybridizing *evolutionary algorithms* (EA) with *local search* techniques (LS) is not exactly a new idea. In fact, several such approaches exist (e.g. genetic local search, hybrid genetic algorithms) and nowadays, they are subsumed under the term *memetic algorithms* (MA) that was invented by [13]. A recent overview is given by [10], together with a suggested taxonomy.

Hovewer, MA usually do not treat EA and LS as coequal techniques. Instead, the local search methods are integrated into the evolutionary algorithm framework. This is straightforward as EAs are considered to have global search capabilities whereas LSs are prone to get stuck in the first local optimum they approach. Nevertheless, we follow a different path by simply applying an *evolution strategy* (ES) and a *quasi-Newton* (QN) method consecutively, without any other information exchange besides communicating the best solution found by the former into the latter. This scheme resembles the simplest possible of such combinations, thereby implying that the EA is able to detect a region near the global optimizer in one go which is then approximated by the local search method. The working hypothesis of commonly used MA differs insofar as the evolutionary algorithm is only required to step into the vicinity of *any* (possibly local) optimizer before the LS method takes over. In stark contrast to the situation investigated here, MAs mostly apply both techniques several times.

†corresponding author: Thomas.Bartz-Beielstein@udo.edu

The main motivation for hybridizing ES and QN by simply applying them consecutively only once is drawn from two sources:

- Combining a global and a local search technique is expected to result in better performance than either of them alone, especially if at least one of the techniques can be tailored to deliver what the other needs to proceed.
- Real-world applications as e.g. design problems enforce short runs: The available time poses hard limits onto the allowed number of evaluations, often less than 10^4 can be afforded. Consequently, frequent switching between global and local search techniques may be inappropriate for such problems.

In recent research, we observe two contradictory trends: (i) to develop more and more new algorithms or (ii) to analyze and understand existing heuristics and to add new features only when necessary. With this work, we lean against the second trend by taking two existing algorithms and combining them in a very simple fashion. However, we do not add new features but rather try to *adapt* an EA to work well in combination with a QN-algorithm by tuning its parameters and adjusting the fraction of shared resources it is allowed to consume.

The paper is organized as follows: Section 2 introduces the algorithms that will be hybridized: an evolution strategy and a QN-method, followed by a brief description of the resulting hybrid algorithm. The experimental methodology is introduced in Sect. 3. It relies on the sequential parameter optimization approach, which has been applied to several optimization tasks from industrial optimization and theoretical computer science. Experiments are presented in Sect. 4. Our focus lies on optimization problems with limited resources. Section 5 analyzes the experimental results, and Sect. 6 summarizes the conclusions drawn from this study.

2 Algorithms

2.1 Evolution Strategies

An ES-algorithm run may be characterized as follows: The parental population is *initialized* at time (generation) $g = 0$. Then λ offspring individuals are generated in the following manner: For each offspring individual, a parent family of size ρ is selected randomly from the parent population. *Recombination* is applied to the object variables and the strategy parameters. The mutation operator is applied to the resulting offspring vector. After evaluation, the next parent population is determined by means of a *selection* procedure. The populations created in the iterations of the algorithm are called *generations* or *reproduction cycles*. Unless a termination criterion is fulfilled, the generation counter (g) is incremented and the process continues with the generation of the next offspring. We consider the parameters or control variables from Table 1. This table shows typical parameter settings. Bäck does not recommend using "standard" without reflection. Considering the no-free lunch debate and current results from experimental research, it is obvious that problems exist where these "standards" fail. Thus it is necessary to adjust the parameters to the specific optimization problem. SPO,

Table 1. Default settings of exogenous parameters of a "standard" evolution strategy [1]. The ES parameters can be described as follows: The symbols μ and λ denote parent and offspring population sizes, respectively. The offspring-parent ratio is defined as $\nu = \lambda/\mu$, $\sigma^{(0)}$ denotes the initial standard deviation, which is used for mutation. Let d be the problem dimension. Then between $n_\sigma = 1$ and d different standard deviations can be used. c_0 and c_1 denote multipliers for the global and local learning rates, respectively, as described in Equation (27) in [4]. Note, [4] use the same c, i.e., $c_1 = c_2$ for global and local learning rates. The parameter ρ describes the number of parent individuals used in recombination and r_d and r_i denote discrete and intermediary recombination, respectively. Intermediate recombination has been used for both object and strategy parameters in our experiments. The symbol κ is the maximum lifespan of an individual, the so-called comma strategies use $\kappa = 1$, whereas plus strategies use $\kappa = \infty$. Parameters, that are tuned, are printed in *boldface*.

Symbol	Parameter	Range	Default
μ	**Number of parent individuals**	$\mathbb{N}$	**15**
ν	**Offspring–parent ratio**	$\mathbb{R}_+$	**7**
$\sigma_i^{(0)}$	Initial standard deviations	$\mathbb{R}_+$	3
n_σ	Number of standard deviations	$\{1, 2, \ldots, d\}$	1
$\mathbf{c_0}$	**Multiplier for the global learning rate**	$\mathbb{R}_+$	**1**
$\mathbf{c_1}$	**Multiplier for the local learning rate**	$\mathbb{R}_+$	**1**
ρ	Mixing number	$\{1, 2, \ldots, \mu\}$	2
r_x	Recombination operator for object variables	$\{r_i, r_d\}$	r_d
r_σ	Recombination operator for strategy variables	$\{r_i, r_d\}$	r_i
κ	Maximum age	$\mathbb{N}$	1

as described in Sect. 3.2, provides one possible technique to avoid poor results caused by wrongly specified parameters. The reader is referred to [2] and [4] for detailed descriptions of these parameters.

2.2 Quasi-Newton Methods

The variable metric method utilized for the experiments in this study is a QN-method. Quasi-Newton methods build up curvature information. Let H denote the Hessian, c a constant vector, and b a constant, then a quadratic model problem formulation of the form $\min_x \frac{1}{2}x^T Hx + c^T x + b$ is constructed. If the partial derivatives of x go to zero, i.e., $\nabla f(x^*) = Hx^* + c = 0$, the optimal solution for the quadratic problem occurs. Hence $x^* = -H^{-1}c$. Quasi-Newton methods avoid the numerical computation of the inverse Hessian H^{-1} by using information from function values and gradients. The MATLAB function `fminunc` uses the formula of [5], [7], [8], and [18] to approximate H^{-1}.

2.3 ES/QN-Hybrid

The ESQN algorithm combines the ES and QN by running them consecutively and initializing the latter with the result of the former. The parameter ES2QN

distributes the available resources to the algorithms, i.e., it defines the percentage of function evaluations for the ES: ES2QN $\in [0.0, 1.0]$. The remainder is assigned to the QN-strategy. For example, if ES2QN is 0, the ES receives no function evaluation and the ESQN is a canonical QN-method. Allotting more time for the ES cuts down resources available to the QN-algorithm and vice versa.

3 Experimental Methodology

3.1 Problem and Algorithm Designs

The concept of *experimental designs* is crucial for our approach. On the one hand, search heuristics such as the Nelder-Mead simplex strategy, genetic algorithms, or particle swarm optimization require the specification of *exogenous* parameters before the algorithm is started. On the other hand, *endogenous* parameters can evolve during the optimization process, e.g., in self-adaptive evolution strategies. We will consider exogenous parameters in the following. By varying the values of the exogenous parameters the experimenter can get some insight into the behavior of an algorithm. This procedure can be described as *active experimentation* in contrast to *passive experimentation*, where the experimenter only observes some phenomena. Passive experimentation predominated experimental research in evolutionary computation until recently. Nowadays, more and more active experimental approaches are developed.

Exogenous parameters will be referred to as *design variables* in the context of statistical design and analysis of experiments. The parameter values chosen for the experiments constitute an algorithm design X_A. Let $\mathcal{D}_A$ denote the set of all possible parameter settings for one algorithm. A *design point* $x_a \in \mathcal{D}_A$ represents one specific parameter setting. *Algorithm* tuning *can be understood as the* process of finding the optimal *design point* $x_a^* \in \mathcal{D}_A$ *for a given problem design* X_P. Tuning leads to results that are tailored for one specific algorithm-optimization problem combination. To discuss the behavior of an algorithm the underlying problem has to be taken into account. A problem being GA easy may be ES hard, and vice versa. Tuning enables a fair comparison of two or more algorithms that should be performed prior to their comparison. This should provide an equivalent budget—for example, a number of function evaluations or an overall run time—for each algorithm.

It is crucial to formulate the goal of the tuning experiments precisely, because in many real-world situations, it is not possible or not desired to find the optimum. A good solution, i.e., a robust solution, is often preferred. This discussion is also relevant for the specification of *performance measures* (PM) in evolutionary computation. There are many different measures for the goodness of an algorithm, i.e., the quality of the best solution, the percentage of runs terminated successfully, or the number of iterations required to obtain the results.

3.2 Sequential Parameter Optimization

Sequential parameter optimization (SPO) is a methodology for the experimental analysis of optimization algorithms to determine improved algorithm designs

Algorithm 1 Sequential parameter optimization

1: **procedure** SPO($\mathcal{D}_A$, $\mathcal{D}_P$) ▷ Algorithm und problem design
2: Select $p \in \mathcal{D}_P$ and set $t = 0$ ▷ Select problem instance
3: $X_A(t) = \{x^{(1)}, x^{(2)}, \ldots, x^{(k)}\}$ ▷ Sample k initial points, e.g., LHS
4: **repeat**
5: $y_j^{(i)} = Y_j(x^{(i)}, p) \forall x^{(i)} \in X_A(t)$ and $j = 1, \ldots, r(t)$ ▷ Fitness evaluation
6: $\overline{Y}^{(i)}(t) = \sum_{j=1}^{r(t)} y_j^{(i)}(t)/r(t)$ ▷ Sample statistic for the ith design point
7: x_b with $b = \arg\min_i(\overline{y}^{(i)})$ ▷ Determine best point
8: $Y(\cdot) = \mathcal{F}(\beta, \cdot) + Z(\cdot)$ ▷ DACE model
9: $X_S = \{x^{(k+1)}, \ldots, x^{(k+s)}\}$ ▷ Generate s sample points, $s \gg k$
10: $y(x^{(i)})$, $i = 1, \ldots, k + s$ ▷ Predict fitness from the DACE model
11: $I(x^{(i)})$ for $i = 1, \ldots, s + k$ ▷ Determine expected improvement, cf. [17]
12: $X_A(t + 1) = X_A(t) \cup \{x^{(k+i)}\}_{i=1}^{m}$ ▷ Add m points with the highest $I(\cdot)$
13: **if** $x_b(t) = x_b(t + 1)$ **then**
14: $r(t + 1) = 2r(t)$ ▷ Increase number of repeats
15: **end if**
16: t = t+1;k=k+m ▷ Increment counters
17: **until** Budget exhausted
18: **end procedure**

and to learn, how the algorithm works. It employs computational statistic methods to investigate the interactions among optimization problems, algorithms, and environments. We consider each algorithm design with associated output as a realization of a stochastic process and use interpolation method to predict unknown values. Our presentation follows concepts introduced in [16], [9], and [11].

Consider a set of m design points $x = \{x^{(1)}, \ldots, x^{(k)}\}$ with $x^{(i)} \in \mathbb{R}^d$. In the *design and analysis of computer experiments* (DACE) *stochastic process model*, a deterministic function is evaluated at these design points. The vector of the k responses is denoted as $y = (y^{(1)}, \ldots, y^{(k)})$ with $y^{(i)} \in \mathbb{R}$. The process model proposed in [16] expresses the deterministic response $y(x^{(i)})$ for a d-dimensional input $x^{(i)}$ as a realization of a regression model $\mathcal{F}$ and a stochastic process Z. Algorithm 1 describes the SPO in a formal manner. The selection of a suitable problem instance is done in the pre-experimental planning phase to avoid floor and ceiling effects (l.2). Latin hypercube sampling can be used to determine an initial set of design points (l.3). After the algorithm has been run with these k initial parameter settings (l.5), the DACE process model is used to discover promising design points (l.10). Note that other sample statistics than the mean, e.g., the median, can be used in l.6. The m points with the highest expected improvement are added to the set of design points, where m should be small compared to s. The update rule for the number of reevalutions $r(t)$ (l.13-15) guarantees that the new best design point $x_b(t + 1)$ has been evaluated at least as many times as the previous best design point $x_b(t)$. Obviously, this is a very simple update rule and more elaborate rules are possible. Other termination criteria exist besides the budget based termination (l.17). Figure 1 illustrates

a typical situation from SPO. Here, small population sizes and high selective pressures are beneficial.

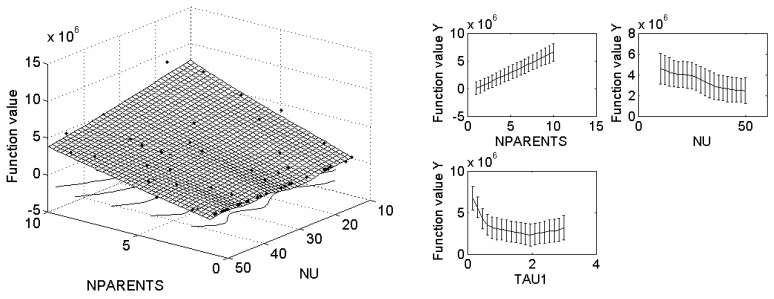

Fig. 1. Typical results from the sequential parameter optimization of the ES. *Left:* Interactions between population size and selective pressure *Right:* Plots of the single effects

A toolbox that implements the sequential parameter optimization is available under the following link: `http://www.springer.com/3-540-32026-1`. Additional material, e.g., the implementation of the evolution strategy used in the following experiments can be downloaded, too. Furthermore, we will provide interfaces to SPO for commonly used search heuristics such as particle swarm optimization, genetic algorithms, or commercial optimization-software packages.

4 Experiments

4.1 Problem Design

We decided to use the deterministic initialization scheme DETEQ, see [3]. It uses one single starting point, i.e., x_0, so that the same initial conditions are used by both algorithms and the hybrid approach.[1] This is a disadvantage for the population based ES because it is forced to spread search points of its initial population (by applying the mutation operator) within a tight cloud around the starting point, rather than distributing them throughout the whole search space. However, our main focus does not lie on a direct comparison of the algorithms, but on the effect of the hybridization.

To check for floor and ceiling effects, the number of function evaluations was varied during the pre-experimental planning phase. This ensures that the problem design is not too easy or to hard for the algorithms under consideration. Floor and ceiling effects are discussed in [6, 3]. To enable a fair comparison, we

[1] Note, x_0 should not be confused with $x^{(0)}$ defined in Algorithm 1. The former describes the starting point for one algorithm run, the latter is one parameter set of the optimization algorithm.

have chosen $t_{\max}$, i.e., the maximum number of function evaluations, as 100 $\times$ problem dimension. This value appears to be very small ES, because ES need a certain amount of function values to adapt they step sizes. However, our experiments reveal some interesting insight into the ES performance that might correct some typical prejudices against ES.

4.2 Algorithm Designs

A suitable algorithm design has to be determined. Clearly, for this specific situation, "standard" parameter settings from the literature are not adequate. Therefore, SPO was used to detect suitable algorithm designs for the ES. Due to the small number of function evaluations, population sizes between 1 and 10 individuals have been used. The selective pressure was chosen from the interval $]0, 10]$. The region of interest for the learning parameters c_0 and c_1 was defined as the interval $[0.1, 3]$. The related ES algorithm designs for the selected functions from [12] are summarized in Table 3. Note, that the parameters from the tuned algorithms show no directly observable patterns, so that no general recommendations can be given for an ES algorithm design that works equally well on every function from the [12] test set.[2]

4.3 Experiments on Moré's Test Problems

Due to the limited space, function definitions are omitted. The reader is referred to [12] and [14] for a full description of these functions. We have included some plots to illustrate some characteristics.

Rosenrock The Rosenbrock function is the first function from the collection described in [12] [15]. Minimum $x^* = (1, 1)$. Optimum $f^* = 0$. Starting point $x_0 = (-1.2, 1)$. This is the famous two-dimensional "banana valley" function.

Experiments with the canonical ES and QN for Rosenbrock's function showed, that QN and ES are able to solve this problem in principle. Now we will tackle the central question from this paper: does it pay to hybridize ES and QN? Therefore, we have generated a series of ES2QN plots, e.g., in Fig. 2 (right). These plots enable a direct comparison of the canonical ES and QN algorithms: if ES2QN is 0 (0 % ES, but 100% QN), the average performance for $n = 10$ runs of the QN algorithm is shown. If ES2QN is 1 (100% ES), the performance of the ES can be seen. Intermediate ES2QN values, i.e., ESQN $\in]0, 1[$, show the performance of hybrid approaches. In addition to the mean value from ten runs, the minimum, maximum, and the `bestof` function values are plotted, because they have a great practical relevance. The `bestof` value is determined from n values as follows: determine the minimum value from m random draws (with replacement) out of n ($m < n$). This procedure is repeated very often, say 1,000,000 times, and the average value is reported. The `bestof` value is larger than the minimum, but smaller than the mean value. We have chosen $m = 5$, because 5 repeated

[2] The QN-method was not tuned, because MATLAB does not provide any interfaces to adjust exogenous parameters.

Table 2. Problem designs for the experiments performed on the [12] test suite. The DETEQ initialization method, the EXH termination criterion, and $n = 10$ repeats are used for all experiments. The experiment's name, the maximum number of function evaluations $t_{\max}$, the problem's dimension d, the starting point x_0 for the initialization of the object variables are reported

Problem design	$t_{\max}$	d	x_0
$x^{(1)}_{\texttt{rosen}}$	200	2	$(-1.2, 1)$
$x^{(1)}_{\texttt{froth}}$	200	2	$(0.5, -2)$
$x^{(1)}_{\texttt{bscp}}$	200	2	$(0, 1)$
$x^{(1)}_{\texttt{bscb}}$	200	2	$(1, 1)$
$x^{(1)}_{\texttt{jensam}}$	200	2	$(0.3, 0.4)$
$x^{(1)}_{\texttt{osborne2}}$	200	11	see [12]
$x^{(1)}_{\texttt{meyer}}$	300	3	$(0.02, 4000, 250)$

runs represent a realistic situation in many real world optimization scenarios. We did not show plots with error bars (or confidence intervals), because for our purpose, the mean, min, max, `bestof` (MMMB plots) provide more information. A comparison of both representations is shown in Fig. 7.

Table 3. ES algorithm designs. Further ES parameters remained constant as described in Sect. 2.1. Rosenbrock: $x^{(1)}_{\text{ES}}$, Freudenstein and Roth: $x^{(2)}_{\text{ES}}$, Powell badly scaled: $x^{(3)}_{\text{ES}}$, Brown badly scaled: $x^{(4)}_{\text{ES}}$, Jenrich and Sampson: $x^{(5)}_{\text{ES}}$, Meyer: $x^{(6)}_{\text{ES}}$, and Osborne 2: $x^{(7)}_{\text{ES}}$.

Algo. design	μ	ν	c_1	c_2
$x^{(1)}_{\text{ES}}$	1	1.5646	0.315154	0.102151
$x^{(2)}_{\text{ES}}$	1	4.35957	0.215921	2.10074
$x^{(3)}_{\text{ES}}$	6	1.09798	2.93576	2.94653
$x^{(4)}_{\text{ES}}$	4	2.29763	1.66219	2.90905
$x^{(5)}_{\text{ES}}$	8	4.70181	1.87773	0.27439
$x^{(6)}_{\text{ES}}$	9	1.239	0.792342	1.93755
$x^{(7)}_{\text{ES}}$	2	6.94046	1.71284	0.537968

Figure 2 clearly indicates that QN outperforms ES and that hybridization worsens the performance for this setting. This result is in accordance with results reported in [14], where the QN algorithm reached a function value of $1.15e-10$ with 150 function evaluations only.

Freudenstein and Roth Minimum $x^* = (5, 4)$. Optimum $f^* = 0$. Starting point $x_0 = (0.5, -2)$. This is function 2 from the [12] test set. Figure 3 shows a 3 dimensional and contour plot. It indicates that hybridization is beneficial and that ES performs slightly better than QN. The ES generates solution candidates

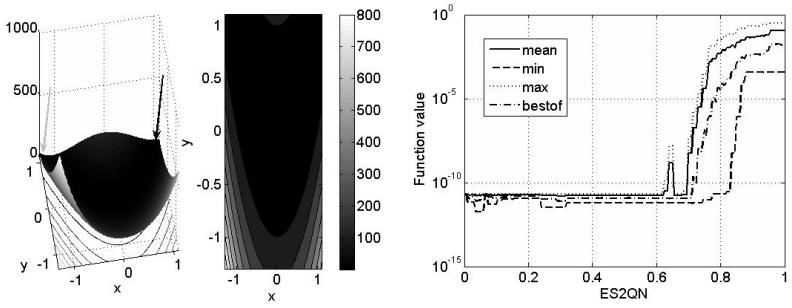

Fig. 2. Rosenbrock. *Left:* The gray arrow depicts the starting point $x_0 = (-1.2, 1)$, the black arrow the optimizer $x^* = (1, 1)$.*Right:* Results from the hybridization ($n = 10$ repeats, this value was used in the following plots, too) clearly demonstrate that hybridization does not improve the algorithm, because QN outperforms ES

that "jump over the saddle ($y \equiv 2$)", and QN can fine tune these solutions. Best results are obtained if approximately 3/4 of the budget is assigned to the ES.

Powell Badly Scaled Minimum $x^* = (1.098\ldots 10^{-5}, 9.106\ldots)$. Optimum $f^* = 0$. Starting point $x_0 = (0, 1)$. This is function 3 from the [12] test set. Figure 4 shows a 3 dimensional and contour plot. This plot illustrates that comparing mean values alone tells not the whole story. Hybridization can improve the performance, but this is not guaranteed. However, it might be a good strategy, if the user can select the best result from several runs (as modeled in the performance measure `bestof`). SPO proposed a 6+6-ES, see Table 3. The following situation could be observed in some runs: The ES was able to detect values close to the optimizer after 3 generations, which could be improved by QN.

Brown Badly Scaled Minimum $x^* = (10^6, 2 \cdot 10^{-6})$. Optimum $f^* = 0$. Starting point $x_0 = (1, 1)$. This is function 4 from the [12] test set. Figure 5 shows a 3 dimensional and contour plot. A first look at the results leads to the conclusion

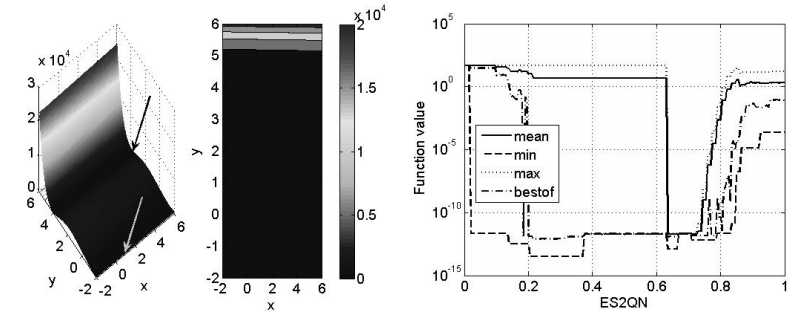

Fig. 3. Freudenstein and Roth. *Left:* The gray arrow depicts the starting point $x_0 = (0.5, 2)$, the black arrow the optimizer $x^* = (5, 4)$.*Right:* Experimental results indicate that hybridization can improve the algorithm's performance

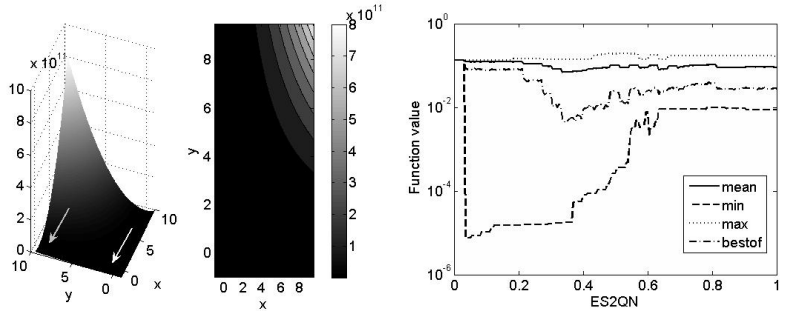

Fig. 4. Powell Badly Scaled. *Left:* The white arrow depicts the starting point $x_0 = (0, 1)$, the gray arrow the optimizer $x^* = (1.098\ldots 10^{-5}, 9.106\ldots)$.*Right:* Results from the hybridization.

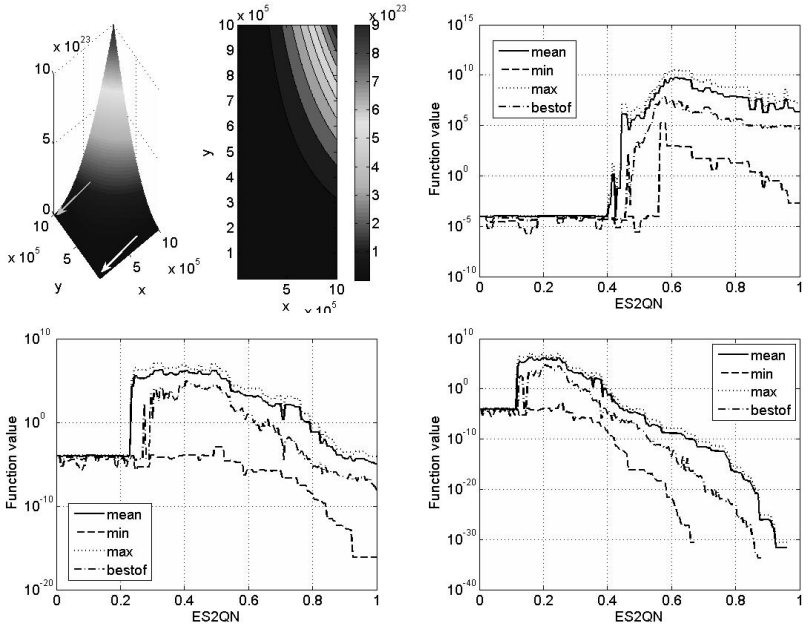

Fig. 5. Brown Badly Scaled. *First row, left:* The white arrow depicts the starting point $x_0 = (1, 1)$, the gray arrow the optimizer $x^* = (10^6, 2 \cdot 10^{-6})$. *Right:* Results from the hybridization, $t_{\max}$= 200. *Second row, left:* $t_{\max}$= 400, *right:* $t_{\max}$= 800. Some curves end abruptly, because the plotted values are zero, which is the known minimum

that QN performs better than the ES, cf. the right graph in the first row. However, this result depends heavily on the number of available function evaluations, i.e., $t_{\max}$. If $t_{\max}$is increased, ES performs better than QN. Hybridization has no positive effect, it is better to use the canonical algorithms. The ES needs some time adapting the step width, but was able to detect the minimizer. QN finds suboptimal solutions with fewer function evaluations. QN could not detect the optimizer, even if $t_{\max}$was increased as can be seen from the graphs in the second row of Fig. 5.

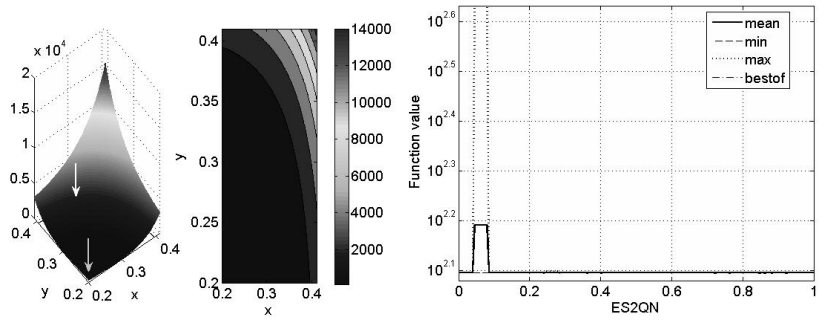

Fig. 6. Jenrich and Sampson. *Left:* The white arrow depicts the starting point $x_0 = (0.3, 0.4)$, the gray arrow the optimizer $x^* = (0.2578\ldots, 0.2578\ldots)$.*Right:* Results from the hybridization illustrate that hybridization worsens the algorithm's performance

Jenrich and Sampson Minimum $x^* = (0.2578\ldots, 0.2578\ldots)$. Optimum $f^* = 124.362\ldots$. Starting point $x_0 = (0.3, 0.4)$. This is function 6 from the [12] test set. Figure 6 shows a 3 dimensional plot and contour plot. Both algorithms perform equally well, there is no benefit in hybridization. Hybridization worsens the performance in some settings, see Fig. 6.

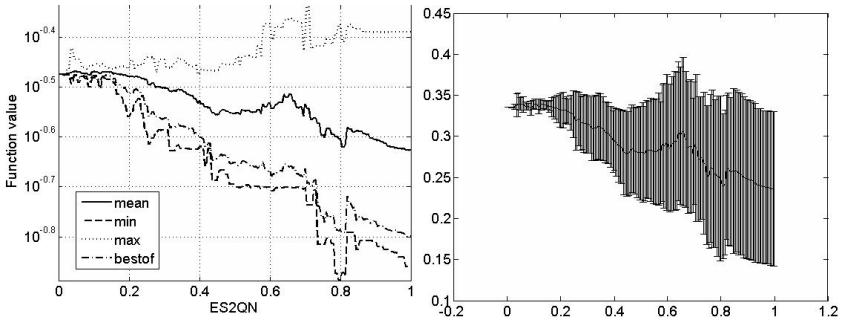

Fig. 7. Osborne 2. *Left:* Results from the hybridization.*Right:* Results from the hybridization illustrate that hybridization does not improve the algorithm's performance

Osborne 2 Osborne 2 was included into the test function set, because it the 11-dimensional function. The test suite from [12] contains 6 two, three, and four dimensional, 1 five, six, and nine dimensional, 9 ten dimensional, and 1 eleven dimensional test function. [14] reports some results from optimization attempts with the MATLAB optimization toolbox: This 11 dimensional problem could not be solved by MATLAB's BFGS without supplying gradient information. And, even with gradient information, more than 10,000 function evaluations were required for finding a point in the vicinity of the global optimizer. [3] Osborne 2 is function 19 from the [12] test set. Figure 7 nicely illustrates the trade off between deterministic (QN) and stochastic (ES) search algorithms. If the user needs a good result with a high reliability, she should use the QN. If she can afford several runs, then ES is the correct choice. There is no guarantee that ES detects a better solution, but a hight probability. Hybridization is not recommended. The plot on the right shows the same data as on the left, but uses error bars.

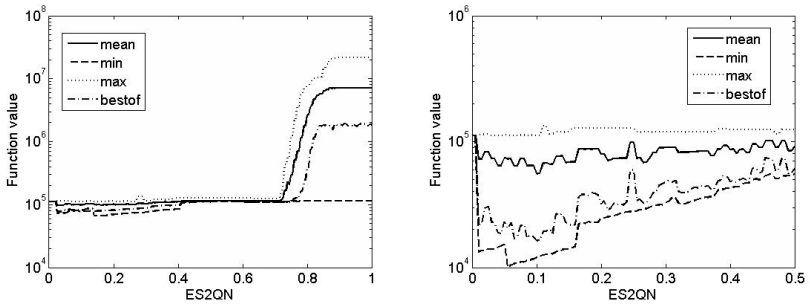

Fig. 8. Meyer. *Left:* Results from the hybridization with 400 function evaluations.*Right:* Results from the hybridization with 1000 function evaluations

Meyer Meyer's function was added to our test set, because it is a three dimensional function on which MATLAB's BFGS method failed. [14] reports a function value of 3.4675e+007 at solution found after 10,002 function evaluations , whereas the best known minimum reads $f^* = 87.9458$. This is function 10 from the [12] test set. Figure 8 suggests that QN performs better than ES. But this is an artefact, because both algorithms failed. They did not find a value in the vicinity of the optimizer. Hence, the problem is too hard for both algorithms. Increasing the computational budget, i.e., $t_{\max}$, does not lead to better results. Therefore, the difference is statistical significant—but not scientifically relevant.

[3] We have obtained slightly different, i.e., better, results, because we used a newer MATLAB release (R14).

5 Analysis

The main research goal of our study addresses the question "Are there situations in which the hybridization of ES and QN methods improve their performance ?" The analysis of the experiments produced no clear picture. QN is more robust than the ES in the traditional definition of robustness, i.e., low standard deviations. This robustness can be seen as an disadvantage, e.g., if the optimization practitioner can afford several runs from which she chooses the best.

Looking at local run properties reveals that the performance improvement is caused by the following effect: The ES explores the search space and detects a suitable starting point that is passed to the QN, which performs a local fine tuning. This is superior to the global search behavior of the ES alone and the local strategy of the QN-methods. However, we could not derive general guidelines, e.g., "choose an ES2QN value of 0.31415 to improve the algorithm's performance". The hybrid approach has also some advantages compared to an approach that performs a sampling of the search space in the first phase and runs a QN method in the second phase, because the region of interest is not known in many situations. The ES jumps to a promising region in the first steps, so that the additional refinement with the QN can be performed efficiently. A comparison to multi-start techniques is of great interest and has not been done in our study.

As expected and mentioned in the abstract, hybridization can improve algorithm's performance, even if the resources are very limited. Restricted resources are standard situations in industrial optimization, because function evaluations are very costly or results must be available immediately, i.e., in optimization via simulation or in real-time optimization scenarios, respectively. We observed the following results that might be transferable to other situations as well: Algorithms that are specialized for certain (simple) optimization scenarios cannot benefit from hybridization. This is understandable, because these algorithms need a certain budget to adapt their internal model, e.g., step sizes in ES or the gradient and Hessian approximation for QN-methods. Switching to another algorithm is costly, it might be beneficial only if no progress can be obtained with the current strategy. Results from Rosenbrock's function support this assumption.

It is important to tune the ES, i.e., to determine suitable algorithm designs. SPO, or related tools, can provide a quick overview of suitable parameter settings. Evolution strategies with standard setting from the literature failed in our scenarios. Not only the algorithms have to be tuned before the experiment is started—it was crucial to find an experimental setup that is neither too hard nor too easy for the algorithms as can be seen from Meyer's function.

6 Summary

No general recommendations—especially for real-world optimization problems—can be given here, because several factors influence the algorithm's performance. Consider the computational budget: Modifications lead to different results. SPO

or related techniques can be applied in this situation, because they can improve the performance significantly. There is no need for hybridization if well tuned algorithms on simple test functions are considered. Only if the problem structure is complex, the combination of global, stochastic search and local, gradient-based strategies is useful. The hybrid ESQN communicates only the best found solutions between its two parts. It may however be beneficial to take over the already learned internal model of the EA (mutation strengths) into the QN-method. Investigating this remains as a task for future research.

Acknowledgment The research leading to this paper was supported by the DFG (Deutsche Forschungsgemeinschaft) as part of the collaborative research center "Computational Intelligence" (531) and by project grant no. 252441, "Mehrkriterielle Struktur- und Parameteroptimierung verfahrenstechnischer Prozesse mit evolutionären Algorithmen am Beispiel gewinnorientierter unscharfer destillativer Trennprozesse".

References

1. T. Bäck. *Evolutionary Algorithms in Theory and Practice*. Oxford University Press, New York NY, 1996.
2. Thomas Bartz-Beielstein. Experimental analysis of evolution strategies—overview and comprehensive introduction. Interner Bericht des Sonderforschungsbereichs 531 Computational Intelligence CI–157/03, Universität Dortmund, Germany, November 2003.
3. Thomas Bartz-Beielstein. *Experimental Research in Evolutionary Computation—The New Experimentalism*. Springer, Berlin, Heidelberg, New York, 2006.
4. H.-G. Beyer and H.-P. Schwefel. Evolution strategies—A comprehensive introduction. *Natural Computing*, 1:3–52, 2002.
5. C. G. Broyden. The convergence of a class of double-rank minimization algorithms. *Journal of the Institute of Mathematics and Its Applications*, 6:76–90, 1970.
6. Paul R. Cohen. *Empirical Methods for Artificial Intelligence*. MIT Press, Cambridge MA, 1995.
7. R. Fletcher. A new approach to variable metric algorithms. *Computer Journal*, 13:317–322, 1970.
8. D. Goldfarb. A family of variable metric updates derived by variational means. *Mathematics of Computing*, 24:23–26, 1970.
9. D.R. Jones, M. Schonlau, and W.J. Welch. Efficient global optimization of expensive black-box functions. *Journal of Global Optimization*, 13:455–492, 1998.
10. Natalio Krasnogor and Jim E. Smith. A Tutorial for Competent Memetic Algorithms: Model, Taxonomy and Design Issues. *IEEE Transactions on Evolutionary Computation*, 5(9):474–488, 2005.
11. S.N. Lophaven, H.B. Nielsen, and J. Søndergaard. DACE—A Matlab Kriging Toolbox. Technical Report IMM-REP-2002-12, Informatics and Mathematical Modelling, Technical University of Denmark, Copenhagen, Denmark, 2002.
12. J. J. More, B. S. Garbow, and K. E. Hillstrom. Testing unconstrained optimization software. *ACM Transactions on Mathematical Software*, 7(1):17–41, 1981.

13. Pablo Moscato. On Evolution, Search, Optimization, Genetic Algorithms and Martial Arts: Towards Memetic Algorithms. Technical Report Caltech Concurrent Computation Program, Report. 826, California Institute of Technology, Pasadena, California, USA, 1989.
14. Arnold Neumaier. "Results for moré/garbow/hillstrom test problems", 2006. `http://www.mat.univie.ac.at/~neum/glopt/results/more/moref.html`. Cited 19 Mai 2006.
15. H.H. Rosenbrock. An automatic method for finding the greatest or least value of a function. *Computer Journal*, 3:175–184, 1960.
16. J. Sacks, W. J. Welch, T. J. Mitchell, and H. P. Wynn. Design and analysis of computer experiments. *Statistical Science*, 4(4):409–435, 1989.
17. T. J. Santner, B. J. Williams, and W. I. Notz. *The Design and Analysis of Computer Experiments*. Springer, Berlin, Heidelberg, New York, 2003.
18. D. F. Shanno. Conditioning of quasi-Newton methods for function minimization. *Mathematics of Computing*, 24:647–656, 1970.