

Comparison of Derivative-Free Algorithms for their Applicability in Self-Optimization of Chemical Processes

Sebastian Soritz,* Daniel Moser, and Heidrun Gruber-Wölfler*^[a]

In this work, several implementations of different derivative-free optimization algorithms are compared for the usage in chemical process optimization. As such, a benchmarking process is carried out, using optimization problems of different types to compare reliability, accuracy, and performance. Finally, using an automated reaction setup and a bespoke Python-based script

featuring a graphical user interface, all algorithms are tested in an optimization of a Suzuki-Miyaura cross-coupling reaction in continuous flow. To increase the scope of comparison, a model function based on the reaction is also used, to allow for a more in-depth comparison without the use of physical resources.

Introduction

For a synthetic chemist, finding optimal parameters, for instance yield, purity, or selectivity of a reaction by varying its conditions, such as temperature and reaction time, is one of the main goals. More generally, optimization strategies are used to minimize or maximize an objective function, by varying parameters that influence the objective function output.

The optimization of chemical processes has traditionally been carried out using trial-and-error laboratory work, for example, the one-variable-at-a-time (OVAT) approach.^[1,2] In recent times, however, more advanced strategies, such as Design of Experiments (DoE) and even more complex machine-learning algorithms have been reported in literature.^[3–5] This step from OVAT to multidimensional optimization has been greatly aided by the increasing automation of laboratory equipment, especially in the context of continuous flow processing.^[6–8] Combining these advanced optimization strategies with the use of automated continuous processes and in-line analytics has led to the field of self-optimization of chemical processes in continuous flow. Self-optimization procedures combine multiple fields, such as chemistry, data science and chemical engineering to minimize tedious laboratory work for researchers.^[9,10] The focus of these single-objective optimization algorithms is to find ideal process parameters to maximize a specific process step outcome, such as the yield for chemical reactions. Other, more mathematically advanced measures


include the space-time yield, as the amount of product formed per unit time and unit reactor volume and the E-factor,^[11] as the ratio between desired product versus waste formed.


Chemical reactions can be regarded as expensive to evaluate due to the cost of reagents and the time that needs to be invested. Therefore, algorithms in use are also primed to focus on minimizing the number of experiments needed to reach their goal. For practical applications, chemical reactions can also be regarded as derivative-free or “black box” functions. This means that no derivatives, like the Hessian or Jacobian matrix, can be inferred from the optimization process, and the underlying function which determines the output parameter is unknown. With regard to chemistry, this is due to the inherent difficulty of understanding and quantifying the underlying principles that govern a reaction, be they of kinetic, thermal, or stoichiometric nature. These requirements narrow the usable self-optimization strategies to specific algorithms or methods.^[12] Additionally, real-world applications in the field of chemistry nearly all exhibit a certain level of noise. One example would be a shifting baseline during the analysis of reaction outcomes, which in continuous flow processes is mostly done through in-line measurements, with the most popular methods being HPLC,^[13] MS,^[14] NMR,^[15] and IR.^[16]

Currently, two of the most widely used methods in literature, which fulfill the criteria mentioned above, are the Nelder-Mead Simplex Algorithm^[17] and the Stable Noisy Optimization by Branch and FIT (SNOBFIT) method.^[18] Exemplarily, Ley et al. used a version of the simplex method in an automated continuous flow setup to optimize an Appel reaction over 30 iterations.^[14] Another approach was done, again using a modified simplex method, by Felpin et al., optimizing a Heck-Matsuda reaction in flow for three different outcomes, namely, yield, throughput and product cost.^[19] Lastly, Fath and Kockmann successfully compared the DoE approach with a modified simplex algorithm for an imine synthesis in flow, using an automated platform capable of responding to process disturbance.^[5]

As for SNOBFIT, a comparison of the Nelder-Mead and SNOBFIT methods were done by Bourne et al., where a super-modified Simplex algorithm was compared with the SNOBFIT

[a] S. Soritz, D. Moser, Prof. Dr. H. Gruber-Wölfler
Institute of Process and Particle Engineering,
Graz University of Technology
Inffeldgasse 13
8010 Graz (Austria)
E-mail: s.soritz@tugraz.at
woelfler@tugraz.at

 Supporting information for this article is available on the WWW under <https://doi.org/10.1002/cmt.202100091>

 © 2022 The Authors. Published by Wiley-VCH GmbH. This is an open access article under the terms of the Creative Commons Attribution Non-Commercial License, which permits use, distribution and reproduction in any medium, provided the original work is properly cited and is not used for commercial purposes.

method studying a continuous flow methylation reaction using an in-line FTIR spectrometer.^[16] A second comparison was carried out by McMullen and Jensen, where a Knoevenagel condensation was performed in flow and optimized using both the Simplex and SNOBFIT methods.^[13]

The comparison of optimization algorithms, also called benchmarking, shows a plethora of data on the subject of optimization algorithms in the computational sciences.^[20–22] Exemplarily, Rios and Sahinidis made a review of a specific set of derivative-free optimization algorithms and their overall performance by systematically comparing 22 different algorithm implementations for 502 test problems.^[20]

The first comparison of derivative-free optimization strategies for chemical processes, however, was recently published by Felton et al.^[23] They developed the “Summit” framework which uses in silico benchmarks of experimentally validated chemical reactions to compare multiple optimization strategies. Their picks included the Nelder-Mead and SNOBFIT methods, as well as three Bayesian optimization methods, such as the TSEMO algorithm,^[24] and Reinforcement learning.^[25] Using benchmark models of both a S_NAr and a cross-coupling reaction, they were able to identify Bayesian optimization as the most efficient among the chosen optimization methods.

A more general but not result-centered approach was conducted by Häse et al. with the development of the Olympus framework for benchmarking noisy optimization problems.^[26] It incorporated multiple benchmarks and analytical surfaces, as well as both derivative- and non-derivative based optimization methods.

While the extant literature in the field of self-optimization has shown the successful application of optimizers, such as the Nelder-Mead and SNOBFIT methods, a wide variety of conceivably suitable algorithms exist in the computational sciences, that have not yet been used in the self-optimization of chemical process steps.

In this work, 16 derivative-free optimization algorithm implementations were compared both theoretically and for a real-life application. All chosen algorithms will be explained briefly below, highlighting their mode of work. In order to compare the different algorithms systematically against each other, a benchmarking procedure was carried out, meaning that the implementations of each algorithm were run on a specified set of test functions or test problems. Here, the “no free lunch” theorem has to be kept in mind, which states that “for any algorithm, any elevated performance over one class of problems is offset by performance over another class”.^[27] This theorem highlights the fact that no algorithm can be considered the optimal choice for every kind of optimization problem. However, there can still be a method which consistently achieves better performance than its competitors for a fixed class of problems. The goal of this work is to find potentially suitable methods currently unused in the field of single-objective chemical process optimization, which can outperform existing algorithms.

Regarding the benchmarking process, all algorithms are compared first via the use of multiple common optimization problems. These problems feature functions with specific land-

scapes which can test the capabilities of optimization algorithms in different ways. The following general procedure of algorithm benchmarking is used in part from Arabas and Opara.^[21] Here, for continuous optimization problems an algorithm is tasked with finding an argument x_{opt} to optimize the set objective function $f : D \rightarrow R$, with $D \in R$, so that $x_{opt} = \arg \min_{x \in D} f(x)$. The bounds in which the algorithm can operate, the domain of the objective function D , is most often a hypercube with the form $D = [l_1, u_1] \times \dots \times [l_n, u_n]$, where n is the number of parameters (often called dimensions) and l_n, u_n are denoting the lower and upper bounds of each dimension. Each algorithm then gets a set number of function calls it can use (also called iterations), where an argument x_i provided by the algorithm can be used to receive the corresponding function output $f(x_i)$.

The algorithms are therefore to be ranked by reaching a specified vicinity to the optimum with a measured number of iterations, while being able to adjust to noise in the function output and random starting points.

After the benchmarking against common optimization problems, all algorithms are then compared for a Pd-catalyzed Suzuki-Miyaura cross-coupling reaction in continuous flow. Cross-coupling reactions, with the Suzuki coupling in general, have seen a rise in popularity in both batch and continuous flow API synthesis^[8,28] and organic chemistry in general.^[29] Additionally, a Suzuki coupling reaction has already been successfully used in a DoE-based automated flow setup, showing a reliable and consistent reproducibility of experiments.^[30]

For this reason, a fully automated flow setup was developed, capable of performing experiments and analyzing the reaction outcome in real-time, using in-line UV-Vis analysis, while being able to run autonomously for up to 40 h. This was combined with a Python-based in-house developed software, capable of hosting different optimization algorithms on one platform. Both the algorithm benchmarking, as well as the Suzuki coupling reaction optimization were carried out using this software.

The Suzuki coupling reaction was to be optimized using three parameters, temperature, residence time and substrate ratio as input parameters. The output parameter for which to optimize was the reaction yield, where a new metric, the progress P , was used to account for analysis drift between algorithm runs. Finally, a model function representing the reaction and its parameters, which was used to test different optimization scenarios without the need of using physical resources, was created.

Optimization Algorithms

In this work, the algorithms used for optimization are categorized into two groups, local and global optimization algorithms. A local solution is defined as an optimum which is smaller than all other feasible points in its close vicinity. A global solution, however, features the lowest function value of all feasible points in the parameter space.

All implemented algorithms share two characteristics. Firstly, as mentioned above, they all operate derivative-free, meaning they cannot infer any derivatives or general structure of the objective function, but instead can only work directly with the objective function output. Secondly, they are single-objective optimizers, meaning they seek to minimize only one objective function output. This is in contrast to multi-objective optimizers, which are trying to find a set of parameters which optimize multiple objectives to the point, where no one objective can be improved without simultaneously lowering the function output for a different objective.^[31–33]

Local Optimizers

Nelder-Mead

The Nelder-Mead Simplex method uses a convex polyhedral (or simplex) with $n+1$ vertices (n being the number of variables used for the optimization process). The algorithm starts by constructing this simplex, using either random or user-determined points. Afterwards, each iteration has the simplex replace the worst vertex by using a geometrical transformation on the center point of the simplex, such as reflection or contraction. Through this, the simplex is able to iteratively converge on a local optimum.^[17]

Subplex

The Subplex algorithm is a new implementation of the Nelder-Mead method, which shares most of its characteristics with its predecessor. However, to improve robustness and efficiency, especially in high-dimension problems, the Subplex (or sbplx) method breaks down the parameter search into a sequence of subspaces in which to optimize the objective function using the original Nelder-Mead method.^[34]

Powell's Method

A very early and simple optimization method which allows for the calculation of a local minimum of a function without using its derivatives. The optimization takes place through a bi-directional search along the axis of each variable used for the optimization process, by using either a golden-section search or the PRAXIS method (Principal Axis, also called "Brent's Method").^[35]

COBYLA (Constrained Optimization BY Linear Approximations)

By forming linear approximations to the objective function via interpolation at the vertices of a simplex (with again $n+1$ vertices, similar to the Nelder-Mead simplex), this algorithm iteratively proposes new vertices for this simplex. Due to the fact that linear approximations tend to be inefficient at a higher

number of variables, this algorithm is suited mostly for small dimension numbers of $n < 9$.^[36]

BOBYQA (Bound Optimization BY Quadratic Approximation)

Similar to COBYLA, this algorithm works by constructing an approximation to the objective function. But instead of linear approximations, BOBYQA employs a quadratic approximation from the black box function output instead. This quadratic model is subsequently used to solve for trust region subproblems.^[37]

Global Optimizers

SNOBFIT (Stable Noisy Optimization by Branch and FIT)

A global optimization algorithm for bound- and soft-constrained optimization problems, especially for noisy and expensive functions. The algorithm uses either a linear or quadratic stochastic model for its surrogate functions. In low dimensions, the algorithm has shown to converge to a global optimum with high efficiency, while being able to handle noise effectively, which is very advantageous for real-life applications such as chemical reactions.^[18]

Dual Annealing

Also called the generalized simulated annealing (GSA) algorithm, this method combines both the classical simulated annealing (CSA) and the fast simulated annealing (FSA) in a generalized form, while also incorporating an additional local search method. This leads to a faster convergence of GSA versus both CSA and FSA, while also being able to escape more easily from a local minimum to search for the true global optimum.^[38] This algorithm utilizes a distorted Cauchy-Lorentz distribution to reduce artificial temperature.^[39] The local search method for this version of Dual Annealing was the Nelder-Mead and BOBYQA method.

AMPGO (Adaptive Memory Programming for Global Optimization)

This optimization algorithm employs a tabu tunneling method combined with a pseudo-cut strategy to add a short-term memory structure to the optimization. These are then used in conjunction with a local optimizer, such as the Nelder-Mead algorithm mentioned above, while being able to stop the algorithm from getting trapped in a local optimum.^[40]

The AMPGO algorithm is fitted with five different local optimizers, namely the Nelder-Mead method, the COBYLA and BOBYQA methods, the PRAXIS method, and finally the Subplex method.

SHGO (Simplicial Homology Global Optimization)

The SHGO algorithm is based on building a complex on a hypersurface homeomorphic and approximating its homology groups to that of a complex on the objective function. The vertices of the complex are constructed by using sampling points over the whole search space (depending on existing constraints). It was developed to efficiently find all local (and the global) minima of the objective function, while still being efficient enough for expensive-to-evaluate functions.^[41] The SHGO method additionally allows for the use of local optimizers as a subroutine, together with a sampling method. For this, the COBYLA method and the Sobol method were chosen for each respective utility.

NOMAD (Nonlinear Optimization by Mesh Adaptive Direct Search)

The NOMAD method is the coded implementation of the Mesh Adaptive Direct Search (MADS) algorithm.^[42] The MADS method itself was evolved from the Generalized Pattern Search (GPS) and is able to optimize constrained black box optimization problems. MADS operates by producing series of meshes with different sizes over the parameter space. It then performs an adaptive search over the meshes to converge to a global optimum. The NOMAD implementation of MADS improves the method by introducing dynamic scaling, and different variants of MADS to be used for specific subproblems.^[43]

ImFil (Implicit Filtering)

The implicit filtering method is a combination of a projected Gauss-Newton algorithm and a deterministic grid-based search algorithm. While the Gauss-Newton algorithm normally needs a differentiable objective function to determine the gradients and the Jacobian matrix, the implicit filtering method instead approximates them with finite differences. The increments for these differences is varied during the optimization process.^[44]

Results and Discussion

Benchmark Functions

In the field of mathematical optimization, a range of classifiers can be used to describe the optimization problem at hand. These can include terms describing the nature of the entire domain of the function, such as convex/non-convex, smooth/non-smooth or continuous/non-continuous. It can also pertain to the area of interest that is considered for the optimization, such as bound or soft constraints, and global/local optimums.

Considering the typical landscape of chemical reaction optimization problems, the following qualifiers were chosen for the choice of benchmarking functions: All functions were bound-constrained, the function output in the domain smooth

and continuous with only continuous variables. Each function only produces one output parameter, making the process a single-objective optimization. Subsequently, the functions were divided into type A and type B functions, to allow for a more nuanced comparison of the optimization methods. Included into these types were noise in the function output as well as the existence of local and global optimums.

The types of functions utilized in this work are briefly outlined in Table 1 and discussed in the following sections. For additional information, see the Supporting Information.

As for the method of benchmark comparison, the criteria summarized in Table 2 were selected using best practice considerations.^[22]

For the optimization method comparison, the success rate (or clearance rate) was chosen to measure reliability, while the fixed-target solve time was used to measure solution quality. Since computational calculation time is negligible compared to running a real-life experiment, the solve time was defined as the number of objective function calls a method needs to reach a predefined vicinity to the global optimum. For this, we use the optimization error, which is defined as the difference between the best function value reached by a method and the global optimum of an optimization problem. Additionally, a limiting budget was used by giving each algorithm a defined number of function calls to use, until the optimization was forcefully terminated.

Type A Functions

The first type of test function, subsequently referred to as type A, contains only distinct global optimums. The function plot in Figure 1 shows an example of this parameters space landscape, while using a log-scaled axis for the function output value to better show the characteristic mentioned above. To additionally

Table 1. Test functions used for algorithm comparison divided into categories A and B.

Type A Function	Type B Function
Beale	Bukin function N.6
Booth	Eggholder
Himmelblau	Holder table function
Goldstein-Price	Schwefel function (2D and 3D)
Matyas	Six hump camel
Rosenbrock	Three hump camel
Sphere	
Styblinski-Tang	

Table 2. Example criteria of benchmark comparison.

Efficiency	Function calls CPU time Memory usage
Reliability	Success rate Constraint violations Fraction of global solutions found
Quality	Fixed-cost solution Fixed-target solve time Computational accuracy

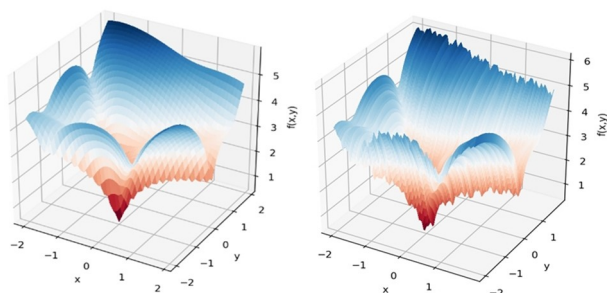


Figure 1. Goldstein-Price function plotted in three dimensions for x, y and $f(x, y)$ without noise (left) and with artificial noise of 20% (right).

test the optimization algorithms in their ability to reliably find the global optimum, all functions of type A were additionally remodeled to include artificial noise. Two modified versions of each function with $\pm 3\%$ and $\pm 5\%$ noise in their function output were utilized. As a demonstration example, the Goldstein-Price function with an additional noise of $\pm 20\%$, for better visibility, can also be seen in Figure 1.

The comprehensive results of the first comparison can be seen in Table 3. The individual values of all algorithms for each test function can be found in the Supporting Information.

Ranking is based on clearance rate, going from highest (green) to lowest (red). Conversely, the average number of iterations needed goes from lowest (green) to highest (red).

To maximize the objectivity of comparison, all algorithms were given 100 optimization runs per function, with a randomized starting point within the bounds for each run. A run was “cleared” in the event of an algorithm reaching an optimization error of below 0.01 for each respective test function. In this case, the iterations needed to get to this point were also recorded. An example of the results for the non-noisy type A functions with the Nelder-Mead method can be seen in Table 4. The method managed to clear 590 out of 800 runs, which translates to a clearance rate of 73.75%, while needing a total of 33,427 iterations which amounts to an average of 56.66 iterations per successful run.

For the type A functions, it can be seen that all algorithms manage to achieve a clearance rate above 60% in the non-noisy category, while needing between 25 and 151 iterations. The best performance is attained by the AMPGO algorithm, with the BOBYQA, PRAXIS and Subplex submethods ranking first to third, followed by the Subplex, dual annealing and SHGO methods. Comparing the overall results with the two most widely used methods for self-optimization setups, Nelder-Mead and SNOBFIT, shows a clear disadvantage of both methods compared to the AMPGO and dual annealing optimizers.

However, the introduction of artificial noise to the test functions changes the results, which shows that the AMPGO method has problems handling noisy functions. Especially the `ampgo_praxis` method is clearly unfit for noisy optimization problems, while the `ampgo_bobyqa` method changes from a

Table 3. Algorithm comparison regarding test functions of type A with and without artificial noise.

Algorithm	Clearance	Iterations	Algorithm	Clearance	Iterations
ampgo_bobyqa	1.0000	58.25	shgo	0.8644	149.95
ampgo_praxis	0.9050	39.30	dual_annealing_nelder	0.8444	91.54
ampgo_sbplx	0.8750	29.86	dual_annealing_bobyqa	0.8438	34.83
subplex	0.8750	29.86	ampgo_sbplx	0.7237	37.03
dual_annealing_bobyqa	0.8750	33.00	subplex	0.7200	35.08
shgo	0.8750	150.71	ampgo_bobyqa	0.6232	68.09
ampgo_cobyla	0.8313	59.11	ampgo_nelder	0.5626	65.32
dual_annealing_nelder	0.7500	90.83	sqnomad	0.5521	43.88
nelder	0.7375	56.66	ampgo_cobyla	0.5516	78.31
snobfit	0.7288	75.83	nelder	0.5395	42.60
sqnomad	0.7188	56.11	snobfit	0.5321	65.58
ampgo_nelder	0.6950	64.07	powell	0.4758	85.16
cobyla	0.6475	39.32	imfil	0.4642	58.25
bobyqa	0.6263	25.16	bobyqa	0.3916	19.73
powell	0.6025	70.46	cobyla	0.3668	29.27
imfil	0.6025	64.59	ampgo_praxis	0.1121	170.05
8 functions, no noise			16 functions, noisy		

Table 4. Results for 100 optimizations runs with each type A test function using the Nelder-Mead algorithm.

	Beale	Styblinski-Tang	Booth	Goldstein	Himmelblau	Matyas	Rosenbrock	Sphere
Cleared	65	21	100	63	100	100	41	100
Iterations	2116	1548	3418	2797	3591	2612	9330	8015

clearance of 100 % to only 62.32 %. Only the `ampgo_sbplx` and `Subplex` optimizers manage to stay in the top five of the ranking. The most promising methods for the noisy type A functions changed to the `SHGO`, `dual_annealing_nelder` and `dual_annealing_bobyqa` methods. However, this comes at the cost of needing a high number of average iterations for both `SHGO` and `dual_annealing_nelder`, while only the `dual_annealing_bobyqa` optimizer stays at a reasonable number. Again, the `Nelder-Mead` and `SNOBFIT` methods, overall, rank in the lower tiers. Also, considering the average number of iterations needed for a successful optimization, it can be said that `ampgo_sbplx` and `dual_annealing_bobyqa` show the most promising results for the type A function, both with and without noise. Considering that most chemical processes, like reactions, behave similarly to a type A function, a strong case can be made for both mentioned optimization methods to replace the `Nelder-Mead` and `SNOBFIT` optimizer that are currently in use.

On a sidenote, two type A functions, namely the `Matyas` and `Sphere` functions, feature a symmetrically shaped landscape with the optimum in the center of the parameter space. To see if any methods profit from these specific conditions, the bounds were adjusted to move the optimum out of the center point. Results for the comparison of 10 optimization runs for each function with a symmetric versus an asymmetric optimum are shown in Table 5. The data shows that the `dual_annealing_nelder` and `SHGO` algorithms respond well to symmetric landscapes; however, these types of situations would be exceedingly rare for real-life applications. For the `SHGO` method, this behavior is due to the sampling method needed for its operation. The sampling is done in a sequence which always operates in the same way and only depends on the bounds of the optimization problem. In this particular case, the optimum for both the `Sphere` and the `Matyas` problem lies symmetrically in the center of the bounds and is therefore automatically

included as the second point in each run. On the other hand, only the `COBYLA` method, and as a result also the `ampgo_cobyla` method, seem to perform better with asymmetric bounds, going from a clearance rate of 78.33 % and 76.67 %, to 81.67 % and 93.33 %, respectively.

Type B Functions

The second test function type, denoted as function type B, comprises functions which feature multiple local optimums with one or more, but not all of them, being a global optimum. These were specifically used to compare the advantages of global optimization algorithms against the local optimizers.

While a global optimizer operates in a way that ensures it does not accidentally converge on only a local minimum, a local optimizer typically does not feature such a characteristic. This can lead to a diminished performance in type B problems for local optimizers, depending on the starting points' proximity to a local or global optimum.

With the `Schwefel` function being used in both its 2D and 3D variant, the total number of type B test functions for type B was seven. All methods were given 50 runs per problem with randomized starting points within bounds. Due to the more demanding landscapes of the functions, the optimization error that had to be reached by an algorithm to finish the optimization successfully, was varied between 0.001 and 5.0. Also, no artificial noise was incorporated for the same reason. The exact values of each algorithm for each function can be found in the Supporting Information.

As can be seen in Table 6, all local optimizers, except for the `Nelder-Mead` and `Subplex` method, with a clearance rate of 57.14 % each, performed poorly compared to the global optimizers.

Table 5. Algorithm comparison regarding specific test functions (including the noisy variants, for a total of six functions) with symmetric optimums.

Algorithm	Clearance	Iterations	Algorithm	Clearance	Iterations
shgo	1.0000	1.00	snobfit	1.0000	28.90
dual_annealing_nelder	1.0000	65.52	ampgo_bobyqa	1.0000	56.15
dual_annealing_bobyqa	1.0000	27.59	ampgo_nelder	0.9833	52.20
snobfit	0.9833	29.71	dual_annealing_bobyqa	0.9833	23.55
nelder	0.9667	51.81	nelder	0.9667	53.55
sqnomad	0.9667	101.55	sqnomad	0.9667	96.31
ampgo_bobyqa	0.9667	60.83	ampgo_sbplx	0.9667	59.52
ampgo_sbplx	0.9667	35.67	powell	0.9333	83.95
bobyqa	0.9167	19.05	ampgo_cobyla	0.9333	88.07
ampgo_nelder	0.9167	67.82	subplex	0.9167	61.53
subplex	0.9167	57.55	shgo	0.9167	148.62
powell	0.9000	91.48	bobyqa	0.8333	18.22
cobyla	0.7833	39.17	cobyla	0.8167	41.43
ampgo_cobyla	0.7667	72.72	dual_annealing_nelder	0.8167	113.84
imfil	0.6167	52.03	imfil	0.6833	52.93
ampgo_praxis	0.4000	33.54	ampgo_praxis	0.4000	58.13
6 functions, symmetric bounds			6 functions, asymmetric bounds		

Table 6. Evaluation of algorithm performance via test functions of type B, featuring both local and global optimums.

Algorithm	Clearance	Iterations
dual_annealing_nelder	0.7143	92.80
dual_annealing_bobyqa	0.7143	75.20
ampgo_nelder	0.6158	121.83
ampgo_bobyqa	0.5940	101.49
ampgo_sbplx	0.5739	62.12
nelder	0.5714	43.25
subplex	0.5714	61.50
snobfit	0.5493	116.06
sqnomad	0.5394	89.09
shgo	0.4286	148.00
ampgo_cobyqa	0.3484	107.20
ampgo_praxis	0.3008	170.88
powell	0.2857	37.00
imfil	0.2611	63.11
bobyqa	0.2537	34.38
cobyqa	0.1429	16.00

7 functions, global and local optimums

In the category of global optimizers, the dual annealing method takes first place, with a clearance rate of 71.43% for both the Nelder-Mead and BOBYQA subroutine. This is followed by the AMPGO method, where the Nelder-Mead, BOBYQA and Subplex submethods all performed similarly well with a clearance rate between 61.58% and 57.39%. Taking also the average number of iterations into account, the dual_annealing_bobyqa method presents the best tradeoff between clearance rate and average iterations, with 71.43% and 75.20, respectively.

Experimental Comparison

For the comparison of the local and global optimizers via the continuous flow Suzuki coupling, some adjustments had to be made to allow for an objective representation of the reaction data over all optimization runs. This was due to the fact that during the in-line UV analysis step, a fluctuation between different optimization runs was observed, leading to reaction yield discrepancies for similar experiment points during optimizations with different algorithms. However, fluctuations within each specific run were monitored to be only minimal and within expectations (as can be seen in Table 7, where pairs of similar points during runs are compared). Therefore, a new system of measurement, namely the progress *P*, was derived in Equation (1) to compare different algorithms. This metric is not directly reliant on in-line analysis data, but instead normalizes the input parameters of the function calls to track the optimization progress.

Table 7. Comparison of experiment point pairs during individual optimization runs.

Exp. point	T [°C]	Residence time [min]	Substrate Ratio [–]	Yield [%]
Nelder 40	99.98	19.13	1.99	72.59
Nelder 38	99.96	19.45	1.99	72.71
Annealing 26	88.18	27.09	1.52	68.28
Annealing 10	87.71	26.98	1.54	68.00
ImFil 16	100	22.91	1.07	69.38
ImFil 13	100	23.46	1.13	69.52

$$P = \frac{\left(\frac{\text{Temperature}-40}{60} + \frac{\text{Residence time}-2}{28} + \frac{\text{Substrate ratio}-1}{1} \right)}{3} \quad (1)$$

Seeing as, on the one hand, the optimum for this reaction is situated in one corner of the reaction space, with the highest temperature, residence time and substrate ratio, and, on the other hand, all algorithms starting point was situated on the opposite corner, the progress *P* describes the proximity of the algorithm to the optimum. Here, *P*=0.00 denotes the starting point with *T*=40 °C, *τ*=2 min, *R*=1, while *P*=1.00 would be the corner optimum, with *T*=100 °C, *τ*=30 min, *R*=2.

As for the experimental runs, all algorithms were given a budget of 40 experiments, with the starting point [40.0, 2.0, 1.0] for [temperature, residence time, substrate ratio]. Then, each algorithms' current best progress was plotted against ongoing experiments. Due to the fact that experimental runs are time- and resource-intensive, not all algorithms were used in the following comparisons. However, a surrogate function, which was modeled after the Suzuki coupling reaction, was used with every method.

Comparison of Local Search Algorithms

As can be seen in Figure 2, which plots the progress *P* of each experiment over the whole optimization process, the Subplex method clearly outperforms the original Nelder-Mead algorithm. It needs fewer iterations to reach the global optimum of *P*=1.00, while all other local methods only reach to about *P*=0.90, and it does so in fewer iterations than the second and third best method, respectively.

Both the BOBYQA and COBYLA methods struggle with the optimization, possibly due to the random noise in the experiment, which leads them to get stuck in a local minimum. The Powell method, while needing some ramp up time due to its operation style, manages to even outperform the Nelder-Mead method; however, it needed the full 40 experiments.

Comparison of Global Search Algorithms

As can be seen in Figure 3, two optimizers managed to reach the global optimum with *P*=1.00, while two more methods managed to reach a *P* value of above 0.90. By far the fastest converging method with only 14 experiments needed was the ampgo_sbplx method, followed by sqnomad, ampgo_nelder

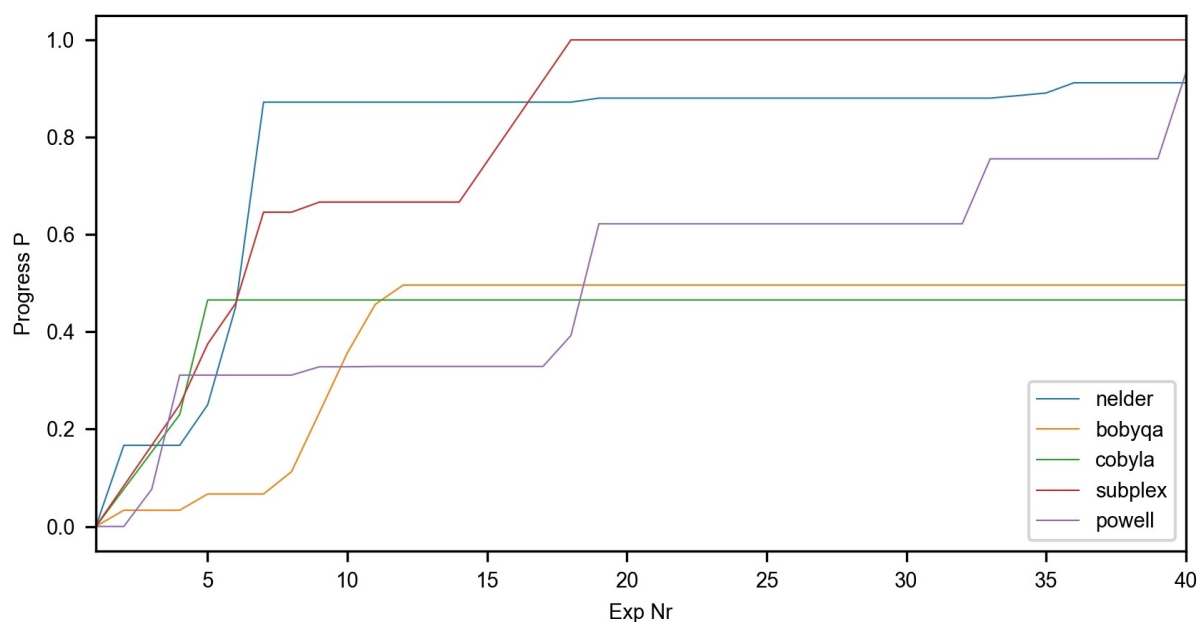


Figure 2. Experimental data comparison of local optimization algorithms. The plot shows the current best progress of each solver over the optimization run.

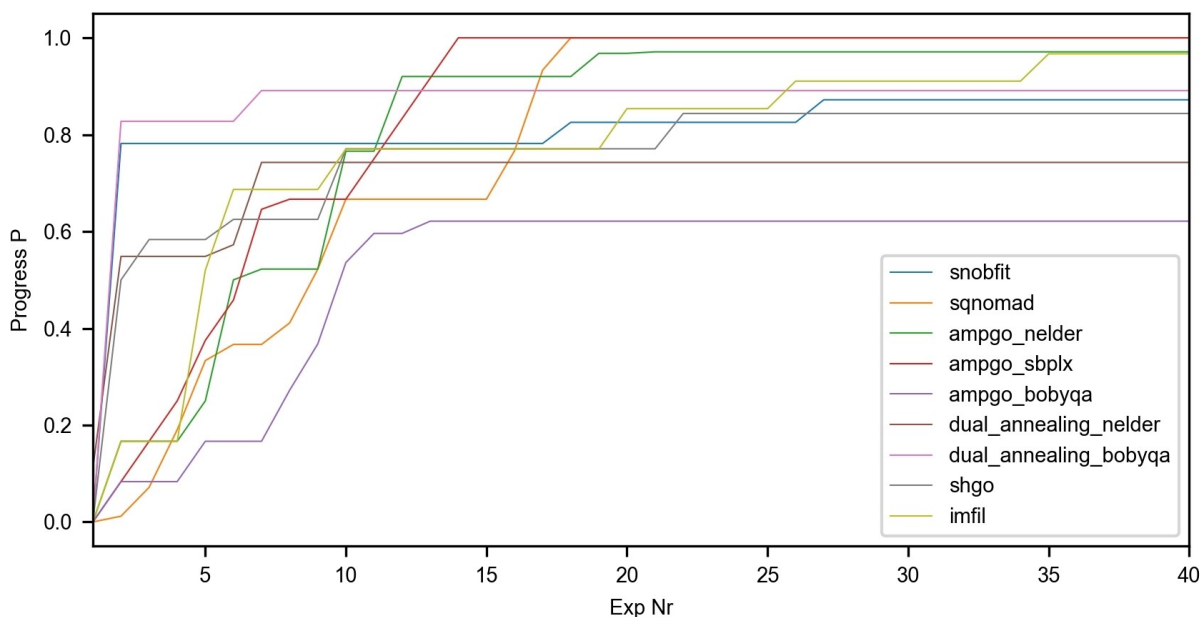


Figure 3. Experimental data comparison of global optimization algorithms. The plot shows the current best progress of each solver over the optimization run.

and the ImFil method. While the dual annealing methods performed very well for both type A and type B test problems, only the dual_annealing_bobyqa method managed to at least achieve a progress of nearly 0.90, admittedly with a low number of needed iterations. The SNOBFIT method also performed in a robust way but was easily outclassed by multiple other methods.

Comparison Using the Suzuki Model Function

Based on results of the Suzuki coupling data, a model function f_{final} was defined as shown in Equation (2) to allow for additional settings in the optimization process without the need for physical resources:

$$f_{\text{pre}}(x, y, z) = 2^{-\frac{10}{x-30}} + 2^{\frac{15}{y}} + 2^{-\frac{5}{z}}$$

$$f_{\text{final}} = \frac{f_{\text{pre}}(x, y, z)}{3 - P} \quad (2)$$

With the model function, all algorithms were now given 100 randomly generated starting points within bounds and a budget of 300 iterations for each run. A run counted as a clear if the algorithms best point was within ± 0.01 of the global optimum. The results are shown in Table 8.

The biggest discrepancies between the Suzuki coupling and model functions results were found with the dual annealing method, where the dual_annealing_nelder optimizer took first place with 100% clearance and the lowest number of iterations. The Subplex and ampgo_sbplx methods in as close second and third, while dual_annealing_bobyqa and SNOBFIT managed to achieve better results than during their experimental runs. For the dual_annealing_bobyqa method, this might be due to the higher number of average iterations needed, which, with 68, is higher than the allotted 40 flow experiments.

Comprehensive Results

Looking at the results as a whole establishes the two main optimizers with the best overall performance, the AMPGO and the dual annealing method. For these two, the ampgo_sbplx and the dual_annealing_bobyqa show the most promise.

The ampgo_sbplx managed to converge very quickly and with high accuracy on most type A problems, both with and without noise (87%/30 iterations and 72%/37 iterations). It successfully cleared 100% of the model function runs with around 18 iterations, and optimized the Suzuki coupling reaction to $P=1.00$ in the least number of iterations of all algorithms. The only caveat are the harder type B problems, where only 57%/62 iterations were achieved.

Table 8. Evaluation of algorithm performance via the model function for the Suzuki coupling in three dimensions.

Algorithm	Clearance	Iterations
subplex	1.0000	17.71
ampgo_sbplx	1.0000	18.33
dual_annealing_nelder	1.0000	14.24
dual_annealing_bobyqa	0.9967	68.24
snobfit	0.9933	24.89
powell	0.9767	32.57
imfil	0.9667	15.36
sqnomad	0.9400	34.42
ampgo_nelder	0.8633	66.12
ampgo_bobyqa	0.8100	137.00
ampgo_cobyla	0.8033	70.59
cobyla	0.7767	8.64
shgo	0.5933	182.20
nelder	0.3467	34.68
bobyqa	0.1267	12.79
ampgo_praxis	0.0000	0.00

3 functions, Suzuki coupling model

The dual_annealing_bobyqa method on the other hand, displayed arguably better results for the type A (87%/33 iterations noise-free type A, 84%/35 iterations noisy type A) and clearly better results for the type B problems (71%/75 iterations). It managed to converge very quickly to $P=0.89$ in the Suzuki coupling reaction, and cleared 99.7% of the model function runs, albeit with an average of 68 iterations.

Both Nelder-Mead and SNOBFIT managed to achieve similar clearance rates for type A and B problems, with the Nelder-Mead method converging in less average iterations, while the SNOBFIT clearly outperformed Nelder-Mead for the Suzuki model function (Nelder-Mead: 74%/57 iteration noise-free type A, 54%/43 iterations noisy type A, 57%/43 iterations type B, 35%/35 iterations model function; SNOBFIT: 73%/76 iteration noise-free type A, 53%/66 iterations noisy type A, 55%/116 iterations type B, 99%/25 iterations model function). Both methods achieved respectable results in the Suzuki reaction optimization, with $P=0.91$ in 36 experiments and $P=0.87$ in 27 experiments, respectively.

Conclusion

In this work, we have compared a number of derivative-free algorithms against each other, especially the often-used Nelder-Mead and SNOBFIT methods. All algorithms were benchmarked against different types of optimization problems with randomized starting points, a Suzuki coupling reaction in continuous flow, and a model function based on the reaction. For this, an automated reaction setup, in conjunction with a Python-based GUI for hosting and recording the optimization algorithms and their performance was developed.

The optimizers that performed the best on average were the ampgo_sbplx and dual_annealing_bobyqa methods, which outperformed the Nelder-Mead and SNOBFIT algorithms in each of the cases in regard to clearance rate. The average number of iterations needed for a successful optimization was lower as well. One reason for these findings might be the combination of both local and global solvers for the AMPGO and dual annealing methods. While the local solvers such as Subplex and BOBYQA offer fast convergence for smaller regions, the global optimizers in the combined method make sure that the local solver does not get stuck in either a local optimum of the objective function or a noise-induced misdirection of the solver. One other possible reason pertains to the number of function evaluations for each run. Compared to optimization in computer sciences where the available budget can number in the thousands, investigating real-life applications with time and resource constraints gives the convergence speed a much higher priority. This leaves solvers such as SHGO (151 iterations) and SNOBFIT (76 iterations) at a disadvantage and favors fast methods such as BOBYQA (25 iterations) and Subplex (30 iterations), taking the average iterations for non-noisy type A functions in Table 3 as a point of reference.

Outlook

To ensure a more objective comparison, all algorithm implementations were used as-is, with the only changes being the sub-methods for some of the global optimizer (see Computational Methods in the Supporting Information). However, fine-tuning the operating parameters of the different implementations could potentially improve performance, which would prove interesting for upcoming studies. Similarly, this work only contains problems with two and three dimensions, leaving high-dimension problems for future research. Finally, applying the methods to more diverse chemical optimization problems could give further understanding of their robustness.

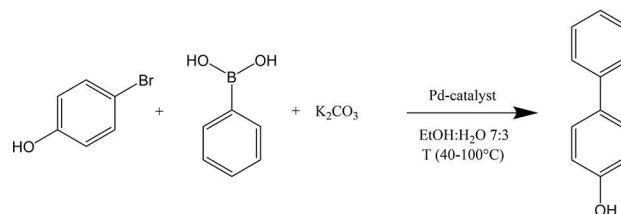
Experimental Section

Model Reaction – Suzuki-Miyaura Cross-Coupling

To compare the optimization algorithms' real-life application, a simple Suzuki-Miyaura cross coupling reaction was chosen, which can be seen in Scheme 2. This reaction has already been extensively studied and the reaction kinetic and mechanism is well enough understood to postulate the optimal reaction conditions in a given reaction space. For the optimization variables, temperature, reaction time or residence time, and the ratio of the coupling partners were chosen (See Table 9). The experiments were carried out in an automated continuous flow setup, which can be seen in Scheme 1. The substrates are mixed in a passive mixing element before entering a heated stainless-steel coil which is used as a plug flow reactor (PFR). After the reaction reaches steady-state, an automated

Table 9. Variables for self-optimization with lower and upper limits.

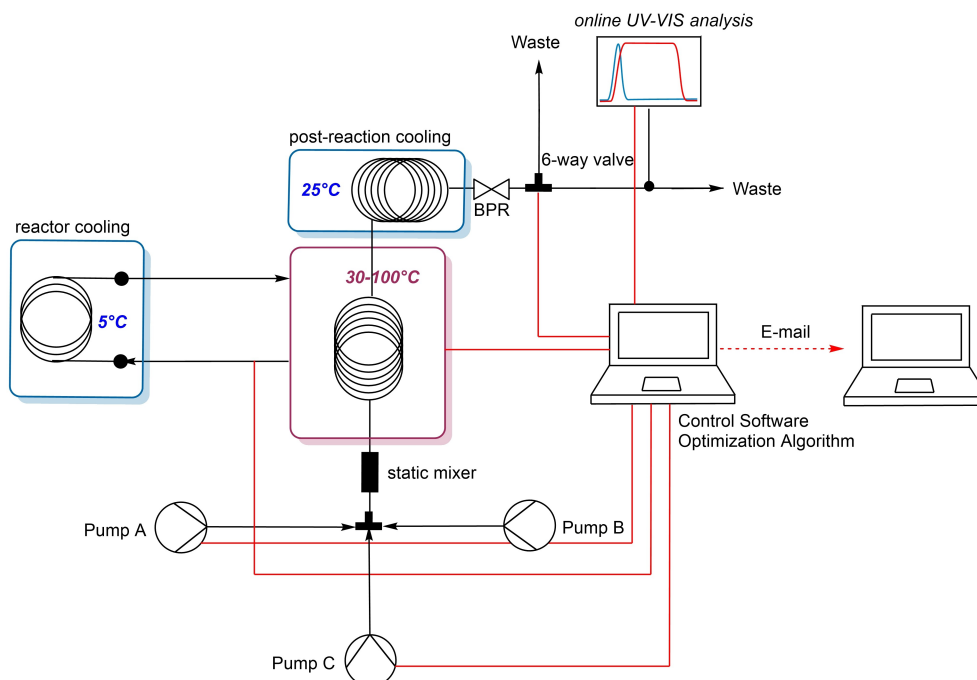
Variable	Lower limit	Upper limit
Temperature T [°C]	40	100
Residence time τ [min]	2	30
Substrate ratio R [–]	1	2



Scheme 2. Reaction scheme for the Suzuki-Miyaura cross-coupling used for self-optimization. Concentrations of starting solutions were 70 mM for each reagent, but were varied over the course of the experiments according to the substrate ratio variable.

valve pumps the reaction solution through a UV-Vis flow cell, where the spectrum of the solution is recorded, and the reaction yield is calculated.

The Suzuki coupling was carried out with 4-bromophenol (TCI Chemicals, 98%) and phenylboronic acid (TCI Chemicals, 97%) catalyzed by a PEPPSI IPr palladium catalyst (Umicore, 98%, Sigma Aldrich) and potassium carbonate (ACS reagent, >99%, Sigma Aldrich). The solvent for the reaction was EtOH:H₂O = 7:3 v/v (ethanol Abs., >99.8%, Sigma Aldrich).



Scheme 1. Automated continuous flow setup. Reaction solutions were pumped using Azura P4.1S HPLC pumps. The solution streams were combined with a Valco tee piece and subsequently mixed using a passive mixing element, custom made by additive manufacturing. The tubular coil reactor was made of 1/32" ID stainless-steel capillaries with a total volume of 5.5 mL. Other capillaries in the setup were again 1/32" ID stainless-steel, or 1/32" ID PTFE. For uniform pressure distribution during the reaction, an IDEX 75 psi (5.1 bar) backpressure regulator was used. For switching between waste and UV-Vis analysis, an Azura VU 4.1, combined with an Elegoo UNO R3 microcontroller, was used. Analysis of the reaction yield was carried out using an AvaLight DS-DUV Spectrometer with an AvaSpec-ULS2048 detector.

Optimization Workflow

To automate the needed laboratory equipment, such as pumps, automated valve, UV-Vis station and reaction heating/cooling, a custom-made Python-based script was developed. The script features a GUI and an interface to support the optimization algorithms imported from different libraries. A table of the algorithm implementations can be found in the Supporting Information.

Conflict of Interest

The authors declare no conflict of interest.

Data Availability Statement

The data that support the findings of this study are available in the supplementary material of this article.

Keywords: algorithms · automation · cross-coupling · flow chemistry · self-optimization

- [1] O. W. Gooding, *Curr. Opin. Chem. Biol.* **2004**, *8*, 297–304.
- [2] P. M. Murray, F. Bellany, L. Benhamou, D. K. Bučar, A. B. Tabor, T. D. Sheppard, *Org. Biomol. Chem.* **2016**, *14*, 2373–2384.
- [3] C. J. Taylor, A. Baker, M. R. Chapman, W. R. Reynolds, K. E. Jolley, G. Clemens, G. E. Smith, A. J. Blacker, T. W. Chamberlain, S. D. R. Christie, B. A. Taylor, R. A. Bourne, *J. Flow Chem.* **2021**, 75–86.
- [4] M. I. Jeraal, N. Holmes, G. R. Akien, R. A. Bourne, *Tetrahedron* **2018**, *74*, 3158–3164.
- [5] V. Fath, N. Kockmann, J. Otto, T. Röder, *React. Chem. Eng.* **2020**, *5*, 1281–1299.
- [6] M. B. Plutschack, B. Pieber, K. Gilmore, P. H. Seeberger, *Chem. Rev.* **2017**, *117*, 11796–11893.
- [7] S. G. Newman, K. F. Jensen, *Green Chem.* **2013**, *15*, 1456–1472.
- [8] M. Baumann, I. R. Baxendale, *Beilstein J. Org. Chem.* **2015**, *11*, 1194–1219.
- [9] A. C. Bédard, A. Adamo, K. C. Aroh, M. G. Russell, A. A. Bedermann, J. Torosian, B. Yue, K. F. Jensen, T. F. Jamison, *Science* **2018**, *361*, 1220–1225.
- [10] N. Holmes, G. R. Akien, A. J. Blacker, R. L. Woodward, R. E. Meadows, R. A. Bourne, *React. Chem. Eng.* **2016**, *1*, 366–371.
- [11] R. A. Sheldon, *ACS Sustainable Chem. Eng.* **2018**, *6*, 32–48.
- [12] A. D. Clayton, J. A. Manson, C. J. Taylor, T. W. Chamberlain, B. A. Taylor, G. Clemens, R. A. Bourne, *React. Chem. Eng.* **2019**, *4*, 1545–1554.
- [13] J. P. McMullen, K. F. Jensen, *Org. Process Res. Dev.* **2010**, *14*, 1169–1176.
- [14] D. E. Fitzpatrick, C. Battilocchio, S. V. Ley, *Org. Process Res. Dev.* **2016**, *20*, 386–394.
- [15] V. Sans, L. Porwol, V. Dragone, L. Cronin, *Chem. Sci.* **2015**, *6*, 1258–1264.
- [16] R. A. Skilton, A. J. Parrott, M. W. George, M. Poliakoff, R. A. Bourne, *Appl. Spectrosc.* **2013**, *67*, 1127–1131.
- [17] J. A. Nelder, R. Mead, *Comput. J.* **1965**, *7*, 308–313.
- [18] W. Huyer, A. Neumaier, *ACM Trans. Math. Softw.* **2008**, *35*, 1–25.
- [19] D. Cortés-Borda, K. V. Kotonova, C. Jamet, M. E. Trusova, F. Zammattio, C. Truchet, M. Rodríguez-Zubiri, F. X. Felpin, *Org. Process Res. Dev.* **2016**, *20*, 1979–1987.
- [20] L. M. Rios, N. V. Sahinidis, *J. Glob. Optim.* **2013**, *56*, 1247–1293.
- [21] J. Arabas, K. Opara, *J. Telecommun. Inf. Technol.* **2011**, 73–80.
- [22] V. Beiranvand, W. Hare, Y. Lucet, *Opt. Eng.* **2017**, *18*, 815–848.
- [23] K. C. Felton, J. G. Rittig, A. A. Lapkin, *Chem. Methods* **2021**, *1*, 116–122.
- [24] A. D. Clayton, A. M. Schweidtmann, G. Clemens, J. A. Manson, C. J. Taylor, C. G. Niño, T. W. Chamberlain, N. Kapur, A. J. Blacker, A. A. Lapkin, R. A. Bourne, *Chem. Eng. J.* **2020**, *384*, 123340.
- [25] Z. Zhou, X. Li, R. N. Zare, *ACS Cent. Sci.* **2017**, *3*, 1337–1344.
- [26] F. Häse, M. Aldeghi, R. J. Hickman, L. M. Roch, M. Christensen, E. Liles, J. E. Hein, A. Aspuru-Guzik, *Mach. Learn.: Sci. Technol.* **2021**, *2*, 035021.
- [27] D. H. Wolpert, W. G. Macready, *IEEE Trans. Evol. Comput.* **1997**, *1*, 67–82.
- [28] D. G. Brown, J. Boström, *J. Med. Chem.* **2016**, *59*, 4443–4458.
- [29] X. F. Wu, P. Anbarasan, H. Neumann, M. Beller, *Angew. Chem. Int. Ed.* **2010**, *49*, 9047–9050; *Angew. Chem.* **2010**, *122*, 9231–9234.
- [30] B. J. Reizman, Y. M. Wang, S. L. Buchwald, K. F. Jensen, *React. Chem. Eng.* **2016**, *1*, 658–666.
- [31] N. Peremzhney, E. Hines, A. Lapkin, C. Connaughton, *Eng. Optim.* **2014**, *46*, 1593–1607.
- [32] E. Bradford, A. M. Schweidtmann, A. Lapkin, *J. Glob. Optim.* **2018**, *71*, 407–438.
- [33] A. M. Schweidtmann, A. D. Clayton, N. Holmes, E. Bradford, R. A. Bourne, A. A. Lapkin, *Chem. Eng. J.* **2018**, *352*, 277–282.
- [34] T. H. Rowan, *PhD Thesis, Univ. Texas* **1990**, 218.
- [35] M. J. D. Powell, *Comput. J.* **1964**, *7*, 39.
- [36] M. J. D. Powell, *Adv. Optim. Numer. Anal.* **1994**, 51–67.
- [37] M. Powell, *NA Rep. NA2009/06* **2009**, 39.
- [38] Y. Xiang, D. Y. Sun, W. Fan, X. G. Gong, *Phys. Lett. A* **1997**, *233*, 216–220.
- [39] Y. Xiang, X. G. Gong, *Phys. Rev. E* **2000**, *62*, 4473–4476.
- [40] L. Lasdon, A. Duarte, F. Glover, M. Laguna, R. Martí, *Comput. Oper. Res.* **2010**, *37*, 1500–1509.
- [41] S. C. Endres, C. Sandrock, W. W. Focke, *J. Glob. Optim.* **2018**, *72*, 181–217.
- [42] C. Audet, A. L. Custódio, J. E. Dennis, *SIAM J. Optim.* **2007**, *18*, 1501–1503.
- [43] C. Audet, S. Le Digabel, V. Rochon Montplaisir, C. Tribes, *NOMAD 4, Softw. Optim. blackbox Probl.* **2021**, 1–22.
- [44] C. Kelley, *ImFil, Implicit Filter. Implementation, North Carolina State Univ.* **2011**, 1–59.

Manuscript received: October 29, 2021

Version of record online: February 15, 2022