SFU

Optimization Algorithms Benchmark Report

Author: Connor Chai

Product Design Optimization Laboratory

Simon Fraser University

August 22, 2025

SFU

Table of Contents

Introduction

This is a technical document detailing the benchmark test running various optimization algorithm implementations with open-source frameworks on a suite of 15 problems. This suite included both single-objective (SO) and multi-objective (MO) problems, constrained and unconstrained problems with set budgets. The nine SO functions range from 10 to 60 dimensions and include one problem with eight constraints. The maximum dimensionality of the multi-objective functions is 30. This suite also includes a discrete variable MO optimization problem and a constrained MO optimization problem with 6 constraints. All of the objective and constraint functions in the benchmark problems are assumed to be black box functions that are usually evaluated via simulations. As a result, none of the algorithms implemented in this test uses gradient information from the original functions. The list of problems can be found in Table A1, and the benchmark problems are defined in Table A2. The 15 problems included in this report are as follows: Rosenbrock 10d, 50d, Michalewicz 10d, 30d, and 60d, Schubert 10d, 30d, and 60d, G7, Zürich den Tiadel (ZDT) 1, ZDT2, ZDT3, ZDT6, Osyczka Kundu (referenced as OSY in this report), and the geartrain design. These problems are considered difficult problems and are widely used for benchmarking optimizers.

The primary purpose of this benchmark test is to compare open-source frameworks, namely Dakota [1] and Python [2], with OASIS.AI [3], a commercial tool. Applications for seeking out the performance of these open-source alternatives include the implementation of these tools for educational purposes and for aiding future research.

Python is a popular programming language and is the foundation for a myriad of well-known libraries for scientific computing. Dakota is a lesser-known tool, developed by Sandia National Laboratories in the United States for running and designing engineering experiments, building surrogate models, and optimization.

The code used to run these benchmark tests is comprised of a host of Python scripts paired with Dakota ".in" files, when applicable. The design of the Python scripts allows the user to replicate the results by automating the process of running 10 instances of each solver on a given benchmark problem, post-processing the data, and graphing the results in separate subfolders. The README in the repository provides further documentation on the code.

Solver Information

A host of algorithms are readily available to be implemented for performing optimization on black box functions. Successful optimization of a function results in finding a value close to the known minima of the objective function. A given optimization algorithm has its own abstract definition

and can be implemented in different libraries with varying results. Each implementation features a variety of unique parameter settings, stopping criteria, and initial search points (when applicable). Because each algorithm has these various parameters, Dual Annealing (DA) implementation in SciPy [4], for example, will potentially yield varying results when compared to the DA approach in Nevergrad [5]. However, in this report, to increase the interpretability of the data by limiting the cluttering of an abundance of solvers, Dividing Rectangles (DIRECT) is the only algorithm left with more than one implementation. Other algorithms, if there are multiple implementations, will have the best performing implementation in the group for comparison.

OASIS.AI was run independently from Empower Operations Corp [3], and the raw results data can be found in the repository.

Unless otherwise specified in the README or later sections of the report, the hyperparameters of each algorithm were set to the defaults according to what is listed in the Dakota keyword reference user manual [6]. From this information, the population size was set to 50 for all evolution-based algorithms, and the budget for training samples for surrogate-based algorithms was set to 10% of the allocated budget. Although better results may be obtained with more complex methods and the fine-tuning of parameters, those are outside the scope of this project. The chosen algorithms and their implementations are outlined below.

Single-Objective Solvers

Beginning with (SO) solvers, SciPy's implementations of Dual Annealing (DA), Differential Evolution (DE), and Dividing Rectangles (DIRECT) methods were used, as SciPy is a well-known library in Python. It is used in other benchmark frameworks like COCO [7]. Although the Basin Hopping and Simplicial Homology Global Optimization (SHGO) methods were readily available in the SciPy library, they were excluded in the tests, as they are, respectively, a hybrid optimization method and a method geared towards lower-dimensional problems.

The NgIohTuned meta-optimizer was implemented from the Nevergrad library, created by Facebook AI Research. Instead of being a standard optimization solver implementing a specific algorithm, Nevergrad internally configures its settings and picks a specific optimization strategy from inside the Nevergrad framework based on the properties of the problem. Unfortunately, there is no guaranteed approach for determining the exact solver strategy that NgIohTuned uses.

Genetic Algorithm (GA) was implemented from the pymoo library [8]. Mesh Adaptive Direct Search was also implemented, from the OMADS library [9]. DIRECT is a deterministic algorithm,

meaning its output remains the same for different run instances. The rest of the solvers are stochastic, meaning that, in the absence of a set seed, the results will vary for each run.

The solvers that were used in Dakota include efficient_global (Efficient Global Optimization (EGO)), genie_direct (an implementation of DIRECT), soga (single objective GA), coliny_ea (an implementation of Evolutionary Algorithm), and surrogate_based_global (an implementation of Surrogate Based Optimization). The descriptions for each of these methods in Dakota can be found in the keyword reference under the "method" tab [6]. Several other Dakota solvers were removed after initial tests. These are discussed in the "Methodology" section under "Single-Objective Problems: Unconstrained".

Table 1 summarizes the solvers implemented in this test. The parenthesized names are abbreviations or acronyms used to reference the solvers. Boldfaced algorithm names indicate an ability to handle nonlinear constraints.

Table 1. List of solvers used for single-objective optimization. Those that can handle constraints are bold-faced.

Dakota	**efficient_global (EGO)**	**genie_direct (GENIE)**	**soga (GA)**	**coliny_ea (EA)**	**surrogate_based_global (SBGO)**	
Python	**Mesh Adaptive Search (OMADS)**	**Differential Evolution (SciPy-DE)**	**Genetic Algorithm (pymoo-GA)**	Dual Annealing (SciPy-DA)	NgIohTuned (Nevergrad)	DIRECT (SciPy-DIRECT)

Multi-Objective Solvers

The MO solvers include Non-dominated Sorting Genetic Algorithm 2 (NSGA-II), Multi-Objective Evolutionary Algorithm based on Decomposition (MOEA/D), and Strength Pareto Evolutionary Algorithm 2 (SPEA2) were implemented from pymoo. Additionally, the MADS implementation from OMADS was used as it had the functionality for MO optimization in addition to SO optimization. Both MO Dakota solvers were used in this test: coliny_ea, the implementation of Evolutionary Algorithm, and MOGA, Dakota's implementation of Multi-Objective Genetic Algorithm. The list of MO optimization solvers used in this report are listed in Table 2.

Table 2. List of solvers used for multi-objective optimization. Those that can handle constraints are bold-faced.

Dakota	**moga (MOGA)**	**coliny_ea (EA)**		
Python	**NSGA-II (pymoo)**	MOEA/D (pymoo)	**SPEA2 (pymoo)**	**MADS (OMADS)**

Single-Objective Problems: Unconstrained

Methodology

As stated earlier, each solver was run 10 times for each problem, and results are graphed with a convergence plot and a box plot. The data for the running minima for each instance was aggregated and processed to create a file with the average running minima for each solver. The best minima for each instance were congregated into a single file for each solver.

Additionally, an initial test of the solvers was run on Rosenbrock 10, and from this, several solvers were excluded from future comparisons (their data from Rosenbrock 10d can be found in Figures A1-A2, Tables A3-A4). Details are as follows.

Ax-BO, the simplified surrogate-based optimization from BoTorch [10], based on PyTorch was originally included, but despite being given a 30% larger budget on Rosenbrock 10d, each of its instances took 30 minutes to converge to a value three orders of magnitude larger than the known minima, with even two failed runs.

SMT's [11] Efficient Global Optimization (EGO) was also initially used but was removed due to values being three to four orders of magnitude larger than the known minimum, and an average runtime of 30 minutes. The inaccuracy and long training time could be explained by a deeper understanding of the algorithm, which inherently does not handle high dimensions very well. In a paper published documenting the EXPO Benchmark, they say that "Bayesian optimization with Gaussian processes is typically applied to problems with less than 10 variables" [12].

Similarly, Dakota's EGO was removed due to weak convergence and significant runtime. With this implementation, each solver run on Rosenbrock 50, Michalewicz 30, and Schubert 30 took around 12 hours. For Schubert 30, resulting minima displayed large deviation from the known minima of -2.4x10^24, ranging from -1e5 to -1e7.

These three methods were also removed from further comparison: ncsu_direct (referenced as NCSU), coliny_direct (referenced as COLINY), and genie_opt_darts (referenced as OPT-Darts). The last one is a modified version of DIRECT, and the first two are two different implementations of DIRECT, each yielding similar or worse results than GENIE, the DIRECT implementation was kept in the future tests.

The convergence chart, boxplot, time data, and solver summaries from the initial test on Roesnbrock 10d, including the discontinued solvers, can be found in Figures A1-A2 and Tables A3-A4, and the raw data from each solver is in the repository.

Results

The results from running the solvers on the SO unconstrained problems are displayed with a convergence graph, in Figures 1-8, and boxplots, in Figures 9-16. The convergence graphs display the convergence of the objective function values of each solver as the number of function evaluations (NFE) increases, plotting the running minima, averaged across all 10 solver instances. The boxplot graphs display the range of best minima for each solver, attained across all 10 of its instances. A boxplot is also called a box-and-whisker plot. The line inside the box shows the median value, and the box spans from the 1st quartile to the 3rd quartile of the data. The ends of the whiskers show the smallest and largest data points that are not outliers. The numerical summaries of data used to build these boxplots, including minimum, maximum, and average minimum values for each solver for each problem, can be found in Tables A6-A15.

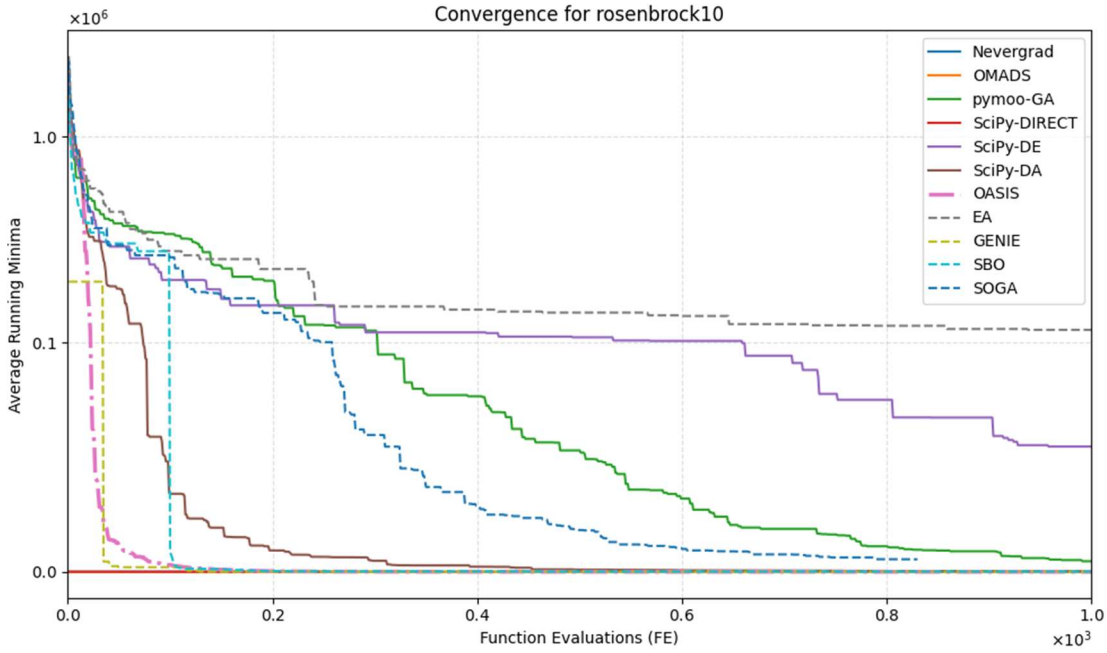


Figure 1. Rosenbrock 10d convergence plot from solver data up to 1000 function evaluations (Python, Dakota, OASIS.AI)

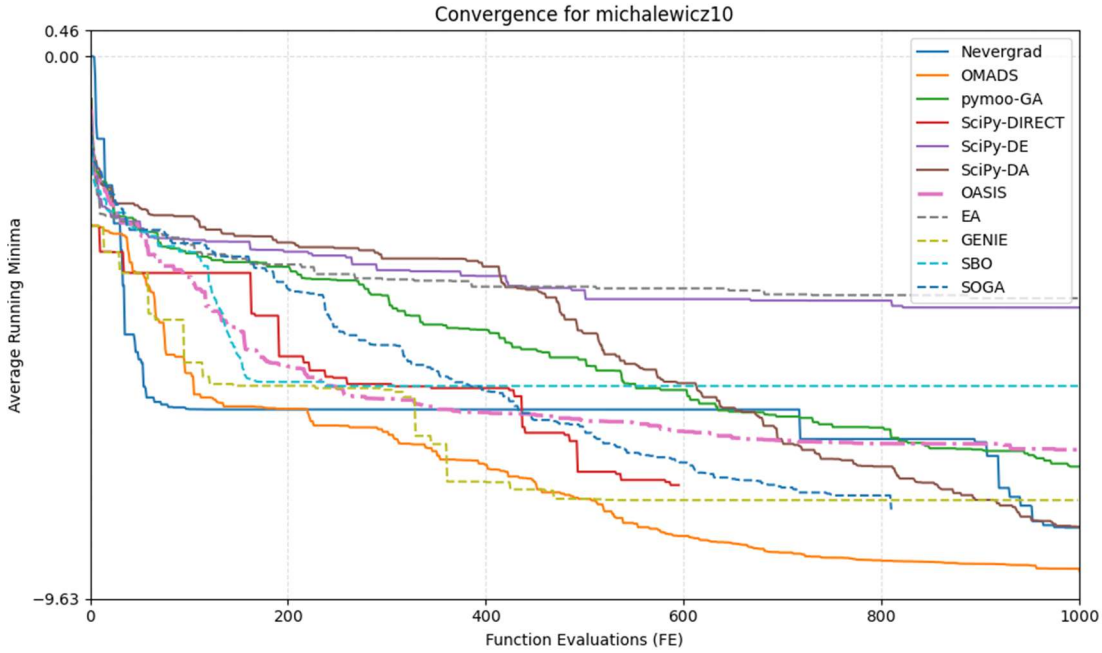


Figure 2. Michalewicz 10d convergence plot from solver data up to 1000 function evaluations (Python, Dakota, OASIS.AI)

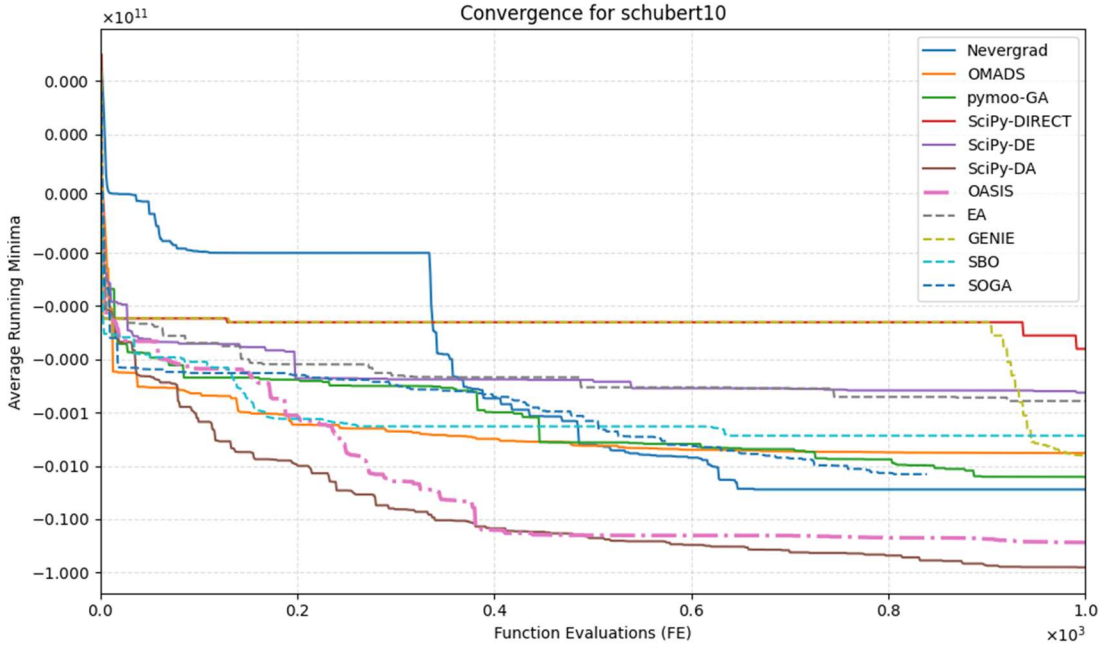


Figure 3. Schubert 10d convergence plot from solver data up to 1000 function evaluations (Python, Dakota, OASIS.AI)

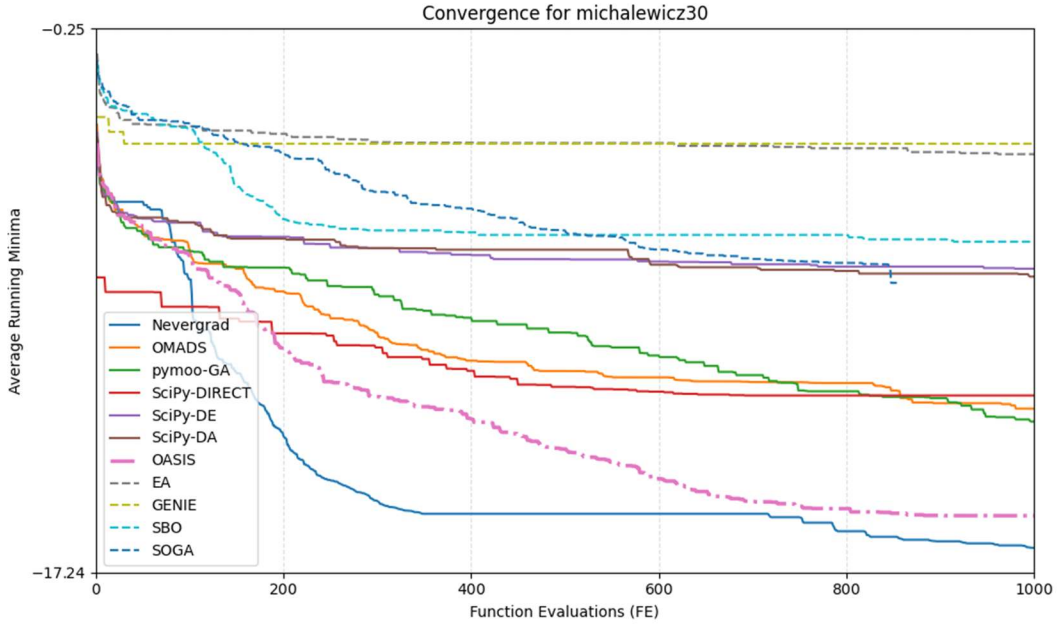


Figure 4. Michalewicz 30d convergence plot from solver data up to 1000 function evaluations (Python, Dakota, OASIS.AI)

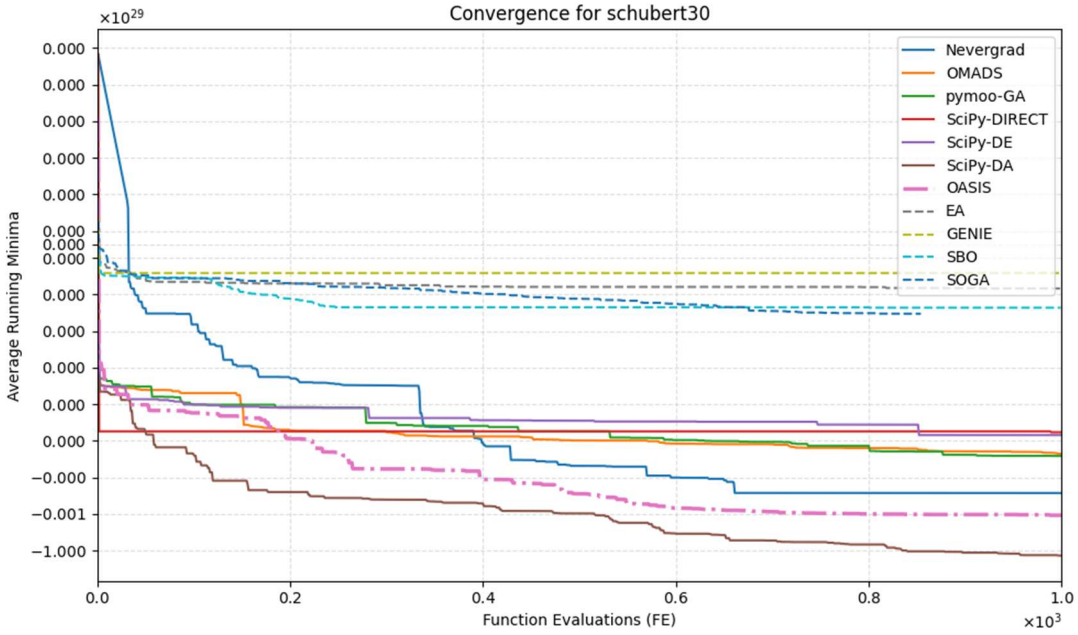


Figure 5. Schubert 30d convergence plot from solver data up to 1000 function evaluations (Python, Dakota, OASIS.AI)

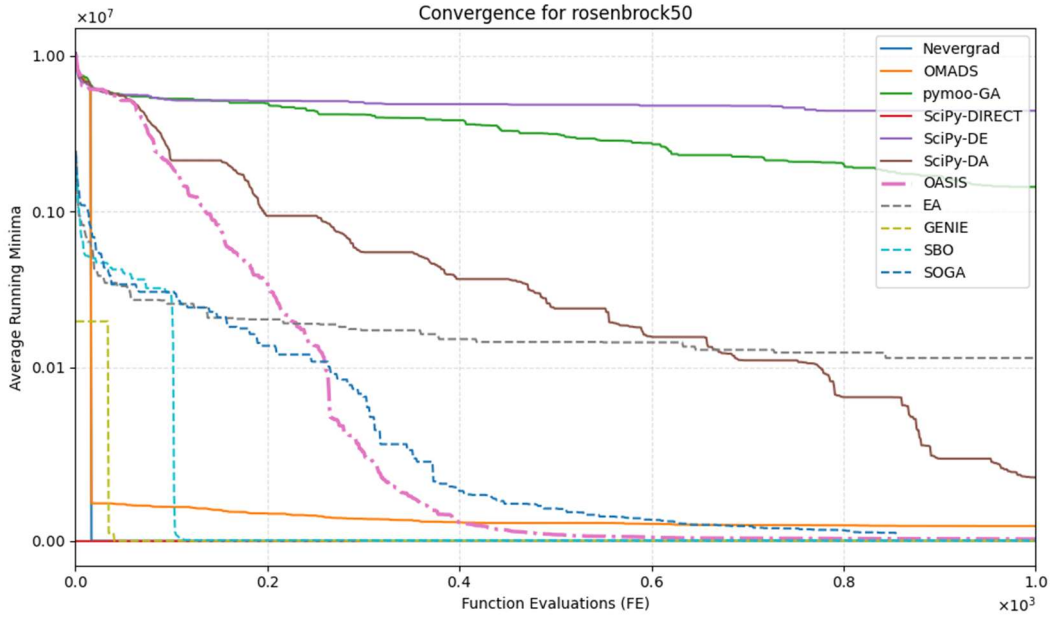


Figure 6. Rosenbrock 50d convergence plot from solver data up to 1000 function evaluations (Python, Dakota, OASIS.AI)

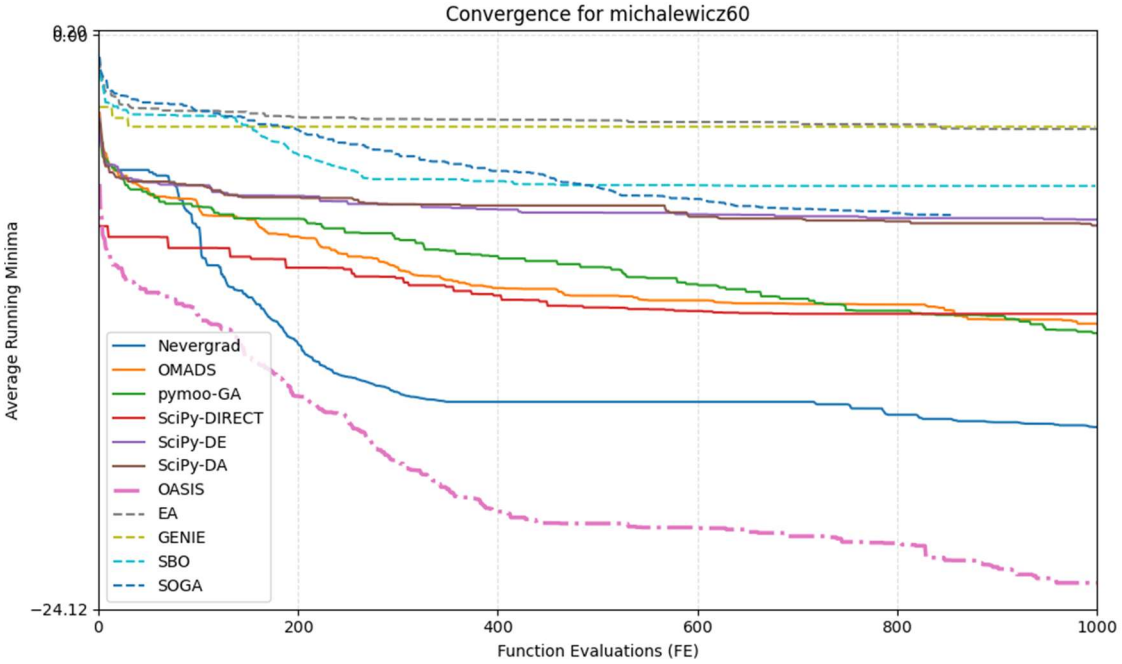


Figure 7. Michalewicz 60d convergence plot from solver data up to 1000 function evaluations (Python, Dakota, OASIS.AI)

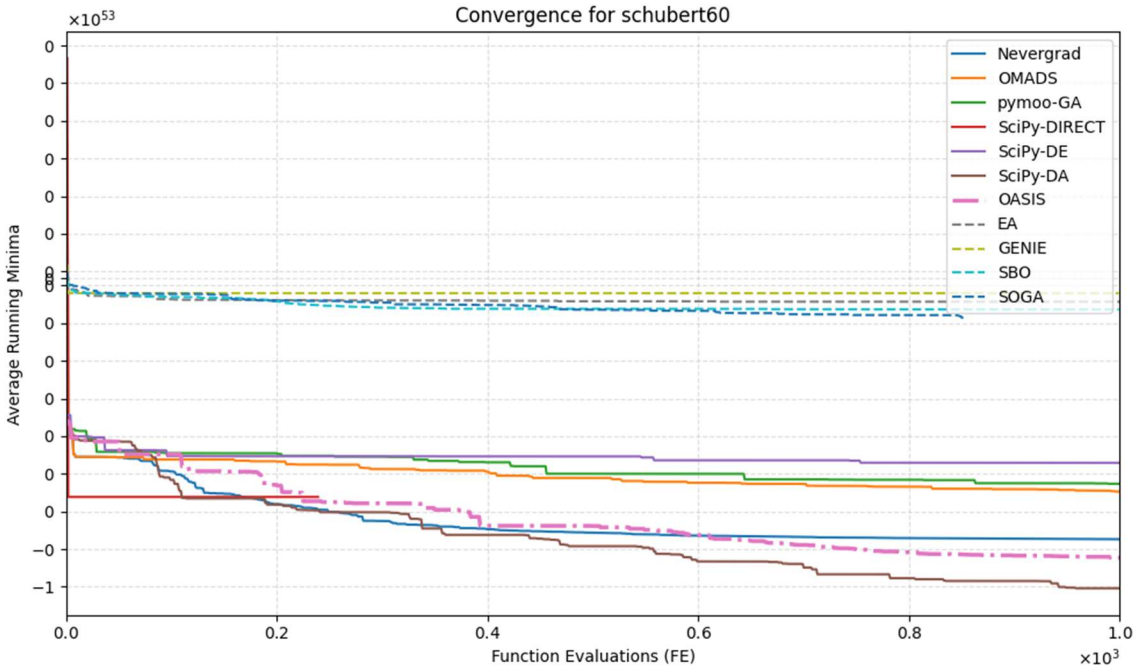


Figure 8. Schubert 60d convergence plot from solver data up to 1000 function evaluations (Python, Dakota, OASIS.AI)

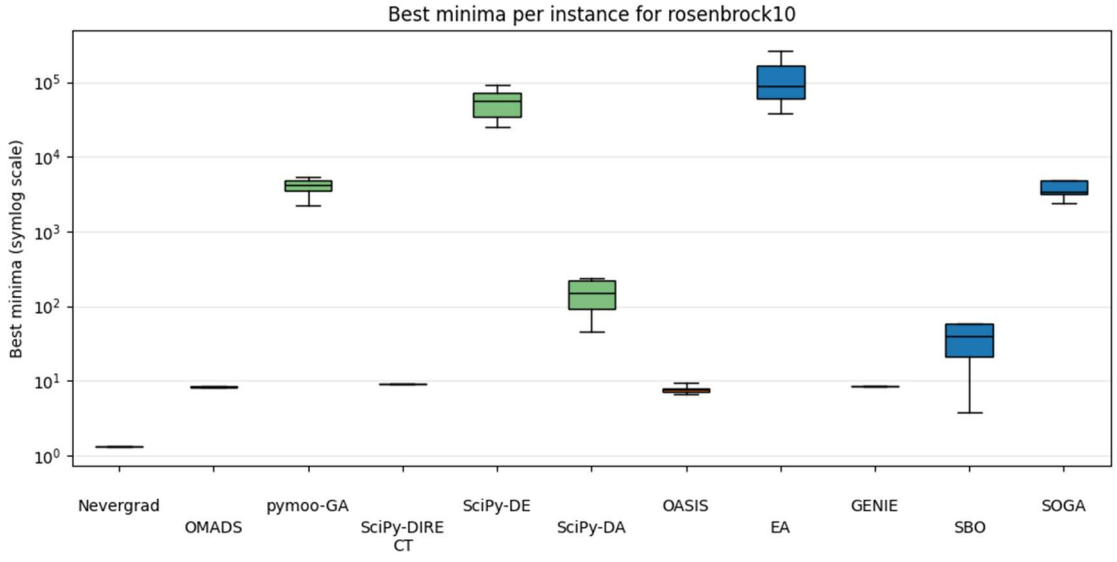


Figure 9. Rosenbrock 10d boxplot of best minima per instance from each solver (Python, Dakota, OASIS.AI)

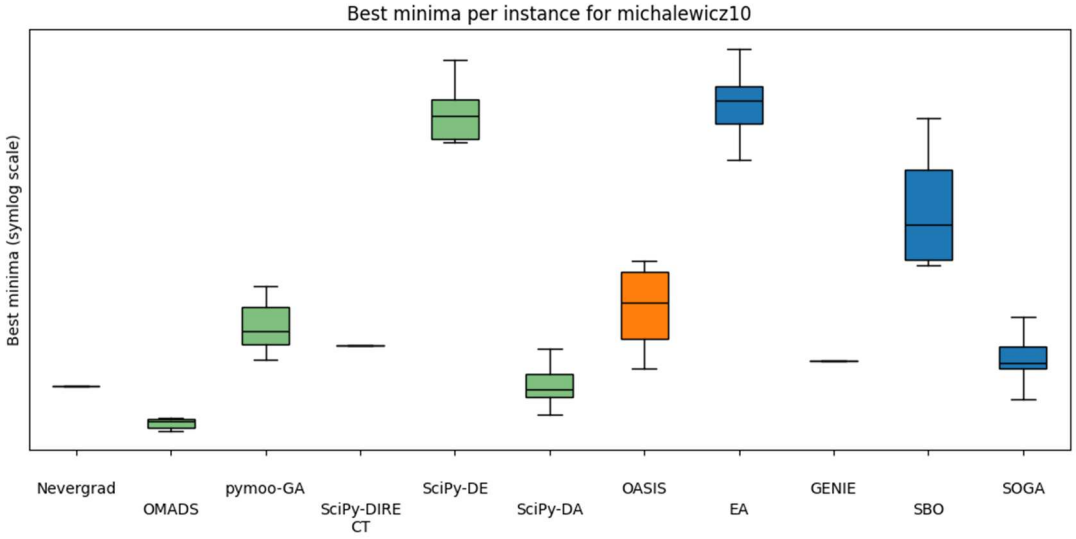


Figure 10. Michalewicz 10d boxplot of best minima per instance from each solver (Python, Dakota, OASIS.AI)

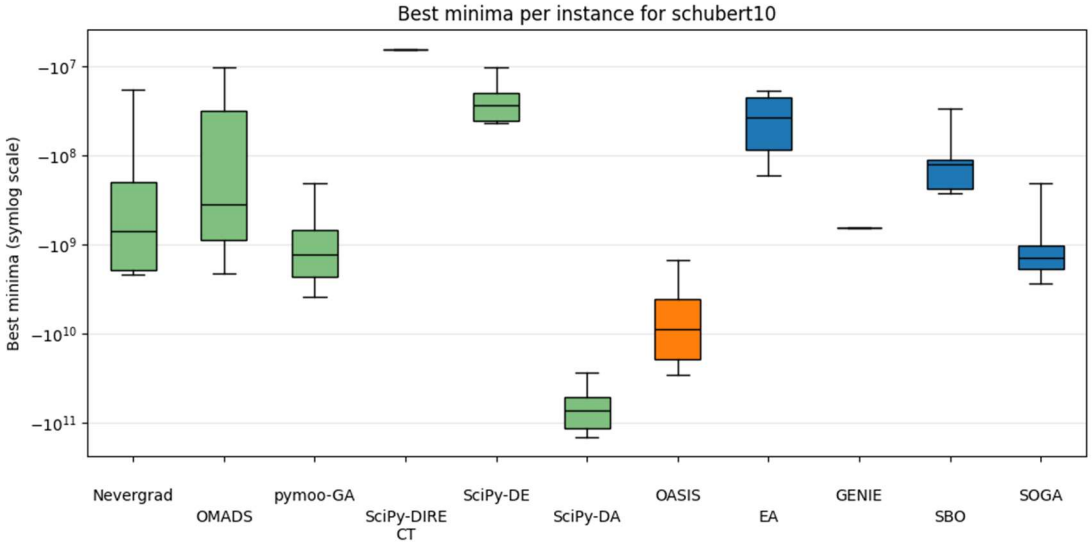


Figure 11. Schubert 10d boxplot of best minima per instance from each solver (Python, Dakota, OASIS.AI)

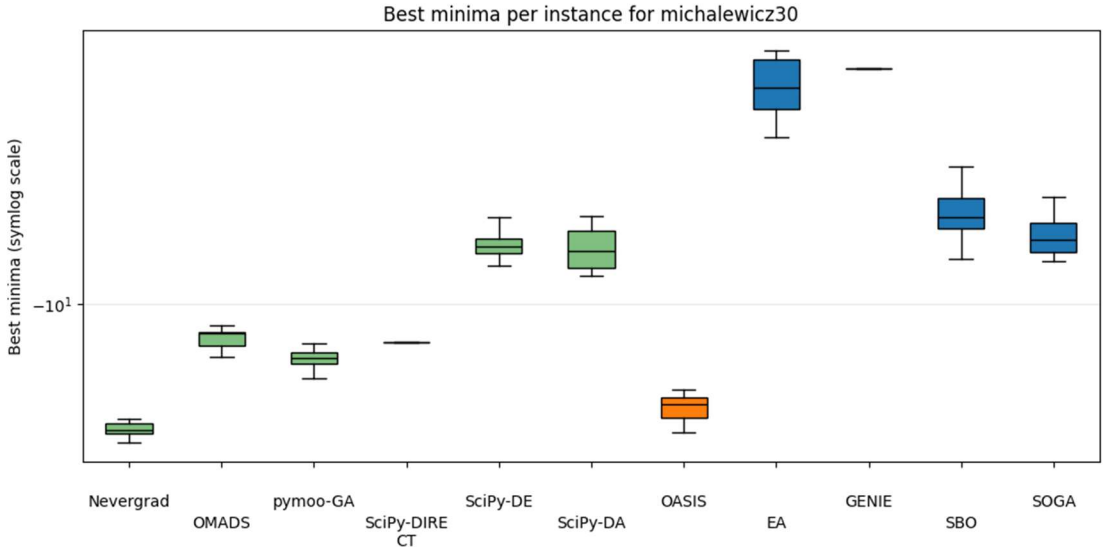


Figure 12. Michalewicz 30d boxplot of best minima per instance from each solver (Python, Dakota, OASIS.AI)

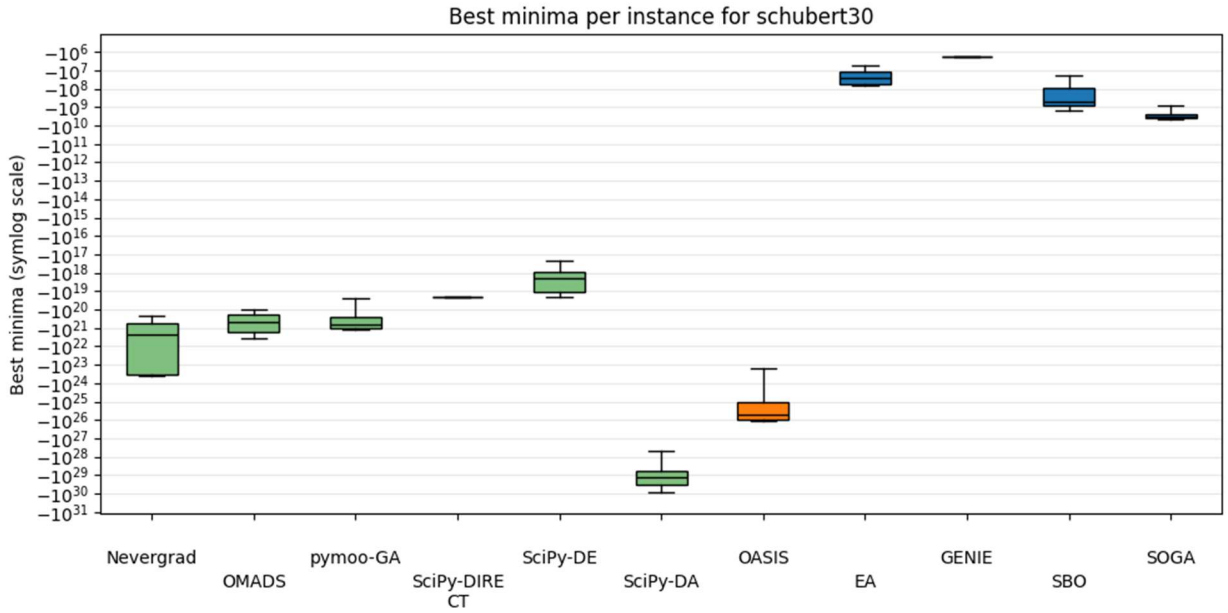


Figure 13. Schubert 30d boxplot of best minima per instance from each solver (Python, Dakota, OASIS.AI)

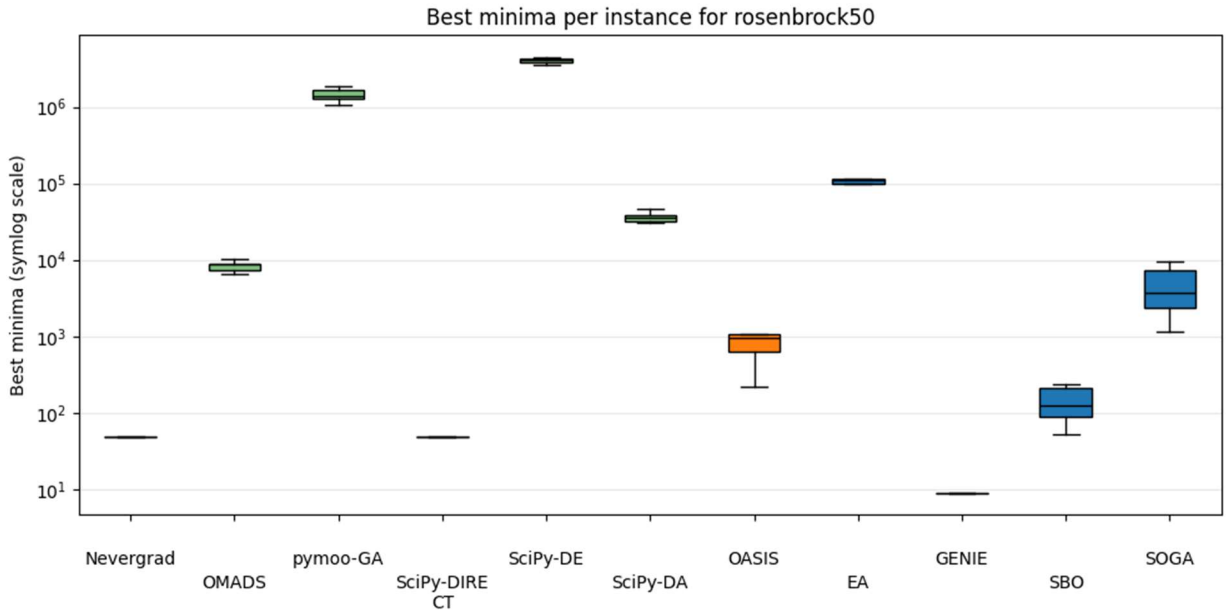


Figure 14. Rosenbrock 50d boxplot of best minima per instance from each solver (Python, Dakota, OASIS.AI)

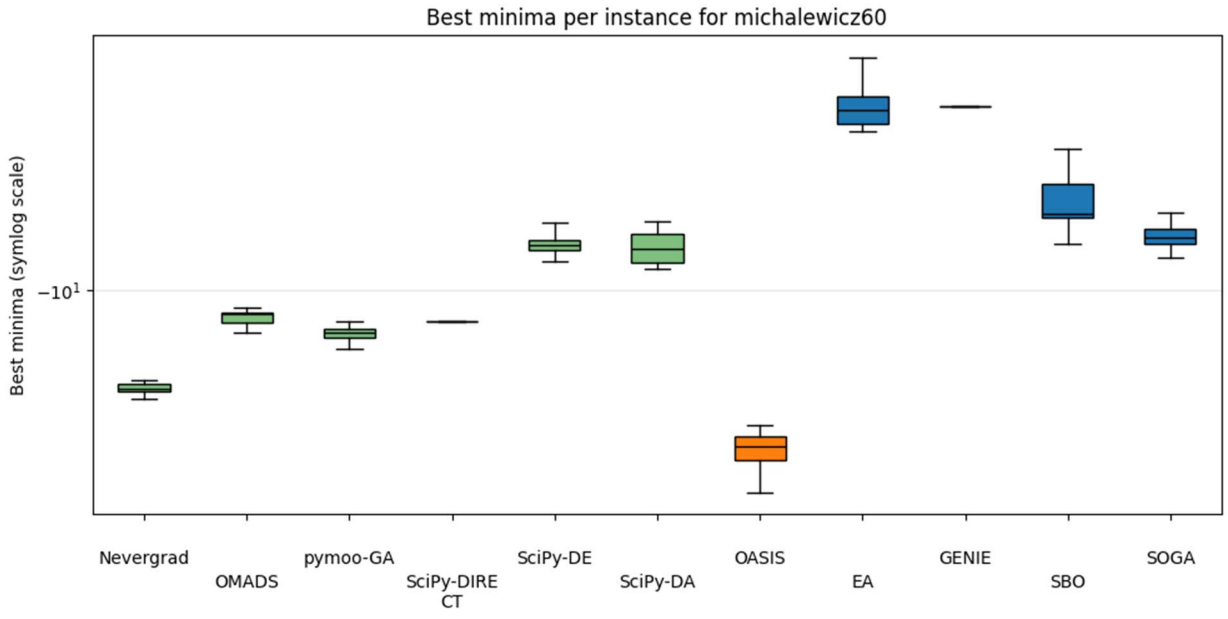


Figure 15. Michalewicz 60d boxplot of best minima per instance from each solver (Python, Dakota, OASIS.AI)

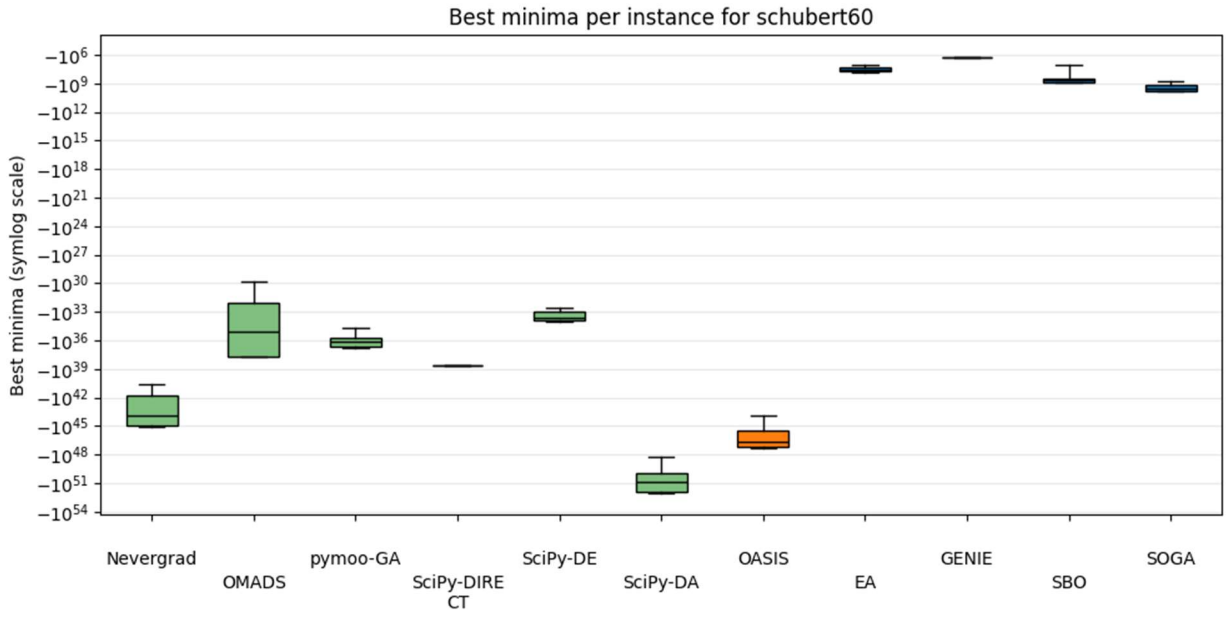


Figure 16. Schubert 60d boxplot of best minima per instance from each solver (Python, Dakota, OASIS.AI)

Discussion

Starting with the convergence plots, it is observed that SciPy-DIRECT and Nevergrad display peculiar behaviour for Rosenbrock 10d and 50d, as their initial starting points are already very near the known minima. The known minima of these problems are 0, and the initial guesses of those two solvers is near 9.0. The cause for this behaviour is most likely explained by the solvers starting their search with the center point $[0, 0, \ldots, 0]^T$ in the bound constraints provided, which is (-10, 10), for all input variables. The value of the Rosenbrock functions at the center point $[0, 0, \ldots, 0]^T$ is exactly 9.0.

From the boxplot for Rosenbrock 10d we observe that GENIE, OMADS, SciPy-DIRECT and Nevergrad are attaining similar or better values than OASIS.AI, bearing in mind these algorithms may both start from the center of the design space, which gains unfair advantage against OASIS.AI and other algorithms. Such advantages continue to Michalewicz 10d and Rosenbrock 50d. Then one can see Nevergrad, SciPy-Direct, and GENIE performed much poorer than OASIS.AI for Schubert 10d, 30d, 60d, as well as for Michalewicz 30d and 60d (except Nevergrad gets similar performance for Michalewicz 30d).

For Michalewicz 10d, several methods outperform OASIS.AI, including OMADS, SOGA, SciPy-DA and pymoo-GA. But OMADS, SOGA, and pymoo-GA lost to OASIS.AI in all of the other cases. Similarly, SBO lost to OASIS.AI in all cases except for Rosenbrock 50d. The only strong contender

is SciPy-DA, which continues to do better than OASIS.AI for Schubert 10d, 30d, and 60d. However, SciPy-DA was significantly behind OASIS.AI for Rosenbrock 10d and 50d, as well as Michalewicz 30d and 60d.

The performance of chosen Dakota solvers appears to be worse than the selected Python algorithms. Dakota's solvers generally perform poorly on the >10 dimensional problems. Although it found a lower minimum on average than OASIS.AI on Rosenbrock 50d, Dakota's SBO took 12 hours per instance to run. Three Python solvers were also able to outperform Dakota's SBO. Nevergrad, OMADS, pymoo-GA, and SciPy-DA significantly outperform SBO in every Michalewicz and Schubert problem, and to some extent outperform SBO on the Rosenbrock problems as well. The time data for SO unconstrained solvers on all problems can be found in Tables A8-A15, and the specifications of the various computers used to run the problem can be found in Figure A3.

Also, on an interesting note when running SBO on Michalewicz 30d, SBO ran for six instances before failing on the 7th, when the algorithm abruptly stopped. This is likely due to an internal algorithm issue within Dakota, resulting from the values generated for that specific seed in Dakota. The ".out" file indicates a sudden termination, and the ".dat" file shows -0 values reappearing. Instances 8-10 were run manually, without the .out files, and the results for the graphing files were calculated without the 7th instance.

Summary for Unconstrained Single-Objective Problems

OASIS.AI demonstrates strong performance and robustness across all test cases. Except for Michalewicz 60, OASIS.AI is outperformed by at least one solver in each of the eight problems. There is, unfortunately, no other algorithm that shows similar strength and consistency in solution quality. This means that overall, the open-source codes are still lagging behind OASIS.AI.

Among all the contenders, SciPy-DA seems to be the top-performing solver for unconstrained single-objective problems. Its runtime is also significantly lower than that of Dakota solvers, specifically Dakota's SBO, which takes at least three hours to run each of its 10 instances.

Single-Objective Problems: Constrained (G7)

Methodology

Due to the time constraint, G7 was the only selected constrained SO problem in the test suite. Benchmarking solvers on this problem followed the same process as for the SO problems, except some solvers were excluded due to their inability to handle constraints, as listed in the last section.

A tolerance of 1e-6 was implemented for all solvers, allowing some flexibility for solvers to satisfy the constraints. The coliny implementation DIRECT was implemented instead of GENIE, as its performance proved to be similar during the initial test for Rosenbrock 10. However, most solvers were unable to find a single feasible solution across all 10 instances, and the findings are discussed later below the following Results section.

Implementation of constraints in Dakota was guided by the constraint handling video resource in the Dakota library [13].

Results

A modified convergence chart is displayed below in Figure 17, along with Table 3, which documents the first feasible solution for each solver instance. Each of the three solvers is graphed with a different line type. The regions where no infeasible solutions exist are depicted with a red colouring for each trend. Once the first feasible solution is attained, across any instance of the solver, a solver's trendline is coloured green, and the resulting value from that point onwards is the average of the feasible running minima across all 10 instances. For example, if there are 10 instances, with eight instances returning feasible values, the average value is computed for those 8 instances and compared with the average running minimum up to that point. If it is found to be lower, it becomes the new running minimum. As a reminder, DIRECT is a deterministic algorithm, so its curve was created with the data from just one instance, and it produced a handful of feasible solutions.

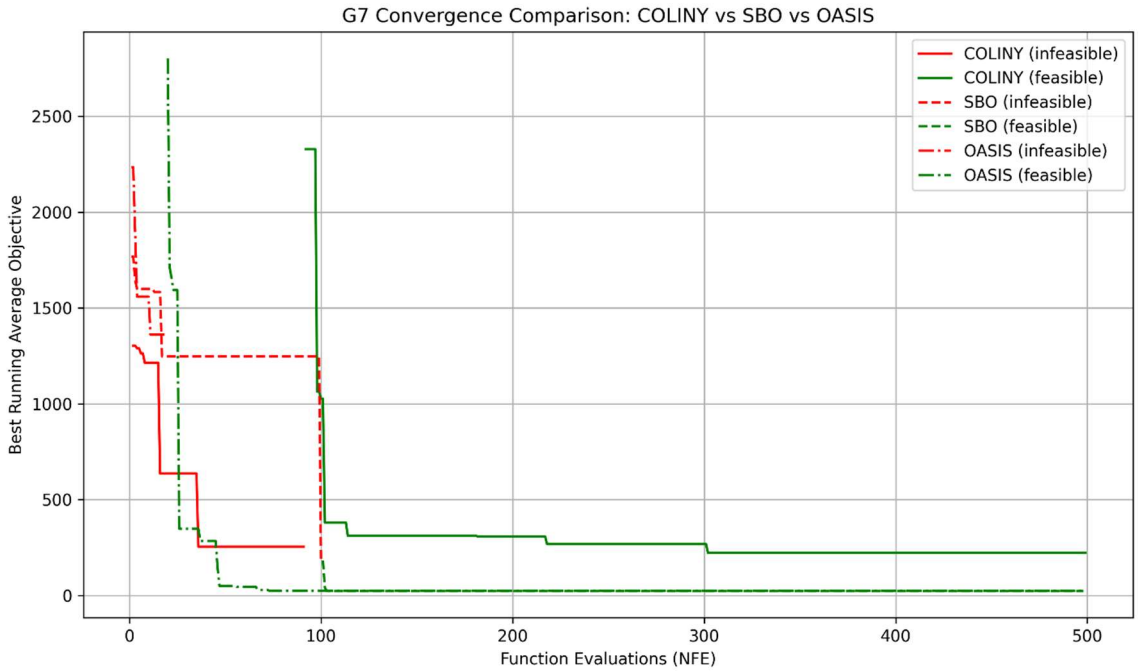


Figure 17. G7 conditional convergence plot

Table 3. First feasible solutions for the successful solvers.

First feasible solution NFE			
Method	SBO	COLINY	OASIS
Run 1	103	91	20
Run 2	102		N/A
Run 3	103		62
Run 4	104		29
Run 5	103		34
Run 6	103		44
Run 7	102		23
Run 8	103		78
Run 9	101		24
Run 10	103		36
Average	102.7	91	38.88889

Discussion

The performance of all solvers in Python libraries was abysmal, and none of them were able to find a single feasible solution. Ax-BO and OMADS terminated at around trial 21 and 234, respectively. Algorithms pymoo-GA and SciPy-EA showed signs of convergence when their budgets were increased by six orders of magnitude.

Some Dakota methods yielded feasible solutions. SOGA in Dakota failed to give any feasible solutions, with 500 evaluations, and produced poor results even with an extended budget of

3,000 evaluations. As for EGO, it produced only one feasible solution at NFE 102 with the objective function value of 459.45, which is far from the known optimum.

From Figure 17, one can see that both SBO and OASIS.AI converge to a similar optimum, while coliny (DIRECT) converged prematurely. Moreover, each run of OASIS.AI consistently finds its first feasible solution significantly earlier than Dakota's SBO. One can see from Table 3 that OASIS.AI found the first feasible solution with an average of 39 NFEs, while SBO used 103 and coliny used 91 NFEs.

Summary for Constrained Single-Objective Problems

All of Python's solvers failed to find a feasible solution, whereas Dakota's coliny DIRECT and SBO solvers provided meaningful results. Dakota's SBO method converges to the same minimum as OASIS.AI, while coliny converges prematurely. Overall, OASIS.AI is superior to the others in terms of being the first to find feasible solutions and the solution quality.

Multi-Objective Problems

Methodology

Similarly to the SOO problems, 10 instances of each MOO solver were run, resulting in a set of non-dominated solutions from the aggregated data of all 10 instances of each solver for each problem. This allowed for the plotting of the non-dominated solutions (f1, f2) for each problem, also known as Pareto Frontiers, in Figures 18-23 and hypervolume calculations [14], in Figure 24. For the constrained multi-objective problem, OSY, only feasible solutions were added to the Pareto Frontier.

All three of the pymoo solvers are evolution-based and use a set population size of 50, as that is the default that Dakota sets for their MOGA implementation.

The parameters for the Mesh Adaptive Search (MADS) implementation on ZDT1, ZDT3, ZDT6, and OSY were copied from the test script in the OMADS repository, except for the budget allocation. One minor change for the unconstrained problems was setting the constraint type to an empty value, as opposed to the penalty barrier specification "[PB]" as shown in the repository. It is unclear as to why this parameter is included in the OMADS test script, as the ZDT problems are all unconstrained. Removing the penalty barrier specification resulted in a noticeably improved Pareto Frontier by providing Pareto points in the left half of the graph. The Pareto Frontiers for OMADS on ZDT1, ZDT2, ZDT3, and ZDT6 with the original penalty barrier constraint specification are shown in Figure A4.

SFU

Results

Hypervolumes are helpful for understanding the performance of solvers relative to one another, especially when the Pareto Frontier of solvers may be overlapping. The higher the hypervolume value for a given solver on a given problem, the better the performance is. Although there are several formulations in the community for hypervolume calculation, Equation 1 was applied to calculate the results in Figure 24, as defined in [1]:

$$HV = \text{area } of \left(\cup_{i=1}^{n} \left[f_1^{(i)}, r_1 \right] \times \left[f_2^{(i)}, r_2 \right] \right) \quad \text{Equation 1}$$

Additionally, the reference point used to define r1 and r2 for the hypervolume is defined by Equation 2.

$$ref = (max(obj1) * 1.01, max(obj2) * 1.01) \quad \text{Equation 2}$$

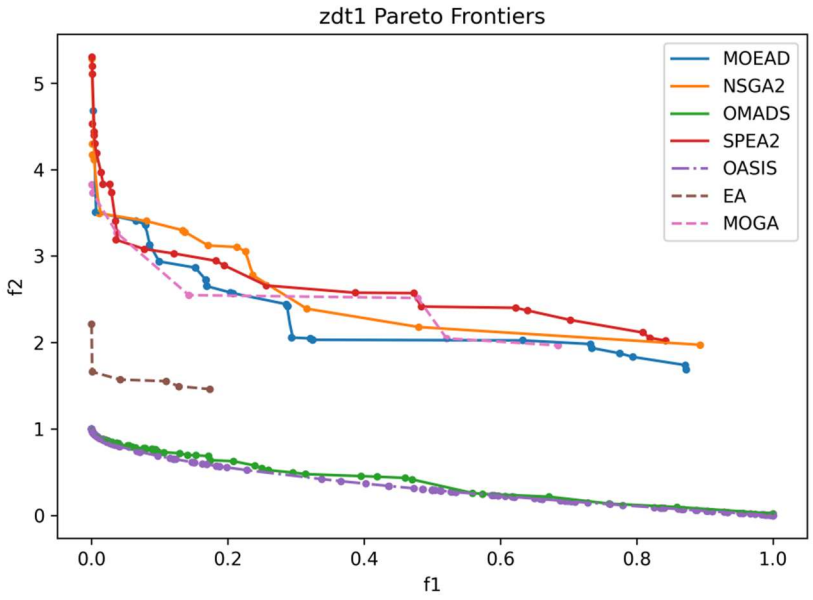


Figure 18. ZDT1 Pareto Frontiers from solvers (Python, Dakota, and OASIS.AI)

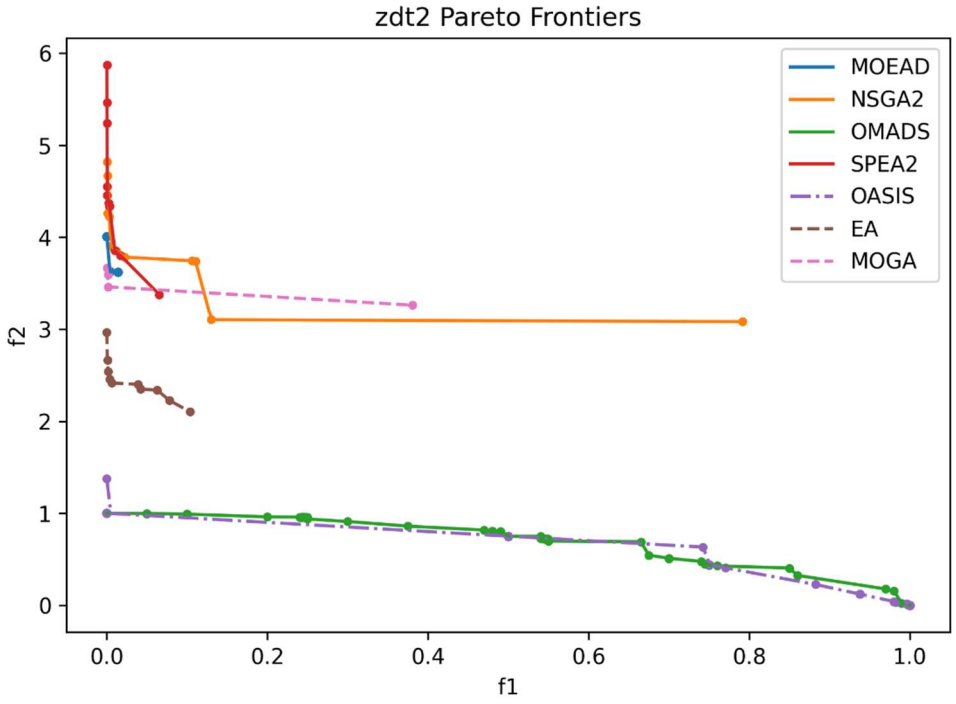


Figure 19. ZDT2 Pareto Frontiers from solvers (Python, Dakota, and OASIS.AI)

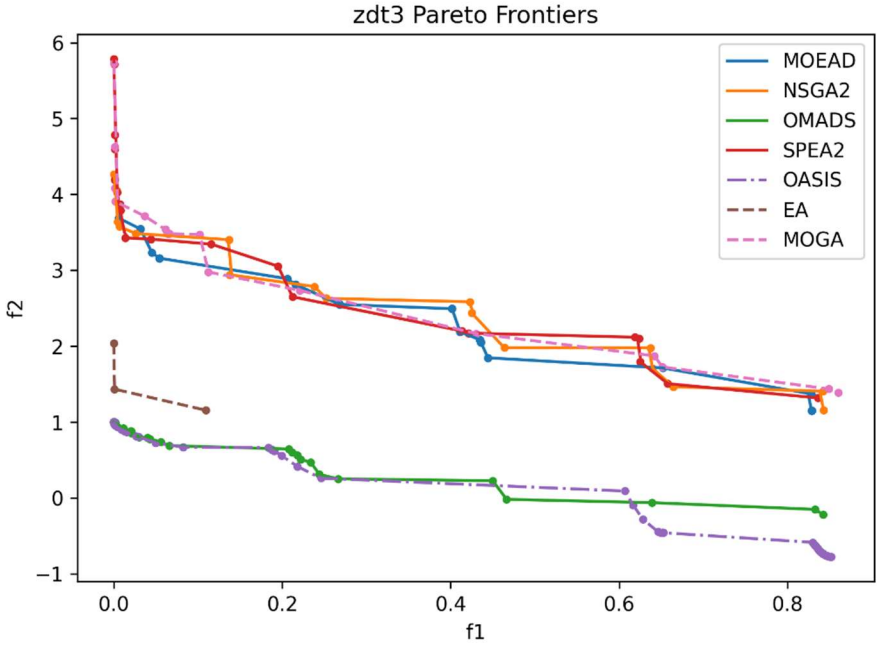


Figure 20. ZDT3 Pareto Frontiers from solvers (Python, Dakota, and OASIS.AI)

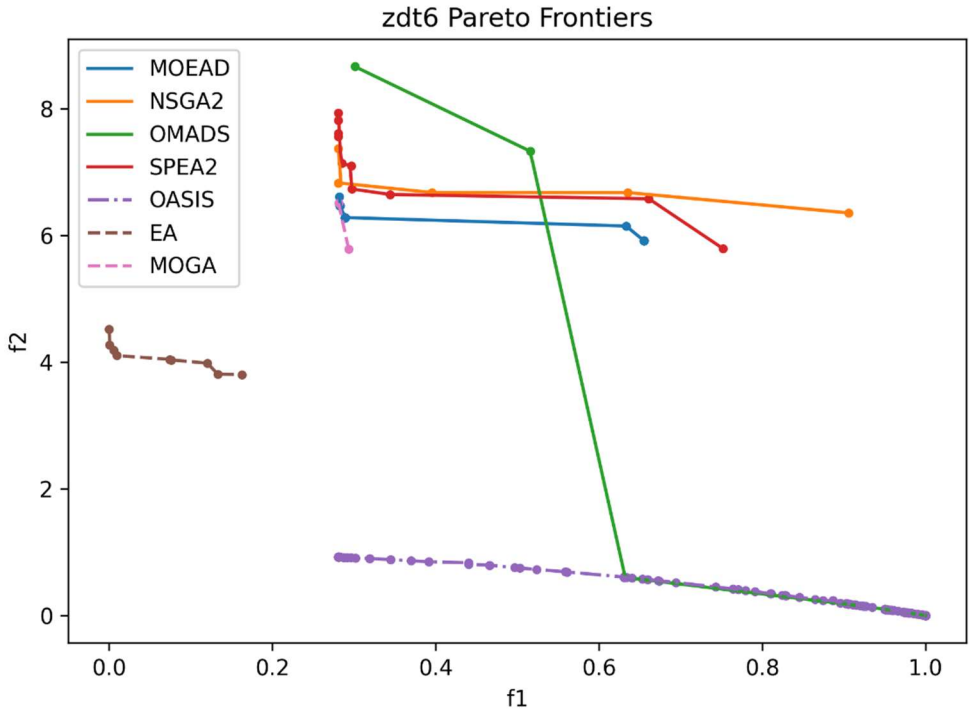


Figure 21. ZDT6 Pareto Frontiers from solvers (Python, Dakota, and OASIS.AI)

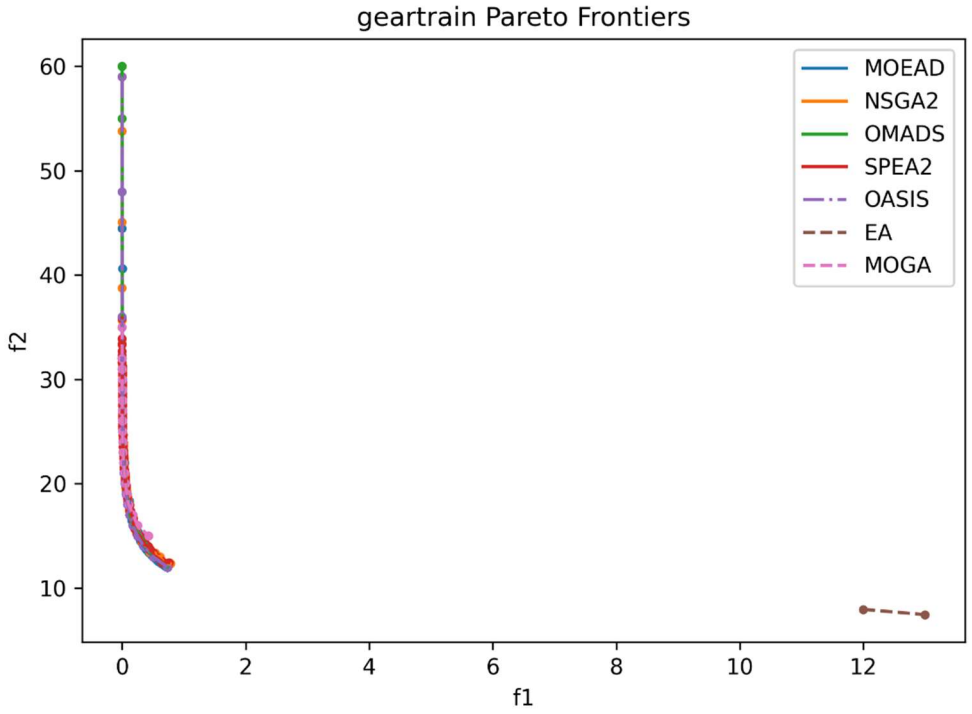


Figure 22. Geartrain (a discrete problem) Pareto Frontiers from solvers (Python, Dakota, and OASIS.AI)

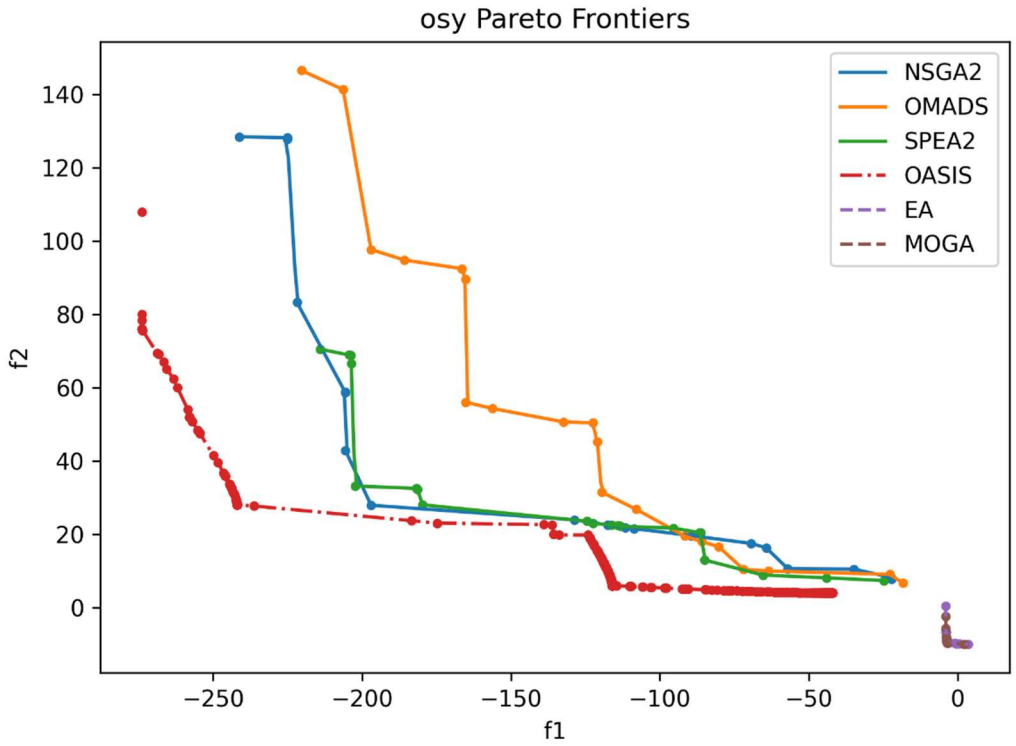


Figure 23. OSY (constrained) Pareto Frontiers from solvers (Python, Dakota, and OASIS.AI)

	MOO Problem Hypervolumes					
Sovler	ZDT1	ZDT2	ZDT3	ZDT6	OSY	Geartrain
MOEAD	3.1478	2.3287	3.0432	1.9302	N/A	635.66
NSGA2	2.8667	2.766	3.0218	1.5292	28073	631.81
OMADS	5.0067	5.2717	4.865	3.3084	22455	635.5
SPEA2	2.8248	2.5425	2.9867	1.7488	27417	630.52
OASIS	5.0588	5.2291	4.9303	5.9754	35707	635.68
EA	3.9122	3.838	4.0371	4.9604	1199	59.571
MOGA	2.9859	2.6179	2.9164	2.1546	1198.1	597.53

Figure 24. Hypervolume of MOO problems. Reference points are calculated relative to the worst f1 and f2 values for each problem. Green, light green, yellow, orange, and red indicate the gradual decrease of hypervolume values.

Discussion

It is observed that EA does not generate very good solutions for any of the problems, and its Pareto Frontiers are in a noticeably separate region from the other solvers. Additionally, the number of resulting Pareto Points is also quite few. EA was run with an increased budget of 1500, as opposed to 250 for ZDT1, and the Pareto Frontier still looked very similar, if not identical. A

possibility is that EA is not very well-suited for solving MO problems. MOGA also displays similar behaviour as it generates few Pareto Points per run. However, its accuracy is much better, and its Pareto Frontier has significant overlap with the other solvers.

The three solvers in pymoo display similar results with hypervolume values and Pareto Frontiers for each problem. OMADS noticeably outperforms all the other solvers for ZDT1, ZDT2, ZDT3, and ZDT6, but lacks prowess for evaluating OSY.

It is also to be noted that all solvers perform equally as well on the Geartrain problem, generating apparently identical Pareto Frontiers, aside from EA. However, it is to be noted that the hypervolumes generated by OASIS.AI are larger than those of other solvers for every problem. The only exception is OMADS in the ZDT2 problem, where it marginally beats OASIS.AI by a score of 0.05.

Summary for Multi-Objective Problems

OMADS appears to be the only competitor to OASIS.AI for the multi-objective problems. ZDT2 is the only problem where OMADS outperforms OASIS.AI, with a slightly larger hypervolume value. Dakota solvers perform poorly and generate few Pareto Points overall.

Conclusion

From the test results, OASIS.AI shows the overall strongest, most consistent performance for all three categories of problems, i.e., unconstrained single-objective problems, constrained single-objective problems, and multi-objective problems.

However, OASIS.AI has some competition from the open-source alternatives that have been listed, particularly the solvers used for unconstrained single-objective problems. It is observed that it was outperformed by SciPy-DA in several problems and by one of the other algorithms in seven test problems. As for the constrained SO problem, SBO managed to converge to a similar feasible minimum as OASIS.AI, but took more NFEs to reach the feasible region. As for MO problems, OMADS had close performance to OASIS.AI on four problems out of six.

If a more in-depth framework were to be required for the intensive, meticulous testing of algorithms, pre-existing benchmark sites like COCO [7] would be a good option, as it generates multiple transformed instances of each problem to prevent the user-provided algorithms from exploiting certain qualities of the problems. One main shortcoming of this study is that some solvers, namely DIRECT implementations, start from the center of the space; with these platforms, this problem can be overcome.

References

[1] Sandia National Laboratories. (2021). *Dakota (Version 21.1)* [Software]. Albuquerque, NM: Sandia National Laboratories.

[2] Python Software Foundation. (2025). *Python Language Reference Manual* (Version 3.13.7). Retrieved from https://docs.python.org/3/reference/index.html

[3] Empower Operations Corp. (2025). *OASIS AI: Next-generation AI-driven design optimization* [Web page]. Retrieved from https://empowerops.com

[4] Virtanen, P., Gommers, R., Oliphant, T. E., Haberland, M., Reddy, T., Cournapeau, D., Burovski, E., Peterson, P., Weckesser, W., Bright, J., van der Walt, S. J., Brett, M., Wilson, J., Millman, K. J., Mayorov, N., Nelson, A. R. J., Jones, E., Kern, R., Larson, E., … SciPy 1.0 Contributors. (2020). SciPy 1.0: Fundamental algorithms for scientific computing in Python. *Nature Methods, 17*(3), 261–272. https://doi.org/10.1038/s41592-019-0686-2

[5] Rapin, J., & Teytaud, O. (2018). *Nevergrad - A gradient-free optimization platform*. GitHub. https://doi.org/10.5281/zenodo.1207017

[6] Sandia National Laboratories. (2023). *Keyword Reference* [Dakota documentation, version 6.22.0]. In *Using Dakota*. Retrieved from https://snl-dakota.github.io/docs/6.22.0/users/usingdakota/reference.html

[7] Hansen, N., Auger, A., Ros, R., Mersmann, O., Tušar, T., & Brockhoff, D. (2021). "*COCO*: A platform for comparing continuous optimizers in a black-box setting". *Optimization Methods and Software, 36*(1), 114–144. https://doi.org/10.1080/10556788.2020.1808977

[8] Blank, J., & Deb, K. (2020). "pymoo: Multi-objective optimization in Python". IEEE Access, 8, 89497–89509. https://doi.org/10.1109/ACCESS.2020.2990567

[9] Bayoumy, A. (2022). OMADS: *A Python implementation of mesh adaptive direct search (MADS)* [Computer software]. GitHub. Retrieved July 2025, from https://github.com/Ahmed-Bayoumy/OMADS

[10] Balandat, M., Karrer, B., Jiang, D. R., Daulton, S., Letham, B., Wilson, A. G., & Bakshy, E. (2019). "BoTorch: A framework for efficient Monte-Carlo Bayesian optimization". arXiv. https://arxiv.org/abs/1910.06403

[11] Saves, P., Lafage, R., Bartoli, N., Diouane, Y., Bussemaker, J., Lefebvre, T., Hwang, J. T., Morlier, J., & Martins, J. R. R. A. (2024). "SMT 2.0: A Surrogate Modeling Toolbox with a

focus on hierarchical and mixed variables Gaussian processes". *Advances in Engineering Software*, 188, 103571. https://doi.org/10.1016/j.advengsoft.2023.103571

[12] Bliek, L., Guijt, A., Karlsson, R., Verwer, S., & de Weerdt, M. (2023). "Benchmarking surrogate-based optimisation algorithms on expensive black-box functions". *Applied Soft Computing, 110744*. https://doi.org/10.1016/j.asoc.2023.110744

[13] Sandia National Laboratories. *003.01 Dakota Training: Optimization - Feb. 2016* [Video]. Sandia National Laboratories. https://digitalops.sandia.gov/Mediasite/Play/a13c912f3e994c4ea010aacd903b12111d

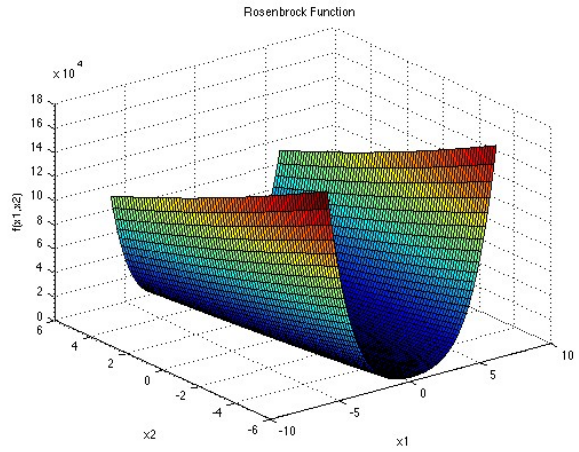
[14] Auger, A., Bader, J., Brockhoff, D., & Zitzler, E. (2009). "Theory of the hypervolume indicator: Optimal μ-distributions and the choice of the reference point". In *Foundations of Genetic Algorithms* (pp. 87–102). ACM. https://doi.org/10.1145/1527125.1527132

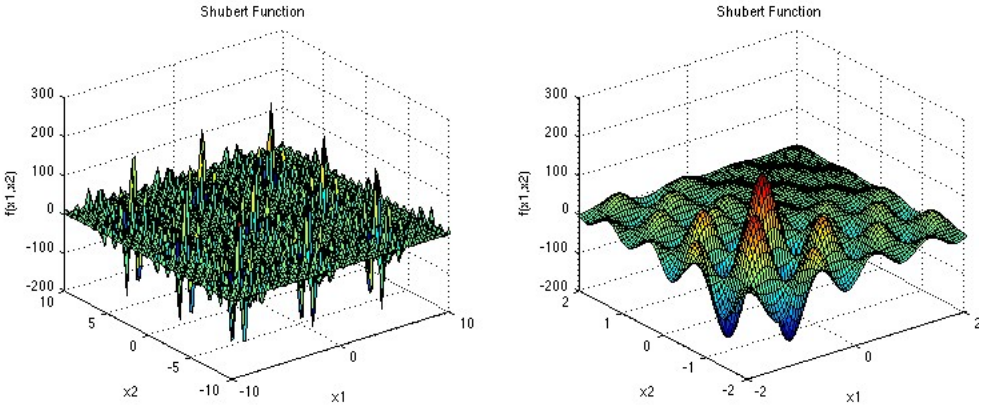
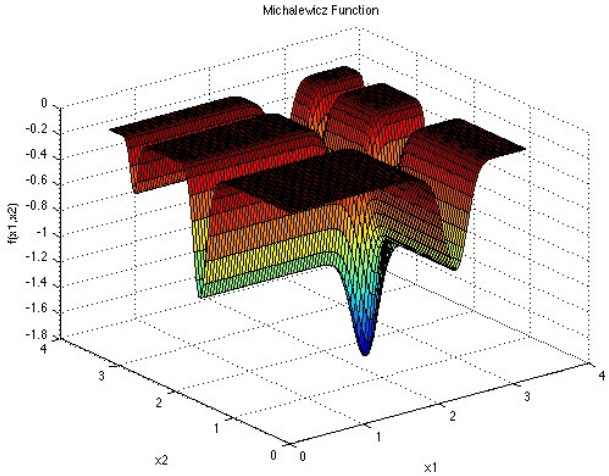
Appendix A

Table A1. The complete list of benchmark problems used in this test. The definitions for Rosenbrock d10, Shubert d10, and Michalewicz d10 are found below in Table A2. Row 6 is excluded from the test.

No	Problem Type	Benchmark	No. of Inputs	No. of Objectives	No. of Constraints	Budget (Func. Eval)
1	Multi-Objective, unconstrained	ZDT 1	30	2	0	250
2	Multi-Objective, unconstrained	ZDT 2	30	2	0	250
3	Multi-Objective, unconstrained	ZDT 3	30	2	0	250
4	Multi-Objective, unconstrained	ZDT 6	10	2	0	250
5	Multi-Objective, expensive constraints	OSY Exp. Cons.	6	2	6	500
6	Multi-Objective, cheap constraints	OSY Cheap Cons.	6	2	6	500
7	Multi-Objective, unconstrained	Geartrain	4 - Discrete	2	0	1000
8	Single Objective, expensive constraints	G7	10	1	8	500
9	Unimodal, unconstrained	Rosenbrock d10	10	1	0	1000
10	Unimodal, unconstrained	Rosenbrock d50	50	1	0	1000
11	Multimodal, unconstrained	Shubert d10	10	1	0	1000
12	Multimodal, unconstrained	Shubert d30	30	1	0	1000
13	Multimodal, unconstrained	Shubert d60	60	1	0	1000
14	Ill-shaped, unconstrained	Michalewicz d10	10	1	0	1000
15	Ill-shaped, unconstrained	Michalewicz d30	30	1	0	1000
16	Ill-shaped, unconstrained	Michalewicz d60	60	1	0	1000

Table A2. All the objective function definitions for benchmark test, for both constrained and unconstrained, and single- and multi-objective. (Equations 3-12)

Function definition with 2-D illustration (when applicable)	Name
$f(x) = sum_{(i=1)}^{(n-1)} \left(100 * \left(x_{(i+1)} - x_i^2\right)^2 + (1 - x_i)^2\right), x_i \in [-10,10]$ $for\ i = 1, \ldots, n$  https://www.sfu.ca/~ssurjano/rosen.html	*Rosenbrock function*

$$f(x) = prod_{(i=1)}^{n}\left(sum_{(j=1)}^{5}\left(j * cos\left((j+1) * x_i + j\right)\right)\right), x_i \in [-5.12, 5.12] \quad for\ i = 1, \dots, n$$  https://www.sfu.ca/~ssurjano/shubert.html	*Schubert function*
$$f(x) = -sum_{(i=1)}^{n}\left(sin(x_i) * \left(sin\left(\left((i+1) * x_i^2\right)/pi\right)\right)^{(2*m)}\right), x_i \in [0, pi] \quad fori = 1, \dots, n; m \in N$$  https://www.sfu.ca/~ssurjano/michal.html	*Michalewicz function*
$$f(x) = x_0^2 + x_1^2 + x_0 * x_1 - 14 * x_0 - 16 * x_1 + (x_2 - 10)^2 + 4 * (x_3 - 5)^2 + (x_4 - 3)^2 + 2 * (x_5 - 1)^2 + 5 * x_6^2 + 7 * (x_7 - 11)^2 + 2 * (x_8 - 10)^2 + (x_9 - 7)^2 + 45, s.t.\ 4 * x_0 + 5 * x_1 - 3 * x_6 + 9 * x_7 \leq 105; 10 * x_0 - 8 * x_1 - 17 * x_6 + 2 * x_7 \leq 0; 8 * x_0 - 2 * x_1 + 5 * x_8 - 2 * x_9 \leq 0; 3 * (x_0 - 2)^2 + 4 * (x_1 - 3)^2 + 2 * x_2^2 - 7 * x_3 \leq 120; 5 * x_0^2 + 8 * x_1 + (x_2 - 6)^2 - 2 * x_3 \leq 40; 0.5 * (x_0 - 8)^2 + (x_1 - 4)^2 + 3 * x_4^2 \leq 30; x_0 * x_1 + x_0 * x_5 - 2 * x_0^2 - 2 * x_1^2 + 14 * x_5 - 6 * x_6 \leq 0; 3 * x_0 - 6 * x_1 + 12 * (x_8 - 8)^2 - 7 * x_9 \leq 0$$	*G7 with constraints*

$f_1(x) = x_0; g(x) = 1\left(9/(n-1)\right) * sum_{(i=1)}^{(n-1)}(x_i);$ $h(x) = 1 - sqrt(f_1/g);\ f_2(x) = g * h; x_i \in [0,1]$	*ZDT1*
$f_1(x) = x_0; g(x) = 1 + \left(9/(n-1)\right) * sum_{(i=1)}^{(n-1)}(x_i);$ $h(x) = 1 - (f_1/g)^2;\ f_2(x) = g * h; x_i \in [0,1]$	*ZDT2*
$f_1(x) = x_0; g(x) = 1 + \left(9/(n-1)\right) * sum_{(i=1)}^{(n-1)}(x_i);$ $h(x) = 1 - sqrt(f_1/g) - (f_1/g) * sin(10 * pi * f_1);\ f_2(x) = g * h;$ $x_i \in [0,1]$	*ZDT3*
$f_1(x) = 1 - exp(-4 * x_0) * \left(sin(6 * pi * x_0)\right)^6;$ $g(x) = 1 + 9 * \left(sum_{(i=1)}^{(n-1)}(x_i)/(n-1)\right)^{(1/4)};$ $h(x) = 1 - (f_1/g)^2; f_2(x) = g * h; x_i \in [0,1]$	*ZDT6*
$f_1(x) = -(25 * (x_0 - 2)^2 + (x_1 - 2)^2 + (x_2 - 1)^2 + (x_3 - 4)^2 +$ $(x_4 - 1)^2)\ ; f_2(x) = x_0^2 + x_1^2 + x_2^2 + x_3^2 + x_4^2 + x_5^2,$ $s.t. x_0 + x_1 - 2 \geq 0; 6 - x_0 - x_1 \geq 0; 2 - x_1 + x_0 \geq 0; 2 - x_0 + 3 * x_1 \geq$ $0; 4 - (x_2 - 3)^2 - x_3 \geq 0; (x_4 - 3)^2 + x_5 - 4 \geq 0,$ $x_0 \in [0,10], x_1 \in [0,10], x_2 \in [1,5], x_3 \in [0,6], x_4 \in [1,5], x_5 \in [0,10]$	*OSY with constraints*
$f_1(x) = \left(1/6.931 - (x_2 * x_1)/(x_0 * x_3)\right)^2; f_2(x) = \max(x_0, x_1, x_2, x_3)\ ;$ $x_i \in [12, 60]\ for\ i = 0, \ldots, 3$	*Geartrain design*

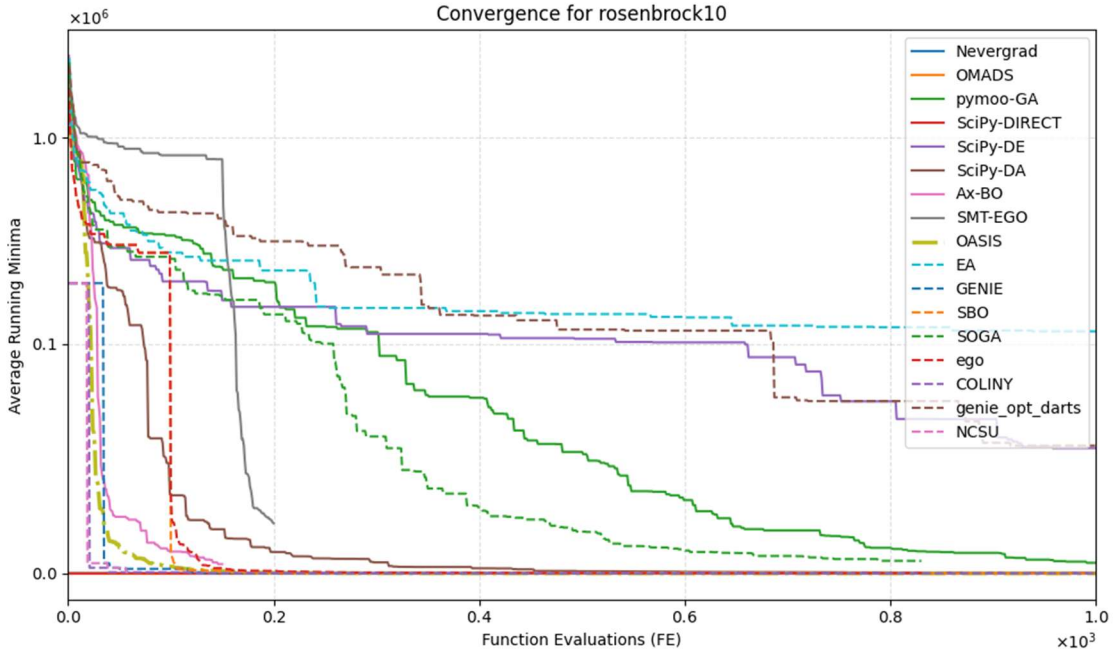


Figure A1. Rosenbrock 10d convergence plot with extra solvers that were excluded from use on future problems: NCSU (ncsu_DIRECT), COLINY (coliny_DIRECT), genie_opt_darts, EGO (Dakota), Ax-BO (from BoTorch), and EGO (SMT)

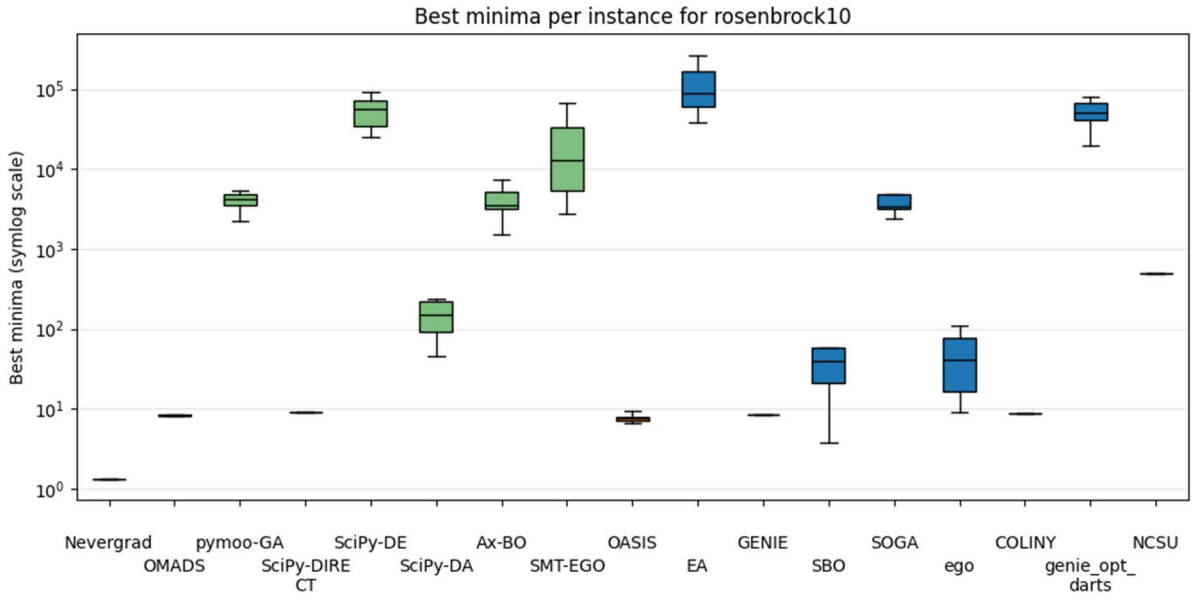


Figure A2. Rosenbrock 10d boxplot with extra solvers that were excluded from use on future problems: NCSU (ncsu_DIRECT), COLINY (coliny_DIRECT), genie_opt_darts, EGO (Dakota), Ax-BO (from BoTorch), and EGO (SMT)

Table A3. Rosenbrock 10d solver per instance runtimes with a total in hours, with extra solvers. Ax-BO and SMT-EGO are missing values for run 1 and 2 due to computer failure after the second runs were completed (results are only written after all instances are run)

Run	Nevergrad	OMADS	pymoo-GA	SciPy-DIRECT	SciPy-DE	SciPy-DA	Ax-BO	SMT-EGO	OASIS	ea	GENIE	sbo	soga	ego	COLINY	genie_opt_darts	NCSU
1	5.11568	47.33692	0.38464	0.00124	0.05152	0.05883	nan	nan	n/a	139.34	134.49	1552.9	103.99	739.5	139.21	995.37	8.15
2	4.60408	48.66469	7.01E-02	0.00319	0.04653	0.05006	nan	nan	n/a	139.49	134.13	1521.41	101.17	2376	137.02	1004.25	8.3
3	4.66587	46.48837	6.88E-02	0.00549	0.04764	0.0461	854.99237	1986.57887	n/a	139.42	135.16	1466.16	109.43	318.17	136.74	991.37	8.15
4	5.03135	50.04265	7.12E-02	0.00331	0.04708	0.04537	799.29808	1895.43265	n/a	139.45	134.36	1460.34	110.14	474.29	136.84	1014.95	8.1
5	3.7293	47.31055	7.37E-02	0.00595	0.04863	0.04726	821.21254	1787.02454	n/a	139.54	134.61	1499.23	106.44	1260.94	137.07	1010.38	8.22
6	4.47899	49.16509	0.07392	0.00198	0.04684	0.04438	850.01756	1888.88374	n/a	139.11	134.9	1493.11	112	3264.15	136.9	1016.25	8.11
7	4.89541	49.7998	0.07724	0.00429	0.05034	0.05	1016.03273	1925.17859	n/a	139.7	136.01	1451.51	103.17	1165.88	137.1	1011.3	8.14
8	5.06504	50.54742	0.07407	0.00321	0.04858	0.04967	824.86491	1839.73435	n/a	139.36	134.73	1329.31	108.72	1565.66	136.62	986.76	8.18
9	5.00292	49.22922	0.07014	0.00696	0.04741	0.05064	809.65406	1540.18124	n/a	139.53	134.82	1497.31	110.4	827.26	136.99	999.29	8.09
10	4.79858	47.68765	0.08631	0.00338	0.04391	0.05653	790.11323	1869.23168	n/a	139.55	134.79	1468.33	109.76	1955.92	137.08	1003.77	8.14
total (hrs)	0.01316	0.13508	0.00029	0.00001	0.00013	0.00014	1.8795	4.09229	n/a	0.38736	0.37444	4.09434	0.29867	3.87438	0.38099	2.78714	0.02266

Table A4. Rosenbrock 10d solver summary with extra solvers for minimum, maximum, and average minima found. Solvers with the same values for all columns are implementations of deterministic algorithms.

Solver	Min	Max	Average
Nevergrad	1.18	1.77	1.39
OMADS	8.06	8.49	8.27
pymoo-GA	2230	9940	4590
SciPy-DIRECT	8.94	8.94	8.94
SciPy-DE	24700	91400	54700
SciPy-DA	45.9	651	214
Ax-BO	1520	7340	4030
SMT-EGO	2790	66200	21900
OASIS	6.67	9.56	7.79
ea	37900	265000	115000
GENIE	8.44	8.44	8.44
sbo	3.84	168	57.2
soga	2430	13200	5180
ego	9	109	49.3
COLINY	8.67	8.67	8.67
genie_opt_darts	19700	113000	55700
NCSU	503	503	503

Table A5. Summary of minimum, maximum, and average for minima found by solvers for Michalewicz 10d, Schubert 10d, and Rosenbrock 10d

Michalewicz 10d			
Solver	Min	Max	Average
Nevergrad	-8.36E+00	-8.36E+00	-8.36E+00
OMADS	-9.30E+00	-8.56E+00	-9.05E+00
pymoo-GA	-7.87E+00	-6.60E+00	-7.28E+00
SciPy-DIRE	-7.61E+00	-7.61E+00	-7.61E+00
SciPy-DE	-5.46E+00	-3.87E+00	-4.46E+00
SciPy-DA	-8.96E+00	-7.65E+00	-8.35E+00
OASIS	-8.03E+00	-6.22E+00	-6.98E+00
EA	-4.90E+00	-3.77E+00	-4.29E+00
GENIE	-7.88E+00	-7.88E+00	-7.88E+00
SBO	-9.21E+00	-4.43E+00	-5.85E+00
SOGA	-8.64E+00	-7.11E+00	-7.85E+00

Schubert 10d			
Solver	Min	Max	Average
Nevergrad	-1.66E+10	-1.82E+07	-2.77E+09
OMADS	-2.10E+09	-1.03E+07	-5.76E+08
pymoo-GA	-3.84E+09	-2.06E+08	-1.63E+09
SciPy-DIRE	-6.36E+06	-6.36E+06	-6.36E+06
SciPy-DE	-1.54E+08	-1.01E+07	-4.23E+07
SciPy-DA	-1.43E+11	-2.72E+10	-8.03E+10
OASIS	-1.78E+11	-1.47E+09	-2.75E+10
EA	-1.64E+08	-1.87E+07	-6.05E+07
GENIE	-6.38E+08	-6.38E+08	-6.38E+08
SBO	-8.95E+08	-2.93E+07	-2.71E+08
SOGA	-2.71E+09	-2.03E+08	-1.44E+09

Rosenbrock 10d			
Solver	Min	Max	Average
Nevergrad	1.18E+00	1.77E+00	1.39E+00
OMADS	8.06E+00	8.49E+00	8.27E+00
pymoo-GA	2.23E+03	9.94E+03	4.59E+03
SciPy-DIRE	8.94E+00	8.94E+00	8.94E+00
SciPy-DE	2.47E+04	9.14E+04	5.47E+04
SciPy-DA	4.59E+01	6.51E+02	2.14E+02
OASIS	6.67E+00	9.56E+00	7.79E+00
EA	3.79E+04	2.65E+05	1.15E+05
GENIE	8.44E+00	8.44E+00	8.44E+00
SBO	3.84E+00	1.68E+02	5.72E+01
SOGA	2.43E+03	1.32E+04	5.18E+03

Table A6. Summary of minimum, maximum, and average for minima found by solvers for Michalewicz 30d, Schubert 30d, and Rosenbrock 50d

Michalewicz 30d			
Solver	Min	Max	Average
Nevergrad	-1.76E+01	-1.36E+01	-1.65E+01
OMADS	-1.24E+01	-9.70E+00	-1.14E+01
pymoo-GA	-1.35E+01	-1.18E+01	-1.25E+01
SciPy-DIRE	-1.17E+01	-1.17E+01	-1.17E+01
SciPy-DE	-8.96E+00	-7.02E+00	-7.96E+00
SciPy-DA	-8.94E+00	-7.00E+00	-8.00E+00
OASIS	-1.82E+01	-1.42E+01	-1.55E+01
EA	-5.08E+00	-3.56E+00	-4.17E+00
GENIE	-3.84E+00	-3.84E+00	-3.84E+00
SBO	-8.32E+00	-5.07E+00	-6.92E+00
SOGA	-8.41E+00	-6.47E+00	-7.61E+00

Schubert 30d			
Solver	Min	Max	Average
Nevergrad	-1.71E+25	-2.00E+20	-1.87E+24
OMADS	-4.30E+21	-9.30E+19	-1.24E+21
pymoo-GA	-6.90E+21	-2.48E+19	-1.72E+21
SciPy-DIRE	-1.94E+19	-1.94E+19	-1.94E+19
SciPy-DE	-2.86E+20	-2.24E+17	-3.34E+19
SciPy-DA	-7.95E+29	-5.17E+27	-2.55E+29
OASIS	-8.14E+26	-1.59E+23	-1.25E+26
EA	-6.64E+07	-5.70E+06	-3.15E+07
GENIE	-1.71E+06	-1.71E+06	-1.71E+06
SBO	-7.85E+09	-1.86E+07	-1.19E+09
SOGA	-1.21E+10	-8.04E+08	-3.76E+09

Rosenbrock 50d			
Solver	Min	Max	Average
Nevergrad	4.83E+01	4.84E+01	4.84E+01
OMADS	6.54E+03	1.02E+04	8.24E+03
pymoo-GA	1.08E+06	1.83E+06	1.44E+06
SciPy-DIRE	4.90E+01	4.90E+01	4.90E+01
SciPy-DE	3.07E+06	4.49E+06	4.00E+06
SciPy-DA	3.04E+04	4.62E+04	3.66E+04
OASIS	2.23E+02	2.57E+03	1.04E+03
EA	5.25E+04	2.01E+05	1.11E+05
GENIE	9.00E+00	9.00E+00	9.00E+00
SBO	5.28E+01	1.34E+03	2.53E+02
SOGA	1.13E+03	9.67E+03	4.68E+03

Table A7. Summary of minimum, maximum, and average for minima found by solvers for Michalewicz 60d and Schubert 60d

Michalewicz 60d			
Solver	Min	Max	Average
Nevergrad	-1.76E+01	-1.36E+01	-1.65E+01
OMADS	-1.24E+01	-9.70E+00	-1.14E+01
pymoo-GA	-1.35E+01	-1.18E+01	-1.25E+01
SciPy-DIRE	-1.17E+01	-1.17E+01	-1.17E+01
SciPy-DE	-8.96E+00	-7.02E+00	-7.96E+00
SciPy-DA	-8.94E+00	-7.00E+00	-8.00E+00
OASIS	-2.86E+01	-2.01E+01	-2.30E+01
EA	-4.37E+00	-2.97E+00	-3.84E+00
GENIE	-3.84E+00	-3.84E+00	-3.84E+00
SBO	-8.59E+00	-4.77E+00	-6.56E+00
SOGA	-8.43E+00	-6.65E+00	-7.55E+00

Schubert 60d			
Solver	Min	Max	Average
Nevergrad	-2.14E+46	-4.06E+40	-2.75E+45
OMADS	-4.37E+38	-7.03E+29	-6.78E+37
pymoo-GA	-2.23E+37	-4.33E+34	-3.89E+36
SciPy-DIRE	-4.91E+38	-4.91E+38	-4.91E+38
SciPy-DE	-8.28E+35	-3.47E+32	-8.65E+34
SciPy-DA	-1.58E+54	-2.05E+48	-1.73E+53
OASIS	-1.93E+49	-8.24E+43	-2.38E+48
EA	-7.18E+07	-1.00E+07	-3.92E+07
GENIE	-1.71E+06	-1.71E+06	-1.71E+06
SBO	-1.69E+09	-1.17E+07	-6.69E+08
SOGA	-2.50E+10	-5.26E+08	-6.98E+09

Table A8. Rosenbrock 10d summary of solver runtimes, for each instance. Includes total runtime in hours.

Rosenbrock 10d											
Run	Nevergrad	OMADS	pymoo-GA	ciPy-DIREC	SciPy-DE	SciPy-DA	OASIS	ea	GENIE	sbo	soga
1	5.11568	47.33692	0.38464	0.00124	0.05152	0.05883	nan	139.34	134.49	1552.9	103.99
2	4.60408	48.66469	0.07005	0.00319	0.04653	0.05006	nan	139.49	134.13	1521.41	101.17
3	4.66587	46.48837	0.06881	0.00549	0.04764	0.0461	nan	139.42	135.16	1466.16	109.43
4	5.03135	50.04265	0.07116	0.00331	0.04708	0.04537	nan	139.45	134.36	1460.34	110.14
5	3.7293	47.31055	0.07374	0.00595	0.04863	0.04726	nan	139.54	134.61	1499.23	106.44
6	4.47899	49.16509	0.07392	0.00198	0.04684	0.04438	nan	139.11	134.9	1493.11	112
7	4.89541	49.7998	0.07724	0.00429	0.05034	0.05	nan	139.7	136.01	1451.51	103.17
8	5.06504	50.54742	0.07407	0.00321	0.04858	0.04967	nan	139.36	134.73	1329.31	108.72
9	5.00292	49.22922	0.07014	0.00696	0.04741	0.05064	nan	139.53	134.82	1497.31	110.4
10	4.79858	47.68765	0.08631	0.00338	0.04391	0.05653	nan	139.55	134.79	1468.33	109.76
total (hr)	0.01316	0.13508	0.00029	0.00001	0.00013	0.00014	nan	0.38736	0.37444	4.09434	0.29867

Table A9. Michalewicz 10d summary of solver runtimes, for each instance. Includes total runtime in hours.

Michalewicz 10d											
Run	Nevergrad	OMADS	pymoo-GA	SciPy-DIREC	SciPy-DE	SciPy-DA	OASIS	ea	GENIE	sbo	soga
1	3.11136	45.74136	0.0723	0.01299	0.06688	0.03635	nan	139.52	134.63	564.58	98.46
2	3.16869	43.51375	0.07828	0.00196	0.05671	0.05103	nan	139.56	134.8	1052.74	106.66
3	3.08655	46.01915	0.07207	0.01705	0.05773	0.0507	nan	139.9	134.85	1334.82	105.96
4	2.99134	44.24328	0.07095	0.01641	0.05862	0.04965	nan	139.58	134.66	1003	104.97
5	3.31391	42.96798	0.07245	0.01642	0.05699	0.05119	nan	140.04	134.67	1329.97	109.73
6	3.15032	42.71963	0.07052	0.0171	0.05616	0.03878	nan	141.28	134.61	1240.06	106.88
7	2.97935	53.34374	0.07171	0.01526	0.05605	0.05007	nan	141.94	134.76	1245.25	109.78
8	3.16062	44.36217	0.07138	0	0.05802	0.06859	nan	140.52	134.8	1283.79	105.87
9	3.13102	44.65778	0.0707	0.0171	0.05791	0.0524	nan	139.77	134.71	1178.05	102.58
10	2.97804	42.53092	0.06867	0.01621	0.05667	0.04264	nan	139.33	134.67	1087.56	108.15
total (hr)	0.00863	0.12503	0.0002	0.00004	0.00016	0.00014	nan	0.38929	0.37421	3.14439	0.29418

Table A10. Schubert 10d summary of solver runtimes, for each instance. Includes total runtime in hours.

Schubert 10d											
Run	Nevergrad	OMADS	pymoo-GA	SciPy-DIREC	SciPy-DE	SciPy-DA	OASIS	ea	GENIE	sbo	soga
1	2.2606	39.97259	0.14327	0.06606	0.08939	0.10568	nan	221.81	168.19	1344.66	137.93
2	2.29041	39.73105	0.11923	0.06308	0.10097	0.07499	nan	228.84	167.6	1181.27	136.8
3	2.08312	39.92059	0.11517	0.05534	0.09892	0.08418	nan	218.2	0.21	1188.01	142.09
4	2.45537	40.02026	0.1014	0.04059	0.09912	0.09302	nan	206.71	168.35	1400.04	134.99
5	2.17088	48.81545	0.11519	0.05525	0.08329	0.08736	nan	216.15	170.42	1421.84	138.9
6	2.09533	39.92605	0.10659	0.05517	0.09997	0.08325	nan	222.41	166.33	1338.47	139.2
7	1.74454	41.713	0.09479	0.05416	0.09933	0.08334	nan	222.26	165.4	1299.15	140.97
8	2.24016	39.43657	0.10038	0.05497	0.09985	0.08292	nan	208.91	165.42	1208.41	141.86
9	1.78081	40.91353	0.11743	0.05521	0.09963	0.08417	nan	193.12	165.61	1164.07	139.47
10	2.53658	40.64881	0.10081	0.05482	0.08386	0.09206	nan	176.81	173.94	2185.81	139.77
total (hr)	0.00602	0.11419	0.00031	0.00015	0.00027	0.00024	nan	0.58756	0.41985	3.81437	0.38666

Table A11. Michalewicz 30d summary of solver runtimes, for each instance. Includes total runtime in hours.

Michalewicz 30d											
Run	Nevergrad	OMADS	pymoo-GA	SciPy-DIREC	SciPy-DE	SciPy-DA	OASIS	ea	GENIE	sbo	soga
1	3.65971	205.9527	0.40501	0.21874	0.31109	0.28567	nan	112.75	108.41	1577.13	94.67
2	3.56858	205.563	0.38036	0.21877	0.31625	0.30204	nan	111.99	108.92	1935.88	92.33
3	3.36454	214.4522	0.40124	0.20572	0.31997	0.28618	nan	112.37	108.44	1730.33	93.14
4	3.25851	202.2292	0.36008	0.19925	0.32157	0.3044	nan	111.91	108.84	1740.64	92.42
5	3.20982	214.5894	0.36248	0.21636	0.29364	0.33823	nan	111.89	108.56	1670.76	93.23
6	3.26787	213.7097	0.36027	0.20011	0.32825	0.28352	nan	112.22	108.79	1650.7	93.74
7	3.58269	210.6857	0.33876	0.21083	0.30594	0.2997	nan	111.87	108.48	nan	93.38
8	3.24826	212.6851	0.39323	0.20392	0.32561	0.30341	nan	112.25	108.62	nan	93.25
9	3.28319	208.8046	0.35842	0.19713	0.31875	0.299	nan	112.21	108.49	nan	92.58
10	3.22958	214.5441	0.3491	0.19221	0.3066	0.29525	nan	111.87	110.39	nan	93.88
total (hr)	0.00935	0.58423	0.00103	0.00057	0.00087	0.00083	nan	0.31148	0.30221	2.86262	0.25906

Table A12. Schubert 30d summary of solver runtimes, for each instance. Includes total runtime in hours.

Schubert 30d											
Run	Nevergrad	OMADS	pymoo-GA	SciPy-DIREC	SciPy-DE	SciPy-DA	OASIS	ea	GENIE	sbo	soga
1	1.50792	39.65663	0.31851	0.21908	0.31283	0.25499	nan	313.09	259.78	1568.73	223.7
2	1.58655	37.84613	0.26616	0.22673	0.31086	0.2306	nan	305.15	259.8	2410.54	224.6
3	1.46922	37.09596	0.25954	0.21017	0.30186	0.22163	nan	312.27	254.8	1616.98	223.12
4	1.44607	36.44197	0.26835	0.21579	0.31517	0.22718	nan	313.16	255.24	5030.71	223.85
5	1.47607	33.87855	0.26733	0.21196	0.30606	0.23754	nan	313.78	255.34	1864.87	223.66
6	1.48859	34.93456	0.26924	0.21868	0.34539	0.23266	nan	312.36	258.45	1851.61	223.42
7	1.45642	41.77081	0.26626	0.20983	0.31036	0.22474	nan	315.04	254.57	1724.55	221.93
8	1.7251	35.55431	0.25556	0.21363	0.31614	0.23799	nan	312.46	256.46	2580.86	223.69
9	1.45052	34.48709	0.26609	0.21039	0.30788	0.25258	nan	290.21	256.18	2252.61	222.5
10	1.50849	34.11728	0.25389	0.20976	0.29939	0.23205	nan	271.32	256.59	2759.46	226.54
total (hr)	0.0042	0.10161	0.00075	0.0006	0.00087	0.00065	nan	0.84968	0.71311	6.57248	0.62139

Table A13. Rosenbrock 50d summary of solver runtimes, for each instance. Includes total runtime in hours.

Rosenbrock 50d											
Run	Nevergrad	OMADS	pymoo-GA	SciPy-DIREC	SciPy-DE	SciPy-DA	OASIS	ea	GENIE	sbo	soga
1	0.3717	12.20198	0.08176	0.00151	0.05054	0.04781	nan	137.78	133.23	43200	114.89
2	0.41864	12.04714	0.06999	0	0.04901	0.02695	nan	137.55	133.48	43200	116.61
3	0.37727	12.08656	0.06737	0	0.03687	0.04813	nan	137.59	133.26	43200	115.67
4	0.37839	12.06604	0.05701	0.01562	0.04676	0.03714	nan	137.53	133.17	43200	115.51
5	0.42953	12.11029	0.07874	0	0.04798	0.0361	nan	137.64	133.23	43200	116.49
6	0.38918	11.98081	0.07487	0	0.04907	0.0353	nan	137.45	133.29	43200	117.2
7	0.36154	12.02753	0.06575	0.00182	0.04085	0.0445	nan	137.56	133.24	43200	116.87
8	0.43741	12.14646	0.06716	0	0.04517	0.02848	nan	137.49	133.18	43200	117.23
9	0.36655	12.13643	0.06466	0	0.04189	0.04659	nan	137.4	133.21	43200	115.88
10	0.36493	12.10753	0.06291	0.01701	0.03114	0.02778	nan	137.44	133.19	43200	115.15
total (hr)	0.00108	0.03359	0.00019	0.00001	0.00012	0.00011	nan	0.38206	0.37013	120	0.32264

Table A14.Michalewicz 60d summary of solver runtimes, for each instance. Includes total runtime in hours.

Michalewicz 60d											
Run	Nevergrad	OMADS	pymoo-GA	SciPy-DIREC	SciPy-DE	SciPy-DA	OASIS	ea	GENIE	sbo	soga
1	2.60264	163.9184	0.35824	0.16106	0.24923	0.22719	nan	nan	108.86	nan	92.54
2	3.19539	160.4676	0.2787	0.15942	0.24448	0.2337	nan	nan	109.34	nan	92.74
3	3.0116	159.3516	0.2943	0.16161	0.24567	0.22677	nan	nan	108.71	nan	92.48
4	2.81427	158.9014	0.28916	0.1603	0.24321	0.22721	nan	nan	109.58	nan	91.59
5	3.07758	159.9999	0.27529	0.15973	0.24416	0.22712	nan	nan	109.05	nan	91.9
6	2.93948	159.1792	0.27634	0.1626	0.24819	0.22694	nan	nan	109.98	nan	92.04
7	2.93149	158.8273	0.28122	0.16141	0.24577	0.23254	nan	nan	109.17	nan	91.5
8	2.68177	159.2305	0.28228	0.16266	0.24419	0.22522	nan	nan	108.57	nan	91.41
9	2.02854	158.9865	0.27605	0.16158	0.24341	0.22607	nan	nan	108.31	nan	92.23
10	2.61171	160.4918	0.27838	0.16116	0.24385	0.22532	nan	nan	110.08	nan	92.26
total (hr)	0.00775	0.44427	0.0008	0.00045	0.00068	0.00063	nan	nan	0.30324	nan	0.25575

Table A15. Schubert 60d summary of solver runtimes, for each instance. Includes total runtime in hours.

Schubert 60d											
Run	Nevergrad	OMADS	pymoo-GA	SciPy-DIREC	SciPy-DE	SciPy-DA	OASIS	ea	GENIE	sbo	soga
1	2.76943	16.66489	1.35616	0.25596	3.41389	1.26233	nan	387.37	475.96	4130.69	223.57
2	2.92231	16.9603	1.23587	0.27989	3.26741	1.24874	nan	387.37	473.66	4713.48	220.34
3	2.35638	49.76992	1.22658	0.25376	3.53986	1.21118	nan	424.83	472.67	12974.39	223.28
4	2.31672	17.58608	1.30885	0.25844	3.47942	1.18715	nan	506.02	475.45	4527.31	307.82
5	2.31995	50.90723	1.27372	0.24698	3.38613	1.27237	nan	506.87	1383.12	5478.11	310.71
6	2.35876	18.59197	1.3387	0.26306	3.40306	1.21738	nan	505.86	258.25	5644.1	313.31
7	2.25028	17.84892	1.33559	0.2495	3.33635	1.20768	nan	507.52	273.86	4029.93	317.8
8	2.60709	52.6647	1.25498	0.27592	3.36189	1.18077	nan	505.36	268.18	6162.68	316.03
9	2.30736	52.63826	1.22393	0.28755	3.33779	1.17903	nan	504.1	264.67	21504.32	313.73
10	2.28464	53.11474	1.2998	0.26111	3.36083	1.20516	nan	506.51	257.79	5315.85	315.07
total (hr)	0.0068	0.09632	0.00357	0.00073	0.00941	0.00338	nan	1.31717	1.27878	20.68913	0.79491

SOO	Problem	Python	Dakota
	G7	ASUS	ASUS
	Rosenbrock10	GB	GB
	Rosenbrock50	PDOL11	PDOL14
	Schubert10	GB	GB
	Schubert30	ASUS	ASUS
	Schubert60	PDOL11	GB
	Michalewicz10	GB	GB
	Michalewicz30	PDOL11	PDOL11
	Michalewicz60	PDOL11	PDOL11
BOO	Problem	Python	Dakota
	ZDT1	ASUS	PDOL11
	ZDT2	PDOL11	PDOL11
	ZDT3	ASUS	PDOL11
	ZDT6	PDOL11	PDOL11
	OSY Cheap Const.	PDOL11	PDOL11
	Geartrain	PDOL11	PDOL11

Computer Specifications
ASUS: 8GB memory, 2.10GHz
GB: 16GB memory, 2.5GHz
PDOL11: 16GB memory, 3.0GHz
PDOL 14: 16GB memory, 3.2GHz

Figure A3. Specifications for computers used to run the solvers in Python and Dakota, along with identification for which computer ran which solver

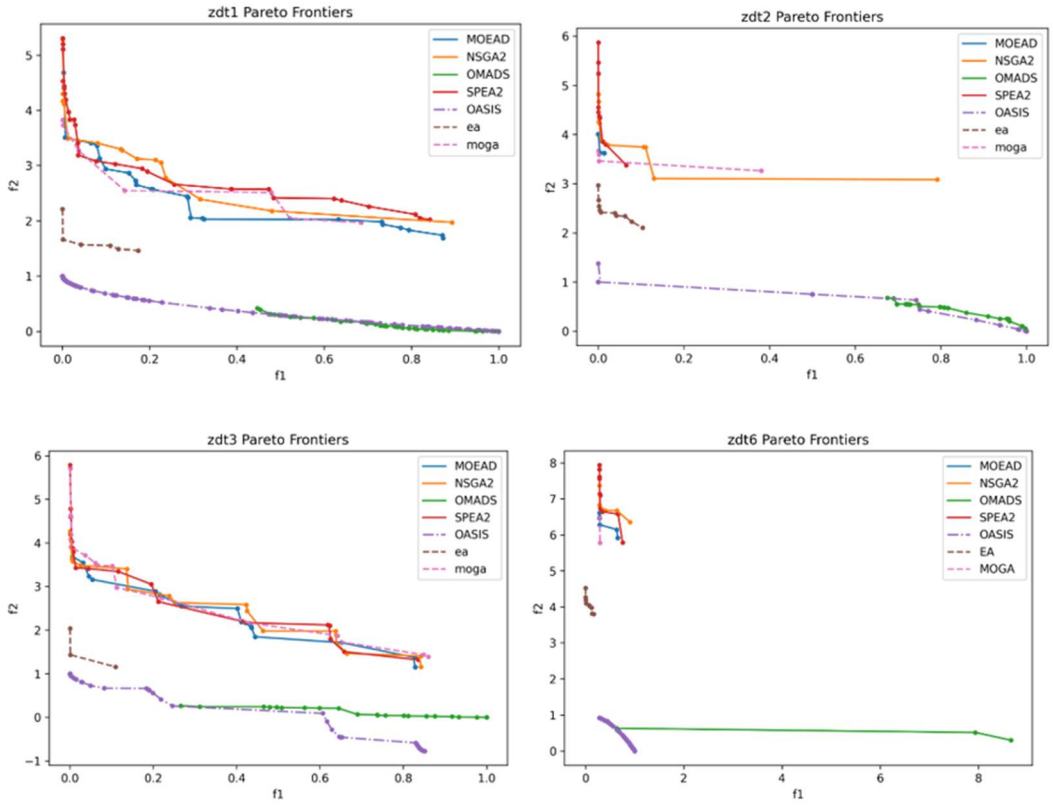


Figure A4. Pareto Frontiers for ZDT1, ZDT2, ZDT3, ZDT6, with OMADS parameter "constraint_type" set to "[PB]"

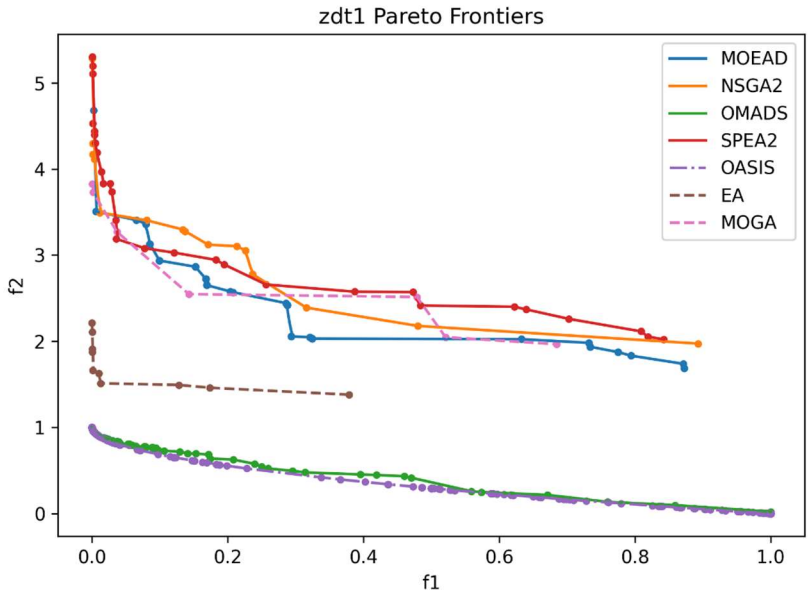


Figure A5. ZDT1 Pareto Frontier, with EA given a budget of 1500, as opposed to 250.