Extension of "Nevergrad", an Optimisation Platform

Ryan Kroon (Author)
Undergraduate Student
University of Adelaide
ryan.kroon@adelaide.edu.au

Dr. Markus Wagner (Supervisor)
Associate Professor
University of Adelaide
markus.wagner@adelaide.edu.au

ABSTRACT

This paper focuses on an optimisation platform called Nevergrad which is a Python library that acts as a toolbox for derivative-free and evolutionary optimisation. It is an open-source project that strongly welcomes any contributions to benefit the platform. The aim is to gain a working understanding of Nevergrad before extending the API via the addition of problem instances that broaden the application and usability of the platform. The problem instances considered for contribution are the Team Pursuit Track Cycling Problem and the Travelling Thief Problem. Extended focus also relates to the benchmarking of algorithms within defined problem scenarios.

1 INTRODUCTION

Released in 2018, Nevergrad is an open-source Python 3.6+ toolkit by Facebook and offers an extensive collection of algorithms for optimisation that avoid gradient calculations. These algorithms are presented in an easy-to-use Python framework that enable scientists whose work involve derivative-free optimisation to implement state-of-the-art algorithms and methods to compare performance in different scenarios[1]. The library contains a wide range of optimisers such as Particle Swarm Optimisation, FastGA, Covariance Matrix Adaptation, Sequential Quadratic Programming, Differential Evolution, Population Control Methods for noise management, Evolution Strategies and many others. Nevergrad also contains a wide range of test functions to evaluate the optimisers which helps developers and researchers find the best optimiser for specific cases or use well-known benchmarks to evaluate how a method compares with the current state of the art.

Since its initial release, Nevergrad has become a widely used research tool. The number of users is likely to increase due to high level of maintenance and testing that is performed on the API, as well as how often new features are added to the platform. The use of optimisation is applicable to many different disciplines, however, more specifically, most machine learning tasks from natural language processing to image classification rely on derivative-free optimisation to tune parameters and possibly hyperparameters in their models. Given the specialty of Nevergrad in derivative-free optimisation, Nevergrad is highly utilised by AI researchers and machine learning specialists[11]. However, the platform is still relevant and applicable to other fields. For example, a recent improvement to Nevergrad is multi-objective optimisation. This type of optimisation is prominent in nearly everyone's life, for instance, take someone who is looking to buy a house, they may want options that are simultaneously cheap, nearby the beach, at least two bathrooms etc. And given the simple ask-and-tell nature of Nevergrad's API, it can be used by people who are by no means experts in Python.

The aim for this project is to provide useful and insightful contributions to Nevergrad that hopefully improve and broadens its functionality. This will be achieved via the integration of two problem instances, the Team Pursuit Track Cycling Problem and the Travelling Thief Problem. Both these problem scenarios will provide a foundation for future benchmarking studies in Nevergrad.

2 MOTIVATION

Nevergrad strongly welcomes any contributions to the platform, so much so that not only is it a reason why Nevergrad is so widely regarded and utilised, but they have teamed up with IOHprofiler, a benchmarking framework, to organise the Nevergrad and IOHprofiler Open Optimisation Competition to reward the most effective contributions[5]. There are two tracks; a performance-oriented track for new algorithms and optimisation methods, and a general benchmarking practices track which essentially considers all other relevant ideas such as suggesting new benchmark problems, performance measures or statistics, visualisation of benchmark data, extending the functionalities of the benchmarking environment, or any other relevant improvement[5]. The submission deadline for both tracks is September 30, 2021 and an extended focus of this project is to participate in this competition.

Nevergrad want to promote and welcome as many contributions to the toolbox as possible because the platform is still new, and the initial release only contained basic artificial test functions. The plan was for contributors to add more instances, including functions that represent physical models. They want to continue to add functionality to help researchers create and benchmark new algorithms. The efforts of this project will hopefully work towards this goal and broaden the functionality of Nevergrad.

Since its release, there have been many contributions and improvements to Nevergrad. Some recent improvements made by the maintainers of Nevergrad include Multi-objective optimisation, Constrained optimisation, simplified problem parametrisation and competence map optimisers which means there are now algorithms to automatically select the best optimisation method, considering the computational budget, the dimension, the type of variables, and the degree of parallelism[11].

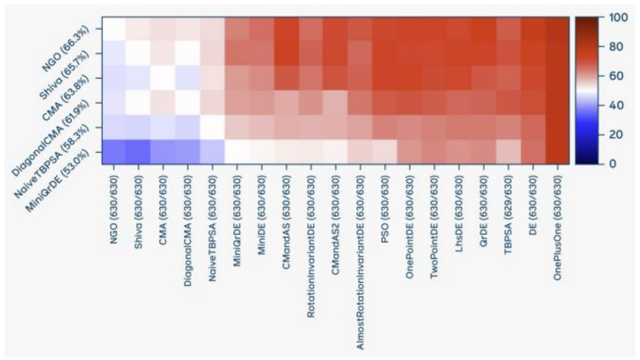


Figure 1: Graphic comparing optimisation methods for a set of example problems

Figure 1 shows the probability of an optimiser outperforming the other algorithms. Algorithms are ranked left to right on performance with the 6 best algorithms also listed on the vertical axis. Yet another recent improvement is an interface with HiPlot, which is Facebook's lightweight visualisation tool, and will allow researchers to easily explore the optimisation process or to use an interactive plot in a Jupyter notebook to analyse the behaviours of different algorithms[2]. The continuous maintenance and addition of features to Nevergrad contributes to the attractiveness of the overall platform. Such is the motivation for this project as it will attempt to facilitate this prospect by providing useful contributions to Nevergrad in terms of new problem instances for benchmarking.

3 LITERATURE REVIEW

The papers read for this project fall into three main categories. The first being the documentation and documents published by Nevergrad explaining the purpose of their platform and how to use it. Papers that present Nevergrad or use Nevergrad as a prominent tool in their work also fall into this category. The second category relates to benchmarking, including the goals of benchmarking, the desirable qualities of a problem scenario, and what constitutes good practice in benchmarking studies. Finally, papers were read in relation to possible problem instances that will be added to Nevergrad.

3.1 Nevergrad, an optimisation platform

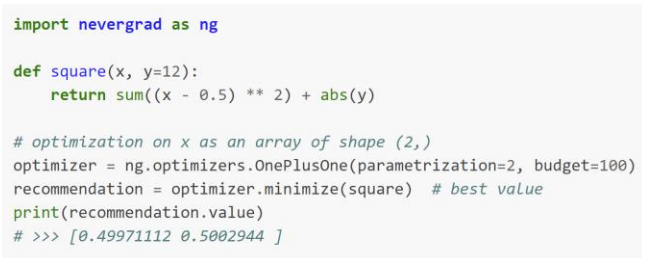
3.1.1 Using and Applying Nevergrad

Installation of Nevergrad is simple and can be done with a pip install. It is also simple to install the master branch of the latest release if needed.

The core object that the user can interact with is called optimiser. This object implements the optimisation method and can be one of several different optimisers[3]. The optimiser facilitates the ask and tell interface with its three main methods being:

- Ask: suggest a candidate on which to evaluate the function to optimise.
- Tell: update the optimiser with the value of the function for a candidate.
- Provide_recommendation: returns the candidate that the algorithm considers the best.

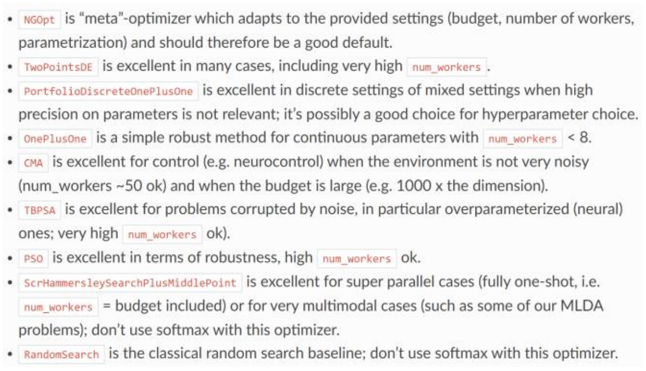
By using an optimiser, minimisation with Nevergrad is simple and an example using the OnePlusOne algorithm is given in figure 2 below.


```
import nevergrad as ng

def square(x, y=12):
    return sum((x - 0.5) ** 2) + abs(y)

# optimization on x as an array of shape (2,)
optimizer = ng.optimizers.OnePlusOne(parametrization=2, budget=100)
recommendation = optimizer.minimize(square)  # best value
print(recommendation.value)
# >>> [0.49971112 0.5002944 ]
```
Figure 2: Basic Example to Minimise a Function

There are many optimisers available and Nevergrad contains a dictionary of all the optimisers which can be printed with the command below. All algorithms have strengths and weaknesses, but Nevergrad has provided a list of algorithms with a few helpful comments and rules of thumb as seen in figure 3 below.

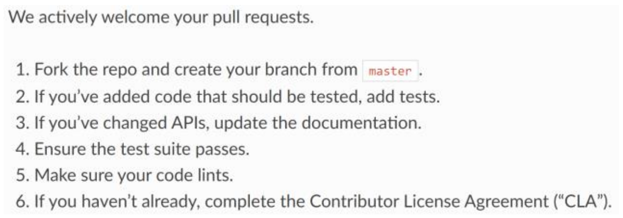


- `NGOpt` is "meta"-optimizer which adapts to the provided settings (budget, number of workers, parametrization) and should therefore be a good default.
- `TwoPointsDE` is excellent in many cases, including very high `num_workers`.
- `PortfolioDiscreteOnePlusOne` is excellent in discrete settings of mixed settings when high precision on parameters is not relevant; it's possibly a good choice for hyperparameter choice.
- `OnePlusOne` is a simple robust method for continuous parameters with `num_workers` < 8.
- `CMA` is excellent for control (e.g. neurocontrol) when the environment is not very noisy (num_workers ~50 ok) and when the budget is large (e.g. 1000 x the dimension).
- `TBPSA` is excellent for problems corrupted by noise, in particular overparameterized (neural) ones; very high `num_workers` ok).
- `PSO` is excellent in terms of robustness, high `num_workers` ok.
- `ScrHammersleySearchPlusMiddlePoint` is excellent for super parallel cases (fully one-shot, i.e. `num_workers` = budget included) or for very multimodal cases (such as some of our MLDA problems); don't use softmax with this optimizer.
- `RandomSearch` is the classical random search baseline; don't use softmax with this optimizer.

Figure 3: List of Optimisers with Comments from Nevergrad

3.1.2 Contributing a benchmark problem to Nevergrad

As seen in figure 5 below, the documentation includes some general instructions before making a pull request and thus making a contribution to Nevergrad.


We actively welcome your pull requests.

1. Fork the repo and create your branch from `master`.
2. If you've added code that should be tested, add tests.
3. If you've changed APIs, update the documentation.
4. Ensure the test suite passes.
5. Make sure your code lints.
6. If you haven't already, complete the Contributor License Agreement ("CLA").

Figure 4: Steps for Contributing to Nevergrad

Despite there being very detailed guidelines for contributing an optimisation algorithm to Nevergrad including where to put the code and which files to update and so forth, such guidelines do not exist for integrating a problem instance. This meant that reading of the source code had to be done to understand how problem functions are defined and run in Nevergrad.

From this source code, it was found that the problem implementation should be stored separately in a folder under the directory /nevergrad/functions. However, for Nevergrad to optimise against this problem, an experiment must be created in the file /nevergrad/benchmark/experiments.py. Experiment settings should be set such as the optimisers to use, budget and number of workers. The driver file of the problem should be imported in this file and called from the experiment. It is helpful to look at other experiments as a template. The driver file should also be edited such that the parameters to be optimised are defined using Nevergrad's parametrisation. See Parametrisation API reference in the documentation to see how this works, but again it is also helpful to look at driver files of other problems.

Although preliminary, this category is very critical to the project as it provides the basic knowledge on how to use Nevergrad. Before making extension to the platform, it is important to understand how the platform works and operates. Most of the reading has focused on the official documentation, source code and papers that present Nevergrad. Further reading included reading about researchers that have applied Nevergrad as a prominent tool in their work. This gave a deeper insight into the capabilities of Nevergrad and the type of work that it is used for.

3.2 Benchmarking Goals

It would be simple if there were some optimal optimisation algorithm that rendered all the other processes superfluous, but instead the incredibly large and ever-growing list of optimisation methods leads to the question of the best strategy. This presents the importance of benchmarking practices. The questions that are asked in benchmarking are essentially:
- how well does a certain algorithm perform on a given problem?
- why does an algorithm succeed/fail on a specific test problem?

Preparing a benchmark study can be intricate and complex. Specifying the goal of the study is very important as it provides the experimental setup being the choice of problem instances, algorithm instances, and performance criteria[6]. These three aspects are fundamental to every benchmark study and need to be clearly defined. However, the motivation for performing a benchmark study can be very diverse. Figure 5 below summarises the most common objectives for a benchmarking study.

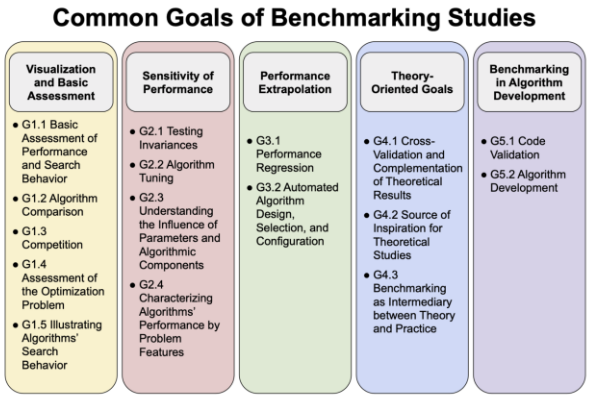

Common Goals of Benchmarking Studies
Visualization and Basic Assessment
G1.1 Basic Assessment of Performance and Search Behavior
G1.2 Algorithm Comparison
G1.3 Competition
G1.4 Assessment of the Optimization Problem
G1.5 Illustrating Algorithms' Search Behavior
Sensitivity of Performance
G2.1 Testing Invariances
G2.2 Algorithm Tuning
G2.3 Understanding the Influence of Parameters and Algorithmic Components
G2.4 Characterizing Algorithms' Performance by Problem Features
Performance Extrapolation
G3.1 Performance Regression
G3.2 Automated Algorithm Design, Selection, and Configuration
Theory-Oriented Goals
G4.1 Cross-Validation and Complementation of Theoretical Results
G4.2 Source of Inspiration for Theoretical Studies
G4.3 Benchmarking as Intermediary between Theory and Practice
Benchmarking in Algorithm Development
G5.1 Code Validation
G5.2 Algorithm Development

Figure 5: Common Goals of Benchmarking Studies

Arguably the most common goal of benchmarking, visualisation and basic assessment, aims to uncover which algorithms work well on a given problem. The study might involve comparing different algorithms on one or more problems and deciding which is the best. Another aspect of this type of goal may also be to assess the problem instance, as benchmarking optimisation heuristics can help to analyse the problem and to understand its characteristics.

The second type of goal, sensitivity of performance, can relate to testing invariances which is when benchmarking is used to test whether an algorithm possesses the correct invariances in terms of certain aspects to the problem. Algorithm tuning and understanding the influence of parameters and algorithmic components are both similar subsections whereby benchmarking is used both to understand how much influence certain parameters have on an algorithm and to also find the best configuration for a given problem.

The third type of goal, performance extrapolation, refers to a 'classical hope' in benchmarking where the generated data can be used to extrapolate the performance of an algorithm for other problem instances that haven't been tested. The hope is that for previously unseen problem instances, the benchmarking results can be utilised to design, select or configure an algorithm.

Theory-oriented goals can refer to using benchmarking studies in order to cross-validate or corroborate the reseults of a previous theoretical study. Vice-versa, empirical results from benchmarking studies can provide a source of inspiration for a theoretical study.

The final type of goal, benchmarking in algorithmic development, can mean to use benchmarking as a tool to verify whether a given program performs as expected, because if an algorithm doesn't perform as expected, then a source code review may be necessary. In addition to this, benchmarking can also be used to identify weak spots to develop better performing algorithms.

Overall, the paper titled Benchmarking in Optimisation: Best Practice and Open Issues is very well written and informative. I have learnt the most common goals for benchmarking as well as the key aspects that set up a good benchmarking study. Given that benchmarking is a significant part of my project, the paper is very relevant to understand the importance of the role of benchmarking in optimisation.

3.3 Problem Instance, Possible Extensions for Nevergrad

Selecting problem instances to add to Nevergrad is a careful process as it plays a key element in algorithm benchmarking. There may be real-world scenarios that present as interesting optimisation problems. However, there are many characteristics to a problem, and the more desirable a problem set is for benchmarking practices, the likelihood that the problem set is desirable for the Nevergrad platform also increases. The first desirable characteristic is that a problem must be diverse. This is because a problem that is difficult for one algorithm may be easy for another algorithm, so it is good to have a diverse problem set where the strengths and weaknesses for different algorithms are discovered. Another important property is that a problem set must be representative of its own problem class. This gives more credit when making claims on an algorithm's performance for the problem class at hand. A good problem scenario should also be scalable and tunable which means that characteristics of the problem, such as the dimensions and the number of objectives, should be adjustable. The final desirable property is that the problem already has known optimal solutions which makes it easier to measure the exact performance of algorithms in relation to the known optimal performance.

Keeping these desirable qualities in mind, there are a number of possible problem instances being investigated under the School of Computer Science, University of Adelaide that can be added to Nevergrad. Therefore, the papers in this section are highly helpful toward choosing which problems are suitable for contribution to Nevergrad. The problem instances which are being considered are described in the following section.

3.4 Benchmark Problems for Nevergrad Contribution

3.4.1 Team Pursuit Track Cycling Problem

Team pursuit track cycling is a sporting event that involves multiple team members that work together to complete a race in the minimum time possible. This takes place on an elliptical track called a velodrome. The cyclists in the team are allowed to take turns riding in the front position which allows the other cyclists to conserve energy through aerodynamic benefit and draft. Therefore, the team of cyclists are able to maintain a higher velocity compared to a single cyclist. The changes to the position of the cyclists are most efficiently made on either of the two banked turns of the track. The front cyclist moves up the bank and rejoins the team at the back[9].

Both men and women compete in this sporting event, however there are differences to the sporting rules for each gender. For the men's event, there are four cyclists in a team and the race totals 4000 metres over 16 laps. Only three of the four cyclists need to complete the race meaning one cyclist usually spends longer than the others in the first position and exhausts his energy and has to retire from the race. For the women's event, there are three cyclists in a team and the race is shorter with 12 laps totaling 3000 metres. However, all three cyclists need to complete the race which means no one can retire and the workload is usually more evenly distributed across the team.

There are many ways to improve the performance of a track cycling team such as the physiological and psychological abilities of the team members and the technical factors of the bicycles. However, the optimisation problem that can be worked lies in the pacing and transition strategy of the cyclists. The pacing strategy (set of continuous variables) describes the power of the cyclist in the first position in between each transition, i.e. each race segment. The transition strategy (set of discrete variables) describes the positions where a transition is to happen such that it can only occur on a banked turn of the track[10].

Overall, the optimisation problem is very complex and is multi-objective where the primary is to minimise the total time needed for a team of cyclists to complete a race. A secondary objective is to maximise the amount of energy the riders have at the end of the race. It is also an interesting problem given that it requires the optimisation of both continuous and discrete variables and contains separate instances for men and women.

3.4.1 Travelling Thief Problem

This problem has been introduced recently and is comparable to real-world problems. It is a combination of the Travelling Salesperson Problem and the Knapsack Problem. There are n cities, and the distance matrix is given. Also, there are m items each of them having a value and weight. Every city contains an item except for the first one. There is a thief who is going to visit these cities exactly once and pick some items from the cities and fill his knapsack which has a maximum weight before returning to the first city. A renting rate R is to be paid per each time unit being on a way. The thief travels at a speed relevant to the weight of the knapsack. The goal is to find a tour of maximal profit[7].

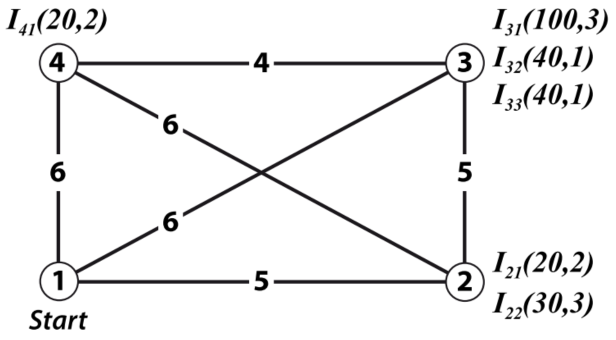


Figure 6: Illustrative Example of TTP

As seen in figure 6, the problem is presented numerically. Each node except the first has a set of items. For example, node 2 has the items I_{21} with profit 20 and weight 2, and item I_{22} with profit 30 and weight 3. Let's assume the problem constraints being a maximum knapsack weight W = 3, renting rate R = 1, minimum speed $v_{min} = 0.1$ and maximum speed $v_{max} = 1$. A tour can be represented as a vector of nodes to be visited in the order T = (x_1, …., x_n) and a packing plan can be represented as a vector of boolean numbers indicating which items to pick up. Therefore, the optimum objective function for the illustrated case is F(T,P) = 50 where T = (1, 2, 4, 3) and P = (0, 0, 0, 1, 1, 0). Hence, the optimal tour is for the thief to travel from city 1 to city 2 then 4 without picking up any items. Next it travels to city 3 where it picks up items I_{32} and I_{33} for a total profit of 80. However, the knapsack now has a weight of 2 which slows the speed of the thief the last part of the tour costs 15 instead of 6. The first 3 parts of the tour have a total cost of 15. So the final objective value is F(T,P) = 80 – 15 – 15 = 50.

Overall, this category of papers is extremely relevant to my project as they provide the crucial understanding of both problems to be implemented and integrated into Nevergrad. The knowledge and understanding I gain about these problems not only helps with the implementation of the problems, further increases my understanding of why these problems are great candidates to be integrated into Nevergrad for future benchmark purposes.

4 METHODOLOGY

The primary goal for this project was to provide useful and insightful contributions to Nevergrad that hopefully improve and broadens its functionality. The methodology to achieve this was to research relevant problem instances that serve as a ground for good benchmarking practices. The original goal was to integrate three appropriate benchmark problems (developed and researched by Adelaide University). The original timetable is shown in table 1.0 below.

Table 1: Original Project Timetable

Week	Description
1	Gain a working understanding of Nevergrad through reading papers and becoming familiar with the source code and how to run it.
2	
3	
4	Gain a working understanding of benchmarking and various problem instances which can be used to extend Nevergrad.
5	
6	Implement and integrate first problem instance – Team Pursuit Track Cycling Problem
7	
8	Implement and integrate second problem instance – Travelling Thief Problem
9	
10	Implement and integrate third problem instance – Wave Energy Converter Problem
11	
12	Finalise the project – ensure all contributions are submitted through a GIT pull request. Write final paper.
13	

This was a best-case scenario timeline for the project, however some challenges arose which meant this couldn't be achieved. Fortunately, the methodology was flexibly designed to accommodate for such contingencies. The plan was simply for less benchmark problems to be integrated into Nevergrad to allow for more time to overcome challenges.

The problem instances were implemented in Java which meant a conversion to Python was necessary, since Nevergrad is a Python package. This presented the challenge of learning Java, and subsequently the conceptual differences between Python and Java, so that I could write efficient Python code that matched the functionality of the Java code. Another challenge was integrating the problems into Nevergrad. Despite there being clear steps to contribute new optimisation algorithms to Nevergrad, there were no guidelines for contributing a problem, and the large repository meant it wasn't obvious where to place the code. It was also a challenge to understand how experiments and parametrisation worked so that the problem could be run inside Nevergrad. Overall, these challenges consequently resulted in only two problems being contributed (not officially as of yet) to Nevergrad, the Team Pursuit Track Cycling Problem and the Travelling Thief Problem.

A secondary goal for the project was to use Nevergrad to optimise at least one of the problems that had been integrated. This involved writing experiments in Nevergrad and utilising different optimisers and experiment settings. This was completed for the Team Pursuit Track Cycling problem and is explained in further detail in the experimental setup.

Another goal of the project is somewhat of an extension and refers to the Open Optimisation Competition. All submissions to Nevergrad through a pull request are eligible. However, for a contribution to be officially added to the Nevergrad source code, it has to be approved by the maintainers of Nevergrad. Subsequently, the contribution be entered into the competition. Therefore, the first challenge will be providing a contribution that is worthy enough to first be officially added to Nevergrad and the competition. The second goal would then be to produce a contribution so worthy that it yields a reward from the competition. The challenge with this goal is beating the majority of the other competitors as it is likely that the other contributors are highly skilled researchers that no doubt have great ideas for extending Nevergrad. The submission deadline is 30 September, 2021.

5 EXPERIMENTAL SETUP

The first step in the experimental setup is to calculate the original results of the Team Pursuit Track Cycling Problem. This is simply calculated using default settings of the pacing strategy and transition strategy. For the men, the default pacing strategy is 550 cycling power for every racing segment and 400 for the women. The default transition strategy alternates between true and false for both the men and women.

The next step is to write an experiment in Nevergrad defining the optimisers, budget, and number of workers. Nevergrad contains many optimisers, but three basic optimisers called NGOpt10, CMA and DE were chosen as they ran smoothly and produced slightly different results opening the door for comparison. When optimising solely the pacing strategy or transition strategy, a small budget (say 1000) is sufficient. However, when optimising both the pacing and transition strategy, the search space becomes considerably larger and so a larger budget is required. For simplicity and fair comparisons to be made across the problem formulations, the budget was set to a reasonable size of 3000, and the number of workers was set to 10.

For each optimiser and problem formulation, 10 runs were completed, and the best score was recorded for each repetition. Standard statistic results can then be calculated such as the mean, standard deviation and best overall score. Violin plots were created using the Seaborn library in the Jupyter Notebook environment. These plots visualized the distribution and probability density of the data, and was useful to compare the performance of the optimisation algorithms within each problem formulation.

It must be noted that while this experimental setup does allow for some comparison between three optimisers, it is less of a priority with the main goal being to ensure the problem is properly integrated into Nevergrad for future benchmarking studies.

6 RESULTS

6.1 Experiment Results and Observations

Using the default settings for the pacing and transition strategy resulted in scores: Men's - 268.1 and Women's - 219.72. The term score is used for the purpose of comparing the optimisers, however represents the time (in seconds) that it takes for the team to finish a race.

Table 2: Optimisation of Men's Pacing Strategy

	NGOpt10	**CMA**	**DE**
1	266.8	266.6	262.4
2	266.8	266.6	258.5
3	266.6	266.8	258.8
4	266.6	266.8	261.4
5	266.8	266.6	259.2
6	266.8	266.8	260.8
7	266.6	266.6	261
8	266.6	266.6	261.2
9	266.6	266.8	262.4
10	266.6	266.6	259.2
Mean	266.68	266.68	260.49
S. Dev	0.09798	0.09798	1.383076
Best	266.6	266.6	258.5

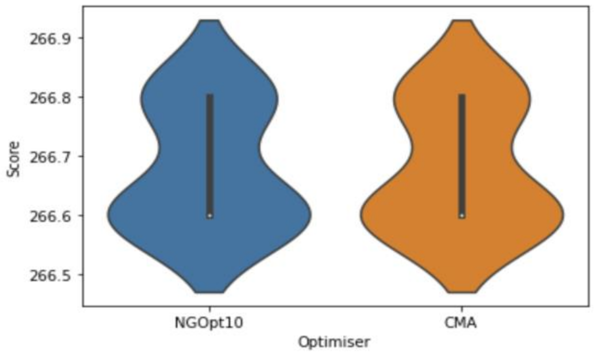


Figure 7: Optimisation of Men's Pacing Strategy, NGOpt and CMA optimisers

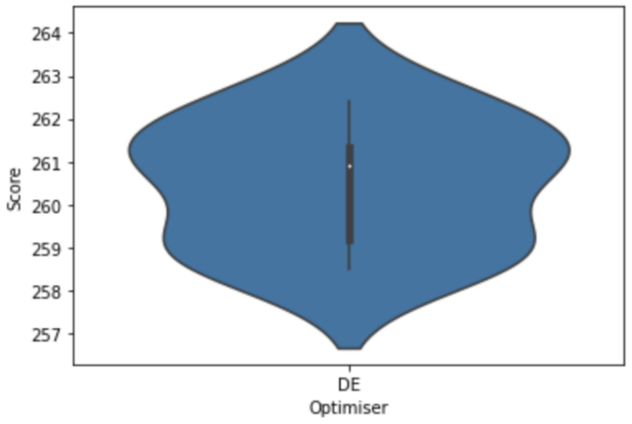


Figure 8: Optimisation of Men's Pacing Strategy, DE optimiser

The violin plot for this problem formulation was split into two figures as the results varied to the extent that the plot became warped and was difficult to visualize.

As seen in figure 6 and 7, NGOpt and CMA produced very extremely similar results with a mean score of 266.6. Despite being less accurate with a much large standard deviation, the DE optimiser considerably outperformed the other two optimisers with a mean score of 260.49.

Table 3: Optimising Men's Transition Strategy

	NGOpt10	**CMA**	**DE**
1	263.28	264.24	264.2
2	264.74	262.7	263.52
3	264.16	262.72	263.4
4	263.3	263.42	263.22
5	263.1	264.5	263.74
6	262.44	262.9	263.36
7	263.72	263.28	263.56
8	263.08	263.52	263.3
9	263.64	263.56	263.3
10	264.52	263.52	263.26
Mean	263.598	263.436	263.486
S. Dev	0.673347	0.563404	0.282
Best	262.44	262.7	263.22

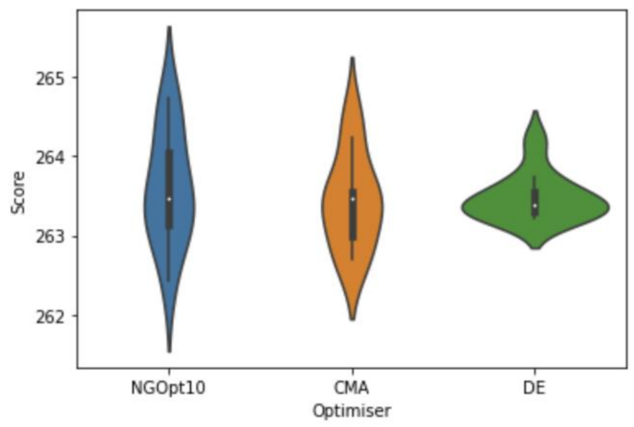


Figure 9: Optimisation of Men's Transition Strategy

In contrast to the previous problem formulation (optimising pacing strategy), figure 8 demonstrated that DE was the most consistent of the optimisers on this occasion. Overall, all three optimisers performed at a similar level with all achieving a mean score around 263.5.

Table 4: Optimisation of Men's Pacing and Transition Strategy

	NGOpt10	CMA	DE
1	340.2	343.24	341.12
2	340.06	339.44	344.44
3	340.84	342.64	349.84
4	341	341.06	348.78
5	344.42	342.74	348.18
6	343.72	343.74	346.66
7	338.16	338.74	350.62
8	343.2	342.98	350.62
9	341.48	340.94	349.94
10	344.3	344.16	348.94
Mean	341.738	341.968	347.914
S. Dev	1.980494	1.741452	2.900552
Best	338.16	338.74	341.12

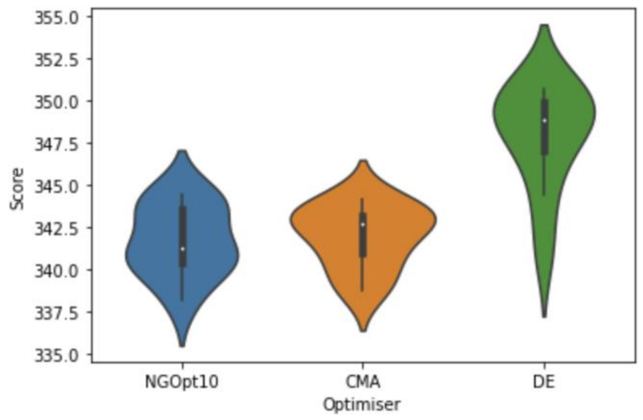


Figure 10: Optimisation of Men's Pacing and Transition Strategy

As seen in figure 9, NGOpt10 and CMA both had similar performance with their mean and best scores almost the same. The DE algorithm did not perform as well as the other two optimisers on this occasion.

Table 5: Optimisation of Women's Pacing Strategy

	NGOpt10	CMA	DE
1	217.12	217.12	Inf
2	217.12	217.12	Inf
3	217.12	217.12	Inf
4	217.12	217.12	Inf
5	217.12	217.12	Inf
6	217.12	217.12	Inf
7	217.12	217.12	Inf
8	217.12	217.12	Inf
9	217.12	217.12	Inf
10	217.12	217.12	Inf
Mean	217.12	217.12	N/A
S. Dev	0	0	N/A
Best	217.12	217.12	Inf

As seen in table 5, the results produced for this problem formulation were quite unexpected. The NGOpt10 and CMA algorithms had identical performance and produced the same optimised score of 217.12 for each test run. However, the strangest result was that the DE algorithm was not able to find a valid score despite being the best performer for the men's version of this problem formulation, i.e. optimisation of men's pacing strategy.

Given these results, the data was not sufficient to produce any visualisations.

Table 6: Optimisation of Women's Transition Strategy

	NGOpt10	CMA	DE
1	219.32	216.68	215.28
2	218	215.32	215.36
3	216.52	216.32	215.48
4	216.32	216.68	216.92
5	216.64	216.76	215.16
6	214.08	214.08	215.52
7	216.68	214.28	216.52
8	214.76	216.48	214.36
9	218.4	216.72	216.56
10	216.52	216.88	215.16
Mean	216.724	216.02	215.632
S. Dev	1.492643	1.011494	0.748716
Best	214.08	214.08	214.36

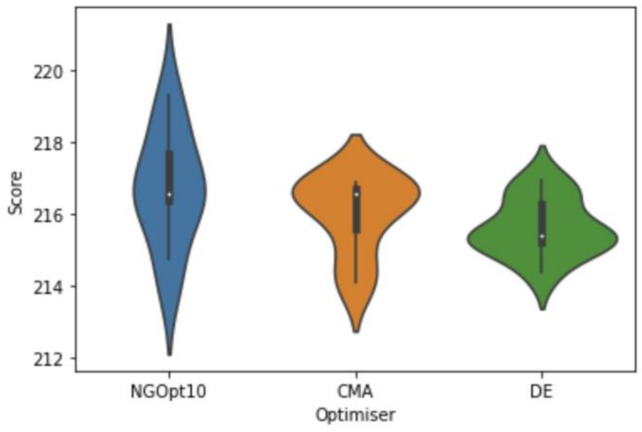


Figure 11: Optimisation of Women's Transition Strategy

As seen in figure 10, all optimisers performed relatively similar with mean scores ranging from 215.6 to 216.8 in addition to very similar overall best scores. The main differences lied in the consistency with the NGOpt10 optimiser being the least consistent while the DE optimiser was the most consistent.

Table 7: Optimisation of Women's Pacing and Transition Strategy

	NGOpt10	**CMA**	**DE**
1	247.34	246.5	251.88
2	248.3	248.26	252.06
3	247.84	248.88	253.82
4	249.06	248.8	250.14
5	251.28	247.16	252.02
6	249.18	247.22	253.4
7	249.1	248.24	251.72
8	249.5	249.5	252.8
9	249.32	250.6	254.62
10	247.7	250.32	250.36
Mean	248.862	248.548	252.282
S. Dev	1.08109	1.284249	1.350095
Best	247.34	246.5	250.14

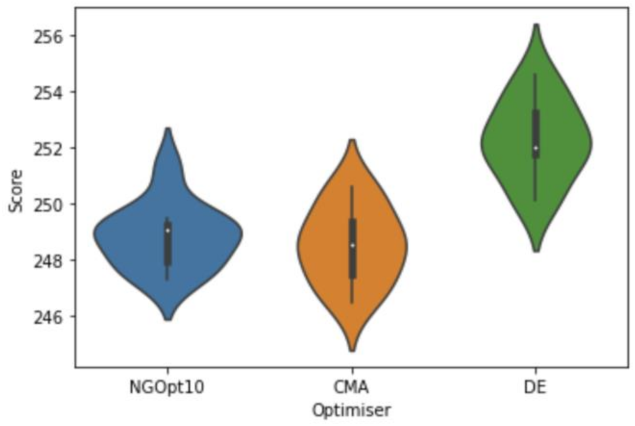


Figure 12: Optimisation of Women's Pacing and Transition Strategy

As seen in figure 11, the overall performance of the optimisers mirrored the results of the men's version of this problem formulation. NGOpt10 and CMA both had similar performance with their mean and best scores almost the same. The DE algorithm did not perform as well as the other two optimisers.

6.2 Final Discussion

Overall, there was not one optimiser that stood out as a 'clear winner' across all the problem formulations. The DE optimiser did not show any trends in its performance. On some occasions, it performed with great success, especially in the case of optimising the men's pacing strategy. However, other times it would perform poorly. One clear trend is that the NGOpt10 and CMA optimisers both had similar performance across all the problem formulations.

The best score found for the men's team was 262.44 (default=268.1) which came from optimising the men's transition strategy. The best score for the women's team 214.08 (default=219.72) which was also a result of solely optimising the transition strategy. This was not expected as it was assumed that optimising both the pacing strategy and transition strategy simultaneously would produce the best results. However, this produced the worst results; around 80 seconds slower for the men and 40 seconds slower for the women.

This could be explained by the much larger search space since the number of variables to optimise is almost doubled for each gender. As stated previously, roughly half these variables are continuous for the pacing strategy, and the other half are discrete for the transition strategy. However, Nevergrad interpreted all the variables as continuous and some 'rabbit code' was used to implement the transition strategy. Hence, the parameters were not setup in the perhaps the correct way which may have had a negative effect on the optimisers' performance.

Another explanation could be that simply the optimisers chosen were not the best and that other optimisers may better handle the larger search space. In addition, perhaps the budget was set too low as only one budget was used being 5000. Yet another note to consider is that the problem is also capable of outputting more information that would help the optimisers. The energy remaining of each rider can be passed to the optimiser. This variable would need to be maximised and serves as a secondary objective.

7 CODE

GitHub Link: https://github.com/facebookresearch/nevergrad.git

The above GitHub link is for Nevergrad. The code described in this paper can be found in the following directories:

/nevergrad/functions/cycling – implementation of Team Track Cycling Problem

/nevergrad/functions/ttp – implementation of Travelling Thief Problem

/nevergrad/benchmark/experiments.py (cycling and ttp functions) – experiments for both problems.

It must be noted that at the time of writing, neither problem instances have been officially contributed to Nevergrad. Both problems should be officially integrated by the end of September 2021.

8 CONCLUSIONS

Developed by Facebook, Nevergrad is a Python library that supports derivative-free and evolutionary optimisation. The platform is growing in popularity and is supported by a strong community of machine learning scientists and other researchers. The maintainers of Nevergrad strongly welcome contributions to the platform in the form of new algorithms, benchmark problems and other ideas. This project focused on the contribution of two benchmark problems, the Team Pursuit Track Cycling Problem and the Travelling Thief Problem, both of which have been thoroughly researched and developed by the School of Computer Science, University of Adelaide.

These two additions will hopefully broaden the scope and application of Nevergrad by introducing new and interesting problem scenarios for future benchmarking studies. The problem instances have not yet been officially contributed to Nevergrad, however will be submitted by the end of September, 2021. The Team Pursuit Track Cycling Problem will be the first to be submitted as it has already essentially been integrated and experimented upon with Nevergrad's API which produced some interesting results.

Due to the six different problem formulations, the results highlighted the complexity of the cycling problem, and why it is a good benchmark problem to be integrated into Nevergrad. It was a simple experiment phase, however the goal was not so much to find the best solution and best optimiser, but rather ensure the problem was integrated correctly into Nevergrad and provide a basis for future benchmarking studies. There is clearly an opportunity for further work which includes testing more optimisers, budgets, and incorporating the second objective of maximising the energy remaining for each cyclist after the race.

REFERENCES

[1] Choudhury, A., 2021. *FB's New Python Library Nevergrad Provides A Collection Of Algorithms That Don't Require Gradient Computation*. Analytics India Magazine. Available at: https://analyticsindiamag.com/fbs-new-python-library-nevergrad-provides-a-collection-of-algorithms-that-dont-require-gradient-computation/

[2] Ai.facebook.com. 2020. *Nevergrad, an evolutionary optimization platform, adds new key features*. Available at: https://ai.facebook.com/blog/nevergrad-an-evolutionary-optimization-platform-adds-new-key-features/

[3] Facebookresearch.github.io. 2019. *Nevergrad - A gradient-free optimization platform — nevergrad documentation*. Available at: https://facebookresearch.github.io/nevergrad/index.html

[4] *Rapin, J., Bennet, P., Centeno, E., Haziza, D., Moreau, A. and Teytaud, O., 2020.* Open Source Evolutionary Structured Optimization. *In Genetic and Evolutionary Computation Conference Companion (GECCO '20 Companion).*

[5] Www-ia.lip6.fr. 2021. *Open Optimization Competition 2021*. Available at: http://www-ia.lip6.fr/~doerr/OpenOptimizationCompetition2021.html

[6] Bartz-Beielstein, T., Doerr, C., van den Berg, D., Bossek, J., Chandrasekaran, S., Eftimov, T., Fischbach, A., Kerschke, P., La Cava, W., Lopez-Ibanez, M., Malan, K., Moore, J., Naujoks, B., Orzechowski, P., Volz, V., Wagner, M. and Weise, T., 2020. *Benchmarking in Optimization: Best Practice and Open Issues*.

[7] Bonyadi, M., Michalewicz, Z. and Barone, L., 2013. The travelling thief problem: the first step in the transition from theoretical problems to realistic problems. Congress on Evolutionary Computation.

[8] Michalewicz, Z., Neumann., Polyakovskiy, S,. Reza, M., Wagner, M., 2014. A Comprehensive Benchmark Set and Heuristics for the Traveling Thief Problem

[9] Nevergrad : A Python toolbox for performing gradient-free optimization. *2019. Available at: https://www.techleer.com/articles/576-nevergrad-a-python-toolbox-for-performing-gradient-free-optimization/#:~:text=Nevergrad%20is%20an%20open%2Dsource,%2Dand%2Dtell%20Python%20framework.*

[10] Wagner, M., 2016. Nested Multi- and Many-Objective Optimization of Team Track Pursuit Cycling. Frontiers in Applied Mathematics and Statistics.

[11] Day, J., Jordan, D., Kroeger, T., Neumann, F., Wagner, M., 2013. Evolving Pacing Strategies for Team Pursuit Track Cycling. Advances in Metaheuristics.

[12] Cs.adelaide.edu.au. 2021. *Optimisation and Logistics (School of Computer Science, The University of Adelaide)*. Available at: https://cs.adelaide.edu.au/~optlog/research/

[13] Teytaud, O. and Rapin, J., 2021. *Nevergrad: An open source tool for derivative-free optimization - Facebook Engineering*. [online] Facebook Engineering. Available at: https://engineering.fb.com/2018/12/20/ai-research/nevergrad/