Hyperparameter Optimization for Evolutionary Algorithms

[Vlad Covali]

1 Introduction and Project Overview

Evolutionary algorithms (EAs) are powerful optimization techniques inspired by natural selection and evolution. They have proven effective in solving complex problems across various fields like engineering, computer science, and biology. However, the performance of these algorithms often depends heavily on careful tuning of their hyperparameters. These hyperparameters, such as population size, mutation rate, and selection pressure, significantly influence how well the algorithm finds optimal solutions.

Manually tuning these hyperparameters is time-consuming and requires a lot of expertise. What makes it even trickier is that the best set of hyperparameters can change depending on the specific problem being solved. This challenge has been a major hurdle in making evolutionary algorithms more widely used in real-world applications.

For example, when using a genetic algorithm to optimize the shape of a car for better aerodynamics, the algorithm's ability to find the best design stongly depends on settings like how often it combines different designs (crossover rate) and how much it randomly changes designs (mutation rate). If these settings are not right, the algorithm might get stuck with a not-so-great design or keep changing designs without actually improving them.

The relationship between hyperparameters and algorithm performance is not straightforward. Sometimes, a small change in a hyperparameter can make a big difference in the results. Also, settings that work well for one type of problem might not work well for another. This means we need a smart way to find the best hyperparameters for each specific problem.

Given these challenges, there is a real need for automated methods to find the best hyperparameters for evolutionary algorithms. If we can develop good techniques for this, it would save a lot of time and potentially lead to much better results across many different applications. It would also make evolutionary algorithms easier to use for people who are not experts in the field.

This project aims to address this need by studying and implementing various hyperparameter optimization techniques for evolutionary algorithms. We will explore how different optimization methods perform with various evolutionary algorithms and try to find the best ways to tune these algorithms for different

types of problems.

2 Selection of Hyperparameter Optimization Algorithms

The core of our project revolves around the effective tuning of evolutionary algorithms. To achieve this, we have selected four diverse and powerful hyperparameter optimization algorithms (HOAs). Each of these algorithms brings unique strengths to the task of finding optimal hyperparameter configurations.

2.1 Covariance Matrix Adaptation Evolution Strategy (CMA-ES)

CMA-ES [1] is a state-of-the-art optimization algorithm particularly well-suited for continuous parameter spaces.

Key Features:

- adaptive step size control for efficient search,
- covariance matrix adaptation to capture parameter dependencies,
- invariance to linear transformations of the search space,

Reasoning: We have chosen CMA-ES for its proven track record in optimizing continuous parameters. Its ability to adapt its search strategy makes it particularly effective for tuning numerical hyperparameters like learning rates, population sizes, or mutation probabilities.

Use Case: In tuning a Differential Evolution algorithm, CMA-ES could efficiently optimize the scale factor and crossover rate, potentially finding an optimal balance between exploration and exploitation much faster than grid search or random search methods.

2.2 Simulated Annealing (SA)

Simulated Annealing [2] is a probabilistic technique for approximating the global optimum of a given function, inspired by the annealing process in metallurgy.

Key Features:

- probabilistic technique for approximating global optima
- can escape local optima through occasional "uphill" moves
- effective for both continuous and discrete parameter spaces
- annealing schedule provides a balance between exploration and exploitation

Reasoning: Simulated Annealing is a versatile optimization algorithm that can be effectively implemented within the Pagmo2 framework. Its ability to escape local optima makes it well-suited for the complex landscapes often encountered in hyperparameter optimization.

Use Case: SA could be used to optimize a wide range of hyperparameters in evolutionary algorithms. For example, it could tune the tournament size, crossover rate, and mutation rate in a genetic algorithm, effectively balancing between exploration of the parameter space and exploitation of good solutions.

2.3 Particle Swarm Optimization for Hyperparameter Tuning (PSO-HT)

PSO-HT [3][4] adapts the well-known Particle Swarm Optimization algorithm specifically for the task of hyperparameter tuning.

Key Features:

- adapts the PSO algorithm specifically for hyperparameter optimization,
- can handle mixed continuous and discrete parameter spaces,
- parallelizable, aligning well with Pagmo2's capabilities.

Reasoning: Implementing PSO-HT leverages Pagmo2's existing PSO implementation, making it a practical choice. It provides an interesting meta-optimization scenario, using an evolutionary algorithm to optimize evolutionary algorithms.

Use Case: PSO-HT could be used to tune the parameters of other evolutionary algorithms, such as Differential Evolution. Its ability to handle mixed parameter spaces makes it versatile for various EA configurations.

2.4 Nelder-Mead Simplex Method

The Nelder-Mead method [5] is a simple yet effective direct search method for multidimensional unconstrained optimization.

Key Features:

- derivative-free optimization method,
- effective for low to moderate-dimensional problems,
- simple to implement and understand.

Reasoning: The Nelder-Mead method is straightforward to implement within the Pagmo2 framework. Its simplicity provides a good baseline for comparison with more sophisticated methods.

Use Case: Nelder-Mead could be used to fine-tune a small number of critical parameters in an evolutionary algorithm, such as the crossover and mutation rates in a genetic algorithm. Its performance on these simpler optimization tasks would provide an interesting comparison to the more complex methods.

3 Selection of Evolutionary Algorithms

With our hyperparameter optimization methods chosen, we now turn to selecting the evolutionary algorithms (EAs) that we will be optimizing. We have chosen four diverse algorithms that represent different approaches within evolutionary computation.

3.1 Differential Evolution (DE)

DE [6] is a powerful algorithm for continuous optimization problems.

Key Features:

- simple yet effective mutation strategy,
- self-referential population evolution,
- strong performance on many continuous optimization benchmarks.

Reasoning: We have chosen DE for its effectiveness in continuous optimization and its sensitivity to hyperparameter settings. The interplay between its scale factor and crossover rate provides an interesting optimization challenge for our HOAs.

Use Case: DE could be applied to engineering design problems, such as optimizing the shape of an airfoil for minimal drag. The performance of DE on such a problem could be significantly improved by finding optimal hyperparameter settings.

3.2 Particle Swarm Optimization (PSO)

PSO [7] is inspired by social behavior models and is effective for both continuous and discrete optimization.

Key Features:

- inspired by bird flocking and fish schooling behaviors,
- balances global and local search through social and cognitive components,
- effective on a wide range of problem types.

Reasoning: We have selected PSO for its different approach to population-based optimization compared to genetic algorithms. Its performance is highly dependent on the balance between exploration and exploitation, which is controlled by its hyperparameters.

Use Case: PSO could be used for training neural networks, where the particles represent different network weights. Optimizing PSO's hyperparameters could lead to faster and more reliable network training.

3.3 Self-adaptive Differential Evolution (jDE)

jDE [8] is an advanced variant of DE that automatically adapts its control parameters.

Key Features:

- automatically adjusts scale factor and crossover rate during the optimization process
- can adapt to different phases of the search process
- often outperforms standard DE on complex problems

Reasoning: We have chosen jDE to explore how our hyperparameter optimization methods perform on an algorithm that already includes some self-adaptation. This presents an interesting challenge: can our HOAs find initial parameter settings or adaptation rates that improve upon jDE's built-in adaptation?

Use Case: jDE could be applied to complex, multi-modal optimization problems in fields like computational chemistry, where the energy landscape of molecular configurations is being explored. Optimizing jDE's hyperparameters could lead to more efficient exploration of these complex landscapes.

3.4 Simple Genetic Algorithm (SGA)

SGA [9] is a classic evolutionary algorithm that forms the foundation of many other evolutionary methods.

Key Features:

- uses biological inspired operators: selection, crossover, and mutation
- flexible representation allows application to a wide range of problem types
- simple to implement but can be powerful when well-tuned

Reasoning: We have included SGA as a baseline algorithm and because its simplicity allows for clear interpretation of how hyperparameter changes affect performance. It also provides a good contrast to the more complex algorithms in our study.

Use Case: SGA could be applied to discrete optimization problems like job shop scheduling. By optimizing the SGA's hyperparameters, we could potentially improve its performance to be competitive with more complex algorithms on these types of problems.

4 Selection of Implementation Framework

To effectively implement and evaluate our chosen hyperparameter optimization algorithms (HOAs) and evolutionary algorithms (EAs), we require a robust and flexible implementation framework. After careful consideration of our project's

specific needs and the characteristics of our selected algorithms, we identified several key requirements:

1. **Support for Diverse Evolutionary Algorithms:** The framework must support various evolutionary algorithms, including Differential Evolution (DE), Particle Swarm Optimization (PSO), jDE, and Simple Genetic Algorithm (SGA).
2. **Flexibility to Implement Custom Algorithms:** The ability to integrate and implement custom algorithms is essential, particularly for our HOAs such as Covariance Matrix Adaptation Evolution Strategy (CMA-ES), Simulated Annealing, PSO Hyper-Tuning (PSO-HT), and the Nelder-Mead Simplex Method.
3. **Handling of Parameter Spaces:** The framework should efficiently handle continuous, discrete, and mixed parameter spaces to meet the diverse needs of our HOAs and EAs.
4. **Parallel and Distributed Computing Capabilities:** To address the computational demands of hyperparameter optimization, the chosen framework must support parallel and distributed computing.
5. **Modern Implementation with C++:** A framework utilizing recent C++ standards ensures high performance, maintainability, and compatibility with our existing development tools.
6. **Active Development and Community Support:** Active maintenance and a supportive community are crucial for long-term viability and access to new features and updates.

Based on these criteria, we evaluated several popular evolutionary algorithm libraries: BEAGLE, Pagmo2, Darwin, Evolving Objects (EO), EALib, and Shark. Below is a detailed comparison of these libraries and the reasoning behind our selection of Pagmo2 as our implementation framework.

4.1 Evaluation of Frameworks

4.1.1 BEAGLE

Features: BEAGLE is a comprehensive, object-oriented framework for evolutionary algorithms, supporting genetic algorithms, evolution strategies, and multi-objective optimization.
Pros:

- **High Flexibility:** BEAGLE is highly customizable, making it suitable for experimenting with different evolutionary algorithms and custom adaptations.
- **Parallel Processing:** Built-in parallel processing capabilities are beneficial for computationally intensive tasks.

Cons:

- **Complexity:** Its broad feature set comes at the cost of a steep learning curve, making it challenging to use efficiently without prior experience.
- **Lack of Active Maintenance:** BEAGLE is no longer actively maintained, raising concerns about long-term stability and the availability of updates.

Conclusion: While BEAGLE's flexibility and parallel capabilities are strong assets, the lack of maintenance and its complexity make it unsuitable for a project requiring ongoing support and easy adaptability.

4.1.2 Pagmo2

Features: Pagmo2 is a cutting-edge C++ library aimed at parallel global optimization. It is actively maintained by the European Space Agency (ESA) and has Python bindings through PyGMO.
Pros:

- **Algorithm Support:** Offers built-in implementations of our selected EAs, such as DE, PSO, and variants of DE that can be adapted for jDE.
- **Extensibility:** Its modular design allows for the straightforward addition of custom algorithms, critical for implementing our HOAs.
- **Parameter Handling:** Pagmo2 has robust support for handling continuous, discrete, and mixed parameter spaces, which is crucial for the diversity of our optimization tasks.
- **Parallelization:** Provides built-in support for parallel and distributed computing, vital for managing the computational load of hyperparameter optimization.
- **Modern Implementation:** Uses the latest C++ standards, enhancing performance and maintainability, which is ideal for the complex algorithms in our project.
- **Active Development:** Backed by ESA, Pagmo2 has a growing community and is actively maintained, ensuring a stable, evolving library.

Cons:

- **Learning Curve:** The need for familiarity with parallel computing concepts makes it challenging for users unfamiliar with these areas.

Conclusion: Despite its steeper learning curve, Pagmo2's extensive features, active development, and focus on evolutionary optimization make it the most suitable choice for our project.

4.1.3 Darwin

Features: Darwin is a simple library focusing mainly on genetic algorithms. It aims to be user-friendly, with limited support for other evolutionary algorithms.
Pros:

- **Ease of Use:** Simple to set up and use, making it ideal for small-scale or straightforward genetic algorithm implementations.

Cons:

- **Limited Scope:** Darwin lacks the necessary features to support diverse evolutionary algorithms beyond genetic algorithms.
- **Minimal Community and Documentation:** Sparse documentation and limited community support make troubleshooting and extending its capabilities challenging.

Conclusion: While Darwin's simplicity is an advantage for small projects, it lacks the versatility and community support required for the breadth of evolutionary algorithms needed in our project.

4.1.4 Evolving Objects (EO)

Features: EO is a flexible library that supports various evolutionary algorithms, including genetic algorithms and evolution strategies. It is designed to be easy to extend.
Pros:

- **Flexibility:** EO is well-suited for users needing to extend and customize evolutionary algorithms.
- **Decent Documentation:** Offers sufficient documentation and examples for learning and experimentation.

Cons:

- **Setup Complexity:** Requires considerable setup and configuration for implementing complex algorithms, which could hinder efficiency in prototyping.
- **Limited Parallel Support:** While it is extendable, EO lacks built-in parallel computing capabilities that are essential for large-scale optimization tasks.

Conclusion: EO's flexibility makes it a good candidate for custom evolutionary algorithm implementations. However, the lack of parallel computing support and complex setup process limit its utility for our project.

4.1.5 EALib

Features: EALib is a lightweight library offering basic tools for evolutionary algorithms.
Pros:

- **Simplicity:** Very easy to learn and implement, suitable for simple evolutionary algorithm applications.

Cons:

- **Lack of Advanced Features:** Does not offer the features required for implementing sophisticated evolutionary algorithms or HOAs.
- **Not Suitable for Complex Projects:** EALib's basic nature makes it unsuitable for handling the variety and complexity of our optimization needs.

Conclusion: EALib's simplicity makes it accessible, but the lack of advanced features and flexibility renders it insufficient for our project's requirements.

4.1.6 Shark

Features: Shark is a comprehensive library designed for machine learning and optimization, including a limited selection of evolutionary algorithms.
Pros:

- **Versatility:** Includes a wide range of optimization and machine learning algorithms, making it an all-in-one solution for hybrid tasks.
- **Good Documentation:** Well-documented and actively maintained, with a strong community of users.

Cons:

- **Machine Learning Oriented:** The focus on machine learning means evolutionary algorithms are not the primary strength of Shark, resulting in limited support and less flexibility for evolving custom EA variations.
- **Complexity:** Its broad scope makes it more complex to navigate, with many features that are unnecessary for a pure evolutionary algorithm project.

Conclusion: Shark's versatility and strong support are advantageous for mixed machine learning projects, but the limited focus on evolutionary algorithms and added complexity make it less suitable for our evolutionary algorithm-centric project.

4.2 Selection Justification

After careful evaluation, **Pagmo2** emerged as the most suitable framework for our project due to the following reasons:

- **Extensive Algorithm Support:** It provides built-in implementations of our chosen evolutionary algorithms and allows for easy customization and extension, crucial for integrating our hyperparameter optimization algorithms.
- **Parallel Computing Capabilities:** Pagmo2's support for parallel and distributed computing will enable efficient handling of computationally intensive tasks, a significant requirement given the complexity of our optimization efforts.
- **Modern and Actively Maintained:** Being actively developed by the European Space Agency, Pagmo2 is a stable and evolving framework, providing the security of continuous updates, bug fixes, and access to community support.

Although Pagmo2 has a steeper learning curve compared to simpler frameworks, its advanced features, flexibility, and alignment with our project's needs make it the ideal choice. Its robust architecture and active support will allow us to efficiently implement, test, and compare our hyperparameter optimization algorithms across various evolutionary algorithms and problem types, ensuring the success of our project.

5 Implementation Strategy and Expected Outcomes

5.1 Implementation Strategy

1. **HOA Implementation:**
    - Develop a common interface for all HOAs, ensuring compatibility with Pagmo2's optimization framework.
    - Implement CMA-ES, leveraging Pagmo2's existing evolutionary strategy components where possible.
    - Create SA implementation, developing the annealing schedule and acceptance probability logic within Pagmo2.
    - Develop Particle Swarm Optimization for Hyperparameter Tuning (PSO-HT), extending Pagmo2's existing PSO implementation for hyperparameter optimization.
    - Implement the Nelder-Mead Simplex method as a baseline comparison algorithm.

2. **EA Integration:**
    - Utilize Pagmo2's built-in implementations of DE and PSO.
    - Extend Pagmo2's DE implementation to create jDE, incorporating self-adaptive mechanisms.
    - Implement SGA using Pagmo2's evolutionary algorithm interface, developing custom genetic operators as needed.
3. **Benchmark Suite Development:**
    - Create a diverse set of optimization problems, including continuous, discrete, and mixed-integer problems.
    - Implement real-world inspired problems that challenge different aspects of our EAs.
    - Develop a standardized performance measurement protocol, considering solution quality, convergence speed, and robustness.
4. **Hyperparameter Optimization Experiments:**
    - Design experiments to test each HOA on each EA across various problem types.
    - Utilize Pagmo2's parallelization capabilities to manage the computational load.
    - Implement logging and visualization tools to track optimization progress and results.
5. **Analysis and Comparison:**
    - Develop statistical analysis tools to compare HOA performance across different EAs and problem types.
    - Create visualization methods to illustrate the impact of hyperparameter choices on EA performance.
    - Implement sensitivity analysis to identify the most crucial hyperparameters for each EA.

5.2 Expected Outcomes

By following this implementation strategy, we anticipate achieving several key outcomes:

1. **HOA Performance Insights:** We expect to gain detailed insights into the strengths and weaknesses of each HOA across different EAs and problem types. For instance, we may find that CMA-ES excels at tuning DE for continuous problems, while SA is more effective for discrete parameter spaces.

2. **EA Hyperparameter Landscape:** Our study should reveal the hyperparameter landscapes of our chosen EAs, identifying which parameters are most crucial for performance and how they interact. This could lead to new guidelines for manually tuning these algorithms.

3. **Optimization Efficiency:** By comparing our HOAs to traditional methods like grid search, we expect to demonstrate significant improvements in the efficiency of finding good hyperparameter configurations.

4. **Algorithm Insights:** The process of optimizing hyperparameters may reveal insights into the behavior of our EAs. For example, we might discover unexpected synergies between certain parameter settings in jDE.

5. **Comparative Analysis:** Our comprehensive benchmarking will provide a valuable resource for the evolutionary computation community, offering detailed performance comparisons across different algorithms and problem types.

6. **Implementation Techniques:** The project will result in a suite of HOA implementations compatible with Pagmo2, which could be valuable for future research and practical applications in the field of hyperparameter optimization.

In conclusion, this implementation strategy, built upon the foundation of Pagmo2 and our carefully selected algorithms, positions our project to make significant contributions to the field of evolutionary computation. By systematically exploring the hyperparameter optimization landscape, we aim to enhance the efficiency and effectiveness of evolutionary algorithms across a wide range of applications.

References

[1] Nikolaus Hansen. *The CMA Evolution Strategy: A Comparing Review*, pages 75–102. Springer Berlin Heidelberg, Berlin, Heidelberg, 2006.

[2] A. Corana, M. Marchesi, C. Martini, and S. Ridella. Minimizing multimodal functions of continuous variables with the "simulated annealing" algorithm—corrigenda for this article is available here. *ACM Trans. Math. Softw.*, 13(3):262–280, September 1987.

[3] Yaru Li and Yulai Zhang. Hyper-parameter estimation method with particle swarm optimization. *arXiv preprint arXiv:2011.11944*, Dec 2020. Version 2.

[4] Pablo Ribalta Lorenzo, Jakub Nalepa, Michal Kawulok, Luciano Sanchez Ramos, and José Ranilla Pastor. Particle swarm optimization for hyperparameter selection in deep neural networks. In *Proceedings of the Genetic and Evolutionary Computation Conference*, GECCO '17, page 481–488, New York, NY, USA, 2017. Association for Computing Machinery.

[5] J. A. Nelder and R. Mead. A Simplex Method for Function Minimization. *The Computer Journal*, 7(4):308–313, 01 1965.

[6] Rainer Storn and Kenneth Price. Differential evolution – a simple and efficient heuristic for global optimization over continuous spaces. *Journal of Global Optimization*, 11(4):341–359, December 1997.

[7] Riccardo Poli, James Kennedy, and Tim Blackwell. Particle swarm optimization. *Swarm Intelligence*, 1(1):33–57, June 2007.

[8] Janez Brest, Sao Greiner, Borko Bošković, Marjan Mernik, and Viljem Zumer. Self-adapting control parameters in differential evolution: A comparative study on numerical benchmark problems. *Evolutionary Computation, IEEE Transactions on*, 10:646 – 657, 01 2007.

[9] David E. Goldberg. *Genetic Algorithms in Search, Optimization, and Machine Learning*. Addison-Wesley, Reading, Massachusetts, 1989.