# 

**UFAZ**

**Project Report**

**DSA M1**

**Meta-Heuristic Algorithms**

**Topic: Backtracking Search Optimization**

**Group members:**

* **Nurlan Abdullayev**
* **Elvin Rustamov**
* **Javid Huseynli**
* **Yusif Hajizada**

17.12.2024

## Abstract

This report explores the effectiveness of the Backtracking Search Optimization (BSA) algorithm in solving four prominent mathematical benchmark functions: Ackley, Rastrigin, Rosenbrock, Schwefel. Our Python implementation of BSA was evaluated across 30 runs to assess its performance in finding global minima within a 30-dimensional search space. The results demonstrate BSA's capability in achieving competitive optimization outcomes across different functions. The results demonstrate that the Python implementation consistently achieves competitive optimization results, producing lower objective values across all benchmarks compared to the MATLAB implementation. However, the MATLAB version demonstrated superior computational efficiency. These findings show the tradeoff between these two implementations: the Python implementation excels in achieving better solutions, the MATLAB implementation offers time savings.

## Keywords

* Backtracking Search Optimization
* Metaheuristics
* Benchmark Functions
* Optimization Algorithms
* Global Minima

## Introduction

Optimization is very important concept in many areas of the science and engineering. It helps us figure out the best solutions, in most of the times, not always, even in really complicated and high-dimensional problems. But traditional optimization methods don’t always work well, especially when the problem is nonlinear, it has multiple solutions, or is just too complex. Because of these issues, new methods called Metaheuristic Algorithms were developed. They’re more flexible and can handle tricky and messy optimization problems much better than the old methods.[1].

Metaheuristics are methods inspired by nature and how it works,operates, like behavior of animals or natural processes. They give a way to explore and find solutions in complex problems without needing things like gradient information or assuming the problem is convex. These methods are pretty useful when traditional approaches don’t work well enough [2]. Among these, Backtracking Search Optimization (BSA) has emerged as a promising algorithm, leveraging iterative improvement and diversification mechanisms to converge towards optimal solutions [3].

This report delves into the application of BSA in solving four a proven and trusted mathematical benchmark functions: Ackley, Rastrigin, Rosenbrock, Schwefel [4]. These functions are famous for having challenging landscapes, which makes them perfect candidates for evaluating the efficacy of optimization algorithms. By implementing BSA in Python and comparing its performance with Matlab-based counterparts, this study aims to elucidate the strengths and limitations of BSA within the context of complex optimization tasks [5].

## Literature Review

### Metaheuristic Algorithms in Optimization

Metaheuristic algorithms have become more popular because they can handle tough optimization problems that classical methods just can't solve [6]. These algorithms, inspired by natural phenomena such as evolution, swarm behavior, and physical processes, offer flexible and robust frameworks for exploring large search spaces [7]. Notable metaheuristics include Genetic Algorithms (GA) [8], Particle Swarm Optimization (PSO) [9] [10], Ant Colony Optimization (ACO) [10], and Grey Wolf Optimizer (GWO) [11], among others.

### Backtracking Search Optimization (BSA)

Backtracking Search Optimization is a fairly new addition to the set of metaheuristic algorithms, but it's already showing a lot of promise in solving complex problems [13].BSA is based on how people often go back and think again their approach when they are trying solving problems. It takes this natural process and turns it into a useful tool for solving complex problems. The algorithm mixes exploration and exploitation to move through tough search spaces. Firstly, it looks at different areas of the problem to understand it better, and then it focuses on the areas that seem more promising. This backtracking step, where it goes back from less useful solutions and works on the better ones, helps to keep the right path, stay on it. The algorithm iteratively refines candidate solutions by backtracking from less promising regions and focusing computational resources on more promising areas, thereby enhancing convergence rates and solution quality By focusing on the parts that are most likely to give the best result, BSA speeds up finding the best solution. Over time, this means faster results and better solutions, especially when compared to traditional methods. This approach works well when facing hard problems that simpler methods just can’t handle [14].

### Application of BSA in Benchmark Functions

Previous researches have demonstrated BSA's potential in solving various benchmark functions and real-world optimization problems [15] [23]. Its ability to adapt and the smart ways it searches for solutions have been pointed out in many different areas. From helping with engineering design to fine-tuning machine learning parameters, this method has shown how useful it can be. Whether it’s figuring out the best design to machine learning parameter tuning or adjusting the settings in a complex model, its flexibility and efficiency make it a great choice. By using different strategies depending on the situation, it can handle a wide range of challenges and help find the best solutions faster.

However, comparative analyses with established algorithms like GA and PSO are essential to position BSA's performance within the broader optimization landscape [16].

### Benchmark Functions as Evaluation Tools

Benchmark functions such as Ackley, Rastrigin, Rosenbrock, and Schwefel serve as standard tests for assessing optimization algorithms [17]. Their diverse characteristics, such as multimodality, nonlinearity, and varying dimensionalities, provide insights into an algorithm's ability to locate global minima amidst numerous local optima [18].

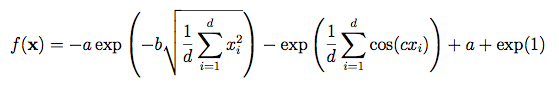
## Problem Definition

Optimization algorithms are evaluated based on their ability to find global minima of specific mathematical functions [19]. This study focuses on four benchmark functions.

Each function is defined mathematically with specific parameters and search space bounds, creating a standardized evaluation framework for optimization algorithms [24].

### Ackley Function

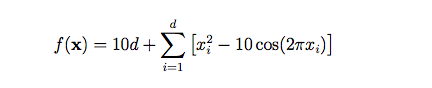
The Ackley function is a widely used benchmark in optimization, defined as:



where a=20, b=0.2, c=2π, and d is the dimensionality. It has a global minimum at x=0 with f(x)=0. Values of x are bound in [-32.768, 32.768] for all dimensions.[20]

### Rastrigin Function

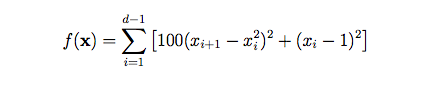
The Rastrigin function is defined as:



where A=10 and d is the dimensionality. The function has a global minimum at x=0 with f(x)=0. It is usually evaluated on x in [-5.12, 5.12] for all dimensions.[21]

### Rosenbrock Function

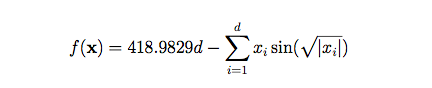
The Rosenbrock function, also known as the Rosenbrock's valley or banana function, is defined as:



where d is the dimensionality. It has a global minimum at x=1 with f(x)=0. Usually evaluated on x in [-5,10] for all dimensions. [22]

### Schwefel Function

The Schwefel function is defined as:



where d is the dimensionality. The global minimum occurs at x=420.9687 for all dimensions, with f(x)=0. Evaluated on x in [-500, 500] for all dimensions.[23]

## Solution Method

### Backtracking Search Optimization Algorithm (BSA)

The Backtracking Search Optimization Algorithm (BSA) is a metaheuristic inspired by the natural backtracking process in problem-solving. BSA combines exploration and exploitation strategies to efficiently navigate the search space, aiming to locate global minima of objective functions [5, 25]. The algorithm has five steps: Initialization, Selection-I, Mutation, Crossover, Selection-II [5].

#### Algorithm Steps

* **Initialization**:

The initial population is randomly generated using the following equation:

Pij ~ U(lowj, upj), i = 1 to N, j = 1 to D.

N is the size of the population, D is the dimension of the problem, lowj and upj are lower and upper boundaries for jth dimension of each individual in the population. U here is the uniform distribution.

* **Selection-I:**

This is the step in which the historical population Pold is obtained. Pold will later be used for the calculation of the search direction. We obtain the initial value for Pold by using:

Pold,ij ~ U(0,1).

We sample two variables a and b from the uniform distribution:

a,b ~ U(0,1).

Using these variables we can change Pold in new iterations:

If a < b then Pold := P.

Here := is the update operation. Once we have obtained Pold we must randomly shuffle it:

Pold := Permuting(Pold).

* **Mutation:**

The initial trial population T is generated with this equation:

Mutant = P + F \* (Pold - P)

(Pold - P) is the search direction matrix and F is the amplitude of this matrix. We used F = 3 \* r, for r ~ N(0,1) where N is the normal distribution.

* **Crossover:**

This is the process in which we obtain the final value of the trial population T. This step starts with the creation of the binary integer matrix of 1s, called map of size NxD. Two strategies are randomly chosen to define the values of map. The first strategy uses ceil(mixrate \* rnd \* D) for rnd ~ U(0,1) to select the number of randomly chosen indices to set to 0. The second strategy randomly selects one value in each row and sets it to 0.

If mapn,m = 1 for n = 1 to N, m = 1 to D, we update T:

Tn,m := Pn,m

Some individuals may overflow and obtain values outside of set boundaries. These values are regenerated by sampling from U(lowj,upj).

* **Selection-II:**

Individuals of T with better fitness value than the corresponding individuals of P replace those individuals in P. If the best individual in P, Pbest has a better fitness value than the global best solution, then it is replaced withPbest and the global best fitness is replaced with the fitness value of Pbest .

#### Pseudo-Code

**Input:**

* obj\_function: Objective function to minimize
* popsize: Number of individuals in the population
* dimensions: Dimensionality of the search space
* bounds: Lower and upper bounds for each dimension
* max\_evals: Maximum number of objective function evaluations
* optimal\_value: Threshold for a value to be considered optimal
* mixrate: Controls the number of elements of individuals that will mutate

**Output:**

* Best solution global\_best
* Corresponding fitness value global\_min

#### 

#### Parameter Settings

* **Dimension of each benchmark**: 30
* **Arithmetic Precision**: 10-16
* **Maximum Evaluations**: 20,000
* **Number of Runs**: 30
* **Population Size**: 30
* **Search Space Bounds**: Depends on benchmark function being evaluated.

### Implementation Details

The BSA algorithm was implemented in Python, leveraging libraries such as NumPy for numerical computations. The implementation includes the following components:

1. **Benchmark Function Definitions**: Accurate mathematical representations of Ackley, Rastrigin, Rosenbrock, and Schwefel functions.
2. **Population Initialization**: Generation of a diverse initial population within the specified bounds.
3. **Fitness Evaluation**: Calculation of objective function values for all candidate solutions.
4. **Growth and Dispersion Phases**: Mechanisms to enhance exploration and exploitation through controlled perturbations and diversifications.
5. **Termination Conditions**: Monitoring of evaluation counts and precision thresholds to determine algorithm convergence.

## Results

The Backtracking Search Optimization Algorithm (BSA) was executed across **30 independent runs** for each of the **four benchmark functions**: Ackley, Rastrigin, Rosenbrock, Schwefel. Each run was conducted in a **30-dimensional search space** with the following parameters:

* **Maximum Objective Function Evaluations**: 20,000
* **Precision Threshold**: 10-16

The **average fitness**, **standard deviation**, and **total runtime** were recorded for each function, as detailed below.

## 

## Discussion

### Performance Analysis of BSA

The comparison between our Python implementation BSA implementation and the MATLAB-based BSA focused on objective values (Mean and Std) and computational time across four minimization benchmark functions: Ackley, Rastrigin, Schwefel, Rosenbrock.

Objective Values (Solution Quality):

* + For the **Ackley** function, the Python implementation achieved a lower objective value (Mean: 0.393), compared to MATLAB's implementation (Mean: 0.666). Though both of these values are quite close to the global minimum 0.
  + In the **Rastrigin** function, the Python implementation (Mean: 28.26) outperformed MATLAB (Mean: 48.49).
  + For the **Schwefel** function, the Python implementation of BSA obtained a significantly lower objective value (Mean: 97.41) compared to the MATLAB implementation (Mean: 2408.87).
  + On the **Rosenbrock** function, the Python implementation achieved a lower objective value (Mean: 71.82) compared to MATLAB (Mean: 131.82).

Our Python implementation consistently produced better results across all of the benchmark functions.

As for the standard deviation (std), our implementation generally had lower std values, with the exception of the Rastrigin function, for which the MATLAB version of BSA showed lower variation.

In terms of computation time, the MATLAB implementation was consistently faster across all benchmark functions:

* + Ackley: 4.08s (MATLAB) vs. 16.2s (Python)
  + Rastrigin: 3.98s (MATLAB) vs. 14.1s (Python)
  + Schwefel: 4.16s (MATLAB) vs. 12.9s (Python)
  + Rosenbrock: 3.43s (MATLAB) vs. 14.7s (Python)

This demonstrates MATLAB's strength in computational efficiency, likely due to its optimized libraries and numerical processing. The Python implementation was noticeably slower.

### Implications and Recommendations

These results show that while the

**To enhance BSA's performance:**

1. **Hybridization**: Integrating BSA with other optimization strategies could improve its ability to escape local minima and find better solutions [26].
2. **Adaptive Parameters**: Dynamically adjusting growth and dispersion rates based on real-time performance metrics might enhance BSA's adaptability to different function characteristics [27].
3. **Parallelization**: Leveraging parallel computing techniques can reduce computational time, allowing for more extensive exploration [28].
4. **Algorithm Refinement**: Incorporating memory-based mechanisms to retain high quality solutions or using elitism can prevent the loss of optimal candidates during optimization [29].

## Conclusion

The comparison of the BSA algorithm implemented in **Python** and MATLAB highlights the following findings:

1. **Solution Quality**: The Python implementation consistently achieved better minimization results across all benchmark functions (Ackley, Rastrigin, Schwefel, and Rosenbrock)..
2. **Computational Efficiency**: The MATLAB implementation was significantly faster, with runtimes approximately 3–4 times lower than the Python implementation.

In conclusion, while the MATLAB implementation excels in speed, our Python implementation provides superior solution accuracy, making it preferable for applications where achieving the lowest possible objective value is critical. The Python implementation could be further optimized for speed, by leveraging parallel processing or optimized libraries like Numba or TensorFlow. Alternatively, BSA could be implemented in more performant languages such as C, C++ etc. to try and combine both the computational efficiency shown by the MATLAB implementation and the solution accuracy of our Python implementation.

## References

1. *Abdel-Basset, M., Abdel-Fatah, L., & Sangaiah, A. K. (2018). Metaheuristic algorithms: A comprehensive review. Computational intelligence for multimedia big data on the cloud with engineering applications, 185-231.*
2. *Osaba, E., Villar-Rodriguez, E., Del Ser, J., Nebro, A. J., Molina, D., LaTorre, A., ... & Herrera, F. (2021). A tutorial on the design, experimentation and application of metaheuristic algorithms to real-world optimization problems. Swarm and Evolutionary Computation, 64, 100888.*
3. *Hassan, B. A., & Rashid, T. A. (2020). Operational framework for recent advances in backtracking search optimisation algorithm: A systematic review and performance evaluation. Applied Mathematics and Computation, 370, 124919.*
4. *Khan, T. A., Taj, T. A., Asif, M. K., & Ijaz, I. (2012, September). Modeling of a standard Particle Swarm Optimization algorithm in MATLAB by different benchmarks. In Second International Conference on the Innovative Computing Technology (INTECH 2012) (pp. 271-274). IEEE.*
5. *Civicioglu, P. (2013). Backtracking search optimization algorithm for numerical optimization problems. Applied Mathematics and computation, 219(15), 8121-8144.*
6. *Kareem, S. W., Ali, K. W. H., Askar, S., Xoshaba, F. S., & Hawezi, R. (2022). Metaheuristic algorithms in optimization and its application: A review. JAREE (Journal on Advanced Research in Electrical Engineering), 6(1).*
7. *Darvishpoor, S., Darvishpour, A., Escarcega, M., & Hassanalian, M. (2023). Nature-inspired algorithms from oceans to space: A comprehensive review of heuristic and meta-heuristic optimization algorithms and their potential applications in drones. Drones, 7(7), 427.*
8. *Ólafsson, S. (2006). Metaheuristics. Handbooks in operations research and management science, 13, 633-654.*
9. *Lessmann, S., Caserta, M., & Arango, I. M. (2011). Tuning metaheuristics: A data mining based approach for particle swarm optimization. Expert Systems with Applications, 38(10), 12826-12838.*
10. *Zaman, Hamid Reza Rafat, and Farhad Soleimanian Gharehchopogh. "An improved particle swarm optimization with backtracking search optimization algorithm for solving continuous optimization problems." Engineering with Computers 38, no. Suppl 4 (2022): 2797-2831.*
11. *Dorigo, M., & Stützle, T. (2003). The ant colony optimization metaheuristic: Algorithms, applications, and advances. Handbook of metaheuristics, 250-285.*
12. *Kumar, A., Singh, S., & Kumar, A. (2021, June). Grey wolf optimizer and other metaheuristic optimization techniques with image processing as their applications: a review. In IOP Conference Series: Materials Science and Engineering (Vol. 1136, No. 1, p. 012053). IOP Publishing.*
13. *Civicioglu, P. (2013). Backtracking search optimization algorithm for numerical optimization problems. Applied Mathematics and computation, 219(15), 8121-8144.*
14. *Tsai, H. C. (2019). Improving backtracking search algorithm with variable search strategies for continuous optimization. Applied Soft Computing, 80, 567-578.*
15. *Chen, D., Zou, F., Lu, R., & Wang, P. (2017). Learning backtracking search optimisation algorithm and its application. Information Sciences, 376, 71-94.*
16. *Parashar, M., Rajput, S., Dubey, H. M., & Pandit, M. (2017, February). Optimization of benchmark functions using a nature inspired bird swarm algorithm. In 2017 3rd international conference on computational intelligence & communication technology (CICT) (pp. 1-7). IEEE.*
17. *Price, K. V., Storn, R. M., & Lampinen, J. A. (2005). Benchmarking differential evolution. Differential Evolution: A Practical Approach to Global Optimization, 135-187.*
18. *Hussain, K., Salleh, M. N. M., Cheng, S., & Naseem, R. (2017). Common benchmark functions for metaheuristic evaluation: A review. JOIV: International Journal on Informatics Visualization, 1(4-2), 218-223.*
19. *Weise, T. (2009). Global optimization algorithms-theory and application. Self-Published Thomas Weise, 361, 153.*
20. *Molga, M., & Smutnicki, C. Test functions for optimization needs (2005).*
21. *Pohlheim, H. GEATbx Examples: Examples of Objective Functions (2005)*
22. *Adorio, E. P., & Diliman, U. P. MVF - Multivariate Test Functions Library in C for Unconstrained Global Optimization (2005)*
23. *Laguna, M., & Marti, R. Experimental Testing of Advanced Scatter Search Designs for Global Optimization of Multimodal Functions (2002)*
24. *Štefek, A. (2011, June). Benchmarking of heuristic optimization methods. In 14th International Conference Mechatronika (pp. 68-71). IEEE..*
25. *Tsai, H. C. (2019). Improving backtracking search algorithm with variable search strategies for continuous optimization. Applied Soft Computing, 80, 567-578.*
26. *Wang, L., Zhong, Y., Yin, Y., Zhao, W., Wang, B., & Xu, Y. (2015). A hybrid backtracking search optimization algorithm with differential evolution. Mathematical problems in engineering, 2015(1), 769245.*
27. *Chen, D., Zou, F., Lu, R., & Li, S. (2019). Backtracking search optimization algorithm based on knowledge learning. Information Sciences, 473, 202-226.*
28. *Jin, Y. F., & Yin, Z. Y. (2020). Enhancement of backtracking search algorithm for identifying soil parameters. International Journal for Numerical and Analytical Methods in Geomechanics, 44(9), 1239-1261.*
29. *Hu, Z., Dai, C., & Su, Q. (2022). Adaptive backtracking search optimization algorithm with a dual-learning strategy for dynamic economic dispatch with valve-point effects. Energy, 248, 123558.*