

# CS471 Project 2

---

Max Shuford

February 7, 2026

## 1 INTRODUCTION

Benchmark objective functions are widely used to evaluate numerical behavior in continuous optimization. Each function exhibits different landscape properties (e.g., unimodal vs. multimodal, separable vs. non-separable, smooth vs. rugged), which affects both (1) the distribution of objective values produced by random samples and (2) the computational effort required to evaluate fitness.

This project implements ten standard benchmark functions and evaluates them using randomized populations. Rather than running an optimizer, the goal is to measure *fitness evaluation outcomes*: objective value statistics and evaluation time in milliseconds for computing fitness for an entire population. This establishes a consistent baseline for comparing benchmark problems under identical sampling conditions.

## 2 SYSTEM DESIGN AND IMPLEMENTATION

The program was refactored into modular components:

- **Population:** An  $n \times m$  matrix ( $\mathbb{R}^{n \times m}$ ), where  $n$  is the number of experiments (default  $n = 30$ ) and  $m \in \{10, 20, 30\}$  is the problem dimension. Each row corresponds to one candidate solution vector.
- **Fitness:** A vector in  $\mathbb{R}^n$  storing the scalar objective function values returned from the problem evaluation.
- **Problem:** A collection of benchmark functions (Schwefel through Egg Holder). Input is a vector from the population; output is a scalar fitness value.

**Configuration and I/O:** All key settings are read from an external input file (dimension  $m$ , population size  $n$ , problem type 1–10, seed, output CSV path). Fitness values are written to a CSV file for external analytics.

**Timing Measurement:** The program measures wall-clock time (milliseconds) for evaluating the fitness of the *entire population* (i.e., timing includes only the loop that computes the fitness vector, excluding file I/O and configuration parsing). This timing value is stored in the output and later analyzed in Python.

### 3 OPTIMIZATION ALGORITHMS

This project evaluates two stochastic optimization strategies: a Blind Algorithm (Random Sampling) and Repeated Local Search (RLS). Both methods were applied to identical benchmark problems, dimensions, and population sizes to enable direct performance comparison.

#### 3.1 BLIND ALGORITHM (RANDOM SEARCH)

The Blind Algorithm is a simple enumerative strategy that evaluates a predefined number of randomly generated candidate solutions without exploiting information from previous evaluations. Each candidate is sampled independently from the search space, and the best observed objective value is retained.

This method serves as a baseline because it does not guide the search toward improved regions of the solution space and provides insight into the raw difficulty of each benchmark function under uniform random sampling.

#### 3.2 REPEATED LOCAL SEARCH

Repeated Local Search (RLS) extends basic local search by performing multiple independent local optimization runs starting from different randomly generated initial solutions. Each local search iteratively explores neighboring solutions and moves to a better neighbor until no further improvement is found.

By restarting the local search process several times, RLS reduces the likelihood of becoming trapped in poor local optima and typically achieves better objective values at the cost of increased computational effort.

### 4 BENCHMARK PROBLEMS (INTRODUCTIONS)

The ten benchmark functions were selected to represent a variety of search landscapes:

1. **Schwefel (1):** Highly multimodal and deceptive; many local minima. Random samples often produce wide objective value ranges due to strong nonlinearities.
2. **De Jong 1 / Sphere (2):** Smooth, convex, unimodal baseline. Fitness values tend to cluster more tightly relative to multimodal functions.

3. **Rosenbrock (3):** Features a narrow curved valley; evaluation is simple but the landscape is nonconvex and nonseparable.
4. **Rastrigin (4):** Strong periodic structure with many local minima. Objective values often show high variance due to oscillatory cosine terms.
5. **Griewank (5):** Many regularly distributed local minima; combines a sum and a product term.
6. **Sine Envelope Sine Wave (6):** Oscillatory behavior with envelope scaling; typically produces rugged response surfaces.
7. **Stretch V Sine Wave (7):** Emphasizes nonlinear scaling and oscillations; sensitive to pairwise variable interactions.
8. **Ackley One (8):** Mixes exponential and trigonometric components; designed to be multimodal with broad flat regions.
9. **Ackley Two (9):** A more interaction-heavy variant; frequently increases computational cost due to additional nonlinear terms.
10. **Egg Holder (10):** Very rugged with steep ridges/valleys; often yields extreme values depending on input range and nonlinear structure.

Table 4.1: Experiments

	Name	$f(x^*)$	Dimension	Range
$f_1$	Schwefel	0	30	$[-512, 512]^n$
$f_2$	De Jong 1	0	30	$[-100, 100]^n$
$f_3$	Rosenbrock's Saddle	0	30	$[-100, 100]^n$
$f_4$	Rastrigin	0	30	$[-30, 30]^n$
$f_5$	Griewank	0	30	$[-500, 500]^n$
$f_6$	Sine Envelope Sine Wave	$-1.4915(n-1)$	30	$[-30, 30]^n$
$f_7$	Stretch V Sine Wave	0	30	$[-30, 30]^n$
$f_8$	Ackley One	$-7.54276 - 2.91867(n-3)$	30	$[-32, 32]^n$
$f_9$	Ackley Two	0	30	$[-32, 32]^n$
$f_{10}$	Egg Holder	–	30	$[-500, 500]^n$

## 5 EXPERIMENTAL PROCEDURE

For each experimental run:

1. Read configuration parameters from the input file:
  - Dimension  $m \in \{10, 20, 30\}$

- Population size  $n$  (default  $n = 30$ )
  - Algorithm type (1,2)
  - Problem type  $p \in \{1, \dots, 10\}$
  - Problem Ranges
  - Output CSV path
  - Random seed (fixed seed or system-time seed)
2. Generate a population matrix  $X \in \mathbb{R}^{n \times m}$  with uniform random values over the function's recommended input range.
  3. Compute fitness values  $f_i = F(X_i)$  for each row vector  $X_i$ .
  4. Measure evaluation time in milliseconds for computing the full fitness vector.
  5. Write results to CSV for external analysis.

**External analytics:** A Python script runs the program multiple times, appends fitness results to a master CSV, and computes descriptive statistics (mean, median, range, standard deviation) and average evaluation time across runs.

Table 5.1: Condensed results for Blind Search ( $n = 30$ ,  $m = 10$ ).

Problem	Avg	Median	Range	SD	T(ms)
$f_1$ Schwefel	4189.701	4191.478	3839.388	635.591	0.017
$f_2$ De Jong 1	32846.825	32480.842	53599.532	9473.073	0.003
$f_3$ Rosenbrock	99517.729	99446.653	99536.537	16131.967	0.006
$f_4$ Rastrigin	3104.307	3071.436	5527.490	827.029	0.014
$f_5$ Griewank	209.546	208.920	369.703	60.650	0.016
$f_6$ Sine Env.	-6.615	-6.548	4.412	0.698	0.017
$f_7$ Stretch V	130.378	128.141	242.330	41.597	0.057
$f_8$ Ackley One	180.745	181.500	187.995	31.152	0.028
$f_9$ Ackley Two	-11301.909	-10014.850	38539.368	6287.169	0.050
$f_{10}$ Egg Holder	-33.531	-15.414	4789.508	845.753	0.029

Table 5.2: Condensed results for Blind Search ( $n = 30$ ,  $m = 20$ ).

Problem	Avg	Median	Range	SD	T(ms)
$f_1$ Schwefel	8383.667	8377.252	5493.023	871.136	0.030
$f_2$ De Jong 1	67087.259	66836.306	92965.401	13956.229	0.004
$f_3$ Rosenbrock	37983194643.099	37268796187.066	78162883384.856	11619149200.990	0.004
$f_4$ Rastrigin	6244.335	6183.579	7734.278	1228.343	0.027
$f_5$ Griewank	419.356	416.988	515.274	83.565	0.031
$f_6$ Sine Env.	-13.813	-13.732	5.826	0.985	0.028
$f_7$ Stretch V	269.961	268.298	390.895	58.015	0.121
$f_8$ Ackley One	384.036	386.127	317.520	46.464	0.054
$f_9$ Ackley Two	-24006.607	-22476.364	55397.100	9406.426	0.101
$f_{10}$ Egg Holder	-96.294	-96.281	8968.480	1265.449	0.057

Table 5.3: Condensed results for Blind Search ( $n = 30$ ,  $m = 30$ ).

Problem	Avg	Median	Range	SD	T(ms)
$f_1$ Schwefel	12494.112	12527.376	7125.014	1052.864	0.049
$f_2$ De Jong 1	99517.729	99446.653	99536.537	16131.967	0.006
$f_3$ Rosenbrock	57993285306.134	57083183260.244	109427463885.913	14848294478.695	0.006
$f_4$ Rastrigin	9209.496	9205.680	9173.487	1486.517	0.043
$f_5$ Griewank	630.332	627.496	657.934	103.807	0.046
$f_6$ Sine Env.	-21.158	-21.122	7.925	1.269	0.043
$f_7$ Stretch V	414.272	412.229	413.166	71.464	0.183
$f_8$ Ackley One	585.728	586.820	380.343	55.818	0.072
$f_9$ Ackley Two	-36510.769	-35763.841	68667.233	11497.971	0.162
$f_{10}$ Egg Holder	-135.488	-189.827	9005.986	1591.709	0.083

Table 5.4: Condensed results for Repeated Local Search ( $n = 30$ ,  $m = 10$ ).

Problem	Avg	Median	Range	SD	T(ms)
$f_1$ Schwefel	2205.834	2202.848	2319.898	400.959	1.518
$f_2$ De Jong 1	185.256	170.910	530.171	74.795	0.449
$f_3$ Rosenbrock	1205533.934	721501.623	31615724.759	2182174.181	0.477
$f_4$ Rastrigin	118.966	110.614	301.709	37.218	2.666
$f_5$ Griewank	2.175	2.093	7.681	0.520	3.212
$f_6$ Sine Env.	-7.939	-7.927	4.575	0.714	1.114
$f_7$ Stretch V	48.797	47.722	85.326	14.121	4.398
$f_8$ Ackley One	111.234	111.467	228.665	42.437	2.889
$f_9$ Ackley Two	-59982.682	-58332.291	86862.375	15256.276	10.083
$f_{10}$ Egg Holder	-2652.318	-2656.550	4173.459	694.508	2.569

Table 5.5: Condensed results for Repeated Local Search ( $n = 30$ ,  $m = 20$ ).

Problem	Avg	Median	Range	SD	T(ms)
$f_1$ Schwefel	5136.667	5133.629	3494.892	587.731	3.392
$f_2$ De Jong 1	949.673	861.055	3800.646	389.498	1.243
$f_3$ Rosenbrock	23531740.498	12813666.879	553513024.807	39329879.692	1.333
$f_4$ Rastrigin	362.528	332.171	1039.494	121.545	7.153
$f_5$ Griewank	6.943	6.435	22.623	2.461	9.096
$f_6$ Sine Env.	-15.870	-15.849	5.931	0.960	2.219
$f_7$ Stretch V	150.978	151.700	166.233	26.654	9.158
$f_8$ Ackley One	276.532	277.682	365.015	64.374	6.068
$f_9$ Ackley Two	-94039.374	-94056.415	150109.959	23480.130	21.378
$f_{10}$ Egg Holder	-4242.272	-4183.297	6034.493	948.177	5.534

Table 5.6: Condensed results for Repeated Local Search ( $n = 30$ ,  $m = 30$ ).

Problem	Avg	Median	Range	SD	T(ms)
$f_1$ Schwefel	8394.497	8387.291	4004.122	674.485	5.178
$f_2$ De Jong 1	2375.118	2136.974	8813.971	965.561	2.287
$f_3$ Rosenbrock	106506854.750	68245644.496	1562899325.776	136394685.542	2.406
$f_4$ Rastrigin	712.360	655.910	2929.458	247.620	12.914
$f_5$ Griewank	15.684	14.526	71.998	5.473	17.313
$f_6$ Sine Env.	-23.691	-23.643	7.036	1.143	3.449
$f_7$ Stretch V	266.137	267.984	214.857	35.053	13.744
$f_8$ Ackley One	458.409	466.851	465.226	83.424	9.139
$f_9$ Ackley Two	-126664.873	-125545.746	169110.327	28490.445	34.697
$f_{10}$ Egg Holder	-5270.983	-5242.376	8564.373	1183.979	8.593

## 6 ANALYSIS AND FINDINGS

### 6.1 BLIND SEARCH VS. REPEATED LOCAL SEARCH

Across all benchmark functions and dimensions, Repeated Local Search consistently outperformed the Blind Algorithm in terms of average and median fitness values. This improvement is expected, as RLS actively exploits local neighborhood information to guide the search toward lower objective values rather than relying on independent random samples.

The performance gap between the two algorithms widened as dimensionality increased, particularly for nonseparable and highly multimodal functions such as Rosenbrock, Schwefel, and Ackley Two. While Blind Search provided a useful baseline for understanding function difficulty, its lack of guidance resulted in higher variance and poorer overall fitness values.

In contrast, Repeated Local Search achieved significantly better solutions but incurred higher computational cost due to iterative neighborhood evaluations and restart overhead. This highlights the tradeoff between solution quality and evaluation time when choosing an optimization strategy.

## 7 CONCLUSION

This project evaluated the performance of two stochastic optimization strategies, Blind Search and Repeated Local Search, across ten continuous benchmark functions and three problem dimensions ( $m = 10, 20, 30$ ). All experiments were conducted using consistent population sizes and parameter settings to allow for direct comparison of solution quality and computational cost.

The Blind Search algorithm provided a useful baseline by characterizing the difficulty of each benchmark under uniform random sampling. As expected, Blind Search produced wide fitness distributions and high variance for multimodal and nonseparable functions such as Schwefel, Rastrigin, Rosenbrock, and Egg Holder. While computationally inexpensive, the lack of search guidance resulted in comparatively poor solution quality, especially as dimensionality increased.

Repeated Local Search consistently outperformed Blind Search in terms of average and median fitness values across all benchmark functions and dimensions. By exploiting neighborhood information and restarting from multiple initial solutions, RLS was able to identify substantially improved solutions and reduce sensitivity to poor starting points. This performance advantage became more pronounced for higher-dimensional and rugged landscapes, confirming the benefits of iterative local improvement in complex search spaces.

The improved solution quality achieved by Repeated Local Search came at the cost of increased evaluation time. Functions involving expensive nonlinear operations, such as Ackley Two and Griewank, exhibited the largest runtime differences between the two algorithms. These results highlight the fundamental tradeoff in optimization between computational cost and solution quality.

Overall, the experimental results validate both the correctness of the benchmark function implementations and the effectiveness of Repeated Local Search compared to unguided random sampling. The project demonstrates how algorithmic structure and problem character-

istics jointly influence optimization performance. The Blind Search results establish a neutral baseline, while the Repeated Local Search results illustrate the advantages of informed local exploration.

Future work could extend this study by increasing the number of repetitions per configuration, visualizing fitness distributions, or incorporating additional metaheuristic approaches such as genetic algorithms or particle swarm optimization to further examine convergence behavior and scalability.