Estimation-of-Distribution Algorithms

Anass Ettahiri Anas Labgoul

15th February 2026

1 Introduction

This report presents the implementation and experimental evaluation of three black-box optimization algorithms for pseudo-Boolean functions: the significance-based compact genetic algorithm (sig-cGA), the compact genetic algorithm (cGA), and the (1+1) evolutionary algorithm ((1+1) EA). All algorithms operate on bit strings and rely only on fitness evaluations.

The work covers implementation design, data structures, algorithmic behavior, complexity considerations, and an empirical comparison on standard benchmarks: OneMax, LeadingOnes, and Jump. Particular attention is given to parameter scaling, runtime behavior, and the practical impact of implementation details.

2 Task 1 — Data Structures and Standard Bit Mutation

2.1 Individual Representation

Individuals are represented by:

- the dimension n,
- a list of n binary values.

This representation provides constant-time access to each bit and linear-time traversal for evaluation and copying. It is simple and well suited for repeated mutation and sampling operations.

2.2 Standard Bit Mutation

Standard bit mutation flips each bit independently with probability $1/n$. The implementation uses a two-step strategy:

1. Sample the number of flipped bits $k \sim \text{Binomial}(n, 1/n)$.
2. Sample k distinct indices uniformly and flip only these bits.

Runtime justification. After copying the parent, the mutation performs exactly k flips and does not scan all bits. The runtime is therefore $O(k)$.

Expected runtime. Since $\mathbb{E}[k] = 1$, the expected runtime (excluding the copy step) is $O(1)$, satisfying the requirement that mutations with zero flips do not cost $\Theta(n)$.

2.3 Frequency Vector Model

The probabilistic model is stored as a vector of n frequencies. Sampling generates each bit independently according to its frequency. Frequency updates are clipped to the interval $[1/n, 1 - 1/n]$ to avoid absorbing states.

Sampling an individual costs $O(n)$ time.

3 Task 2 — Significance Testing and History Structures

3.1 Significance Function

The significance test determines whether the observed bit history deviates sufficiently from its expectation under the current frequency. For history length m and frequency p, the threshold is

$$mp + \varepsilon \max\{\sqrt{mp \ln n}, \ln n\}.$$

If the observed count exceeds this bound, the decision is UP or DOWN; otherwise the frequency remains unchanged. All required quantities are maintained incrementally, so the test runs in constant time.

3.2 Simplified History

The simplified history stores only:

$$(m,\ \#1,\ \#0).$$

Adding a bit is $O(1)$. Only the full history is tested for significance. This version is computationally lightweight and memory efficient.

3.3 Original History with Exponential Blocks

The original-history variant uses a linked list of blocks storing block size and number of ones. Blocks summarize exponentially increasing segments of recent history. When three consecutive blocks have equal size, the first two are merged.

During implementation, special care was required in the consolidation procedure. After merging two consecutive blocks, the traversal pointer must be advanced to the node following the removed block. Concretely, we explicitly updated the pointer so that after a merge the variable tracking the next node takes the value `next = next.next`. Without this correction, the algorithm may revisit already processed nodes or skip necessary comparisons, producing incorrect block structures and distorted subsequence summaries. This small pointer update proved essential for correct history maintenance.

Maximum number of merges. Each merge doubles block size. Therefore, a single insertion can trigger at most $O(\log m)$ merges, where m is total history length.

Worst-case example (merge cascade). Consider a consolidated list that contains exactly two blocks of each size:

$$[1, 1,\ 2, 2,\ 4, 4,\ 8, 8,\ \ldots,\ 2^L, 2^L],$$

ordered from head (most recent) to tail (oldest). Insert a new bit when the head already summarizes at least $\ln(n)$ bits, so the algorithm creates a new head block of size 1. The list becomes

$$[1,\ 1, 1,\ 2, 2,\ 4, 4,\ \ldots,\ 2^L, 2^L],$$

which contains three consecutive size-1 blocks. Consolidation therefore merges the first two size-1 blocks into a size-2 block, yielding

$$[1,\ 2,\ 2,2,\ 4,4,\ \dots].$$

Now the first three size-2 blocks are consecutive, so another merge creates a size-4 block at the head. This repeats: each merge at size 2^j creates a new block of size 2^{j+1} which, together with the two existing blocks of that size, forms a triple and triggers the next merge. Thus, the insertion causes a cascade of merges at sizes $1, 2, 4, \dots, 2^L$, i.e., $\Theta(\log m)$ merges where m is the current history length.

4 Task 3 — Full sig-cGA Implementation

The sig-cGA maintains a frequency vector restricted to $\{1/n, 1/2, 1-1/n\}$ and one history per bit position.

Each iteration:

1. sample two individuals,
2. select the fitter one,
3. append winning bits to histories,
4. scan subsequences,
5. apply the significance test,
6. update the frequency and reset the history at first significant deviation.

Sampling costs $O(n)$ per individual. History scanning costs up to $O(\log m)$ per bit in the worst case.

5 Task 4 — Competing Algorithms

5.1 Compact Genetic Algorithm (cGA)

The cGA samples two individuals and updates each frequency by $\pm 1/K$ according to the fitter sample. Frequencies are clipped to borders after each update. Each iteration costs $O(n)$.

5.2 (1+1) Evolutionary Algorithm

The (1+1) EA maintains one current solution and generates one mutated offspring per iteration. The offspring replaces the parent if fitness improves. The method is simple and mutation-driven.

6 Task 5 — Benchmarks and Termination

Benchmarks implemented:

- OneMax — number of ones,
- LeadingOnes — prefix of ones,

- Jump_k — deceptive landscape with a valley.

Termination occurs when the optimum is found or a fixed evaluation budget is exhausted. Each configuration is repeated multiple times with fixed seeds for reproducibility.

7 Task 6 — Experimental Results and Discussion

7.1 Choice of Numerical Parameter Values

Parameter values were selected to balance theoretical scaling, observability of performance differences, and computational feasibility.

Problem sizes $n \in \{100, 200, 300\}$ are large enough to show scaling effects while allowing repeated trials. Each configuration was repeated 10 times to reduce variance without excessive computation.

Budgets were smaller for OneMax and larger for LeadingOnes and Jump due to their structural difficulty. The chosen limits allow both partial and full successes, enabling meaningful comparisons.

For cGA, values of K were chosen near theoretical scales and varied by constant factors to observe both fast and slow adaptation regimes.

For sig-cGA, several ε values were tested to compare conservative and reactive update behavior.

For Jump, small values $k = 2$ and $k = 3$ were used to introduce deception while keeping success probability measurable.

7.2 Observed Patterns

Performance depends strongly on landscape structure. Smooth functions favor frequency drift (cGA), while prefix-based structure allows mutation-driven progress ((1+1) EA). Deceptive landscapes highlight robustness differences.

Scaling effects dominate at larger n: fixed budgets sharply reduce success rates across all algorithms.

Variance differs across methods: cGA is relatively stable, mutation search more variable, sig-cGA very conservative.

7.3 Algorithmic Interpretation

cGA benefits from continuous small updates and shows stable convergence when K is well chosen. Too large K slows learning; too small increases noise.

The (1+1) EA performs well when improvements are incremental but struggles when coordinated changes are needed.

sig-cGA updates only when statistical evidence is strong. Under limited budgets this leads to very slow adaptation.

8 Conclusions

The experiments suggest:

- cGA provides the best reliability–speed tradeoff on smooth problems.
- (1+1) EA remains competitive on incremental landscapes.
- sig-cGA is theoretically strong but practically sensitive to thresholds and budgets.

- Parameter scaling and evaluation budgets strongly influence observed performance.

We also believe that part of the difficulties encountered with the sig-cGA implementation may stem from a detail in the significance test: in one branch of the test, the number of observed zeros is compared against the expected number of ones. This mismatch in the reference expectation can bias the decision rule and delay or prevent frequency updates. Correcting this comparison is likely to improve the practical behavior of the algorithm.

Overall, incremental frequency-update EDAs such as cGA are the most robust choice under constrained budgets, while significance-based updates require careful implementation and calibration.