Estimation-of-Distribution Algorithms

Implementation and Experimental Study

1 Introduction

This report presents the implementation and empirical 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). Each method operates on bit strings and relies solely on fitness evaluations, making them suitable for simulation-based or analytically intractable objectives.

The project covers data structure design, algorithmic implementation, runtime considerations, and a systematic experimental comparison on standard benchmark functions: OneMax, LeadingOnes, and Jump. The focus is on both correctness and practical efficiency, with attention to parameter scaling and statistical reliability.

2 Task 1 — Data Structures and Standard Bit Mutation

2.1 Individual Representation

Individuals are represented as objects containing:

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

This structure allows constant-time access to any bit and linear-time traversal for evaluation and copying. The representation is simple, explicit, 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 guarantee. After copying the parent, the mutation performs exactly k flips and does not scan all n bits. Hence the runtime is $O(k)$.

Expected runtime. Since $\mathbb{E}[k] = 1$, the expected mutation runtime (excluding copying) is $O(1)$. This satisfies the requirement that mutations with zero flips do not incur $\Theta(n)$ time.

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. Each frequency update is clipped to the interval $[1/n,\, 1 - 1/n]$ to avoid absorbing states at the extremes.

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

3 Task 2 — Significance Testing and History Structures

3.1 Significance Function

The significance test evaluates whether the observed bit history deviates sufficiently from its expected value under the current frequency. For a history of length m and frequency p, a threshold of the form

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

is used. If the number of ones (or zeros) exceeds this bound, the decision is UP (or DOWN); otherwise the frequency remains unchanged.

All required quantities are stored incrementally, so the test runs in constant time.

3.2 Simplified History

The simplified history stores only:

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

Adding a bit updates counters in $O(1)$ time. Only the full history is tested for significance. This version is memory-efficient and computationally cheap.

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 represent exponentially increasing segments of recent history. When three consecutive blocks have equal size, the first two are merged.

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

Worst-case example. A merge cascade occurs when two blocks already exist at every size level and a new smallest block creates a third equal-size block, triggering merges across all levels.

4 Task 3 — Full sig-cGA Implementation

The sig-cGA maintains:

- a frequency vector with values restricted to $\{1/n, 1/2, 1 - 1/n\}$,
- one history object per bit position.

Each iteration:

1. sample two individuals,
2. select the fitter one,

3. update each bit history with the winner's bit,

4. scan subsequences and apply the significance test,

5. update the frequency and reset history at the 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 $\pm 1/K$ depending on which bit value appears in the fitter sample. Frequencies are clipped to borders after each update.

Each iteration costs $O(n)$ time.

5.2 (1+1) Evolutionary Algorithm

The (1+1) EA maintains a single current solution. Each iteration generates one mutated offspring and accepts it if fitness is not worse. The method is simple and uses only mutation and elitist selection.

6 Task 5 — Benchmarks and Termination

The following benchmark functions are implemented:

- **OneMax** — counts the number of ones.
- **LeadingOnes** — counts the length of the leading prefix of ones.
- **Jump**$_k$ — introduces a deceptive valley before the optimum.

Termination occurs when either the optimum is found or a fixed evaluation budget is exhausted. Multiple independent runs with fixed seeds ensure reproducibility.

7 Task 6 — Experimental Results and Discussion

7.1 Choice of Numerical Parameter Values

Numerical parameters were chosen to balance theoretical guidance, experimental observability, and computational feasibility.

Problem sizes $n \in \{100, 200, 300\}$ are large enough to show scaling effects while still allowing repeated runs. Each configuration was executed 10 times, which provides stable averages while keeping runtime manageable.

Budgets were smaller for OneMax and larger for LeadingOnes and Jump, reflecting their increasing structural difficulty. The chosen limits allow partial and full successes to appear, which helps distinguish algorithm behavior.

For cGA, the parameter K was selected around theoretical scaling laws and varied by moderate factors to observe both fast-adaptation and slow-adaptation regimes.

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

For Jump, small values $k = 2$ and $k = 3$ were used. These already introduce deception while keeping success probability measurable within finite budgets.

7.2 Observed Behavioral Patterns

Performance differences are strongly landscape-dependent. Smooth functions favor frequency-based drift (cGA), while prefix-dependent functions allow mutation-based progress ((1+1) EA). Deceptive landscapes highlight robustness differences.

Scaling effects dominate at larger n: fixed budgets cause success rates to drop across all methods. This shows that budget scaling is as important as algorithm choice.

Variance also differs across algorithms. cGA shows relatively stable behavior across runs, while mutation-based search exhibits higher spread. The sig-cGA behaves conservatively, with infrequent updates under strict thresholds.

7.3 Algorithmic Interpretation

The cGA's incremental updates create steady directional drift, explaining its strong OneMax performance and moderate Jump robustness. Parameter K controls the stability–speed tradeoff.

The (1+1) EA performs well when improvements can be accumulated locally but struggles when coordinated bit changes are required.

The sig-cGA requires strong statistical evidence before updating frequencies. Under limited budgets this leads to slow adaptation, making it sensitive to threshold tuning and history growth.

7.4 Practical Lessons

Simpler adaptive rules are more robust under limited budgets. Statistical-update EDAs require careful calibration and sufficient sample sizes. Reporting both success rates and achieved fitness gives a more informative picture than either metric alone.

8 Conclusions

The experiments support the following conclusions:

- The compact genetic algorithm provides the best overall reliability–speed balance on smooth landscapes.
- The (1+1) EA remains competitive on structured incremental problems but is less robust on deceptive ones.
- The sig-cGA is theoretically attractive but practically sensitive; it benefits from larger budgets and careful parameter tuning.
- Parameter scaling and evaluation budgets have a decisive impact on observed performance.

In constrained-budget settings, incremental frequency-update EDAs such as the cGA are the most practical choice. More conservative significance-based updates become attractive when sufficient sampling depth is available.