arXiv:2605.28703v2 [cs.NE] 7 Sep 2026

A Fresh Look at Lamarckian Evolution and the Baldwin Effect

Inès Benito[1[0009−0005−2435−2152]], Johannes F. Lutzeyer[1[0000−0001−9512−7917]], and Benjamin Doerr[1[0000−0002−9786−220X]]

Laboratoire d'Informatique (LIX), CNRS, École Polytechnique, Institut Polytechnique de Paris, France
{firstname.lastname}@polytechnique.edu

Abstract. Baldwinian and Lamarckian evolution have existed for a long time in evolutionary algorithms (EAs) without ever dominating the academic literature or practical applications. In this work, we use modern empirical and theoretical methods to revisit Lamarckian and Baldwinian evolution and rigorously compare them with the generic Darwinian evolution. On the empirical side, we run a comprehensive suite of experiments on graphs from six different datasets from the recent GraphBench benchmark on Maximum Independent Set and Maximum Cut problems. Our results show that Baldwinian and Lamarckian evolution consistently outperform Darwinian evolution, confirming the great potential of local search augmented evolutionary algorithms. Notably, in the great majority of cases, all EAs outperform recent deep learning baselines and approach the performance of highly specialised heuristic and exact solvers. We furthermore report a high-performing set of generalist parameters for all studied evolution types that we hope will be of use to practitioners in future. On the theoretical side, we extend the existing DeceptiveLeadingBlocks benchmark to arbitrary block length k. For all constant k, we then prove asymptotically tight runtime bounds for the $(1 + 1)$ EA in the three evolution types on this benchmark. For Baldwinian evolution, these are independent of k, whereas for the other two evolution types, the runtimes steeply increase with growing value of k.

Keywords: Memetic Algorithms · Baldwin Effect · Runtime Analysis.

1 Introduction

Evolutionary algorithms (EAs) are population-based metaheuristics inspired by the principles of natural selection [14]. Starting with a population of candidate solutions, they iteratively apply evolutionary operators (crossover, mutation and selection) to retain the fittest individuals for the next generation. Unlike random search, selection pressure biases the search towards promising regions of the solution space, enabling EAs to scale to large and complex problems.

In some cases, standard EAs are effective at exploration, rapidly identifying promising regions of the search space, but can be weaker at exploitation [14]. EAs augmented with a local search step have been proposed to address this

imbalance, giving rise to memetic algorithms [24]. Two distinct paradigms have emerged. In Lamarckian evolution, each offspring is replaced by its locally improved version before selection [32]. In Baldwinian evolution [2, 16], the offspring is left unchanged but evaluated on the fitness of the solution returned by local search. A third approach, also known as "partial Lamarckianism" [17], combines Lamarckian and Baldwinian evolution, which we refer to as LB evolution. In LB evolution, the locally improved solution replaces the offspring with probability p_{LB}. In practice, most work on this topic has favoured either a pure Lamarckian approach or LB evolution [14]. Empirical studies have highlighted a trade-off between Lamarckian and Baldwinian evolution: the former tends to converge faster, while the latter is more reliable on harder instances [32, 4].

Despite their long history, Lamarckian and Baldwinian evolution have not recently been empirically evaluated on state-of-the-art benchmarks at scale. Moreover, to our knowledge, there are no runtime analyses of Baldwinian EAs and only sporadic ones of Lamarckian EAs under the name of memetic algorithms.

In this work, we take a fresh look at all four evolution types using both large-scale empirical evaluation and modern theoretical tools. On the empirical side, we conduct a comprehensive suite of experiments on the GraphBench benchmark [27], covering small and large graphs from three random graph models on the Maximum Independent Set and Maximum Cut problems. Our results show that Baldwinian evolution outperforms Lamarckian evolution on the Maximum Independent Set problem, while both substantially outperform deep learning baselines and match or approach the performance of highly specialised solvers. In our experiments, Darwinian evolution is consistently the weakest, demonstrating the great potential of local search augmentation in practice. We also report a robust set of generalist parameters that incurs an average solution quality loss below 1% compared to problem-specific tuning for all local-search-based evolution types, providing a set of reference values for practitioners. The complete source code reproducing our experiments is publicly available[1].

On the theoretical side, we extend the DeceptiveLeadingBlocks (DLB) benchmark [20] from block length $k = 2$ to arbitrary $k \geq 2$ and prove asymptotic upper and lower bounds on the runtime of Darwinian, Lamarckian, and Baldwinian variants of the $(1 + 1)$ EA. In terms of iterations, Baldwinian evolution is asymptotically faster than Lamarckian, which is in turn asymptotically faster than Darwinian evolution for all $k \geq 2$. When accounting for the cost of the local search procedure, the precise ordering depends on the implementation, but Baldwinian evolution outperforms the others from a small value of k onwards, aligning with the empirical advantage of Baldwinian evolution on hard instances.

2 Related Work

We now provide an overview of the existing literature on Baldwinian, Lamarckian and LB evolution. Hinton and Nowlan [16] showed in the context of Neural

[1] https://github.com/Hypatia-II/polerina

networks that Baldwinian evolution modifies the search space by smoothing the fitness landscape, thereby creating a basin of attraction around the global optimum. Whitley, Gordon and Mathias [32] also found that Baldwinian evolution allows for accelerated convergence towards a global optimum.

Comparative studies showed that while Lamarckian evolution tends to converge faster on easier problems, it is more susceptible to getting trapped in local optima. Baldwinian evolution, on the other hand, is often slower but more reliably converges to the global optimum for difficult problems [32, 4]. Lamarckian evolution has also been applied in other domains, such as evolutionary robotics [23, 22]. A detailed discussion of the encoding requirements for Lamarckian evolution is provided in Appendix A.

Houck et al. [17] studied LB evolution on bounded nonlinear and combinatorial optimisation problems, arguing it provides the most robust performance across diverse problem sets. They find, however, that p_{LB} is problem-specific.

Given the little understanding of the relation of Darwinian, Lamarckian, and Baldwinian evolution in evolutionary computation in general, it is not surprising that there are very few theoretical works on this topic. The only work regarding the Baldwin effect which we are aware of is the seminal paper by Whitley, Gordon, and Mathias [32]. It defines a moderately deceptive function on bitstrings of length four and studies how a simple evolutionary algorithm in the infinite population model copes with this problem. The mathematical study is restricted to the analysis of how the fitness landscape changes, which determines the change of the frequencies of the 16 individuals in the population in one iteration. How this affects the global behaviour of the algorithm is then numerically evaluated. The landscape analysis exhibits several interesting properties of Lamarckian and Baldwinian evolution which still shape today's understanding of these two types of evolution. With a problem size of four and the infinite population model, however, the results are hard to transfer into the asymptotic runtime analysis perspective, which dominates the theoretical analysis of EAs today. We note that in [32] in both the Lamarckian and the Baldwinian EA the local search procedure performs only one hillclimbing step.

While the term Lamarckian evolution is rarely used in these works, most memetic algorithms in fact use Lamarckian evolution, that is, they apply a local search improvement heuristic to each solution generated. We refer to the latest of these works [26] or the survey [29] for a detailed discussion. Overall, these works show that local search can often speed up evolutionary algorithms, however, a complete picture of when and how memetic algorithms profit from local search has not yet been obtained. In [26], local search helps greatly in leaving local optima, but the hurdle benchmark studied there has the particularly suitable property that each local optimum has a Hamming neighbour from which local search can lead to a better local optimum. When this property is absent, memetic algorithms using local search can face enormous difficulties, as shown in [28].

3 Empirical Analysis

We now present an empirical comparison of the four evolution types: Darwinian, Baldwinian, Lamarckian, and LB across two combinatorial optimisation problems on graphs. We introduce the two studied problems and their fitness functions in Section 3.1, we then define the evolutionary algorithms and the local search used in Section 3.2. Subsequently, we report our experimental setup in Section 3.3 and provide our main empirical results in Section 3.4.

3.1 Problem Formulations

We study two discrete combinatorial optimisation problems defined on graphs: Maximum Independent Set (MIS) and the Maximum Cut (MC). We focus on their unweighted versions where all edges are assigned unit weights. Let $G = (V, E)$ be an undirected graph, where $V = \{v_1, \ldots, v_n\}$ is the set of vertices and $E \subseteq \{\{v_i, v_j\} : v_i, v_j \in V\}$ is the set of edges. Let A be the adjacency matrix of G, where $A_{ij} = 1$ if $\{v_i, v_j\} \in E$ and 0 otherwise. A candidate solution is represented by a binary vector $x \in \{0,1\}^n$, where $x_i = 1$ if vertex v_i is selected and $x_i = 0$ otherwise. The MIS problem seeks the largest subset of vertices such that no two vertices in the subset are connected by an edge. The MC problem seeks a partition of the vertex set that maximises the number of edges between the two parts. Formal problem formulations are given in Appendix B. We now proceed to define problem specific fitness functions.

For the MIS problem, we allow non-independent sets in the population and use a penalised fitness function to handle infeasible individuals. This relaxation enables the EAs to explore infeasible regions of the search space, potentially facilitating the crossing of fitness valleys. Infeasible individuals are penalised proportionally to the number of independence violations they incur, steering the search towards feasible solutions. To determine the total number of violations, we evaluate the number of internal edges within the subgraph induced by x, which is computed as $E_{\text{int}}(x) = \frac{1}{2} x^\top A x$. A set is independent if and only if $E_{\text{int}}(x) = 0$. The EA fitness $F_{\text{MIS}}(x)$ is hence defined as

$$F_{\text{MIS}}(x) = \begin{cases} \sum_{i=1}^{n} x_i, & \text{if } E_{\text{int}}(x) = 0; \\ -E_{\text{int}}(x), & \text{otherwise.} \end{cases} \tag{1}$$

For the MC problem, since every binary vector $x \in \{0,1\}^n$ defines a valid partition of V, there are no constraints to penalise. The EA fitness $F_{\text{MC}}(x)$ is therefore simply formulated as

$$F_{\text{MC}}(x) = (1 - x)^\top A x. \tag{2}$$

3.2 Evolutionary Algorithms and Local Search

To tackle the MIS and MC problems, we work with evolutionary algorithms with different evolution types. The core of our implementation is a $(\mu + \lambda)$ Evolutionary Algorithm which maintains a population P of fixed size μ, initialised

Algorithm 1: $(\mu+\lambda)$ EA with Darwinian, Baldwinian, Lamarckian, or LB evolution to maximise the fitness function $f : \{0,1\}^n \to \mathbb{R}$.

```
Initialise population P with μ individuals randomly sampled from {0,1}^n;
repeat
    for i ← 1 to λ do
        Randomly sample x_1, x_2 from P;
        with prob. r_c: y_i ← Mutate(Crossover(x_1, x_2)); else
          y_i ← Mutate(x_1);
        if Darwinian then f_{y_i} ← f(y_i);
        else y_{i,LS} ← LocalSearch(y_i);
        if Baldwinian then f_{y_i} ← f(y_{i,LS});
        if Lamarckian then f_{y_i} ← f(y_{i,LS}); y_i ← y_{i,LS};
        if LB then f_{y_i} ← f(y_{i,LS}); with prob. p_LB: y_i ← y_{i,LS};
    P ← the μ fittest from P ∪ {y_1, ..., y_λ};
until forever;
```

by sampling from $\{0,1\}^n$. For every generation, the λ offspring y_i are generated independently: with probability r_c, offspring y_i results from a uniform crossover between two randomly selected parents, followed by mutation; otherwise, with probability $1-r_c$, offspring results from the mutation of a single randomly sampled parent. The mutation is a standard Bernoulli mutation where each bit x_i is independently flipped with probability $r_{\text{mut}} = 1/n$, yielding an expected number of one bit-flip per offspring. For Baldwinian, Lamarckian and LB evolutions, the offspring undergo a local search. In our local search, we employ a steepest ascent and we search until local optimality [14]. The fitness of the local optimum found is used for evaluation. For Lamarckian evolution, the improved offspring always replaces the initial one, for LB evolution the replacement occurs with probability p_{LB}, and in Baldwinian evolution, the initial offspring is preserved. For Darwinian evolution, the offspring are evaluated directly after mutation. In all EAs, the offspring y_i are evaluated once and their fitness is stored in f_{y_i}. Finally, the μ fittest individuals from the combined pool of parents and offspring are retained for the next generation. The pseudo-code is given in Algorithm 1.

MIS Local Search. For the MIS problem, local search acts as a repair mechanism. For a given node v_i, we define the independence violation count as the number of its neighbours in the selected set, denoted by $c_i(x) = x_i \sum_{j=1}^{n} A_{ij} x_j$. The repair strategy iteratively selects the node v_i with the highest $c_i(x)$ and removes it from the set ($x_i \leftarrow 0$). If multiple vertices share the same $c_i(x)$, a random tie-breaking rule is applied. This process repeats until $E_{\text{int}}(x) = 0$.

MC Local Search. For the Maximum Cut problem, the local search iteratively flips nodes from one partition set to the other. In each step, the vertex that yields the maximum positive gain in cut size is identified. To ensure computational efficiency, we maintain an active list of nodes with positive gains, avoiding a full scan of all nodes at each iteration. When a node is flipped, the gains of its neighbours are updated incrementally and the active list is dynam-

ically adjusted. Random tie-breaking is also used when multiple flips yield the same maximum gain. We stop when no additional gains are possible.

Difference Between Repair and Local Search. The two local search procedures above differ in nature. For MC, the local search optimises the objective (cut size) until no improving flip exists, guaranteeing a local optimum. For MIS, the procedure is a repair mechanism: it only removes vertices to restore feasibility and never attempts to augment the independent set, yielding a feasible but not necessarily locally optimal solution. One could extend this work, as suggested by a reviewer, by following the repair step with a proper local search step for the MIS problem.

3.3 Experimental Setup

We evaluate the four evolution types on six graph datasets from the GraphBench suite. We now detail the benchmark and experiments. Implementation details can be found in Appendix C.2.

Benchmark. We conducted experiments using the GraphBench benchmarking suite [27]. Specifically, we focus on the "Combinatorial Optimization" domain, which provides standardised datasets for the MIS and MC problems. GraphBench provides three random graph models: Erdős-Rényi (ER), Barabási-Albert (BA), and RB-Graphs with two size scales (small and large). Small-scale graphs contain 200-300 nodes and large-scale graphs 700-1200 nodes. Full dataset characteristics are available in [27]. GraphBench benchmarks supervised and unsupervised learning approaches. Our EAs fall into the unsupervised category, i.e., no ground-truth labels are required to run EAs. As an upper-bound reference, GraphBench provides solver-generated solutions (see Appendix C.1). For MIS, solutions are produced by Karlsruhe Maximum Independent Sets (KaMIS) [19], a highly specialised heuristic solver. For MC, solutions are obtained by formulating the problem as an integer programming problem and solving it with Gurobi Optimisation, LLC [15] with a timeout of 3600 seconds.

Experiments. Since the EAs employ an elitist $(\mu + \lambda)$ selection strategy, the best found solution is always retained in the population. Hence, we report the maximum fitness score (independent set size for MIS or cut set size for MC) found in the final population obtained with a fixed computational budget of 40,000 fitness evaluations per experiment.

The EA parameter search space is provided in Table 3 (Appendix C). All four evolution types share the same grid over population size μ, number of offspring λ, crossover rate r_c, initialisation type, and mutation type. The Lamarckian probability p_{LB} is additionally tuned for LB evolution. This results in a total of 252 parameter combinations tested. The grid search is conducted on the training split of the 3 graph kinds (ER, BA, and RB-Graphs), in the two sizes (small and large). For the rest of the paper, and in accordance with GraphBench, we refer to the association of a graph kind and a size as a dataset. To find the optimal parameters, we sample 0.2% of each of the six datasets, yielding 70 graphs per dataset and 420 graphs in total. The 252 parameter combinations are evaluated on all 420 graphs with one repetition per graph, resulting in $252 \times$

Table 1: Mean score (↑, ± std) for EA evolution vs. GraphBench baselines across datasets for MIS and MC. Underlined bold typesets the best EA and best baseline. Bold typesets the values within standard deviation of the best values.

Problem	Type	ER Small	ER Large	BA Small	BA Large	RB Small	RB Large
MIS	GIN	25.42 ± 0.41	26.28 ± 0.41	100.16 ± 3.67	135.00 ± 0.72	17.29 ± 0.33	14.00 ± 0.32
	GT	22.98 ± 0.47	24.98 ± 0.29	99.58 ± 6.45	114.26 ± 0.60	16.54 ± 0.48	13.41 ± 0.14
	MLP	23.18 ± 0.02	24.26 ± 0.45	95.11 ± 2.04	114.49 ± 0.76	16.11 ± 0.10	13.04 ± 0.21
	DeepSets	23.05 ± 0.06	24.22 ± 0.06	95.08 ± 0.17	114.89 ± 0.02	16.02 ± 0.03	13.18 ± 0.04
	Solver	**33.60 ± 1.43**	**45.64 ± 0.63**	**142.86 ± 16.54**	**433.77 ± 19.17**	**20.80 ± 1.82**	**42.55 ± 4.45**
	Darwinian	26.47 ± 1.44	30.73 ± 1.54	140.18 ± 1.34	405.72 ± 4.69	16.96 ± 0.75	31.20 ± 1.23
	Baldwinian	**33.00 ± 0.30**	**42.06 ± 0.90**	**143.42 ± 0.03**	**433.63 ± 0.32**	**20.04 ± 0.21**	**36.89 ± 0.52**
	Lamarckian	30.37 ± 0.94	37.79 ± 1.06	**143.12 ± 0.33**	429.02 ± 2.03	**19.45 ± 0.47**	**37.14 ± 0.88**
	LB	**32.79 ± 0.40**	**40.66 ± 1.72**	**143.38 ± 0.06**	**433.53 ± 0.36**	**20.10 ± 0.21**	**37.61 ± 0.88**
MC	GIN	2327.9 ± 24.8	20878.0 ± 107.9	397.0 ± 0.6	1044.1 ± 0.6	2106.7 ± 14.6	24748.0 ± 87.8
	GT	2172.7 ± 91.8	16534.0 ± 278.0	363.8 ± 0.6	986.9 ± 3.1	1925.7 ± 32.8	21524.0 ± 184.0
	MLP	1866.7 ± 67.6	7335.4 ± 57.5	308.7 ± 0.2	929.2 ± 4.1	1727.7 ± 165.1	20357.0 ± 249.6
	DeepSets	33.6 ± 20.8	27.7 ± 6.8	1.1 ± 0.8	154.3 ± 151.5	140.0 ± 155.5	3575.9 ± 730.0
	Solver	**2835.5**	**23884.0**	**460.9**	**1260.4**	**2920.1**	**33914.0**
	Darwinian	2861.5 ± 14.1	23548.9 ± 44.3	398.4 ± 4.5	1186.1 ± 8.8	2805.9 ± 12.5	31622.6 ± 297.6
	Baldwinian	**2910.1 ± 0.7**	**23857.0 ± 9.1**	**412.6 ± 0.6**	**1248.1 ± 2.7**	**2822.1 ± 0.1**	**32001.3 ± 0.1**
	Lamarckian	**2910.4 ± 0.5**	**23862.1 ± 5.8**	**412.7 ± 0.5**	**1247.8 ± 2.6**	**2822.2 ± 0.0**	**32001.3 ± 0.1**
	LB	**2910.4 ± 0.5**	**23861.6 ± 6.1**	**412.7 ± 0.5**	**1247.6 ± 2.7**	**2822.1 ± 0.1**	**32001.3 ± 0.1**

$420 = 105{,}840$ EA runs per problem. For each (evolution type, dataset) pair, the selected configuration is the one that maximises the mean objective value over the 70 sampled graphs. This procedure is performed for MIS and MC.

Once the optimal parameters are found for each dataset (6) and evolution type (4), we evaluate the corresponding EAs on the full test set of 7,500 graphs using three independent random seeds, for a total of $7{,}500 \times 6 \times 4 \times 3 = 540{,}000$ EA runs per problem. We report the mean objective value and its mean standard deviation. For each graph the standard deviation is computed across the three seeds; the reported standard deviation is the mean of the 7,500 per-graph values.

3.4 Results and Analysis

In this section, we report the results of our experiments and compare them against GraphBench baselines. We analyse solution quality via mean score, i.e., independent set size or cut set size, then examine convergence trajectories and diversity of both optimal solutions found and population over the runs. We also report runtime and present a single cross-problem parameter configuration.

Solution Quality. Following the grid search protocol described in Section 3.3, we identified the optimal EA parameters (provided in Appendix C Table 4) and report the corresponding mean scores for MIS and MC in Table 1.

Across all datasets and both problems, every EA evolution type substantially outperforms the four deep learning baselines (GIN, GT, MLP, DeepSets). The following discussion therefore focuses on the comparison with the solvers.

In Table 1, we observe that in the great majority of cases, the best-performing $p_{\text{LB}} \in \{0.15, 0.3, 0.5, 0.9\}$ for LB tracks whichever evolution type dominates:

0.15 on MIS and 0.9 on MC, pulling LB towards the stronger single evolution type (Appendix C Table 4). Hence, in our experiment, LB offers no systematic advantage: its best configuration simply converges to the highest performing evolution, and can be discarded in favour of the simpler single-mode alternatives.

Darwinian evolution is the weakest EA across all datasets, with both lower mean scores and higher standard deviations. On MIS, the gap to the best repair-based EA ranges from 2% on BA Small to 27% on ER Large, suggesting that without local search, evolutionary operators alone are less effective at navigating the highly constrained MIS feasibility landscape. On MC the gap is smaller (within 5% of the best EA on all datasets).

For MIS, Baldwinian evolution achieves the strongest performance, within standard deviation, on all datasets. It also nearly matches the specialised KaMIS solver: reaching 98.21% of the KaMIS score on ER Small, 93.18% on ER Large, 99.98% on BA Small, 99.92% on BA Large. For MC, Lamarckian and Baldwinian methods reach similar scores with differences within standard deviation. Taken together, Baldwinian evolution is the most consistent choice: it achieves near-optimal MIS performance, matches the best evolution type on MC, and does so with the lowest seed sensitivity of all EA types.

Surprisingly for MC, all four EAs outperform the solver on ER Small. Also, on ER Large, the best EA reaches 99.9% of the solver's score and 99.0% on BA Large, showing that local-search-augmented EAs can be competitive in practice while operating at a fraction of the computational cost (see Table 6 in Appendix C.7).

Beyond the results shared in Table 1, we recorded nine distinct graphs in ER-Large for which EAs found independent sets that are larger than those detected by KaMIS: six using Baldwinian evolution and three using LB evolution ($p_{\mathrm{LB}} = 0.15$). This is noteworthy given that KaMIS is a solver specifically engineered for MIS, combining kernelisation, local search, and branch-and-reduce, whereas our EAs are general-purpose methods with a simple repair mechanism and no problem-specific design. The fact that such a generalist approach can match and occasionally exceed a dedicated specialist solver exemplifies the effectiveness of generalist evolutionary search for combinatorial optimisation.

Convergence. Figure 1 in Appendix C.4 shows the mean fitness against the number of fitness evaluations. On MIS, Baldwinian evolution converges faster than LB, with both reaching comparable optima (Table 1, with RB Large as the sole exception). On MC, all three local search EAs reach near-identical final scores and their convergence curves are largely indistinguishable, consistent with the near-tied values in Table 1; Baldwinian evolution converges marginally slower on some instances. Across both problems, Darwinian evolution requires more fitness evaluations to converge and consistently plateaus at a lower optimum. Overall, local search mechanisms consistently accelerate convergence, with Baldwinian evolution offering the best speed-quality trade-off.

Solution and Population Diversity. We report the diversity of solutions produced by each evolution type in Table 5 in Appendix C.5, where we provide the mean number of unique optimal solutions found during a run. Note that

Table 2: Parameter universality cost (%, ↓): test-set relative loss from a single configuration across datasets and problems instead of per (dataset, problem) tuning. Bold typesets the most robust EA. Cross-problem configurations (μ, λ, r_c): Darwinian $(50, 250, 1.0)$; Baldwinian $(250, 50, 0.9)$; Lamarckian $(10, 10, 0.0)$; LB $(250, 250, 1.0)$ $(p_{\text{LB}} = 0.15)$.

Problem	Evol. Type	ER Small	ER Large	BA Small	BA Large	RB Small	RB Large	Avg.
MIS	Darwinian	4.98	0.00	2.74	0.00	2.59	**−0.11**	1.70
	Baldwinian	0.22	0.00	0.00	0.00	**0.00**	0.09	0.05
	Lamarckian	0.47	**−0.18**	0.00	**−0.06**	**0.00**	0.00	0.04
	LB	**0.00**	−0.13	**−0.01**	0.00	0.10	0.18	**0.02**
MC	Darwinian	0.29	0.53	0.84	1.65	0.10	0.89	0.72
	Baldwinian	**0.00**	**0.01**	**0.03**	**0.11**	**0.00**	0.00	**0.02**
	Lamarckian	0.81	0.68	1.34	1.85	0.05	0.04	0.80
	LB	0.02	0.05	0.10	0.29	0.00	**0.00**	0.08

for Baldwinian evolution, unique solutions are tracked from the locally-searched offspring before they are discarded, ensuring a fair comparison across evolution modes. Darwinian evolution consistently finds very few unique solutions, confirming that local search not only improves mean score but also broadens solution diversity. Figure 2 in Appendix C.6 shows population diversity, measured as mean pairwise Hamming distance, over fitness evaluations. Baldwinian evolution consistently maintains higher population diversity than Lamarckian evolution throughout the runs. For MIS, Baldwinian evolution sustains high population diversity throughout and simultaneously achieves the highest mean scores and the highest solution diversity (Tables 1 and 5). However, diversity alone does not guarantee performance: Darwinian evolution maintains higher diversity than Lamarckian yet reaches lower optima, and LB sustains high diversity on ER Large without achieving the highest mean score. For MC, all local-search-based variants achieve similar performance (Table 1), yet Lamarckian and LB find substantially more unique solutions (Table 5) despite very low diversity.

Computational Cost. Table 6 in Appendix C.7 reports the mean execution time (in seconds). These results have been obtained by running on a sample of 10 graphs, 3 seeds. Since all EAs share the same 40,000 fitness evaluation budget, runtime differences reflect the per-evaluation overhead of each strategy. As expected, Darwinian evolution is the fastest, benefiting from the absence of local search overhead. Conversely, Baldwinian evolution is the most computationally expensive, while keeping to a computational cost that is realistic in practice. Lamarckian evolution significantly mitigates the local search cost by keeping locally optimised individuals in subsequent generations.

Yet across all EA variants, absolute runtimes remain negligible compared to the MC Gurobi solver, which runs with a 3600-second timeout per graph. Even Baldwinian evolution, the most expensive EA type, finishes in under 18 seconds on RB Large, yielding speedups of 200×-2,000× over Gurobi across datasets.

Cross-Problem Parameter Robustness. Running a separate parameter search per problem and dataset is costly. We therefore quantify the loss incurred

by using a single configuration per evolution type across problems. For each evolution type, we identify the cross-problem best configuration and evaluate it on the held-out test set against the per-(dataset, problem) best configuration; more details in Appendix C.8. The relative loss incurred by using the cross-problem configuration is reported in Table 2. The overall losses are small, confirming that cross-problem parameter transfer is viable for all evolution types, and the negative values indicate that the cross-problem configuration occasionally generalises better than per-pair tuning on unseen graphs. Baldwinian evolution is the most robust, with an average loss of 0.05% on MIS and 0.02% on MC and a maximum of 0.22%. LB evolution is comparable, achieving the lowest average loss on MIS (0.02%). Lamarckian evolution incurs a low average loss on MIS (0.04%) but reaches 0.80% on MC, mostly from the BA datasets. Darwinian evolution is the most sensitive, reaching an average loss of 1.70% on MIS and 0.72% on MC.

4 Theoretical Analysis

We complement our experimental findings with a mathematical runtime analysis on a natural generalisation of the established DeceptiveLeadingBlocks benchmark introduced by Lehre and Nguyen [20]. Our results will align with our experimental findings, but more importantly, with the clarity of the mathematical analysis, they will also explain why Baldwinian evolution can be superior to Lamarckian and Darwinian evolution.

The DeceptiveLeadingBlocks Benchmark. We regard the following benchmark problem, which extends the DeceptiveLeadingBlocks benchmark proposed by Lehre and Nguyen [20] in a natural fashion, namely from block length $k = 2$ to arbitrary block lengths $k \geq 2$. Let $k \in \mathbb{N}_{\geq 2}$ and n be a multiple of k. We view the set $[1..n]$, denoting the integers one to n, of bit-positions as partitioned, in a left-to-right fashion, into blocks of length k. The objective value (fitness) of a bit-string x is strongly influenced by the number of blocks, contiguously counted from left to right, that contain only ones. The subsequent block still contributes to the fitness, but to a lesser degree and in a deceptive manner, that is, the contribution is larger when the block contains fewer ones. Consequently, bit-strings composed of a sequence of all-ones blocks followed by an all-zero block are true local optima.

More formally, for a fixed value of k mostly suppressed in the following notation, and for all $\ell \in [1..n/k]$ we call $B_\ell = [(\ell-1)k+1..\ell k]$ the *ℓ-th block*. We write $x_{B_\ell} = (x_{(\ell-1)k+1}, \dots, x_{\ell k}) \in \{0,1\}^k$ to denote the *ℓ-th block* of $x \in \{0,1\}^n$. For $x \neq (1,\dots,1)$, we let $c(x) := \min\{\ell \in [1..n/k] \mid x_{B_\ell} \neq (1,\dots,1)\}$ denote the first block that is not all-ones and call $x_{B_{c(x)}}$ the *critical block of x*. For $x = (1,\dots,1)$, we artificially define $c(x) = n/k + 1$. With these definitions, the DeceptiveLeadingBlocks function with block length k is defined by

$$\mathrm{DLB}(x) := \mathrm{DLB}_k(x) := k(c(x)-1) + (k - 1 - \|x_{B_{c(x)}}\|_1)$$

for all $x \in \{0,1\}^n$. We see that the $c(x)-1$ leading all-ones blocks each contribute k to the fitness, whereas the critical block in a deceptive manner contributes $k - 1 - \|x_{B_{c(x)}}\|_1 \in [0..k-1]$.

As said above, the DeceptiveLeadingBlocks was introduced in [20] for block length $k = 2$ and has received a considerable amount of attention by the theory community since then. Probably, it is the third most prominent multimodal benchmark in the runtime analysis community, behind the Jump and Cliff benchmarks. It differs from these in that a typical EA runs into local optima more frequently than just once.

Lehre and Nguyen [20] used the DLB benchmark to demonstrate that estimation-of-distribution algorithms (EDAs) can have enormous difficulties with local optima, shortly after [5, 7] had shown that they cope surprisingly well with the local optimum of the Jump and Cliff benchmark. Specifically, [20] proved that the UMDA algorithm with parameter settings $\mu = \Theta(\lambda)$ and $\lambda = o(n)$ has an expected runtime of at least $e^{\Omega(\lambda)}$ on DLB, in contrast to the $O(n^3)$ runtime guarantees they showed for many standard evolutionary algorithms. This clear message was questioned in [11], where it was shown that the UMDA with parameters avoiding genetic drift [13] can optimise DLB in $O(n^2 \log n)$ function evaluations with high probability. Subsequently, a $\Theta(n^2)$ expected runtime was shown for the Metropolis algorithm [30] and an $O(n \log n)$ runtime with high probability for the significance-based compact genetic algorithm from [10].

Mathematical Runtime Analysis. Our main results are the following, asymptotically tight runtime guarantees.

Theorem 1. *Let $k \in \mathbb{N}_{\geq 2}$. Then the expected runtimes, measured in iterations, on the DeceptiveLeadingBlocks benchmark, of the $(1+1)$ EA with Darwinian, Lamarckian, and Baldwinian evolution satisfy the following estimates.*

- *Darwin: $\Theta(n^{k+1})$.*
- *Lamarck: $\Theta(n^k)$.*
- *Baldwin: $\Theta(n^2)$.*

Here the asymptotics are for growing problem size n and constant k.

Note critically that we stated the runtimes in terms of iterations as this is the most complete information. When comparing the runtimes over different evolution types, it is important to take into account that an iteration of the Darwinian $(1+1)$ EA mostly consists of a single fitness evaluation (of the offspring), whereas one iteration in the two other evolution types consists mostly of running the local search procedure once on the result of mutation. We discuss the implications of this after the sketch proof of Theorem 1.

Proof Sketch: To present the proof ideas, let us make precise the algorithm we regard. The $(1+1)$ EA works with a parent population of size one (which is why we call this individual the *parent individual*). It is initialised randomly. Then, in each iteration of the main loop, an offspring is generated from the parent via bit-wise mutation with mutation rate $1/n$, that is, by flipping each bit independently with probability $1/n$. If the offspring is at least as good as the parent (in terms of the fitness), then it is *accepted*, that is, replaces the parent; otherwise it is discarded. As common in the mathematical runtime analysis of evolutionary algorithms [25, 1, 18, 33, 12], we do not specify a termination criterion as we

Algorithm 2: The $(1+1)$ EA to maximise $f : \{0,1\}^n \to \mathbb{R}$.

```
1 Let x be chosen uniformly at random from {0,1}^n;
2 repeat
3 |   y ← Mutate(x);
4 |   if f(y) ≥ f(x) then x ← y;
5 until forever;
```

are interested in how long it takes until an optimal solution is generated (via mutation or a local search routine). The pseudocode of this algorithm can be found in Algorithm 2. Precisely, this is the Darwinian version of the $(1+1)$ EA. For the Lamarckian one, any solution that is generated, that is, both the random initial individual and all offspring, is immediately replaced by the result from optimising it via a local search routine. For the Baldwinian $(1+1)$ EA, the individuals are left unchanged, but as their fitness we use the fitness of the result of applying the local search routine to the individual. In Theorem 1, we assume that the local search routine is the best-improvement hillclimber with respect to the standard neighbourhood structure of the discrete hypercube, that is, we repeat checking all neighbours (solutions differing in exactly one bit) of our current solution and replacing it with the best one (breaking ties randomly) until no neighbour is strictly better.

The proof of our result is slightly too technical to give a short summary here, so let us only describe some arguments that show the differences between the three algorithm variants. The main difference is how the three algorithms can leave the local optima of the DLB_k problem. Here the Darwinian EA has no other chance than to flip the k zeros of the critical block to ones as all nearer solutions have a lower fitness. Flipping k particular bits takes time $\Theta(n^k)$. The remaining factor of $\Theta(n)$ stems from the fact that $\Theta(n)$ times a local optimum has to be left. For the Lamarckian EA, when trapped in a local optimum, flipping $k-1$ of the k zeros in the critical block suffices. The resulting solution is a neighbour of a solution having one more leading all-ones block, hence the local search routine will move to that solution and possibly to even better solutions, but in any case with this mutation the local optimum is left. Flipping $k-1$ out of k fixed bits takes time $\Theta(n^{k-1})$, which explains the factor-n speed-up over the Darwinian EA. In contrast, the Baldwinian EA does not see any local optima. Imagine that its current solution x is a local optimum with j leading all-ones blocks. This clearly has fitness $jk+k-1$ also for the Baldwinian EA since local search does not find a better solution. However, any solution differing from x in up to $k-2$ bits of the critical block has the same Baldwinian fitness since local search would return to x (only for the determination of the fitness!). This allows the Baldwinian EA to perform an unbiased walk on this plateau of constant Baldwinian fitness. When this walk has reached a solution with $k-2$ ones in the critical block, then an easy one-bit flip gives a solution in the basin of attraction of a strictly better solution. This constitutes an increase of the Baldwinian fitness, and prevents the algorithm to return to the previous local optimum. That this way of leaving the

local optimum only takes $O(n)$ time is not totally easy to prove and we use an existing runtime analysis for generalised Needle functions [8] for this purpose. We note that we over-simplified the situation a little here – the true plateau of constant Baldwinian fitness is much larger since also solutions with incomplete earlier blocks can be accepted if these blocks contain $k-1$ ones. We have to leave the details to the following formal proof, which can be found in Appendix D.

Towards a Fair Comparison. We now discuss what the result in Theorem 1 means when taking into account the different costs of one iteration due to the local search procedure. Such a comparison is non-trivial since it depends on how efficiently the local search procedure is implemented. As seen in the experimental part of this work, with a moderate understanding of the underlying problem, local search can be implemented significantly more efficiently than via n fitness evaluations per local search step. Since this depends on the particular problem, and on the extent to which it is understood, we here take the pessimistic (for Lamarckian and Baldwinian evolution) view that one iteration of the local search procedure uses n fitness evaluations, and we measure the performance in terms of fitness evaluations.

In the Lamarckian $(1+1)$ EA, the parent is a local optimum and the offspring differs from it in an expected constant number of bits. This implies that an expected constant number of previous all-ones blocks were destroyed, and each need one iteration of local search to be repaired (unless the repair gets stuck prematurely). If the current critical block is not modified so that local search transforms it into an all-ones block, then at most $k-1 = O(1)$ local search iterations may touch this block. Only with probability $O(n^{-(k-1)})$, the current critical block is modified so that local search improves it (in one iteration) to an all-ones block. Above that, only $O(n)$ fitness levels remain, so at most $O(n)$ further iterations of the local search procedure are possible (in reality, this number, in expectation, is only $O(1)$, but we do not need this here). In summary, we see that each local search call in the Lamarckian $(1+1)$ EA performs an expected number of $O(1)$ iterations. When counting one iteration of the local search procedure as n fitness evaluations, the runtime of the Lamarckian $(1+1)$ EA would be $O(n^{k+1})$, on par with the Darwinian $(1+1)$ EA.

For the Baldwinian $(1+1)$ EA, estimating the cost of the local search procedure is a little harder. As an upper bound, we can argue that we have only $n+1$ fitness levels, hence at most n iterations per call of the local search procedure, hence per iteration of the $(1+1)$ EA. Unfortunately, this estimate is asymptotically tight. Consider a block and a time where this block contributes to the Baldwinian fitness, that is, local search had transformed this block and all previous ones into all-ones blocks. Since we have an elitist algorithm, this property remains fulfilled throughout the further run of the algorithm. In particular, this block will always contain either $k-1$ or k ones. Since the algorithm has no strong preference between these two states, in the long run, both states will occur with constant rate (to make this argument precise, we would need to compute the transition probabilities, which are both $\Theta(1/n)$ for switching the state, and the

mixing time, which is $O(n)$ as follows from elementary properties of two-state Markov chains). Hence the expected number of blocks that contribute to the fitness, but have $k-1$ ones (and thus need a local search iteration), of a solution with fitness $\Theta(n)$ is $\Theta(n)$. Hence in a constant fraction of the iterations, the local search procedure performs $\Omega(n)$ iterations. When counting a local search iteration as n fitness evaluations, this leads to a runtime of $\Theta(n^4)$ fitness evaluations for the Baldwinian $(1+1)$ EA. Hence in this view, the Baldwinian $(1+1)$ EA is worse than the two others for the easiest case of $k = 2$, all three are asymptotically equal for $k = 3$, and Baldwinian evolution outperforms the others for all $k \geq 4$.

In a practical application, problem-specific information might allow for more efficient implementations of the local search routine, either because not all neighbours of the current solution need to be inspected, or because the fitness of a solution known to be a neighbour can be computed more efficiently than a general fitness evaluation. In this case, the Lamarckian $(1+1)$ EA would have an asymptotic advantage over the Darwinian one for all k. Since an iteration in any case takes at least one fitness evaluation, the runtime would still be at least $\Omega(n^k)$ fitness evaluations, showing that the advantage of the Lamarckian $(1+1)$ EA is limited to a factor of $O(n)$, and that the Baldwinian $(1+1)$ EA also without accounting for such improvements outperforms it for $k \geq 5$.

Overall, a precise performance comparison depends critically on the local search procedure's cost. The Baldwinian $(1+1)$ EA will outperform the others from a certain small value of k on. This aligns with the general belief in the community that Baldwinian evolution is more suitable for more difficult problems.

5 Conclusion

We have taken a fresh look at Darwinian, Baldwinian, Lamarckian, and LB evolution, which have mostly existed independently in the EA literature without empirical or formal theoretical comparisons. On the empirical side, we demonstrated that local search augmentation consistently improves performance over Darwinian evolution. Baldwinian evolution is the most consistent choice overall, while Lamarckian evolution is a competitive alternative on MC. The LB hybrid offers no systematic advantage over either. Finally, we identified a generalist parameter configuration per evolution type, providing a practical off-the-shelf starting point for future studies. On the theoretical side, we extended the DLB benchmark to arbitrary block length $k \geq 2$ and proved tight asymptotic runtime bounds for the $(1+1)$ EA: Baldwinian evolution runs in $\Theta(n^2)$ iterations, Lamarckian evolution in $\Theta(n^k)$, and Darwinian evolution in $\Theta(n^{k+1})$.

When accounting for the cost of the local search procedure in fitness evaluations, the runtime ranking depends on implementation, but Baldwinian evolution outperforms the others from a small value of k onwards. These results have clear practical implications: local search augmentation should more frequently be considered over Darwinian evolution; in particular, Baldwinian evolution is the most promising starting point in the absence of problem-specific knowledge.

In addition, our generalist parameter configurations for the different evolution types could be a useful practical reference.

Acknowledgements

This work was supported by FMJH Program PGMO and the French National Research Agency (ANR) via the "GraspGNNs" JCJC grant (ANR-24-CE23-3888).

References

1. Auger, A., Doerr, B. (eds.): Theory of Randomized Search Heuristics. World Scientific Publishing (2011)
2. Baldwin, J.M.: A new factor in evolution. The American Naturalist **30**, 441–451 (1896)
3. Böttcher, S., Doerr, B., Neumann, F.: Optimal fixed and adaptive mutation rates for the LeadingOnes problem. In: Parallel Problem Solving from Nature, PPSN 2010. pp. 1–10. Springer (2010)
4. Castillo-Valdivieso, P.Á., García-Arenas, M., García Castellano, F.J., Merelo Guervós, J.J., Prieto, A., Rivas, V.M., Romero, G.: Lamarckian evolution and the Baldwin effect in evolutionary neural networks. CoRR **abs/cs/0603004** (2006)
5. Doerr, B.: A tight runtime analysis for the cGA on jump functions: EDAs can cross fitness valleys at no extra cost. In: Genetic and Evolutionary Computation Conference, GECCO 2019. pp. 1488–1496. ACM (2019)
6. Doerr, B.: Probabilistic tools for the analysis of randomized optimization heuristics. In: Doerr, B., Neumann, F. (eds.) Theory of Evolutionary Computation: Recent Developments in Discrete Optimization, pp. 1–87. Springer (2020), also available at https://arxiv.org/abs/1801.06733
7. Doerr, B.: The runtime of the compact genetic algorithm on Jump functions. Algorithmica **83**, 3059–3107 (2021)
8. Doerr, B., Kelley, A.J.: The runtime of Randomized Local Search on the generalized Needle problem. IEEE Transactions on Evolutionary Computation **29**, 1864–1872 (2025). https://doi.org/10.1109/TEVC.2024.3453776
9. Doerr, B., Kötzing, T.: Lower bounds from fitness levels made easy. Algorithmica **86**, 367–395 (2024)
10. Doerr, B., Krejca, M.S.: Significance-based estimation-of-distribution algorithms. IEEE Transactions on Evolutionary Computation **24**, 1025–1034 (2020)
11. Doerr, B., Krejca, M.S.: The univariate marginal distribution algorithm copes well with deception and epistasis. Evolutionary Computation **29**, 543–563 (2021)
12. Doerr, B., Neumann, F. (eds.): Theory of Evolutionary Computation—Recent Developments in Discrete Optimization. Springer (2020), also available at http://www.lix.polytechnique.fr/Labo/Benjamin.Doerr/doerr_neumann_book.html
13. Doerr, B., Zheng, W.: Sharp bounds for genetic drift in estimation-of-distribution algorithms. IEEE Transactions on Evolutionary Computation **24**, 1140–1149 (2020)
14. Eiben, A.E., Smith, J.E.: Introduction to Evolutionary Computing, Second Edition. Springer (2015)
15. Gurobi Optimization, LLC: Gurobi Optimizer Reference Manual (2024)

16. Hinton, G.E., Nowlan, S.J.: How learning can guide evolution. Complex Systems **1** (1987)
17. Houck, C.R., Joines, J.A., Kay, M.G., Wilson, J.R.: Empirical investigation of the benefits of partial Lamarckianism. Evolutionary Computation **5**, 31–60 (1997)
18. Jansen, T.: Analyzing Evolutionary Algorithms – The Computer Science Perspective. Springer (2013)
19. Lamm, S., Schulz, C., Strash, D., Williger, R., Zhang, H.: Exactly solving the maximum weight independent set problem on large real-world graphs. In: Algorithm Engineering and Experiments, ALENEX 2019. pp. 144–158. SIAM (2019)
20. Lehre, P.K., Nguyen, P.T.H.: On the limitations of the univariate marginal distribution algorithm to deception and where bivariate EDAs might help. In: Foundations of Genetic Algorithms, FOGA 2019. pp. 154–168. ACM (2019)
21. Lehre, P.K., Witt, C.: Black-box search by unbiased variation. Algorithmica **64**, 623–642 (2012)
22. Luo, J., Miras, K., Longhi, C., Weissl, O., Eiben, A.E.: Lamarckian inheritance improves robot evolution in dynamic environments. IEEE Transactions on Evolutionary Computation pp. 1–1 (2025)
23. Luo, J., Miras, K., Tomczak, J., Eiben, A.E.: Enhancing robot evolution through Lamarckian principles. Scientific Reports **13**, 21109 (2023)
24. Moscato, P.: On evolution, search, optimization, genetic algorithms and martial arts: Towards memetic algorithms. Caltech concurrent computation program, C3P Report (1989)
25. Neumann, F., Witt, C.: Bioinspired Computation in Combinatorial Optimization – Algorithms and Their Computational Complexity. Springer (2010)
26. Nguyen, P.T.H., Sudholt, D.: Memetic algorithms outperform evolutionary algorithms in multimodal optimisation. Artificial Intelligence **287**, 103345 (2020)
27. Stoll, T., Qian, C., Finkelshtein, B., Parviz, A., Weber, D., Frasca, F., Shavit, H., Siraudin, A., Mielke, A., Anastacio, M., Müller, E., Bechler-Speicher, M., Bronstein, M.M., Galkin, M., Hoos, H.H., Niepert, M., Perozzi, B., Tönshoff, J., Morris, C.: GraphBench: Next-generation graph learning benchmarking. CoRR **abs/2512.04475** (2025)
28. Sudholt, D.: Hybridizing evolutionary algorithms with variable-depth search to overcome local optima. Algorithmica **59**, 343–368 (2011)
29. Sudholt, D.: Memetic evolutionary algorithms. In: Auger, A., Doerr, B. (eds.) Theory of Randomized Search Heuristics: Foundations and Recent Developments, pp. 141–169. World Scientific (2011)
30. Wang, S., Zheng, W., Doerr, B.: Choosing the right algorithm with hints from complexity theory. Information and Computation **296**, 105125 (2024)
31. Wegener, I.: Simulated annealing beats Metropolis in combinatorial optimization. In: Automata, Languages and Programming, ICALP 2005. pp. 589–601. Springer (2005)
32. Whitley, L.D., Gordon, V.S., Mathias, K.E.: Lamarckian evolution, the Baldwin effect and function optimization. In: Parallel Problem Solving from Nature, PPSN 1994. pp. 6–15. Springer (1994)
33. Zhou, Z.H., Yu, Y., Qian, C.: Evolutionary Learning: Advances in Theories and Algorithms. Springer (2019)

A Encoding Requirements for Lamarckian Evolution

We begin by supplementing our review of related work with a note on a subtle advantage that Baldwinian evolution has been observed to gain. A critical requirement for Lamarckian evolution is that improvements found by local search must be successfully written back into the encoding of an individual. In the combinatorial settings considered in this work, this mapping is trivial since local search operates directly on the encoding, and the improved solution can therefore replace the offspring. In other domains, such as evolutionary robotics, the encoding describes how to build an individual rather than the individual itself, and local search operates on the resulting structure [23, 22]. Translating improvements back into the encoding is then a non-trivial design problem, and is sometimes impossible. Baldwinian evolution leaves the encoding untouched and only modifies the fitness of the individual, which is a structural advantage in settings with complex representations.

B Problem Formulations

In this section, we formally define the two combinatorial optimisation problems studied in this paper. Recall that we let $G = (V, E)$ be an undirected graph, where $V = \{v_1, \ldots, v_n\}$ is the set of vertices and $E \subseteq \{\{v_i, v_j\} : v_i, v_j \in V\}$ is the set of edges. A candidate solution is represented by a binary vector $x \in \{0,1\}^n$, where $x_i = 1$ if vertex v_i is selected and $x_i = 0$ otherwise.

B.1 Maximum Independent Set (MIS)

An independent set of a graph G is a subset of vertices $S \subseteq V$ such that no two vertices in S are connected by an edge. The MIS problem seeks to find an independent set S of maximal cardinality $|S|$. Using the binary representation x, where $x_i = 1$ if $v_i \in S$ and $x_i = 0$ otherwise, the problem can be formulated as

$$\begin{aligned} \text{maximise} \quad & \phi_{\text{MIS}}(x) = \sum_{i=1}^{n} x_i \\ \text{subject to} \quad & x_i + x_j \leq 1, \quad \forall \{v_i, v_j\} \in E \\ & x_i \in \{0,1\}, \quad i = 1, \ldots, n. \end{aligned} \tag{3}$$

B.2 Maximum Cut (MC)

The Maximum Cut problem involves partitioning the vertex set V into two disjoint sets S and $V \setminus S$. The goal is to maximise the number of edges that have one endpoint in S and the other in $V \setminus S$. Using the binary encoding x, where $x_i = 1$ if $v_i \in S$ and $x_i = 0$ if $v_i \in V \setminus S$, the problem can be formulated as

$$\begin{aligned} \text{maximise} \quad & \phi_{\text{MC}}(x) = \sum_{\{v_i, v_j\} \in E} (x_i(1 - x_j) + x_j(1 - x_i)) \\ \text{subject to} \quad & x_i \in \{0,1\}, \quad i = 1, \ldots, n. \end{aligned} \tag{4}$$

Unlike MIS, MC is an unconstrained optimisation problem, as every binary vector x corresponds to a valid partition of the graph.

C Additional Experimental Details

This appendix section collects the technical details that support the empirical analysis in Section 3.4. It includes a note on solver performance (Appendix C.1), implementation choices for the large-scale parameter search (Appendix C.2), the full parameter search space and best per-dataset configurations (Tables 3 and 4, Appendix C.3), convergence trajectories (Figure 1, Appendix C.4), population diversity over runs (Figure 2, Appendix C.6), diversity of optimal solutions found (Table 5, Appendix C.5), mean execution times (Table 6, Appendix C.7), and the procedure for selecting the universal parameter combination and computing the universality cost (Appendix C.8).

C.1 A Note On the Displayed Solver Performance

While all baseline results come from the GraphBench paper, we recomputed the MIS solver results from the true labels available in the test set provided with the GraphBench benchmark. This recomputation of the results led to a small and ultimately inconsequential, difference in the reported solver values.

C.2 Implementation Details

Given the scale of the GraphBench datasets and the large number of EA runs required by the parameter search (105,840 runs per problem), and the use of local search steps, computational efficiency is critical. The graph's topology is represented using sparse adjacency matrices in Compressed Sparse Row (CSR) format. Fitness evaluation for both problems is implemented as sparse matrix-vector multiplication via SciPy's sparse routines, reducing the complexity of fitness evaluation from $O(n^2)$ with a dense adjacency matrix to $O(|E|)$ with the sparse format, since only the $|E|$ non-zero entries are visited during the matrix-vector product. This allows for efficient scaling to the larger benchmark instances investigated in this study. The local search and local search logic for both problems is implemented in JIT-compiled Python using Numba to achieve near-native execution speeds. The complete source code reproducing our experiments can be accessed on github: https://github.com/Hypatia-II/polerina.

C.3 Parameter Configurations

Table 3 reports the full parameter search space shared by all four evolution types.

Table 3: Parameter Search Space for Evolutionary Types: Darwinian, Baldwinian, Lamarckian and LB. †Only applied in the LB evolution type.

Parameter	**Values Tested**
Population Size (μ)	{10, 50, 250}
Initialisation Type	Random
Number of Offspring (λ)	{10, 50, 250}
Crossover Rate (r_c)	{0.0, 0.5, 0.9, 1.0}
Mutation Type	Bernoulli
Lamarckian Probability (p_{LB})†	{0.15, 0.3, 0.5, 0.9}

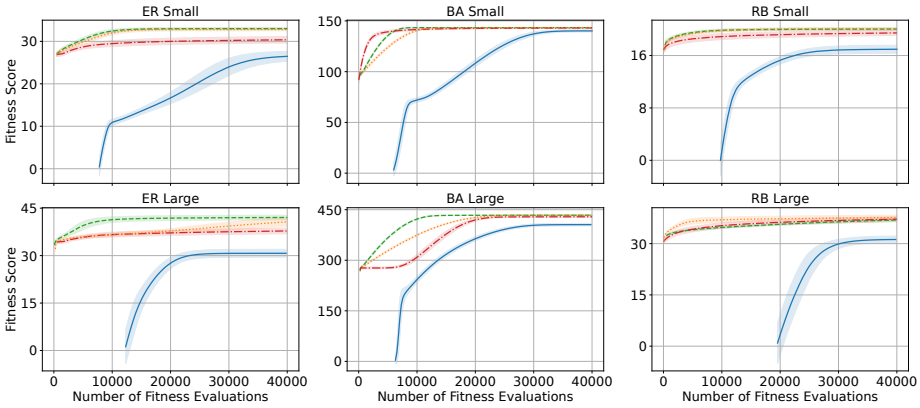
Table 4 reports the best configuration selected for each evolution type, dataset, and problem after grid search on the training split.

Table 4: Best evolution type parameter configuration for MIS and MC. All configurations used Bernoulli mutation and random initialisation. Values shown as $\mu/\lambda/r_c$; Darwinian, Baldwinian, and Lamarckian evolution always have $p_{\text{LB}} = 0$ (—).

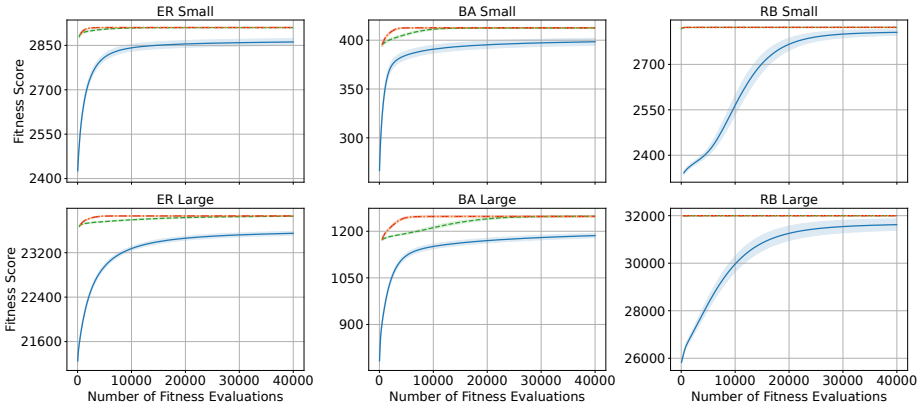
		MIS		**MC**	
Dataset	**Type**	$\boldsymbol{\mu/\lambda/r_c}$	$\boldsymbol{p_{\text{LB}}}$	$\boldsymbol{\mu/\lambda/r_c}$	$\boldsymbol{p_{\text{LB}}}$
ER Small	Darwinian	250/50/1.0	—	10/10/0.0	—
	Baldwinian	250/250/1.0	—	250/50/1.0	—
	Lamarckian	250/250/0.0	—	250/250/1.0	—
	LB	250/250/1.0	0.15	250/250/1.0	0.9
ER Large	Darwinian	50/250/1.0	—	10/10/0.5	—
	Baldwinian	50/50/1.0	—	250/50/1.0	—
	Lamarckian	250/250/0.0	—	250/250/1.0	—
	LB	250/10/1.0	0.15	250/250/1.0	0.9
BA Small	Darwinian	250/250/1.0	—	10/10/0.5	—
	Baldwinian	250/50/0.9	—	250/250/0.9	—
	Lamarckian	10/10/0.0	—	250/250/0.9	—
	LB	250/250/0.9	0.15	250/250/0.9	0.9
BA Large	Darwinian	50/250/1.0	—	10/10/1.0	—
	Baldwinian	250/10/1.0	—	250/250/1.0	—
	Lamarckian	250/10/1.0	—	250/250/0.9	—
	LB	250/250/1.0	0.15	250/250/0.9	0.9
RB Small	Darwinian	250/10/0.5	—	250/250/1.0	—
	Baldwinian	250/50/0.9	—	10/10/1.0	—
	Lamarckian	10/10/0.0	—	50/250/1.0	—
	LB	250/10/1.0	0.15	50/50/0.9	0.5
RB Large	Darwinian	50/250/0.9	—	10/50/1.0	—
	Baldwinian	250/250/1.0	—	250/250/0.9	—
	Lamarckian	10/10/0.0	—	250/50/0.9	—
	LB	50/10/1.0	0.15	250/10/0.9	0.9

C.4 Convergence

We now provide the mean convergence speed results in Figure 1.



(a) MIS



(b) MC

Fig. 1: Mean fitness score (↑) over fitness evaluations per EA evolution across datasets for MIS and MC. Curves use the per-(dataset, problem) best parameter configuration. Darwinian (solid blue), Baldwinian (dashed green), Lamarckian (dash-dot red), LB (dotted orange).

C.5 Diversity of Optimal Solutions

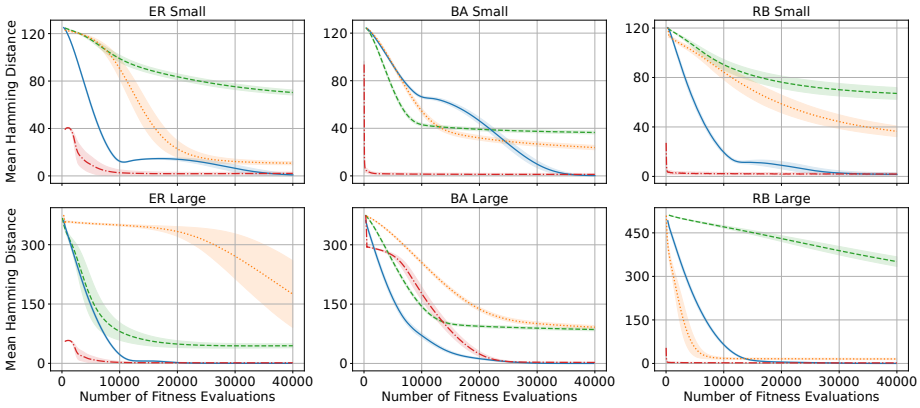
Table 5 provides results on the diversity of the optimal solutions found by the different evolution types.

Table 5: Mean unique optimal solutions (↑, ± std) per EA evolution across datasets for MIS and MC. Bold typesets the most diverse EA.

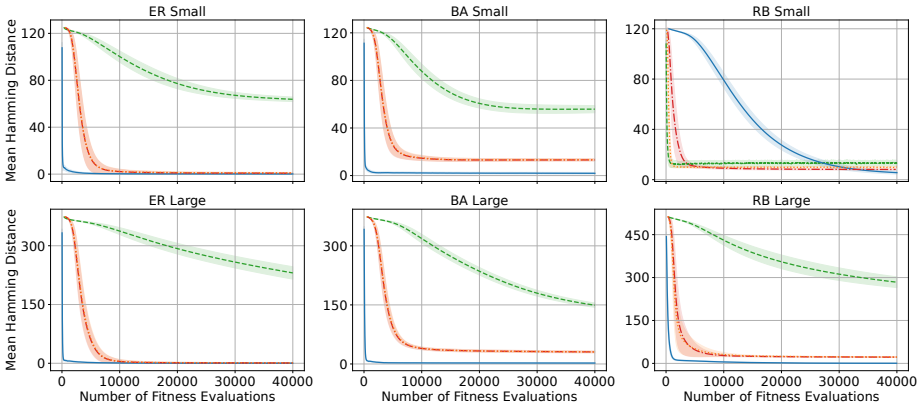
Problem	Evol. Type	ER Small	ER Large	BA Small	BA Large	RB Small	RB Large
MIS	Darwinian	3.7 ± 2.4	1.5 ± 0.6	5.7 ± 2.7	1.7 ± 0.7	15.0 ± 10.2	3.2 ± 2.0
	Baldwinian	**71.9 ± 56.6**	**94.9 ± 101.1**	**14290.2 ± 1156.0**	**6055.2 ± 2696.0**	**12651.4 ± 3766.2**	1933.8 ± 1278.8
	Lamarckian	28.6 ± 21.6	24.2 ± 16.6	442.8 ± 207.3	329.8 ± 152.8	1388.3 ± 549.7	856.3 ± 477.7
	LB	57.0 ± 47.9	34.3 ± 41.4	7915.6 ± 886.7	2276.3 ± 1138.7	7651.9 ± 2541.1	**5739.1 ± 2545.3**
MC	Darwinian	7.3 ± 5.7	3.7 ± 2.6	1133.0 ± 753.8	405.1 ± 335.7	99.1 ± 66.4	49.1 ± 25.3
	Baldwinian	8.3 ± 4.0	2.5 ± 2.0	14475.7 ± 3012.8	2253.4 ± 1870.7	8781.2 ± 350.2	8974.9 ± 734.5
	Lamarckian	**9.0 ± 3.9**	**8.0 ± 6.4**	**30502.3 ± 1931.0**	**27529.0 ± 2885.1**	10935.8 ± 323.4	**12030.7 ± 461.3**
	LB	**9.0 ± 3.9**	**8.0 ± 6.4**	30412.5 ± 2015.7	27449.5 ± 2967.7	**11250.6 ± 199.3**	12024.1 ± 526.9

C.6 Population Diversity

Figure 2 provides results on the population diversity throughout the runs for each evolution types.



(a) MIS



(b) MC

Fig. 2: Mean pairwise Hamming distance as a measure of population diversity over fitness evaluations per EA evolution across datasets for MIS and MC. Curves use the per-(dataset, problem) best parameter configuration. Darwinian (solid blue), Baldwinian (dashed green), Lamarckian (dash-dot red), LB (dotted orange).

C.7 Runtime

Table 6 provides runtime results for the different evolution types.

Table 6: Mean runtime in seconds (↓, ± std) per EA evolution across datasets for MIS and MC. Bold typesets the fastest EA.

Problem	Evol. Type	ER Small	ER Large	BA Small	BA Large	RB Small	RB Large
MIS	Darwinian	**0.36** ± **0.00**	**1.97** ± **0.02**	**0.17** ± **0.00**	**0.61** ± **0.01**	0.75 ± 0.01	**2.54** ± **0.10**
	Baldwinian	1.93 ± 0.45	13.49 ± 0.43	0.64 ± 0.02	4.50 ± 0.13	1.65 ± 0.07	19.69 ± 0.51
	Lamarckian	0.53 ± 0.42	2.12 ± 0.02	0.32 ± 0.00	2.12 ± 0.06	**0.56** ± **0.02**	3.40 ± 0.07
	LB	0.84 ± 0.18	12.17 ± 1.01	0.37 ± 0.02	2.19 ± 0.06	1.29 ± 0.07	4.60 ± 0.20
MC	Darwinian	**0.44** ± **0.00**	**2.62** ± **0.01**	0.26 ± 0.01	**0.57** ± **0.00**	**0.29** ± **0.00**	**2.45** ± **0.07**
	Baldwinian	1.60 ± 0.44	11.07 ± 0.25	0.53 ± 0.01	2.19 ± 0.05	1.74 ± 0.05	17.97 ± 0.42
	Lamarckian	0.89 ± 0.42	4.93 ± 0.09	**0.24** ± **0.01**	0.94 ± 0.02	0.64 ± 0.02	6.63 ± 0.24
	LB	0.76 ± 0.16	5.02 ± 0.06	0.25 ± 0.00	1.10 ± 0.01	0.64 ± 0.02	8.96 ± 0.41

C.8 Cross-Problem Parameter Combination Selection and Universality Cost

We now detail the procedure behind the parameter universality cost reported in Table 2. For each evolution type, the goal is to select a universal parameter combination that performs well across both problems (MIS and MC) on all datasets, and to quantify the performance lost by using it in place of a specialised combination tuned per (problem, dataset) pair.

Cross-Problem Parameter Combination Selection. For each (problem, dataset, parameter combination), we compute a mean score by averaging over repetitions and graphs. These raw scores have scales that vary across datasets and problems, and therefore cannot be compared directly. We thus normalise them within each (problem, dataset, evolution type) group: every parameter combination's mean score is divided by the best mean score achieved for that evolution type on that (problem, dataset) pair. This places all values on a common $(0, 1]$ scale, where 1 marks the best-performing combination of that evolution type on that (problem, dataset) pair. For each evolution type, we then average these normalised scores across all (problem, dataset) pairs and select the highest-scoring combination as the universal one. The resulting combinations are listed in the caption of Table 2.

Universality Cost. For each (evolution type, problem, dataset) cell, we compare the mean score of the universal parameter combination, $s_{\text{universal}}$, against the score of the best-performing combination for that (problem, dataset) pair, $s_{\text{specialised}}$. The relative loss is

$$\text{loss} = \frac{s_{\text{specialised}} - s_{\text{universal}}}{s_{\text{specialised}}} \times 100\%.$$

Table 2 reports this loss for each problem, dataset and evolution type, along with the per-problem averages.

D Formal Proof of Theorem 1

We now provide the full proof of Theorem 1. Note that since we treat k as a constant, in the following proof we do not optimise for the dependence of our runtime bounds on k. In fact, our result is insightful already for $k = 2$ and $k = 3$, hence determining the asymptotics also in k is less important.

Proof (of Theorem 1). We start with proving the upper bound for the Darwinian EA, where we can use a standard fitness level argument. Assume that the current solution of the EA has $c(x) - 1$ leading all-ones blocks and that the critical block contains $\ell \in [1..k]$ zeros. Then with probability at least $n^{-\ell}(1-1/n)^{n-\ell} \geq \frac{1}{e}n^{-k}$, exactly the zeros in the critical block are flipped and consequently the offspring contains at least $c(x)$ leading all-ones blocks. Hence the expected waiting time for increasing the number of leading all-ones blocks is $O(n^k)$. Note that such an offspring is accepted, that is, becomes the new parent, and note further, that the quantity $c(x)$ for the parent individual cannot reduce over time due to the elitist selection of the $(1 + 1)$ EA. Consequently, after at most $n/k = O(n)$ such improvements, the EA has found the optimum. Adding the waiting times for the improvements gives the desired bound for the expected runtime of $O(n^{k+1})$.

For the lower bound, we use a recent result [9] that allows us to prove lower bounds via fitness level arguments and level visiting probabilities. We use the true fitness levels, that is, for each $i \in [0..n]$ the i-st fitness level consists of all solutions having fitness exactly i. We are particularly interested in the case $i = jk + k - 1$, that is, the solutions having j leading all-ones blocks and then only zeros in the critical block. We first argue that each such fitness level is visited with probability at least $\frac{1}{2^k}$ (this is a very pessimistic estimate, but it suffices for our purposes). Fix $j \in [0..n/k - 1]$. Let x be the first solution generated by the algorithm that has at least j leading all-ones blocks. If x is the random initial solution, then, conditional on having at least j leading all-ones blocks, the probability that $x_{jk+1}, \dots, x_{jk+k}$ are all zero is 2^{-k}. In this case, we have $\mathrm{DLB}(x) = jk + k - 1$. Let now x not be the initial solution. Then it was generated from some solution y having less than j leading all-ones blocks. A simple induction shows that in y and in all solutions previously generated by the algorithm, the bits at positions $jk + 1, \dots, jk + k$ are independently and uniformly distributed (since this is true for the initial solution and since also in the following, the values of these bits never had an influence on the run of the algorithm). This is a classic argument known from the analysis of LeadingOnes, see [3, Lemma 1] or [21, Lemma 1], that has been extended to DLB_2 in [30, Lemma 5]. Since x is an offspring of such a y and since all we condition on is that x has the first jk bits equal to one, also in x the bits at positions $jk + 1, \dots, jk + k$ are independently and uniformly distributed. Consequently, they are all zero with probability 2^{-k}. We have thus shown that the first solution x with at least j leading all-ones blocks with probability 2^{-k} has fitness $jk+k-1$. Since any run of the algorithm at some time visits such a solution x, we know that it visits the $(jk + k - 1)$-st fitness level with probability at least $v_{jk+k-1} := 2^{-k}$. When on this fitness level, the probability to leave it in one iteration is exactly

$p_{jk+k-1} := (1-1/n)^{jk} n^{-k}$, since any solution with higher fitness has the first $jk+k$ bits equal to one. By [9, Theorem 8], the expected runtime of our algorithm is at least $\sum_{i=0}^{n-1} \frac{v_i}{p_i}$, where v_i is a lower bound for the probability to visit the i-st fitness level and p_i is an upper bound for the probability to leave it in one iteration. Using the values we just estimated and only regarding these fitness levels, we obtain the lower bound

$$\sum_{j=0}^{n/k-1} \frac{v_{jk+k-1}}{p_{jk+k-1}} \geq \sum_{j=0}^{n/k-1} 2^{-k} n^k = \Omega(n^{k+1}),$$

since we treat k as a constant.

We now continue with the proof for the Lamarckian $(1+1)$ EA. By definition, all its parent solutions are local optima. By definition of the DLB problem, these are exactly the bit strings with jk leading ones followed by k zeros, for some $j \in [0..n/k-1]$, together with the global optimum, the all-ones string. For the upper bound, we use a fitness level argument for these fitness levels. Assume that the current solution has jk leading ones followed by k zeros, for some $j \in [0..n/k-1]$. Then with probability exactly $(1-1/n)^{jk+1} n^{-(k-1)} \geq \frac{1}{e} n^{-k+1}$ the offspring has $jk+k-1$ leading ones followed by a zero. Now the local search procedure finds two types of improving neighbours, namely $k-1$ in which a one in the critical block is flipped to zero and the one in which the last zero in the critical block is flipped to one. The latter having more leading all-ones blocks than the former, it also has the highest fitness, hence this is the next solution visited by the local search procedure. Since the local search procedure does not accept worsenings, these $j+1$ leading all-ones blocks will not be touched anymore. Consequently, we have obtained a solution with fitness better than the parent, that is, we have left the fitness level. We have thus shown that each of the n/k non-optimal fitness levels are left with probability at least $\frac{1}{e} n^{-k+1}$ per iteration. By the classic fitness level theorem [31], this gives an upper bound of $(n/k)en^{k-1} = O(n^k)$ iterations for the expected runtime.

For the lower bound, we again use the fitness level method for lower bounds, with some extra care with respect to the local search step. We estimate the visiting probabilities for the fitness levels. Recall that for the Lamarckian algorithm only solutions with fitness $kj+k-1$, that is, having j leading all-ones blocks and then k zeros, will be taken as parent individual. Let x be the first individual forming the parent population that has fitness at least $kj+k-1$. This individual is either the random initial individual y to which local search was applied, or a mutation offspring y to which local search was applied. Since all previous individuals had a lower fitness (also after applying local search), as in the lower-bound proof for Darwinian evolution, the bits $y_{kj+1}, \ldots, y_{kj+k}$ are uniformly and independently distributed. In particular, with probability $1-(k+1)2^{-k}$, at least two of them are zero. Consequently, in this case, the local search procedure applied to y will, after having possibly optimised earlier blocks to all-ones blocks, optimise the $(j+1)$-st block to an all-zero block. This shows that the Lamarckian EA visits the fitness level $kj+k-1$ with probability $v_{kj+k-1} = 1-(k+1)2^{-k} \geq 1/4$. When on such a fitness level, that is, when the

parent individual has j leading all-ones blocks followed by an all-zeros block, to leave this fitness level to a higher level it is necessary that the mutation flips at least $k-1$ of the zeros in the critical block (otherwise, the local search cannot transform the critical block into an all-ones block). This happens with probability $kn^{-k+1} + n^{-k} \leq (k+1)n^{-k+1} =: p_{kj+k-1}$. Using the fitness-level theorem for lower bounds again, we obtain a lower bound for the expected runtime of $\sum_{j=0}^{n/k-1} \frac{v_{kj+k-1}}{p_{kj+k-1}} \geq (n/k)(1/4)n^{k-1}/(k+1) = \frac{n^k}{12(k+1)} = \Omega(n^k)$.

We finally turn to the Baldwinian EA. Since the fitness of any individual is the fitness of a local optimum, the same fitness levels are visited as for the Lamarckian EA, but a wider set of parent individuals may occur. More specifically, if the Baldwinian fitness of an individual is $jk+k-1$, then the individual can be any that has at least $k-1$ ones in each of the first j blocks and at most $k-2$ ones in the $(j+1)$-st block (and these are all possibilities).

On this fitness level, the algorithm performs a Markov chain as follows. Whether the offspring is accepted is determined by the first j blocks, namely whether the offspring has again at least $k-1$ ones in each of these blocks. This happens with probability at least $(1-1/n)^{kj} \geq \frac{1}{e}$, simply by regarding the event that none of the first kj bits is flipped. In the $(j+1)$-st block, each bit is flipped independently with probability $1/n$. By a simple union bound, the probability that some bit there is flipped is at most $k/n = \Theta(1/n)$, hence the typical event is that no bit is flipped. Again by a union bound over the sets of two bits in this block, the probability that more than one bit is flipped in this block is at most $\binom{k}{2}n^{-2} = \Theta(1/n^2)$. Hence, very roughly speaking, what happens in the $(j+1)$-st block is a pure random walk, slowed down by a factor of $\Theta(n/k)$, with very exceptional events that we move by more than one step. This almost-random walk ends when at least $k-1$ ones in the critical block are reached, as then the local search will bring us to a higher fitness level. All this allows us to use a runtime analysis of the randomized local search (RLS) heuristic on generalized needle functions [8], as follows. Consider a sequence of

$$t = 16 \cdot 2^k n$$

iterations. The probability that in this time interval at least once two bits in the $(j+1)$-st block are flipped is at most $t\binom{k}{2}n^{-2} = O(1/n)$. Let us call an iteration a *move* if the actions of the mutation operator are such that the offspring is accepted (as discussed, this is decided in the first j blocks) and that exactly one bit in the $(j+1)$-st block is flipped. The probability that in an iteration none of the first kj bits flips and exactly one in the $(j+1)$-st block is exactly $k(1/n)(1-1/n)^{kj+k-1} \geq \frac{k}{en}$. Hence this is a lower bound for the probability of a move. The expected number of moves in t iterations therefore is at least $\frac{tk}{en} = \frac{16 \cdot 2^k k}{e} =: E$. By a multiplicative Chernoff bound together with an argument that the expectation can be replaced by a lower bound for it (e.g., Theorem 1.10.5 and Section 1.10.1.8 in [6]), we obtain that the probability to have less than $E/2$ moves in these t iterations is at most $\exp(-E/8) \leq \exp(-8/e)$. Note that a move means that in the $(j+1)$-st block, we do a RLS step. By Theorem 1 of [8], the expected number of RLS steps it takes to reach at least $k-1$ ones, regardless of

the starting point, is at most

$$\sum_{\ell=0}^{k-2} \binom{k}{\leq \ell} \Big/ \binom{k-1}{\ell} \leq k2^k/(k-1) \leq 2^{k+1},$$

where $\binom{k}{\leq \ell} := \sum_{i=0}^{\ell} \binom{k}{i}$. Hence the probability that $E/2$ moves do not suffice, by Markov's inequality, is at most $2^{k+1}/(E/2) = e/4$. Adding up the failure probabilities, we see that the probability that t iterations do not let us leave the fitness level is at most $O(1/n) + \exp(-8/e) + e/4$, which is at most $3/4$ when assuming n to be large enough (which we may since we only prove an asymptotic bound). This proves that any interval of t iterations lets us leave the current fitness level with probability at least $1/4$. This gives an expected time of at most $4t$ for leaving one level, and hence an expected runtime of $4t(n/k) = O(n^2)$.

For the lower bound, we use again the fitness level method with visiting probabilities. The probability v_{kj+k-1} to visit a fitness level is the same as for the Lamarckian EA, since all arguments given there apply here as well. We estimate the probability p_{kj+k-1} to leave a level by $p_{kj+k-1} \leq k/n$, simply because to leave a level it is necessary to flip at least one bit in the critical block. From these $p_{kj+k-1} = O(1/n)$ and $v_{kj+k-1} = \Theta(1)$, noting that we have $\Omega(n)$ levels, we obtain the desired lower bound of $\Omega(n^2)$. □