RUNTIME ANALYSIS OF CROSSOVER-BASED EVOLUTIONARY NEURAL ARCHITECTURE SEARCH FOR MULTICLASS CLASSIFICATION PROBLEMS

Anonymous authors
Paper under double-blind review

ABSTRACT

Crossover-based evolutionary algorithms (EAs), which generate offspring through recombination and mutation, have been widely applied to neural architecture search for automated design of deep neural architectures. However, it remains unclear whether and when the crossover operator benefits the evolutionary neural architecture search (ENAS) algorithm. To bridge this gap, this work analyzes its role by conducting runtime analysis, which is an essential theoretical aspect of EAs for understanding whether evolutionary operators can accelerate the search for optimal or approximate solutions and under which conditions such acceleration occurs. Specifically, we first develop a runtime analysis framework that captures both progress and potential regress occurring in each generation. Based on this, we analyze the runtime of the ENAS algorithm with crossover on multiclass classification problems, and the results reveal that crossover may not always accelerate ENAS, particularly when it is frequently applied to similar or identical individuals. Motivated by the observation, we further introduce a diversity-preserving mechanism through a duplicate-based tie-breaking rule into the ENAS algorithm with crossover to prevent similar or identical individuals from surviving to the next generation, and analyze its effect on the expected runtime. Our results show that maintaining population diversity may improve the efficiency of crossover. We hope this work may provide theoretical support to deepen the understanding of crossover in ENAS and offer insights into designing effective ENAS algorithms.

1 INTRODUCTION

Evolutionary neural architecture search (ENAS) Real et al. (2017); Liu et al. (2023); Miikkulainen et al. (2024) employs evolutionary techniques, primarily evolutionary algorithms (EAs), for the automated design of deep neural architectures Elsken et al. (2019), and has been widely used in real-world applications Sun et al. (2019b); Yang et al. (2023). Early ENAS algorithms, such as Regularized Evolution Real et al. (2019) and AmoebaNet Real et al. (2017), relied solely on mutation to explore neural architectures. Later studies introduced crossover to recombine partial solutions from different architectures, leading to notable performance improvements, such as Genetic CNN Xie & Yuille (2017), CNN-GA Sun et al. (2019a), and EvoCNN Yang et al. (2023). In principle, in the traditional EA-based optimization algorithms, the crossover operator plays the role of exploitation, which can reasonably complement the exploration ability led by the mutation operator, collectively resulting in better performance. However, in practice, the role of crossover in ENAS remains unclear: in some cases, it indeed improves search efficiency, whereas in others, such as the experiments conducted for AmoebaNet Real et al. (2017), introducing crossover brought no observable performance gain. These diverse empirical results raise fundamental questions about whether and when crossover benefits ENAS.

In the evolutionary computation community, runtime analysis of algorithms has long served as a cornerstone for understanding the working principles of EAs in the past few decades Auger & Doerr (2011); Neumann & Witt (2010); Zhou et al. (2019); Doerr & Neumann (2020). It measures the expected number of fitness evaluations until an optimal or approximate solution is searched for the first time. Through this theoretical lens, researchers have systematically analyzed how key algorithmic components, including the mutation operator and crossover operator, affect the runtime of EAs.

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

Early studies mainly focused on mutation-based EAs, which are EAs without crossover, aiming to understand whether a specific mutation operator improves search efficiency and under which conditions it is most effective. Typical studies compared the runtime of algorithms using one-bit mutation versus bit-wise mutation to reveal which mutation operator performs better Doerr et al. (2008); Doerr & Goldberg (2013); Witt (2013), and also investigated how to set the mutation probability to achieve the fastest convergence Droste et al. (2002); Doerr et al. (2017b;a). Beyond mutation-based EAs, subsequent research has extended the analysis to crossover-based EAs (referred to as "XEAs" in this paper and as GAs in the community Oliveto et al. (2022))[1], which include both mutation and crossover operators, to better understand the role of crossover and the conditions under which crossover benefits evolutionary search.

A key insight from the existing runtime analyses is that crossover can reduce runtime by recombining complementary partial solutions that mutation alone would generate only with exponentially small probability. For instance, Lengler Lengler (2019) proved that (μ+1)-EA with mutation probability c/n requires exponential time on HOTTOPIC when $c \geq 2.13$, where n denotes the problem dimension (i.e., the length of the solution encoding), whereas (μ+1)-XEA with a sufficiently large parent population size μ and offspring population size 1 can solve the same problem in expected runtime $\mathcal{O}(n \log n)$ with arbitrary mutation probability. Similarly, Corus and Oliveto Corus & Oliveto (2018; 2020) proved that (μ+1)-XEA outperforms (μ+1)-EA with standard bit mutation when solving ONEMAX. Moreover, runtime analyses clarify under which conditions crossover provides a speedup. Specifically, Sudholt Sudholt (2017) proved that (μ+λ)-XEA, where λ denotes the offspring population size, is at least twice as fast as (μ+λ)-EA with standard bit mutation on ONEMAX, provided both use a diversity-preserving tie-breaking rule that favors individuals with fewer duplicates and older individuals in the environmental selection phase. Crossover is also effective in algorithms with diversity-enhancing mechanisms for artificial problems like colouring problems inspired by the Ising model Sudholt (2005) and shortest path problems Friedrich et al. (2009). In addition, a number of studies have investigated the role of crossover on other benchmark problems, such as JUMP_k Dang et al. (2017); Opris et al. (2024), LEADINGONES Cerf & Lengler (2024), and Royal Road Opris et al. (2025); several works also studied the impact of crossover in multiobjective optimization Doerr & Qu (2023); Dang et al. (2024); Opris (2025). These works provide further insights into how crossover interacts with mutation and population diversity under different fitness landscapes. However, since many of these analyses are tailored to specific problem structures, their results cannot directly yield general conditions under which crossover is beneficial.

Related runtime analyses have studied EAs for optimizing the parameters of fixed neural networks Fischer et al. (2023; 2024). These studies differ from ENAS, which evolves the architecture itself. Although runtime analyses have been extensively studied for EAs, the runtime analysis of ENAS remains poorly understood and requires more in-depth analysis. To the best of our knowledge, existing studies Lv et al. (2024; 2025) on ENAS have focused primarily on algorithms that employ only mutation operators. For example, Lv et al. Lv et al. (2024) performed the first runtime analysis of (μ+λ)-ENAS algorithms on binary classification problems and showed that increasing the population size may lead to longer runtime. Building upon this line of work, Lv et al. Lv et al. (2025) extended the analysis to the more challenging multiclass classification problems and studied the (1+1)-ENAS algorithm, demonstrating that one-bit and bitwise mutation operators exhibit the same expected runtime in a block-based search space.

In contrast, the runtime analysis of ENAS using crossover remains unexplored. It is challenging both to determine whether crossover can accelerate the search and to identify the conditions under which it is effective. First, neural architectures are typically encoded in hierarchical or cell-based encodings Liu et al. (2018); Zoph et al. (2018); Elsken et al. (2019), and in a real-valued manner, which induces complex dependencies among different encoding components. Such couplings make it difficult to characterize how crossover recombines partial architectural information, and thus hinder rigorous analysis of whether crossover can provably accelerate the search. Second, crossover can be effective only when the population contains sufficiently different parent architectures with complementary useful components. However, evolutionary selection may quickly reduce population diversity, making such parent pairs difficult to obtain and preserve. As mentioned earlier, the runtime analysis community has begun to investigate how population diversity can enhance the effectiveness of crossover. Therefore, to address the question of when crossover benefits ENAS, it

[1] In the following, we use the term "EA" to refer to evolutionary algorithms without crossover, in order to distinguish it from "XEA".

is natural to investigate how promoting population diversity can influence its effectiveness. Consequently, to make some progress toward a better understanding of crossover in ENAS, we regard the following questions: (1) how fast does the (μ+λ)-ENAS algorithm with crossover operator find the optimal neural architecture for a machine learning problem, and (2) how does population diversity influence the efficiency of crossover?

To address these questions, we initiate the mathematical runtime analysis of ENAS with crossover. Our main contributions are summarized as follows:

1. We develop a runtime analysis framework for ENAS with crossover by establishing a layered transition model that captures progress and regress caused by recombination. Based on this, we analyze the expected runtime of the $(\mu + \lambda)$-ENAS algorithm with crossover to determine how fast it finds an optimal architecture for the multiclass classification (MCC) problem (Theorem 1). The results suggest that crossover may offer limited benefit with similar individuals.
2. We introduce a diversity-preserving mechanism through a duplicate-based tie-breaking rule to prevent crossover from degenerating into mere replication when similar or identical individuals dominate the population. We then analyze the expected runtime of the algorithm in Theorem 3. The results reveal that enforcing diversity via the duplicate-based tie-breaking rule can improve the effectiveness of crossover in solving MCC, as supported by empirical results.

2 PRELIMINARIES

This section introduces the MCC problem and the ENAS algorithm studied in this paper.

2.1 MULTICLASS CLASSIFICATION PROBLEM

We consider the multiclass classification benchmark MCC proposed in Lv et al. (2025). The input region $\{x \in \mathbb{R}^2 : \|x\|_2 \leq 1\}$ is divided into $n = 2rM$ equal sectors for M classes, where $r \geq 2$. Each sector consists of a triangle and a circular segment. The first rM sectors are labeled $1, \ldots, M$ in order, repeated r times, with all points in each sector sharing the same label. For the remaining rM sectors, the triangles are labeled in the same order, but each segment has the label following that of the triangle in the same sector: if the triangle belongs to class m, the segment belongs to class $m + 1$ (or class 1 when $m = M$). Thus, each class contains r sectors, r triangles, and r segments. Figure 1(a) illustrates the case of $M = 3$ and $r = 2$.

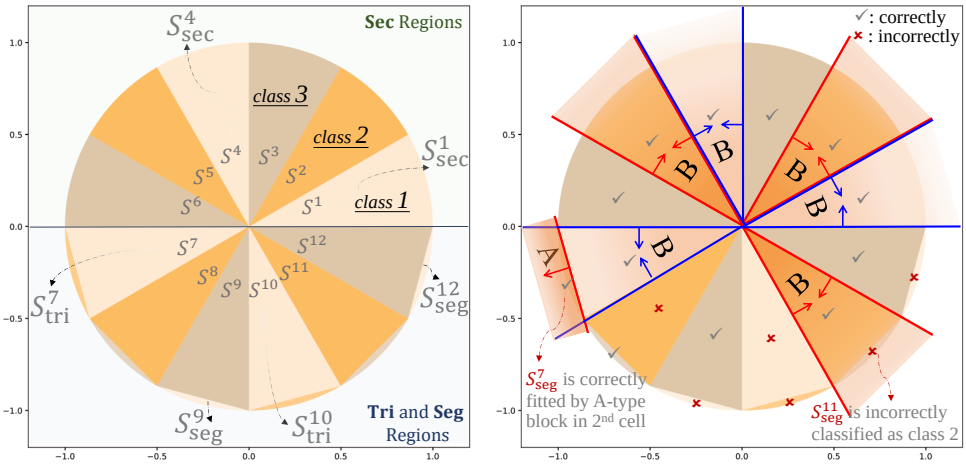
Definition 1 (MCC Problem Lv et al. (2025))**.** *Let the inputs consist of the points on the unit circle $S := \{x \in \mathbb{R}^2 \mid \|x\|_2 \leq 1\}$, with each point labeled as class $y \in [1..M]$, where $M \geq 2$. The unit circle S is evenly divided into $n = 2rM$ sectors with angle $\frac{2\pi}{n}$, where $r \geq 2$. Thus, $S = \cup_{k=1}^{n} S_{\text{sec}}^k$, with S_{sec}^k representing the k-th sector. Each sector S_{sec}^k can be further divided into a triangle S_{tri}^k with area $\frac{1}{2}\sin(\frac{2\pi}{n})$ and a segment (a segment is cut from a sector by a chord) S_{seg}^k with area $\frac{\pi}{n} - \frac{1}{2}\sin(\frac{2\pi}{n})$. Let the points with label m be distributed across r segments, r sectors, and r triangles, with the corresponding regions denoted as Seg^m, Sec^m, and Tri^m, respectively. These regions are defined as*

$$\text{Sec}^m = \bigcup_{k=1}^{r} S_{\text{sec}}^{m+(k-1)M}, \quad \text{Tri}^m = \bigcup_{k=r+1}^{2r} S_{\text{tri}}^{m+(k-1)M},$$

$$\text{Seg}^{m\neq 1} = \bigcup_{k=r+1}^{2r} S_{\text{seg}}^{(m-1)+(k-1)M}, \text{Seg}^1 = \bigcup_{k=r+1}^{2r} S_{\text{seg}}^{kM}.$$

Then, the set of points with label $m \in [1..M]$ is $\mathcal{L}^m = \text{Seg}^m \cup \text{Sec}^m \cup \text{Tri}^m$.

To solve MCC, we use the neural architecture introduced in Lv et al. (2025). The architecture contains $M - 1$ cells, where cell m is used to identify points of class m. Each cell combines three types of blocks to cover regions of the input space: an A-type block covers a circular segment, a B-type block covers a sector, and a C-type block covers a triangle. Together, the blocks in a cell form the region used to identify its class. Points not covered by any cell are assigned to class M. The blocks are implemented using predefined threshold neurons, as detailed in Appendix A.1.



(a) MCC with $M = 3$ and $r = 2$ (b) Decision Regions

Figure 1: Illustration of MCC and the regions constructed by an example neural architecture Lv et al. (2025). (a) The problem with $M = 3$ and $r = 2$, where colors indicate class labels. (b) Decision regions of $\boldsymbol{x} = \{(0, 3, 0), (1, 3, 0)\}$. Blue and red arrows indicate the regions constructed by cells 1 and 2, respectively; uncovered points are assigned to class 3. Check marks and crosses indicate correct and incorrect classification.

An neural architecture is encoded as $\boldsymbol{x} = \{(n_A^1, n_B^1, n_C^1), \ldots, (n_A^{M-1}, n_B^{M-1}, n_C^{M-1})\}$, where $n_A^m, n_B^m, n_C^m \in \mathbb{N}_0$ denote the numbers of A-, B-, and C-type blocks in m-th cell. For example, $\boldsymbol{x} = \{(0, 3, 0), (1, 3, 0)\}$ in Figure 1(b) has two cells: the first uses three B-type blocks for class 1, and the second uses one A-type block and three B-type blocks for class 2. Points outside both cell regions are assigned to class 3.

Optimizing this architecture means adjusting the block counts to maximize the area of correctly classified regions. However, a block may also cover points of another class. For example, a B-type block covers an entire sector, including both its triangle and segment. If these parts have different labels, covering the triangle also covers a segment of another class. The fitness function below accounts for such classification errors. Since each class contains only $2r$ triangle parts and $2r$ segment parts, block counts greater than $2r$ provide no additional fitness improvement.

We evaluate an architecture $\boldsymbol{x}$ by its validation accuracy, defined as the fraction of the input area that it classifies correctly. This area can be calculated by counting correctly classified triangle and segment parts. Lemma 1 gives the corresponding fitness function in terms of the block counts.

Lemma 1 (Lv et al. (2025)). *For an architecture $\boldsymbol{x}$, its validation accuracy on* MCC *is*

$$\mathcal{F}(\boldsymbol{x}) = \frac{\mathrm{Ar}_{\mathrm{tri}}(\mathbb{I}_{\boldsymbol{x}} + 2r) + \mathrm{Ar}_{\mathrm{seg}}(\mathbb{J}_{\boldsymbol{x}} + 2r - \epsilon_{\boldsymbol{x}})}{\pi}, \tag{1}$$

where $\mathrm{Ar}_{\mathrm{tri}}$ and $\mathrm{Ar}_{\mathrm{seg}}$ are the areas of a triangle and a circular segment, respectively, and π is the total input area. The three quantities are

$$\mathbb{I}_{\boldsymbol{x}} = \sum_{m=1}^{M-1} \min\{n_B^m + n_C^m, 2r\}, \tag{2}$$

$$\mathbb{J}_{\boldsymbol{x}} = \sum_{m=1}^{M-1} \left(\min\{n_B^m, r\} + \min\{n_A^m, r + \max\{r - n_B^m, 0\}\}\right), \tag{3}$$

$$\epsilon_{\boldsymbol{x}} = \max\{0, \min\{n_B^{M-1} - r, r - n_C^{M-1}\}\}. \tag{4}$$

Both $\mathbb{I}_{\boldsymbol{x}}$ and $\mathbb{J}_{\boldsymbol{x}} - \epsilon_{\boldsymbol{x}}$ lie in $[0..2r(M-1)]$.

The quantity $\mathbb{I}_{\boldsymbol{x}}$ counts the triangle parts correctly covered by cells $1, \ldots, M-1$. Each class has $2r$ triangle parts: r individual triangles and r within its sectors. Both B- and C-type blocks cover triangle parts, so cell m contributes $\min\{n_B^m + n_C^m, 2r\}$.

The quantity $\mathbb{J}_{\boldsymbol{x}}$ counts the segment parts correctly covered by these cells. In cell m, B-type blocks cover the segment parts of up to r sectors, giving the term $\min\{n_B^m, r\}$. A-type blocks cover the r individual segments and any segment parts of sectors not covered by B-type blocks. There are $\max\{r - n_B^m, 0\}$ such sectors, so at most $r + \max\{r - n_B^m, 0\}$ A-type blocks contribute.

The two $2r$ terms in Eq. (1) account for the triangle and segment parts of class M, which are classified by default when left uncovered. The penalty $\epsilon_{\boldsymbol{x}}$ subtracts the segments of class M incorrectly covered by cell $M-1$. This occurs when $n_B^{M-1} > r$ and $n_C^{M-1} < r$: B-type blocks used to cover triangles of class $M-1$ also cover neighboring segments of class M.

Let $N = 2r(M-1) = n - 2r$. An architecture is globally optimal when $\mathbb{I}_{\boldsymbol{x}} = \mathbb{J}_{\boldsymbol{x}} - \epsilon_{\boldsymbol{x}} = N$, giving $\mathcal{F}(\boldsymbol{x}) = 1$. For example, setting $n_A^m = n_B^m = n_C^m = r$ in every cell yields an optimal architecture. The condition $\mathbb{I}_{\boldsymbol{x}} = N$ holds exactly when $n_B^m + n_C^m \geq 2r$ for every cell.

These quantities allow us to divide the runtime analysis into two phases: reaching an architecture with $\mathbb{I}_{\boldsymbol{x}} = N$, and then reaching one with $\mathbb{J}_{\boldsymbol{x}} - \epsilon_{\boldsymbol{x}} = N$ while preserving $\mathbb{I}_{\boldsymbol{x}} = N$. Adding a B- or C-type block can improve a deficient triangle contribution, while adding an A- or C-type block can improve a deficient segment contribution or reduce the penalty. Thus, every non-optimal architecture admits a fitness improvement by changing one block count, although changes to redundant blocks may leave fitness unchanged.

2.2 (μ+λ)-ENAS

We consider the $(\mu+\lambda)$-ENAS algorithm in Algorithm 1, which searches the architecture space $\mathcal{S} = \{\mathbb{Z}\times\mathbb{Z}\times\mathbb{Z}\}^{M-1}$. In each generation, it produces λ offspring and retains the best μ individuals from the parents and offspring. Each offspring is generated from two parents selected independently and uniformly at random. With probability $p_c = 1/2$, uniform crossover independently copies each cell (n_A^m, n_B^m, n_C^m) from either parent with equal probability; otherwise, one parent is copied uniformly at random. Each cell of the resulting offspring is then mutated independently with probability $p_m = c/(M-1) < 1$, where $c > 0$ is a constant. A selected cell undergoes one mutation operation defined below. Crossover and mutation are commonly used in ENAS Xie & Yuille (2017); Liu et al. (2023).

Definition 2 (Mutation Operation)**.** *Given any cell (n_A^m, n_B^m, n_C^m) of a solution, the algorithm randomly selects a block type $V \in \{A, B, C\}$ and applies one of the following operations uniformly at random: (1)* ***Addition:*** *adds a V-type block (i.e., $n_V^m \leftarrow n_V^m + 1$); (2)* ***Deletion:*** *deletes a V-type block (i.e., $n_V^m \leftarrow n_V^m - 1$) when $n_V^m > 0$; (3)* ***Modification:*** *randomly selects a block type $W \in \{A, B, C\}\backslash V$ and then performs Deletion on V while performs Addition on W when $n_V^m > 0$ (i.e., $n_V^m \to n_V^m - 1$ and $n_W^m \to n_W^m + 1$); otherwise, n_V^m and n_W^m remain unchanged.*

3 Runtime Analysis of (μ+λ)-ENAS

This section analyzes the expected runtime of $(\mu + \lambda)$-ENAS with crossover on Mcc. We first outline the analysis and present the bound in Theorem 1. We then analyze the mutation-only variant in Theorem 2 to provide a baseline for comparison.

3.1 Analysis Outline

Inspired by the fitness-level analysis in Sudholt (2017), we use a layered transition model to capture progress and regress caused by recombination. Our analysis consists of four steps:

Step 1: Phase decomposition. Using the fitness function in Eq. (1), we divide the optimization into two phases:

- Phase 1 ends when an individual satisfies $\mathbb{I}_{\boldsymbol{x}} = N$.
- Phase 2 ends when an individual satisfies both $\mathbb{I}_{\boldsymbol{x}} = N$ and $\mathbb{J}_{\boldsymbol{x}} - \epsilon_{\boldsymbol{x}} = N$.

Step 2: Population space partition. We partition the population space $\mathcal{X}$ into levels $\mathcal{X}_0, \ldots, \mathcal{X}_N$, where $\mathcal{X}_i$ contains populations whose largest $\mathbb{I}$-value is i. For Phase 2, we further partition $\mathcal{X}_N$ into $\mathcal{X}_N^0, \ldots, \mathcal{X}_N^N$. A population belongs to $\mathcal{X}_N^j$ if the largest $(\mathbb{J} - \epsilon)$-value among its individuals satisfying $\mathbb{I} = N$ is j.

Algorithm 1 (μ+λ)-ENAS with Crossover Operator

Initialization:
Let $t := 0$. Create μ initial solutions by choosing $\boldsymbol{x}^{(q)} \in \{[0..n/4]\times[0..n/4]\times[0..n/4]\}^{M-1}$ $(1 \leq q \leq \mu)$ uniformly at random. Let the multiset $\xi_0 := \{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(\mu)}\}$ be the initial population.
Optimization:
while *termination criterion not met* **do**
  $X' \leftarrow \xi_t$.
  Population updating phase:
  for $q = 1$ to λ **do**
    Select parents $\boldsymbol{x}_1$ and $\boldsymbol{x}_2$ from ξ_t uniformly at random.
    Sample u from the uniform distribution over $[0, 1]$;
    if $u < p_c$ **then**
      Generate an offspring $\boldsymbol{y}$ by applying the uniform crossover to $\boldsymbol{x}_1$ and $\boldsymbol{x}_2$
    else
      Generate an offspring $\boldsymbol{y}$ by copying either $\boldsymbol{x}_1$ or $\boldsymbol{x}_2$ uniformly at random
    end if
    for $m = 1$ to $M - 1$ **do**
      Sample w from uniform distribution over $[0, 1]$;
      if $w < p_m$ **then**
        Update $\boldsymbol{y}$ by applying the mutation operation in Definition 2, i.e., applying *Addition*, *Deletion*, or *Modification*, each with probability $1/3$
      end if
    end for
    $X' \leftarrow X' \cup \{\boldsymbol{y}\}$
  end for
  Environmental selection phase:
  Create the multiset ξ_{t+1} by selecting top μ individuals from X', and break ties according to a tie-breaking rule.
  Let $t := t + 1$
end while

Step 3: Distance function. Individuals at the same level may contribute differently across cells. To describe these differences in Phase 1, we count the cells with unequal I-values, as defined below.

Definition 3 ($\mathbb{I}$-distance)**.** *Given two solutions $\boldsymbol{x}_1$ and $\boldsymbol{x}_2$, the $\mathbb{I}$-distance between them is defined as $d_{\mathbb{I}}(\boldsymbol{x}_1, \boldsymbol{x}_2) = \sum_{m=1}^{M-1} \mathbf{1}_{\{I^m_{\boldsymbol{x}_1} \neq I^m_{\boldsymbol{x}_2}\}}$, where $I^m_{\boldsymbol{x}} = \min\{n^m_B + n^m_C, 2r\}$ is the contribution of cell m to $\mathbb{I}_{\boldsymbol{x}}$. The indicator equals one when the two cell contributions differ, and zero otherwise.*

In Phase 2, the distance is defined using each cell's contribution to $\mathbb{J}_{\boldsymbol{x}} - \epsilon_{\boldsymbol{x}}$. Specifically, these contributions are $J^m_{\boldsymbol{x}}$ for $m < M - 1$ and $J^{M-1}_{\boldsymbol{x}} - \epsilon_{\boldsymbol{x}}$ for the last cell, where $J^m_{\boldsymbol{x}}$ is the m-th summand in Eq. (3).

Step 4: Partition of each level. For Phase 1, we divide populations at level i into three cases:

- Case $i.1$: at least one individual has $\mathbb{I} < i$.
- Case $i.2$: all individuals have $\mathbb{I} = i$ and identical I-values in every cell.
- Case $i.3$: all individuals have $\mathbb{I} = i$, but at least two individuals have positive $\mathbb{I}$-distance.

The same distinction is used in Phase 2, based on the cell contributions to $\mathbb{J} - \epsilon$ among individuals with $\mathbb{I} = N$. Individuals with $\mathbb{I} < N$ are treated as belonging to a lower level.

The transitions are illustrated in Figure 2, where i^+ denotes a higher level. From Case $i.1$, the population can reach Case $i.2$, Case $i.3$, or i^+. From Case $i.2$, it can reach Case $i.3$ or i^+. Case $i.3$ can lead to an improvement, but uniform tie-breaking may also remove all individuals with different cell contributions and return the population to Case $i.2$.

3.2 Analysis Result

Theorem 1 gives the expected runtime of (μ+λ)-ENAS with crossover on MCC. Fitness ties during environmental selection are broken uniformly at random, as is standard when no tie-breaking rule is specified. We denote this algorithm by (μ+λ)-ENAS$^{\text{crossover}}_{\text{uni}}$. The proof combines the fitness-level

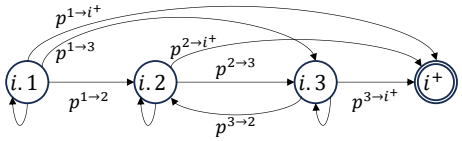


Figure 2: Transitions among population states at fitness level i.

method Wegener (2003); Sudholt (2013) with a case-based transition analysis. Following the two-phase decomposition, each phase is partitioned into $N+1$ fitness levels. Within each level, Cases $i.1$–$i.3$ distinguish the population according to the fitness and cell-wise diversity of its individuals. For Case $i.1$, we apply the population takeover bound from Lemma 2 of Sudholt (2017) to estimate the time until all individuals reach the current best level. For Cases $i.2$ and $i.3$, we derive lower bounds on the one-generation transition probabilities and use geometric waiting-time arguments. The analysis accounts for both improvements generated by mutation or crossover and the possible fallback from Case $i.3$ to Case $i.2$ under uniform tie-breaking. Summing these waiting times over all levels produces harmonic sums and yields the bound in Theorem 1. The second phase is analyzed analogously by replacing $\mathbb{I}$ with $\mathbb{J}-\epsilon$. The complete proof is provided in Appendix A.2.1.

Theorem 1. *The expected runtime until the (μ+λ)-ENAS$^{\text{crossover}}_{\text{uni}}$ with bit-wise mutation probability $p_m = c/(M-1)$ and crossover probability $p_c = 1/2$ finds the optimum for* MCC *is $\mathbb{E}[T] = \mathcal{O}((e^c/c^2)r^2M^3(\mu+\lambda)\cdot\max\{\ln\mu\ln(rM), rM/\mu\})$, when $\mu=\omega(r)$ and $\lambda=\omega(rM)$. Furthermore, when $\mu\ln\mu = o(rM/\ln(rM))$, the bound can be simplified to $\mathcal{O}((e^c/c^2)r^3M^4(1+\lambda/\mu))$.*

The above analysis yields three observations. First, the standard mutation setting $c=1$ is not necessarily optimal for the crossover-based algorithm. The factor e^c/c^2 in Theorem 1 is minimized at $c=2$, suggesting $p_m = 2/(M-1)$. Since this is based only on an upper bound, we examine it empirically in Section 5. Second, the bound does not establish an improvement over mutation-only $(1+1)$-ENAS, whose runtime is $O(rM\ln(rM))$ and $\Omega(rM\ln M)$ shown in Lv et al. (2025). Third, the transition from Case $i.3$ to Case $i.2$ in Figure 2 suggests that diversity loss may increase runtime, motivating diversity-preserving mechanisms to improve crossover.

3.3 MUTATION-ONLY (μ+λ)-ENAS

We consider the mutation-only variant of Algorithm 1, denoted by (μ+λ)-ENAS$^{\text{mutation}}_{\text{uni}}$. Since mutation does not recombine cells from different individuals, it suffices to track the best individual using the fitness-level method. Theorem 2 gives the upper bound on its expected runtime. Unlike the crossover-based algorithm analyzed above, mutation does not recombine cells from different individuals. It is therefore sufficient to track the best individual in the population using a standard fitness-level argument. The complete proof is provided in Appendix A.2.2.

Theorem 2. *Let T be the runtime of (μ+λ)-ENAS$^{\text{mutation}}_{\text{uni}}$ with bit-wise mutation probability $p_m = c/(M-1)$ for finding an optimum of* MCC*, where $c>0$ is a constant. Then, $\mathbb{E}[T] = \mathcal{O}(\mu + rM\max\{\lambda, \frac{\mu e^c}{c}\ln(rM)\})$.*

For $\mu=\lambda=1$, Theorem 2 yields an upper bound of $\mathcal{O}((e^c/c)rM\ln(rM))$, consistent with the bounds previously derived for $(1+1)$-ENAS on MCC Lv et al. (2025). Since e^c/c is minimized at $c=1$, the upper bound suggests the standard mutation probability $p_m = 1/(M-1)$ for the mutation-only algorithm. Moreover, comparing Theorems 1 and 2, we observe that the derived upper bound for the crossover-based algorithm is not smaller than that of the corresponding mutation-only algorithm. Thus, the current upper-bound analysis does not establish that crossover alone accelerates ENAS. One possible reason is that the population may repeatedly lose diversity, leading to crossover being applied to individuals with identical or similar cell-wise structures.

4 POPULATION DIVERSITY IN CROSSOVER-BASED ENAS

This section introduces a duplicate-based tie-breaking rule applied in environmental selection to preserve population diversity and analyzes its effect on the expected runtime of ENAS with crossover.

4.1 DUPLICATE-BASED TIE-BREAKING RULE

We first explain how duplicates are identified in the architecture space considered here. In a binary search space, duplicate solutions can be identified by checking whether their bit strings are identical. Here, the architecture encoding consists of cells and the block counts within each cell, while crossover operates on whole cells. Comparing block counts alone does not capture whether the cells make different contributions: two cells with different block counts may have the same I- and J-values. We therefore compare architectures through their cell contributions and define the outer encoding of $\boldsymbol{x}$ as $\hat{\boldsymbol{x}} = \{(I^1_{\boldsymbol{x}}, J^1_{\boldsymbol{x}}), \ldots, (I^{M-1}_{\boldsymbol{x}}, J^{M-1}_{\boldsymbol{x}})\}$, where $I^m_{\boldsymbol{x}}$ and $J^m_{\boldsymbol{x}}$ are the contributions of cell m to $\mathbb{I}_{\boldsymbol{x}}$ and $\mathbb{J}_{\boldsymbol{x}}$ in Eqs.2 and 3, respectively. Two solutions $\boldsymbol{x}$ and $\boldsymbol{z}$ are duplicates if $\hat{\boldsymbol{x}} = \hat{\boldsymbol{z}}$. For other hierarchical encodings, a suitable cell-level mapping could similarly identify duplicates. We study this approach on MCC, leaving validation in other architecture spaces for future work.

Algorithm 2 selects the best μ individuals by fitness. When a fitness tie must be broken, individuals are selected one at a time. An individual whose outer encoding has not yet been selected is preferred. If all remaining encodings are already represented, the rule favors those with fewer copies in the selected population. Any remaining ties are broken uniformly at random.

Algorithm 2 Duplicate-based Tie-breaking Rule

Input: Candidate multiset X' and population size μ
Output: Next-generation population ξ_{t+1}
Sort X' by decreasing fitness to obtain list L
$f_{\mathrm{thr}} \leftarrow \mathcal{F}(L[\mu])$
$H \leftarrow \{\boldsymbol{x} \in X' \mid \mathcal{F}(\boldsymbol{x}) > f_{\mathrm{thr}}\}$
$E \leftarrow \{\boldsymbol{x} \in X' \mid \mathcal{F}(\boldsymbol{x}) = f_{\mathrm{thr}}\}$
$\xi_{t+1} \leftarrow H$
while $|\xi_{t+1}| < \mu$ **do**
  for each individual $\boldsymbol{x} \in E$ **do**
    $d(\boldsymbol{x}) \leftarrow |\{\boldsymbol{z} \in \xi_{t+1} \mid \hat{\boldsymbol{z}} = \hat{\boldsymbol{x}}\}|$
  end for
  Select an individual $\boldsymbol{x} \in E$ minimizing $d(\boldsymbol{x})$, breaking ties uniformly at random
  Add one copy of $\boldsymbol{x}$ to ξ_{t+1}
  Remove the selected copy of $\boldsymbol{x}$ from E
end while
return ξ_{t+1}

4.2 ANALYSIS RESULT

Next, we analyze (μ+λ)-ENAS with crossover and duplicate-based tie-breaking, denoted by (μ+λ)-ENAS$^{\text{crossover}}_{\text{dup}}$. Theorem 3 gives an upper bound on its expected runtime for finding an optimum of MCC. The proof follows the framework of Theorem 1, with the key step being to exclude transitions from Case $i.3$ back to Case $i.2$. Once the population moves from Case $i.2$ to Case $i.3$, the rule preserves individuals with distinct outer encodings and prevents the population from returning to a state in which all individuals have identical cell-wise contributions. This eliminates repeated transitions between Cases $i.2$ and $i.3$ and reduces the expected time required to leave the current fitness level. Due to space limitations, the complete proof of Theorem 3 is provided in Appendix A.2.3.

Theorem 3. *The expected runtime until the (μ+λ)-ENAS$^{\text{crossover}}_{\text{dup}}$ with bit-wise mutation probability $p_m = c/(M-1)$ and crossover probability $p_c = 1/2$ finds the optimum for* MCC *is* $\mathcal{O}((e^c/c^2)r^2M^3(\mu+\lambda)\ln\mu\ln(rM))$.

Comparing Theorems 1 and 3, duplicate-based tie-breaking yields a smaller expected runtime upper bound when $rM/\mu > \ln\mu\ln(rM)$. Under this condition, the bound is reduced from $\mathcal{O}((e^c/c^2)r^2M^3(\mu+\lambda)\cdot rM/\mu)$ to $\mathcal{O}((e^c/c^2)r^2M^3(\mu+\lambda)\ln\mu\ln(rM))$. This suggests that preserving population diversity can improve crossover effectiveness, particularly when μ is not too large relative to the problem size rM.

5 EXPERIMENTS

This section experimentally compares the runtime of $(\mu + \lambda)$-ENAS with crossover under the uniform and duplicate-based tie-breaking rules on MCC. Specifically, the number of classes M ranges from 2 to 24, and the parameter r ranges from 2 to 10, both with a step size of 2. We consider three population settings: $(\mu + \lambda) \in \{(2 + 1), (2 + 2), (4 + 4)\}$. The crossover probability is fixed at $1/2$, while the mutation parameter c is set to either 1 or 2. Figure 3 reports the average number of fitness evaluations required to find an optimal solution over 1,000 independent runs. In the figure, "dup" and "uni" denote the duplicate-based and uniform tie-breaking rules, respectively; "C" and "NC" indicate the presence and absence of crossover; and "c2" denotes the setting $c = 2$.

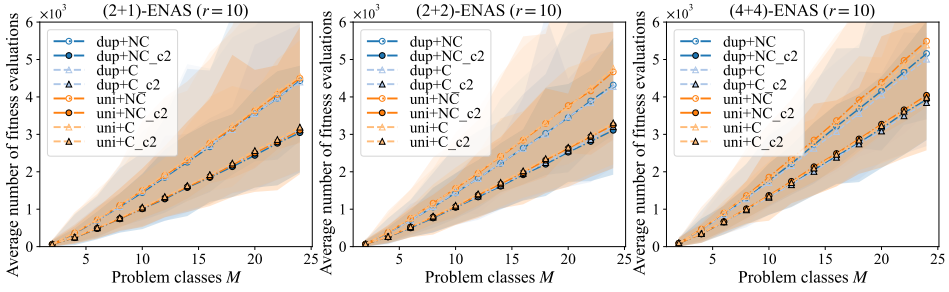


Figure 3: Average number of fitness evaluations of the $(\mu+\lambda)$-ENAS algorithm with crossover for solving the MCC problem. "dup" and "uni" denote the duplicate-based and uniform tie-breaking rules, respectively, and are represented by blue and yellow lines. "C" and "NC" indicate the presence or absence of crossover, corresponding to triangular and circular markers, respectively. "c2" denotes the mutation setting $c = 2$ (vs. the standard $c = 1$) and is marked by black outlines.

The experimental results show that crossover provides little noticeable improvement under the smaller population settings $(2+1)$ and $(2+2)$. A clearer improvement is observed only for $(4+4)$, as shown by the comparison between "uni+C" and "uni+NC". In contrast, introducing the duplicate-based tie-breaking rule leads to a more evident reduction in runtime, as illustrated by the comparison between "dup+C" and "uni+C". This suggests that preserving population diversity enables crossover to combine different cell-wise structures more effectively. Moreover, the crossover-based algorithm with the larger mutation probability $2/(M-1)$, denoted by "dup+C+c2", generally converges faster than its counterpart using the standard mutation probability $1/(M-1)$, denoted by "dup+C". This observation supports the theoretical indication that moderately increasing the mutation strength may improve the search efficiency of crossover-based ENAS. Overall, the experimental findings are consistent with the theoretical results in Theorems 1–3.

6 CONCLUSION

In this paper, we initiate the expected runtime analysis of ENAS with crossover on the MCC problem. We first derive an upper bound on the expected runtime of the $(\mu + \lambda)$-ENAS algorithm with crossover in Theorem 1 and compare it with the corresponding mutation-only algorithm in Theorem 2. The results do not establish a runtime advantage of crossover under uniform tie-breaking, suggesting that crossover may provide limited benefit when applied to similar individuals. We then introduce a duplicate-based tie-breaking rule to preserve population diversity and analyze its effect on the expected runtime in Theorem 3. The comparison shows that this rule can reduce the derived upper bound for the crossover-based algorithm, suggesting that maintaining population diversity may improve the effectiveness of crossover. Since the tightness of these upper bounds remains unclear, we further provide empirical results to complement the theoretical findings. An important direction for future work is to derive matching lower bounds, which would enable a more precise comparison among different ENAS variants. We also hope that this work will encourage further runtime analyses of ENAS, including evolutionary multiobjective optimization Qian et al. (2012) and parent selection Osuna et al. (2017); Opris & Antipov (2026), thereby deepening the theoretical understanding of ENAS and providing insights into the design of more effective algorithms.

AI USE STATEMENT

We used generative AI tools for grammar checking. We did not use AI tools to generate the research results. We take responsibility for the final content of this work, including text, claims or artifacts produced with the aid of generative AI.

REFERENCES

Anne Auger and Benjamin Doerr. *Theory of Randomized Search Heuristics - Foundations and Recent Developments*. World Scientific, Singapore, 2011.

Sacha Cerf and Johannes Lengler. How population diversity influences the efficiency of crossover. In *Proceedings of the 18th International Conference on Parallel Problem Solving from Nature (PPSN'24)*, pp. 102–116, Hagenberg, Austria, 2024. Springer.

Dogan Corus and Pietro S Oliveto. Standard steady state genetic algorithms can hillclimb faster than mutation-only evolutionary algorithms. *IEEE Transactions on Evolutionary Computation*, 22(5): 720–732, 2018.

Dogan Corus and Pietro S Oliveto. On the benefits of populations for the exploitation speed of standard steady-state genetic algorithms. *Algorithmica*, 82(12):3676–3706, 2020.

Duc-Cuong Dang, Tobias Friedrich, Timo Kötzing, Martin S Krejca, Per Kristian Lehre, Pietro S Oliveto, Dirk Sudholt, and Andrew M Sutton. Escaping local optima using crossover with emergent diversity. *IEEE Transactions on Evolutionary Computation*, 22(3):484–497, 2017.

Duc-Cuong Dang, Andre Opris, and Dirk Sudholt. Crossover can guarantee exponential speed-ups in evolutionary multi-objective optimisation. *Artificial Intelligence*, 330:104098, 2024.

Benjamin Doerr and Leslie Ann Goldberg. Adaptive drift analysis. *Algorithmica*, 65:224–250, 2013.

Benjamin Doerr and Frank Neumann. *Theory of Evolutionary Computation: Recent Developments in Discrete Optimization*. Springer, Cham, Switzerland, 2020.

Benjamin Doerr and Zhongdi Qu. Runtime analysis for the NSGA-II: Provable speed-ups from crossover. In *Proceedings of the 35th AAAI Conference on Artificial Intelligence (AAAI'23)*, pp. 12399–12407, Washington, DC, 2023.

Benjamin Doerr, Thomas Jansen, and Christian Klein. Comparing global and local mutations on bit strings. In *Proceedings of the 10th Annual Conference on Genetic and Evolutionary Computation (GECCO'08)*, pp. 929–936, Atlanta, GA, 2008.

Benjamin Doerr, Christian Gießen, Carsten Witt, and Jing Yang. The (1+λ) evolutionary algorithm with self-adjusting mutation rate. In *Proceedings of the 13th Genetic and Evolutionary Computation Conference (GECCO'17)*, pp. 1351–1358, Berlin, Germany, 2017a.

Benjamin Doerr, Huu Phuoc Le, Régis Makhmara, and Ta Duy Nguyen. Fast genetic algorithms. In *Proceedings of the Genetic and Evolutionary Computation Conference*, pp. 777–784, 2017b.

Stefan Droste, Thomas Jansen, and Ingo Wegener. On the analysis of the (1+1) evolutionary algorithm. *Theoretical Computer Science*, 276(1-2):51–81, 2002.

Thomas Elsken, Jan Hendrik Metzen, and Frank Hutter. Neural architecture search: A survey. *Journal of Machine Learning Research*, 20(1):1997—2017, 2019.

Paul Fischer, Emil Lundt Larsen, and Carsten Witt. First steps towards a runtime analysis of neuroevolution. In *Proceedings of the 17th International Workshop on Foundations of Genetic Algorithms (FOGA'23)*, pp. 61–72, Potsdam, Germany, 2023.

Paul Fischer, John Alasdair Warwicker, and Carsten Witt. A runtime analysis of bias-invariant neuroevolution and dynamic fitness evaluation. In *Proceedings of the 26th Annual Conference on Genetic and Evolutionary Computation (GECCO'24)*, pp. 1560–1568, Melbourne, VIC, Australia, 2024.

Tobias Friedrich, Pietro S Oliveto, Dirk Sudholt, and Carsten Witt. Analysis of diversity-preserving mechanisms for global exploration. *Evolutionary Computation*, 17(4):455–476, 2009.

Johannes Lengler. A general dichotomy of evolutionary algorithms on monotone functions. *IEEE Transactions on Evolutionary Computation*, 24(6):995–1009, 2019.

Hanxiao Liu, Karen Simonyan, Oriol Vinyals, Chrisantha Fernando, and Koray Kavukcuoglu. Hierarchical representations for efficient architecture search. In *Proceedings of the 6th International Conference of Learning Representation (ICLR'18)*, Vancouver, BC, Canada, 2018.

Yuqiao Liu, Yanan Sun, Bing Xue, Mengjie Zhang, Gary G. Yen, and Kay Chen Tan. A survey on evolutionary neural architecture search. *IEEE Transactions on Neural Networks and Learning Systems*, 34(2):550–570, 2023.

Zeqiong Lv, Chao Bian, Chao Qian, and Yanan Sun. Runtime analysis of population-based evolutionary neural architecture search for a binary classification problem. In *Proceedings of the 26th Annual Conference on Genetic and Evolutionary Computation (GECCO'24)*, pp. 358–366, Melbourne, VIC, Australia, 2024.

Zeqiong Lv, Chao Qian, Yun Liu, Jiahao Fan, and Yanan Sun. Runtime analysis of evolutionary nas for multiclass classification. In *Proceedings of the 42nd International Conference on Machine Learning (ICML'25)*, Vancouver, Canada, 2025. PMLR.

Risto Miikkulainen, Jason Liang, Elliot Meyerson, Aditya Rawal, Dan Fink, Olivier Francon, Bala Raju, Hormoz Shahrzad, Arshak Navruzyan, Nigel Duffy, et al. Evolving deep neural networks. In *Artificial Intelligence in the Age of Neural Networks and Brain Computing (Second Edition)*, pp. 269–287. Academic Press, 2024.

Frank Neumann and Carsten Witt. *Bioinspired Computation in Combinatorial Optimization - Algorithms and Their Computational Complexity*. Springer, Berlin, Germany, 2010.

Pietro S Oliveto, Dirk Sudholt, and Carsten Witt. Tight bounds on the expected runtime of a standard steady state genetic algorithm. *Algorithmica*, 84:1603–1658, 2022.

Andre Opris. A many-objective problem where crossover is provably indispensable. In *Proceedings of the 37th AAAI Conference on Artificial Intelligence (AAAI'25)*, pp. 27108–27116, Philadelphia, PA, USA, 2025.

Andre Opris and Denis Antipov. Parent selection mechanisms in elitist crossover-based algorithms. In *Proceedings of the 27th Annual Conference on Genetic and Evolutionary Computation (GECCO'26)*, San Jose, Costa Rica, 2026.

Andre Opris, Johannes Lengler, and Dirk Sudholt. A tight $o(4^k/p_c)$ runtime bound for a (μ+ 1) GA on Jump$_k$ for realistic crossover probabilities. In *Proceedings of the 26th Genetic and Evolutionary Computation Conference (GECCO'24)*, pp. 1605–1613, Melbourne, VIC, Australia, 2024.

Andre Opris, Sebastian Sonntag, and Dirk Sudholt. A royal road function for permutation spaces: an example where order crossover is provably essential. In *Proceedings of the 27th Genetic and Evolutionary Computation Conference (GECCO'25)*, pp. 1631–1640, Malaga, Spain, 2025.

Edgar Covantes Osuna, Wanru Gao, Frank Neumann, and Dirk Sudholt. Speeding up evolutionary multi-objective optimisation through diversity-based parent selection. In *Proceedings of the 19th Genetic and Evolutionary Computation Conference (GECCO'17)*, pp. 553–560, Berlin, Germany, 2017.

Chao Qian, Yang Yu, and Zhi-Hua Zhou. On algorithm-dependent boundary case identification for problem classes. In *12th International Conference on Parallel Problem Solving from Nature (PPSN'12)*, pp. 62–71, Taormina, Italy, 2012. Springer.

Esteban Real, Sherry Moore, Andrew Selle, Saurabh Saxena, Yutaka Leon Suematsu, Jie Tan, Quoc V Le, and Alexey Kurakin. Large-scale evolution of image classifiers. In *Proceedings of the 34th International Conference on Machine Learning (ICML'17)*, pp. 2902–2911, Sydney, Australia, 2017. PMLR.

Esteban Real, Alok Aggarwal, Yanping Huang, and Quoc V Le. Regularized evolution for image classifier architecture search. In *Proceedings of the 33rd AAAI Conference on Artificial Intelligence (AAAI'19)*, pp. 4780–4789, Honolulu, HI, 2019.

Dirk Sudholt. Crossover is provably essential for the Ising model on trees. In *Proceedings of the 7th Annual Conference on Genetic and Evolutionary Computation (GECCO'05)*, pp. 1161–1167, 2005.

Dirk Sudholt. A new method for lower bounds on the running time of evolutionary algorithms. *IEEE Transactions on Evolutionary Computation*, 17(3):418–435, 2013.

Dirk Sudholt. How crossover speeds up building block assembly in genetic algorithms. *Evolutionary Computation*, 25(2):237–274, 2017.

Yanan Sun, Bing Xue, Mengjie Zhang, and Gary G Yen. Completely automated CNN architecture design based on blocks. *IEEE Transactions on Neural Networks and Learning Systems*, 31(4): 1242–1254, 2019a.

Yanan Sun, Bing Xue, Mengjie Zhang, and Gary G Yen. Evolving deep convolutional neural networks for image classification. *IEEE Transactions on Evolutionary Computation*, 24(2):394–407, 2019b.

Ingo Wegener. Methods for the analysis of evolutionary algorithms on pseudo-Boolean functions. In *Evolutionary Optimization*, pp. 349–369. Springer, 2003.

Carsten Witt. Tight bounds on the optimization time of a randomized search heuristic on linear functions. *Combinatorics, Probability and Computing*, 22:294–318, 2013.

Lingxi Xie and Alan Yuille. Genetic CNN. In *Proceedings of the IEEE International Conference on Computer Vision (ICCV'17)*, pp. 1379–1388, Venice, Italy, 2017.

Shangshang Yang, Xiaoshan Yu, Ye Tian, Xueming Yan, Haiping Ma, and Xingyi Zhang. Evolutionary neural architecture search for transformer in knowledge tracing. In *Advances in Neural Information Processing Systems 36 (NeurIPS'23)*, pp. 19520–19539, New Orleans, LA, USA, 2023.

Zhi-Hua Zhou, Yang Yu, and Chao Qian. *Evolutionary Learning: Advances in Theories and Algorithms*. Springer, Singapore, 2019.

Barret Zoph, Vijay Vasudevan, Jonathon Shlens, and Quoc V Le. Learning transferable architectures for scalable image recognition. In *Proceedings of the 35th IEEE Conference on Computer Vision and Pattern Recognition (CVPR'18)*, pp. 8697–8710, Salt Lake City, UT, USA, 2018.

A Appendix

A.1 Neural Architecture for Solving Mcc

This part details the cell-based neural architecture introduced in Lv et al. (2025) for solving Mcc. As illustrated in Figure 4, the architecture consists of $M-1$ cells followed by M class-score neurons and a Softmax layer. The m-th cell, $m \in [1..M-1]$, acts as a binary detector primarily associated with class m and outputs either 0 or 1. The resulting cell outputs are aggregated to determine the predicted class.

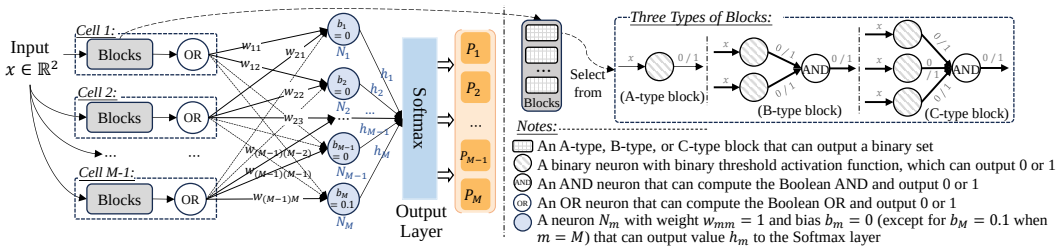


Figure 4: Cell-based neural architecture for Mcc Lv et al. (2025). Each cell contains A-, B-, and C-type blocks and outputs the OR of their binary outputs. The $M-1$ cell outputs form a binary vector passed to M class-score neurons. A Softmax layer converts the scores into probabilities, and the class with the highest probability is predicted.

Each block is constructed from binary threshold neurons. Let $H(b, \varphi)$ denote the half-plane represented by a threshold neuron, where the bias b specifies the distance of its boundary from the origin and φ specifies the direction of its unit normal vector. A block containing q threshold neurons represents the following region:

$$\mathcal{R} = S \cap \bigcap_{j=1}^{q} H(b_j, \varphi_j), \tag{5}$$

where S is the unit disk. The block outputs 1 if the input lies in $\mathcal{R}$ and outputs 0 otherwise.

The architecture employs the following three types of blocks.

- An A-type block contains one threshold neuron with bias $\cos(\pi/n)$. Its boundary intersects the unit circle along a chord, and the corresponding half-plane forms a circular segment.
- A B-type block contains two threshold neurons with zero biases. Their directions satisfy

$$|\varphi_1 - \varphi_2| \in \left\{ \pi - \frac{2\pi}{n}, \pi + \frac{2\pi}{n} \right\}.$$

  The intersection of the two half-planes with the unit disk forms a sector of angle $2\pi/n$.
- A C-type block contains three threshold neurons. Two neurons have zero biases and form the two radial boundaries of a sector. The third neuron has bias $\cos(\pi/n)$ and introduces a chord boundary. The direction differences are chosen such that the intersection of the three half-planes forms the triangular part of a sector.

We assume that the remaining direction parameters are optimized during fitness evaluation so that each block is aligned with an elementary region of Mcc. Consequently, an A-type block can represent a segment, a B-type block can represent a complete sector, and a C-type block can represent a triangle. Since a sector is the union of its segment and triangle parts, it can also be represented jointly by one A-type block and one C-type block.

Let $\mathcal{R}_{m,t}$ denote the region represented by the t-th block in cell m. The output of this cell is the OR of its block outputs:

$$z_m(\boldsymbol{x}) = \mathbb{I}\left[\boldsymbol{x} \in \bigcup_t \mathcal{R}_{m,t}\right] \in \{0, 1\}, \qquad m \in [1..M-1], \tag{6}$$

where $\mathbb{I}[\cdot]$ denotes the indicator function.

The $M-1$ cell outputs are then passed to M class-score neurons. For $i \in [1..M]$, the score of neuron N_i is

$$h_i(\boldsymbol{x}) = b_i + \sum_{m=1}^{M-1} w_{i,m} z_m(\boldsymbol{x}), \tag{7}$$

where

$$w_{i,m} = \begin{cases} 1, & i = m, \\ 0.5, & i = m+1, \\ 0.4, & i = m-1, \\ 0, & \text{otherwise}, \end{cases} \qquad \begin{array}{l} \\ i \in [1..M], \\ m \in [1..M-1]. \\ \\ \end{array} \tag{8}$$

The biases are fixed as

$$b_i = \begin{cases} 0, & i \in [1..M-1], \\ 0.1, & i = M. \end{cases} \tag{9}$$

The Softmax probability of class i is

$$P_i(\boldsymbol{x}) = \frac{\exp(h_i(\boldsymbol{x}))}{\sum_{j=1}^{M} \exp(h_j(\boldsymbol{x}))}, \qquad i \in [1..M]. \tag{10}$$

Since Softmax preserves the ordering of the scores, the predicted label is equivalently given by

$$\widehat{y}(\boldsymbol{x}) = \arg \max_{i \in [1..M]} h_i(\boldsymbol{x}).$$

The specified weights resolve the overlaps between neighboring cells. If only cell m covers an input, then $h_m(\boldsymbol{x}) = 1$, whereas the scores contributed to its neighboring classes are at most 0.5 and 0.4; hence, the input is classified as class m. More importantly, suppose that a B-type block in cell m covers both a triangle of class m and its adjacent segment of class $m+1$. If an A-type block in cell $m+1$ also covers this segment, then $z_m(\boldsymbol{x}) = z_{m+1}(\boldsymbol{x}) = 1$ and

$$h_m(\boldsymbol{x}) = 1 + 0.4 = 1.4, \qquad h_{m+1}(\boldsymbol{x}) = 0.5 + 1 = 1.5.$$

The overlapped segment is therefore assigned to class $m+1$. Finally, if no cell covers the input, then all cell outputs are zero. In this case, N_M is the only neuron with a positive score, due to $b_M = 0.1$, and the input is classified as class M.

The last cell is an exception to the overlap-correction mechanism. A B-type block in cell $M-1$ may cover both a triangle of class $M-1$ and its adjacent segment of class M. Since no cell is associated with class M, this segment cannot be corrected by an A-type block in a subsequent cell. Such incorrectly covered segments give rise to the penalty $\epsilon_{\boldsymbol{x}}$ in Lemma 1.

A.2 Proofs

A.2.1 Proof of Theorem 1

Proof of Theorem 1. We first present the probability bounds of a mutation operation (defined in Definition 2) causing the I-value or J-value (calculated by Eq. (2) and Eq. (3)) of a cell to increase, decrease, or remain unchanged, respectively. Here, γ_+, γ_-, and γ_0 denote the corresponding probabilities. Specifically, (1) for cell in $\boldsymbol{x} \in \cup_{i=0}^{2r-1} \mathcal{S}_i$, we have $\gamma_+ \in [2/9, 1/3], \gamma_- \in [1/6, 1/3]$, and $\gamma_0 \in [1/3, 4/9]$; (2) for cell in $\boldsymbol{x} \in \cup_{j=0}^{2r-1} \mathcal{S}_{2r}^j$, we have $\gamma_+ \in [1/9, 1/3], \gamma_- \in [1/18, 1/2]$, and $\gamma_0 \in [1/9, 7/9]$. Some of these bounds have been used in Lv et al. (2025) and will also be used in our analysis. As their derivations are straightforward and do not affect the final asymptotic runtime bound, we omit the detailed proofs.

To prove the runtime results, we begin with the first optimization phase, in which the population space is partitioned into $N+1$ levels, with all populations in level i having the same $\mathbb{I}$-value. Let the current population X be at level i ($0 \leq i < N$). Next, we analyze the runtimes spent in escaping from Case $i.1$, Case $i.2$, and Case $i.3$, respectively.

Case $i.1$: The condition for leaving Case $i.1$ is that all of the individuals in the current population X have $\mathbb{I}$-value at least i. Once left, the algorithm never returns to this case. Let G denote the event that a solution $\boldsymbol{x}$ with $\mathbb{I} = i$ generates an offspring with $\mathbb{I}$-value greater than or equal to i via mutation. The probability that event G occurs is denoted by P_G. Then, the expected number of fitness evaluations until all individuals in X reach $\mathbb{I} = i$ is at most $\mathcal{O}((\mu+\lambda)\log\mu)/((1-p_c)P_G)$, which is derived by Lemma 2 in Sudholt (2017). We now estimate the probability P_G, which depends on at least three occurrences during offspring generation:

- $\boldsymbol{x}$ creates a clone offspring with probability of $(1-p_m)^{M-1} \geq 1/(e^c + \mathcal{O}(1/(M-1)))$, where the inequality holds by $p_m = c/(M-1) < 1$ for some constant c and $\forall c/(M-1) < 1$

$$\begin{aligned}
&(1-\frac{c}{M-1})^{M-1-1} \geq (1-\frac{c}{M-1})^{M-1-2} \\
&\geq e^{-c}(1-\frac{c^2}{M-1}) \geq e^{-c} - \mathcal{O}(\frac{1}{M-1}) \\
&= (e^c + \mathcal{O}(\frac{1}{M-1}))^{-1}.
\end{aligned} \tag{11}$$

- $\boldsymbol{x}$ creates an offspring $\boldsymbol{y}$ ($\boldsymbol{y} \neq \boldsymbol{x}$) that belongs to fitness level i by increasing the I-value of one cell and decreasing it in another cell, which is denoted as event E. To achieve this, it is necessary to select one cell from the $|\boldsymbol{o}|_0$ cells whose I-values are not optimal, and mutate it to increase its I-value by one (i.e., probability $p_m \cdot \gamma_+$); then, another cell is selected from the at least $\max\{|\boldsymbol{o}|_1 + |\boldsymbol{o}|_0^{I<2r} - 1, 1\}$ cells whose I-values are not zero (where the term $|\boldsymbol{o}|_0^{I<2r} - 1 \geq 0$ accounts for the situation that the transition γ_+ occurs in a cell whose I-value is less than $2r$), and make it decrease I-value by one (i.e., probability $p_m \cdot \gamma_-$); simultaneously, the I-value of each other cell will remain unchanged (i.e., probability $1-p_m$). Therefore, event E occurs with a probability of at least

$$\begin{aligned}
&|\boldsymbol{o}|_0\gamma_+ \max\{|\boldsymbol{o}|_1 + |\boldsymbol{o}|_0^{I<2r} - 1, 1\}\gamma_- p_m^2(1-p_m)^{M-1-2} \\
&\geq \left(M - 1 - \frac{i}{2r}\right) \cdot \frac{2}{9} \cdot 1 \cdot \frac{1}{6} \cdot p_m^2 \cdot (1-p_m)^{M-1-2} \\
&\geq \frac{1-i/N}{27(M-1)} \frac{c^2}{(e^c + \mathcal{O}(1/(M-1)))},
\end{aligned} \tag{12}$$

  where the first inequality holds by $\gamma_+ \geq 2/9$, $\gamma_- \geq 1/6$, and $|\boldsymbol{o}|_0 \geq M - 1 - |\boldsymbol{o}|_1 \geq M - 1 - i/(2r)$ (estimated by Eq. (3) in Lv et al. (2025)).

- $\boldsymbol{x}$ creates an offspring $\boldsymbol{y}$ that belongs to fitness level j ($j > i$), which happens with probability of at least

$$\begin{aligned}
&\gamma_+ \cdot |\boldsymbol{o}|_0 \cdot p_m \cdot (1-p_m)^{M-1-1} \\
&\geq \frac{\gamma_+ \cdot |\boldsymbol{o}|_0}{M-1} \cdot \frac{c}{e^c + \mathcal{O}(1/(M-1))} \\
&\geq \frac{2}{9}\frac{M-1-i/(2r)}{M-1}\frac{c}{e^c + \mathcal{O}(1/(M-1))} \\
&\geq \frac{2}{9}\frac{(1-i/N)c}{e^c + \mathcal{O}(1/(M-1))}.
\end{aligned} \tag{13}$$

Therefore, when only the mutation operator functions on parent $\boldsymbol{x}$, the probability of producing an offspring $\boldsymbol{y}$ with fitness level j ($j \geq i$) is

$$\begin{aligned}
P_G \geq &\frac{1}{e^c + \mathcal{O}(1/(M-1))} \\
&+ \frac{1}{27}(1-\frac{i}{N})\frac{1}{M-1}\frac{c^2}{(e^c + \mathcal{O}(1/(M-1)))} \\
&+ \frac{2}{9}(1-\frac{i}{N})\frac{c}{e^c + \mathcal{O}(1/(M-1))} \\
\geq &\frac{1}{e^c + \mathcal{O}(1/(M-1))}(1-\frac{i}{N}) \cdot \frac{c^2}{M-1}.
\end{aligned}$$

Calculating the runtime of $\mathcal{O}((\mu+\lambda)\log\mu)/((1-p_c)P_G)$ to all of fitness level $i \in [0..N-1]$, the overall time spent in all Case $i.1$ is

$$\begin{aligned}
\mathbb{E}[T_{i.1}] &\leq \sum_{i=0}^{N-1} \frac{\mathcal{O}((\mu+\lambda)\log\mu)}{P_G(1-p_c)} \\
&\leq \frac{\mathcal{O}(N(\mu+\lambda)\log\mu)(M-1)(e^c+\mathcal{O}(\frac{1}{M-1}))}{(1-p_c)c^2} \sum_{i=0}^{N-1} \frac{1}{1-i/N} \\
&\leq \mathcal{O}(N(\mu+\lambda)\log\mu)\frac{(M-1)(e^c+\mathcal{O}(1/(M-1))}{c^2(1-p_c)} N\ln N \\
&\leq \mathcal{O}\Big(\frac{e^c}{c^2}(M-1)N^2(\mu+\lambda)\ln\mu\ln N\Big). \qquad (14)
\end{aligned}$$

Since a fallback may occur between cases $i.2$ and $i.3$ (i.e., from $i.3$ back to $i.2$), and the analysis of escaping from Case $i.2$ depends on the probability of escaping from Case $i.3$ (i.e., $p_{3\to i^+}$), we first analyze the process of the algorithm escaping from Case $i.3$.

Case $i.3$: In this case, at least two individuals in the population have an $\mathbb{I}$-distance greater than one (referred to distinct individuals for simplicity). The algorithm can permanently leave Case $i.3$ by successfully jumping to Case i^+. To estimate the probability of such a jump, denoted as $p^{3\to i^+}$, it is sufficient to consider a successful crossover between two distinct individuals.

Assuming that two different parents $\boldsymbol{x}_1$ and $\boldsymbol{x}_2$ are selected for crossover, this happens with probability $2(\mu-1)/\mu^2$. Let $D \leq M-1$ denote their $\mathbb{I}$-distance, and let $D_1 \geq 1$ be the number of cells (among these D cells) where $\boldsymbol{x}_1$ has a higher I-value than $\boldsymbol{x}_2$. Consequently, there are $D-D_1$ cells where $\boldsymbol{x}_2$ has a higher I-value than $\boldsymbol{x}_1$. To generate an offspring that is strictly better than the current one in Case $i.3$, it is sufficient that the offspring y: (i) inherits all the D_1 advantageous cells from $\boldsymbol{x}_1$, and (ii) inherits at least one of the $D-D_1$ advantageous cells from $\boldsymbol{x}_2$. Since crossover independently selects each cell from either parent with probability $1/2$, the above event occurs with probability at least

$$\left(\frac{1}{2}\right)^{D_1}\left(1-\left(\frac{1}{2}\right)^{D-D_1}\right).$$

If mutation does not destroy this beneficial structure, which happens with probability at least $(1-p_m)^{M-1}$, then the probability of generating a better offspring by crossover is at least

$$\left(\frac{1}{2}\right)^{D_1}\left(1-\left(\frac{1}{2}\right)^{D-D_1}\right)(1-p_m)^{M-1}.$$

Since $D-D_1 \geq 1$, we have

$$1-\left(\frac{1}{2}\right)^{D-D_1} \geq \frac{1}{2}.$$

Moreover, since $D \leq M-1$ and $D-D_1 \geq 1$, it holds that $D_1 \leq M-2$, and thus

$$\left(\frac{1}{2}\right)^{D_1+1} \geq \left(\frac{1}{2}\right)^{M-1}.$$

Therefore, the probability of generating a better offspring by crossover is at least

$$\frac{1}{2^{M-1}}(1-p_m)^{M-1} \geq \frac{1}{2^{M-1}\big(e^c+\mathcal{O}(1/(M-1))\big)}.$$

In addition, a better offspring can also be generated solely by mutation, without crossover. This occurs with a probability of at least $(1-p_c)\cdot(2/9)(1-i/N)c/(e^c+\mathcal{O}(1/(M-1)))$, as estimated in Eq. (13).

Since the selection mechanism preserves offspring with higher fitness, the probability of jumping from Case $i.3$ to Case i^+ is

$$p^{3\to i+} \geq p_c\cdot\frac{2(\mu-1)}{\mu^2}\cdot\frac{1}{2^{M-1}\big(e^c+\mathcal{O}(1/(M-1))\big)} + (1-p_c)\cdot\frac{2}{9}\frac{c\big(1-i/N\big)}{e^c+\mathcal{O}(1/(M-1))}$$

$$\geq \frac{1}{e^c + \mathcal{O}(1/(M-1))}\left(\frac{2(\mu-1)}{\mu^2}\frac{p_c}{2^{M-1}} + \frac{2c(1-p_c)}{9}\Big(1-\frac{i}{N}\Big)\right) \tag{15}$$

Therefore, the overall time spent in Case $i.3$ ($i \in [0, N-1]$) is

$$\begin{aligned}
\mathbb{E}[T_{i.3}] &\leq \sum_{i=0}^{N-1}\Big(e^c + \mathcal{O}\Big(\frac{1}{M-1}\Big)\Big)\left(\frac{2(\mu-1)p_c}{\mu^2 2^{M-1}} + \frac{2c(1-p_c)}{9}\Big(1-\frac{i}{N}\Big)\right)^{-1} \\
&\leq \Big(e^c + \mathcal{O}\Big(\frac{1}{M-1}\Big)\Big)\min\left\{\sum_{i=0}^{N-1}\frac{\mu^2 2^{M-1}}{2(\mu-1)p_c}, \sum_{i=0}^{N-1}\frac{9}{2c(1-p_c)\Big(1-\frac{i}{N}\Big)}\right\} \\
&= \Big(e^c + \mathcal{O}\Big(\frac{1}{M-1}\Big)\Big)\min\left\{\frac{\mu^2 2^{M-1}}{2(\mu-1)p_c}N, \frac{9}{2c(1-p_c)}N\ln N\right\} \\
&= \mathcal{O}\left(\frac{e^c}{c}N\ln N\ \min\left\{\frac{c\,\mu\,2^{M-1}}{\ln N},\ 1\right\}\right),
\end{aligned} \tag{16}$$

where the first inequality holds by $\forall a > b > 0 : 1/(a+b) \leq 1/a \leq 1/b$.

Case $i.2$: The condition for leaving this case is that either population contains at least two individuals whose $\mathbb{I}$-distance is not zero, or population contains one individual with $\mathbb{I}$-value greater than i, i.e., the algorithm successfully makes the jump from Case $i.2$ to either Case $i.3$ or Case i^+. In this case, the crossover operator is ineffective because all candidate parents have identical I^m-value for every $m \in [1..M-1]$, and thus, the successful jump relies solely on the mutation operator. Based on this, we analyze the probabilities of leaving Case $i.2$ to Case $i.3$ and Case i^+, respectively.

The probability of jumping from Case $i.2$ to i^+ is

$$p^{2\to i+} \geq \frac{\mu}{\mu+\lambda}\cdot\frac{2}{9}\Big(1-\frac{i}{N}\Big)\frac{c}{(e^c + \mathcal{O}(1/(M-1)))},$$

where the term before "$\cdot$" holds by the probability of selecting one offspring with fitness j ($j > i$) into the next generation is at least $1 - \binom{\mu+\lambda-1}{\mu}/\binom{\mu+\lambda}{\mu} = 1 - \lambda/(\mu+\lambda) = \mu/(\mu+\lambda)$, and the left term holds by Eq. (13).

Then, we calculate the probability of jumping from Case $i.2$ to Case $i.3$. This requires generating at least one offspring $\boldsymbol{y}$ with fitness i and $d_{\mathbb{I}}(\boldsymbol{x}, \boldsymbol{y}) > 0$ where $\boldsymbol{x}$ denotes the parent, which occurs with probability $p^{2\to 3}$. To produce such an offspring, the mutation operator must simultaneously decrease the I-value of one cell (whose I-value is greater than 0) by one and increase the I-value of another cell (whose I-value is less than N) by one. This happens with probability of at least

$$\Big(1-\frac{i}{N}\Big)\frac{1}{27(M-1)}\frac{c^2}{(e^c + \mathcal{O}(1/(M-1)))}. \tag{17}$$

The selection mechanism ensures that such offspring enter the next generation with probability at least $\mu/(\mu+\lambda)$. Thus, we have

$$p^{2\to 3} \geq \frac{\mu}{\mu+\lambda}\cdot\frac{1}{27}\Big(1-\frac{i}{N}\Big)\frac{1}{M-1}\frac{c^2}{e^c + \mathcal{O}(1/(M-1))}. \tag{18}$$

Since the algorithm incorporates a strategy to maintain the diversity of the population, the algorithm may return from Case $i.3$ back to Case $i.2$ with probability $p^{3\to 2}$. Therefore, when calculating the expected runtime spent in Case $i.2$, we focus on the jump from Case $i.2$ to Case $i.3$ without returning under the condition that Case $i.3$ either returns to $i.2$ or jumps to $i+$, which happens with probability

$$p^{2\to 3}\cdot\frac{p^{3\to i+}}{p^{3\to 2} + p^{3\to i+}} = p^{2\to 3}\cdot\left(1 - \frac{p^{3\to 2}}{p^{3\to 2} + p^{3\to i+}}\right), \tag{19}$$

where $p^{2\to 3}$ can be derived by Eq. (18), $p^{3\to i+}$ can be derived by Eq. (15), and $p^{3\to 2}$ will be analyzed in the following.

To calculate the transition probability $p^{3\to 2}$, we define the majority individuals as the largest subset of best individuals in the population that have identical I-value for each cell (i.e., the $\mathbb{I}$-distance among these majority individuals is zero), and let the current population have w such majority individuals, where $0 < w < \mu$. For the population to return to Case $i.2$, the algorithm must generate at least $\mu - w$ offspring whose $\mathbb{I}$-distance to any majority individual is zero, while the remaining $\lambda - (\mu - w)$ offspring can be arbitrary. Subsequently, the selection mechanism selects a total of μ majority individuals from the combined pool of parents and offspring to form the next generation, which happens with a probability of at most $\binom{\lambda+w}{\mu}/\binom{\lambda+\mu}{\mu} < \binom{\lambda+\mu-1}{\mu}/\binom{\lambda+\mu}{\mu} = \mu/(\mu+\lambda)$. Now we focus on the crucial step of generating offspring belonging to the majority individuals. We pessimistically assume that the $\mathbb{I}$-distance between the individuals in the population is at most 2 (similar to the assumption used in the calculation of p_r in Corus & Oliveto (2018)). Under this assumption, three situations can make the population in Case $i.3$ generate an offspring belonging to the majority individuals:

- Two parents are selected from the majority individuals (i.e., probability $(w/\mu)^2 \leq ((\mu-1)/\mu)^2$ since $w < \mu$), and the mutation does not change the I-value of each cell (i.e., probability $(1 - p_m + p_m \cdot \gamma_0)^{M-1}$);
- One parent is chosen from the majority individuals and the other from the minority individuals (i.e., probability $2w(\mu-w)/\mu^2 \leq 1/2$ since $w < \mu$), then (1) the crossover selects both differing cells from the majority individual and mutation does not change the I-value of each cell (i.e., probability $(1/2)^2 \cdot (1 - p_m + p_m \cdot \gamma_0)^{M-1}$), or (2) the crossover selects both differing cells from the minority individual and mutation modifies the I-values of both cells to match those of the majority individual (i.e., probability $(1/2)^2 \cdot (\gamma_{i,i^+} \cdot \gamma_{i,i^-}) \cdot p_m^2 (1 - p_m + p_m \cdot \gamma_0)^{M-1-2}$, where γ_{i,i^+} and γ_{i,i^-} represents the probability of I-value increase or decrease after mutation, both less than 1), or (3) one differing cell is taken from each parent and mutation changes the I-value of the cell taken from the minority individual to match that of the majority individual (i.e., probability $(1/2) \cdot \max\{\gamma_{i,i^+}, \gamma_{i,i^-}\} \cdot p_m (1 - p_m + p_m \cdot \gamma_0)^{M-1-1}$);
- Two parents are selected from the minority individuals (i.e., probability $((\mu - w)/\mu)^2 \leq (\mu - 1)^2/\mu^2$) and mutation changes at least two different cells to match the majority individual (i.e., probability $p_m^2 (1 - p_m + p_m \cdot \gamma_0)^{M-1-2}$).

Hence, the probability of a population jumping from Case $i.3$ to Case $i.2$ is

$$
\begin{aligned}
p^{3\to 2} \leq \frac{\mu}{\mu+\lambda} \Big[& \frac{(\mu-1)^2}{\mu^2} \left(1 - p_m + p_m \gamma_0\right)^{M-1} \\
& + \frac{1}{2} \Big(\frac{1}{4} \left(1 - p_m + p_m \gamma_0\right)^{M-1} \\
& + \frac{1}{4} \left(\gamma_{i+} \gamma_{i-}\right) p^2 \left(1 - p_m + p_m \gamma_0\right)^{M-1-2} \\
& + \frac{1}{2} \max\{\gamma_{i-}, \gamma_{i+}\} p_m \left(1 - p_m + p_m \gamma_0\right)^{M-1-1} \Big) \\
& + \frac{(\mu-1)^2}{\mu^2} p_m^2 \left(1 - p_m + p_m \gamma_0\right)^{M-1-2} \Big]^{\mu-w} \\
\leq & \frac{(\mu-1)^2 + \mu^2(\frac{1}{8} + \mathcal{O}(\frac{1}{M-1})) + ((\mu-1)^2 + \mu^2)\mathcal{O}(\frac{1}{(M-1)^2})}{e^{2c/3}\mu(\mu+\lambda)} \\
\leq & \frac{(\mu-1)^2 + \mu^2(\frac{1}{8} + \mathcal{O}(\frac{1}{M-1}))}{e^{2c/3}\mu(\mu+\lambda)},
\end{aligned} \tag{20}
$$

where the second inequality holds by $\gamma_0 \leq 1/3$, $(1 - p_m + p_m \cdot \gamma_0)^{M-1} \leq e^{-2c/3}$, and $(1 - p_m + p_m \cdot \gamma_0)^{M-1-1} \leq (1 - p_m(1-\gamma_0))^{M-1-2} \leq e^{-2c/3}/(1 - (2c/3)/(M-1))^2$. By substituting Eq. (20) into Eq. (19) and performing calculation analogous to Eq. (31), we have that the overall runtime spent in Case $i.2$ is

$$
\mathbb{E}[T_{i.2}] = \sum_{i=0}^{N-1} \left(p^{2\to i^+} + p^{2\to 3} - p^{2\to 3} \cdot \frac{p^{3\to 2}}{p^{3\to 2} + p^{3\to i+}} \right)^{-1}
$$

$$\leq \sum_{i=0}^{N-1} \left(\frac{\mu}{\mu+\lambda} \frac{1}{27} \Big(1 - \frac{i}{N}\Big) \frac{c(6 + c/(M-1))}{e^c + \mathcal{O}(1/(M-1))} - p^{2\to 3} \frac{p^{3\to 2}}{p^{3\to 2} + p^{3\to i+}} \right)^{-1}. \tag{21}$$

Among the equation, $p^{2\to 3}p^{3\to 2}$ represents the probability that population jumps from Case $i.2$ to Case $i.3$ and then returns to Case $i.2$, whose upper bound is

$$p^{2\to 3}p^{3\to 2} \leq \frac{\mu^2}{(\mu+\lambda)^2} \Big(1 - \frac{i}{N}\Big) \frac{1}{M-1} \frac{c^2}{e^{2c/3}};$$

and $p^{3\to 2} + p^{3\to i+}$ denotes the probability that population jumps from Case $i.3$ to Case $i.2$ or Case i^+, whose lower bound is

$$\begin{aligned} p^{3\to 2} + p^{3\to i+} \geq{}& \frac{\mu}{e^{2c/3}(\mu+\lambda)} \\ &+ \Big(e^c + \mathcal{O}\Big(\frac{1}{M-1}\Big)\Big)^{-1} \left(\frac{2}{3} \frac{\mu-1}{\mu^2} \frac{p_c}{2^{(M-1)/2}} \right. \\ &\left. + \frac{2c(1-p_c)}{9} \Big(1 - \frac{i}{N}\Big) \right) \end{aligned}$$

When the term $(e^c + \mathcal{O}(1/(M-1)))^{-1} 2c(1-p_c)/9(1-i/N))$ dominates $p^{3\to 2} + p^{3\to i+}$ (i.e., $\mu = \omega(r)$), and the difference between the two terms in Eq. (21) is at least $1/N^2$ (i.e., $\lambda = \omega(rM)$), we have

$$\mathbb{E}[T_{i.2}] = \mathcal{O}((1+\lambda/\mu) \cdot (e^c/c^2) \cdot (M-1)N^3). \tag{22}$$

In this case, we have

$$\mathbb{E}[T_i] = \mathcal{O}\Big(\frac{e^c}{c^2}(M-1)N^2(\mu+\lambda) \cdot \max\Big\{ \ln\mu \ln N, \frac{N}{\mu} \Big\}\Big). \tag{23}$$

When $\mu \ln \mu = o(N/\ln N)$, it can be further simplified as

$$\mathbb{E}[T_i] = \mathcal{O}\Big(\frac{e^c}{c^2}(M-1)N^3\Big(1 + \frac{\lambda}{\mu}\Big)\Big), \tag{24}$$

where $N = 2r(M-1)$.

The analysis of the second optimization phase is similar to that of the first phase, with one key difference: it focuses on the change of $(\mathbb{J} - \epsilon)$-value, whose range is $0 \leq \mathbb{J} - \epsilon \leq N-1$, instead of the $\mathbb{I}$-value. This means the number of fitness levels in the second phase remains N. Since this difference does not affect the asymptotic result of the runtime spent on each fitness level case, the expected runtime for the second optimization phase is the same as that of the first phase. □ □

A.2.2 Proof of Theorem 2

Proof of Theorem 2. We divide the optimization process into the same two phases used in the analysis of the crossover-based algorithm. For a population X, define

$$D_1(X) = \min\{V_1(\boldsymbol{x}) \mid \boldsymbol{x} \in X\}, \qquad V_1(\boldsymbol{x}) = N - \mathbb{I}_{\boldsymbol{x}}, \tag{25}$$

$$D_2(X) = \min\{V_2(\boldsymbol{x}) \mid \boldsymbol{x} \in X\}, \qquad V_2(\boldsymbol{x}) = N - \big(\mathbb{J}_{\boldsymbol{x}} - \epsilon_{\boldsymbol{x}}\big). \tag{26}$$

Phase 1 ends when $D_1(X) = 0$. Starting from the end of Phase 1, Phase 2 ends when $D_2(X) = 0$.

We first analyze Phase 1. Suppose that the current population has distance $D_1(X) = d$, where $d \in [1..N]$. Let $\boldsymbol{x}^*$ be an individual attaining the current minimum distance. Since parent selection is uniform, the probability of selecting $\boldsymbol{x}^*$ is at least $1/\mu$. The total deficit of $\boldsymbol{x}^*$ is d, and the contribution of each cell is at most $2r$. Therefore, $\boldsymbol{x}^*$ contains at least $d/(2r)$ cells whose I-values have not reached their maximum. A sufficient event for reducing the distance is that the algorithm selects one of these cells for mutation, performs an improving mutation operation on this cell, and leaves all other cells unchanged.

According to the mutation probabilities used in the proof of Theorem 1, an improving mutation operation occurs with probability at least $1/9$. Hence, one offspring reduces the distance by at least one with probability

$$q_d \geq \frac{1}{\mu} \cdot \frac{d}{2r} \cdot \frac{c}{M-1} \cdot \frac{1}{9} \cdot \left(1 - \frac{c}{M-1}\right)^{M-2} = \Omega\left(\frac{cd}{\mu e^c r M}\right).$$

Here, we use that c is a constant and $(1 - \frac{c}{M-1})^{M-2} = \Omega(e^{-c})$.

Since λ offspring are independently generated from the current population, the probability that at least one offspring reduces the distance is at least $1 - (1-q_d)^\lambda$. Therefore, the expected number of fitness evaluations required to leave distance level d is

$$\frac{\lambda}{1-(1-q_d)^\lambda} = \mathcal{O}\left(\max\left\{\lambda, \frac{1}{q_d}\right\}\right)\mathcal{O}\left(\max\left\{\lambda, \frac{\mu e^c r M}{cd}\right\}\right). \tag{27}$$

The first equality follows from $1 - (1-q_d)^\lambda = \Omega\left(\min\{1, \lambda q_d\}\right)$.

Summing Eq. (27) over all possible distance levels gives

$$\begin{aligned}
\mathbb{E}[T_1] &= \mathcal{O}\left(\sum_{d=1}^{N} \max\left\{\lambda, \frac{\mu e^c r M}{cd}\right\}\right) \\
&= \mathcal{O}\left(N\lambda + \frac{\mu e^c r M}{c}\sum_{d=1}^{N}\frac{1}{d}\right) \\
&= \mathcal{O}\left(rM\lambda + \frac{\mu e^c}{c} rM\ln(rM)\right),
\end{aligned} \tag{28}$$

where $N = 2r(M-1) = \Theta(rM)$.

The analysis of Phase 2 is analogous. Once Phase 1 has ended, elitist environmental selection preserves at least one solution satisfying $\mathbb{I}_{\boldsymbol{x}} = N$. For a population with $D_2(X) = d$, selecting a current best individual and applying an improving mutation to one of its non-optimal cells reduces $D_2(X)$ with probability $\Omega\left(\frac{cd}{\mu e^c r M}\right)$. Consequently,

$$\mathbb{E}[T_2] = \mathcal{O}\left(rM\lambda + \frac{\mu e^c}{c} rM\ln(rM)\right). \tag{29}$$

Including the μ fitness evaluations used for initialization and combining Eq. (28) and Eq. (29), we obtain

$$\begin{aligned}
\mathbb{E}[T_{\text{mutation}}] &= \mathcal{O}\left(\mu + rM\lambda + \frac{\mu e^c}{c} rM\ln(rM)\right) \\
&= \mathcal{O}\left(\mu + rM \cdot \max\left\{\lambda, \frac{\mu e^c}{c}\ln(rM)\right\}\right).
\end{aligned} \tag{30}$$

This completes the proof. □

A.2.3 Proof of Theorem 3

Proof of Theorem 3. According to the elitist selection strategy and the duplicate-based tie-breaking rule, we have the following conclusions for the population at level i:

1. The population can leave Case $i.1$ by jumping to Case $i.2$, Case $i.3$, or a higher level i^+;
2. The population can leave Case $i.2$ by jumping to Case $i.3$ (this transition is irreversible since the duplicate-based tie-breaking rule) or a higher level i^+;
3. The population can leave Case $i.3$ by jumping to a higher level i^+.

The expected runtimes for the population to leave Cases $i.1$ and $i.3$ are given as

$$\mathbb{E}[T_{i.1}] = \mathcal{O}\Big(\frac{e^c}{c^2}(M-1)N^2(\mu+\lambda)\ln\mu\ln N\Big),$$
$$\mathbb{E}[T_{i.3}] = \mathcal{O}\left(\frac{e^c}{c}N\ln N\ \min\left\{\frac{c\,\mu\,2^{M-1}}{\ln N},\ 1\right\}\right).$$

Next, we analyze the expected runtime required for the population to leave Case $i.2$.

Case $i.2$: The condition for leaving this case is that the population contains at least two individuals whose $\mathbb{I}$-distance is nonzero, or one individual whose $\mathbb{I}$-value is greater than i, i.e., the algorithm successfully jumps from Case $i.2$ to Case $i.3$ or i^+. As shown in the proof of Eq. (1), the probabilities of these transitions, i.e., $p^{2\to 3}$ and $p^{2\to i+}$, satisfy

$$p^{2\to 3} \geq \frac{c^2}{e^c + \mathcal{O}(1/(M-1))},$$
$$p^{2\to i+} \geq \frac{\mu}{\mu+\lambda}\cdot\frac{2}{9}\Big(1-\frac{i}{N}\Big)\frac{c}{e^c+\mathcal{O}(1/(M-1))}.$$

Hence, the expected runtime for the algorithm to leave Case $i.2$ can be expressed as

$$\begin{aligned}
\mathbb{E}[T_{i.2}] &\leq \sum_{i=0}^{N-1}\frac{1}{p^{2\to i+}+p^{2\to 3}}\\
&\leq \sum_{i=0}^{N-1}\Big(\frac{\mu}{\mu+\lambda}\frac{1}{27}\Big(1-\frac{i}{N}\Big)\frac{c(6+c/(M-1))}{e^c+\mathcal{O}(1/(M-1))}\Big)^{-1}\\
&\leq \frac{\mu+\lambda}{\mu}\frac{e^c+\mathcal{O}(1/(M-1))}{c(6+c/(M-1))/27}N\ln N\\
&= \mathcal{O}\Big(\Big(1+\frac{\lambda}{\mu}\Big)\frac{e^c}{c^2}(M-1)N\ln N\Big).
\end{aligned} \tag{31}$$

By integrating the expected runtimes for leaving the three cases, the total expected runtime in the first optimization phase is

$$\begin{aligned}
\mathbb{E}[T_i] &= \mathbb{E}[T_{i.1}] + \mathbb{E}[T_{i.2}] + \mathbb{E}[T_{i.3}]\\
&\leq \mathcal{O}\Big(\frac{e^c}{c^2}(M-1)N^2(\mu+\lambda)\ln\mu\ln N\Big)\\
&\quad + \mathcal{O}\Big(\Big(1+\frac{\lambda}{\mu}\Big)\frac{e^c}{c^2}(M-1)N\ln N\Big)\\
&\quad + \mathcal{O}\left(\frac{e^c}{c}N\ln N\ \min\left\{\frac{c\,\mu\,2^{M-1}}{\ln N},\ 1\right\}\right).\\
&= \mathcal{O}\Big(\frac{e^c}{c^2}(M-1)N^2(\mu+\lambda)\ln\mu\ln N\Big).
\end{aligned} \tag{32}$$

The analysis of the second optimization phase differs from the first phase in that the dominant fitness variable changes from $\mathbb{I}$ to $(\mathbb{J}-\epsilon)$. However, in this phase, the fitness landscape is still divided into N levels, and the asymptotic form of the runtime for leaving each level remains unchanged. Therefore, the total expected runtime in the second phase is still $\mathcal{O}\big((e^c/c^2)(M-1)N^2(\mu+\lambda)\ln\mu\ln N\big)$. Combining the analyses of both phases yields the total expected runtime stated in the theorem, which completes the proof. □