BLACK-BOX COMBINATORIAL OPTIMIZATION WITH ORDER-INVARIANT REINFORCEMENT LEARNING

Anonymous authors
Paper under double-blind review

ABSTRACT

We introduce an order-invariant reinforcement learning framework for black-box combinatorial optimization. Classical estimation-of-distribution algorithms (EDAs) often rely on learning explicit variable dependency graphs, which can be costly and fail to capture complex interactions efficiently. In contrast, we parameterize a multivariate autoregressive generative model trained without a fixed variable ordering. By sampling random generation orders during training - a form of information-preserving dropout - the model is encouraged to be invariant to variable order, promoting search-space diversity and shaping the model to focus on the most relevant variable dependencies, improving sample efficiency. We adapt Generalized Reinforcement Policy Optimization (GRPO) to this setting, providing stable policy-gradient updates from scale-invariant advantages. Across a wide range of benchmark algorithms and problem instances of varying sizes, our method frequently achieves the best performance and consistently avoids catastrophic failures.

1 INTRODUCTION

Black-box optimization (Audet & Kokkolaras, 2016; Brochu et al., 2010) consists of maximizing a function $f : \mathcal{X} \to \mathbb{R}$ over the discrete space $\mathcal{X}$ without any structural or analytical knowledge of f. The function f is typically costly to evaluate (e.g., computationally expensive simulation, querying a physical experiment, or executing a complex algorithm). The interactions among the variables of f are not available, making black-box optimization particularly challenging, especially in high-dimensional and structured discrete domains (Doerr et al., 2019; Larranaga, 2002).

A wide range of methods and concepts have been explored to solve Black-box optimization problems. Among them, Bayesian optimization (BO) is a model-based optimization framework that constructs a probabilistic surrogate model over the objective function and uses an acquisition function to determine where to sample next in the search space. It is particularly effective for global optimization under tight evaluation budgets, making it well-suited for expensive black-box problems (Forrester & Keane, 2009; Frazier, 2018; Shahriari et al., 2015). Evolutionary Algorithms (EAs) are also recognized as powerful methods for solving discrete black-box optimization problems. These metaheuristics operate by iteratively evolving a population of candidate solutions through variation operators (mutation, crossover) and selection mechanisms. Unlike Bayesian optimization, EAs do not build explicit models of the objective function, making them more flexible and easier to implement (Back, 1996; Eiben & Smith, 2015).

As a specific subclass of EAs, Estimation-of-Distribution Algorithms (EDAs) are stochastic black-box optimization methods that guide the search for optima by explicitly learning and sampling from a probabilistic model P of promising candidate solutions by means of a distribution that captures patterns among high-performing solutions (Larranaga, 2002; Mühlenbein & Paass, 1996). EDAs can be conceptually positioned between the two main paradigms of black-box optimization, EAs and BO. Some widely used and effective EDAs such as the Covariance Matrix Adaptation Evolution Strategy (`CMA-ES`) (Hansen & Ostermeier, 2001; Hansen, 2016)—designed for continuous landscapes—and Population-Based Incremental Learning (`PBIL`) (Baluja, 1994)—for discrete landscapes—can also be interpreted within the Information-Geometric Optimization (IGO) framework (Ollivier et al., 2017). This connection provides a formal interpretation of EDAs as performing natural gradient descent in the space of probability distributions, thus explaining their ability to finetune solutions and converge reliably in continuous or discrete spaces. While continuous EDAs—particularly `CMA-ES`—have attracted significant attention, a less explored body of research focuses on EDAs for discrete and combinatorial spaces. Early work in this area has demonstrated the effectiveness of multivariate discrete EDAs in applications such as scheduling, routing, and constraint satisfaction problems (Lozano, 2006). Algorithms such as Mutual Information Maximizing Input Clustering (`MIMIC`) (De Bonet et al., 1996) and Bayesian Optimization Algorithm (`BOA`) (Pelikan, 2005) model dependencies between variables using directed acyclic graphs, enabling them to learn the structure of the search space and capture conditional dependencies among decision variables.

In this paper, we revisit discrete multivariate EDAs by using a multivariate distribution parameterized by neural networks to model the distribution of each variable conditionally on the others. The resulting highly flexible model is capable of capturing complex interactions between variables while controlling the total number of parameters in the joint generative distribution, which scales polynomially with instance size. A neural network is associated with each variable and trained in parallel using modern reinforcement learning techniques—based on policy gradients such as Generalized Reinforcement Policy Optimization (GRPO) (Shao et al., 2024)—which have proven highly successful in rapidly converging on effective policies, especially when discrete action choices must be made in complex environments. The solution generation process is modeled as a sequential assignment of variable values. Inspired by recent work (Pannatier et al., 2024)—which proposes permutation-invariant autoregressive generation to mitigate exposure bias and increase robustness—and in contrast to classical EDAs such as `MIMIC` and `BOA`, which rely on an explicitly learned generation order, we adopt a more agnostic stance. Rather than assuming or learning a sparse directed acyclic graph, which may not reflect the true underlying structure of complex combinatorial problems, we advocate for a multivariate undirected generative model that is invariant to the order of variable generation. While previous works (Kim et al., 2022; Kwon et al., 2020) use RL-based construction methods for optimization problems where symmetries occur in the solution or problem spaces (typically permutations), we focus on the order-invariance of the generation process. Furthermore, we show that learning the model with random orders corresponds to a form of structural dropout (Pal et al., 2020) inspired by recent advances in permutation-invariant modeling and conditional masking in generative neural networks (Uria et al., 2016), where random subsets of the context are provided during training. This technique enables each variable to depend on varying combinations of others, allowing the model to flexibly learn interactions without committing to a fixed generation path. We experimentally show that the resulting model is more robust to structural uncertainty and better suited to complex, high-dimensional combinatorial search spaces. In our approach, the critical NP-hard combinatorial optimization problem at the core of graph learning used in Bayesian multivariate EDAs (like `BOA`) is replaced by a single continuous optimization problem.

The remainder of this paper is organized as follows. Section 2 introduces the discrete black-box optimization problem, reviews related work and discuss the motivations for this work. Section 3 presents the derivation of our proposed RL-EDA approach, which builds on a GRPO RL backbone and is designed to tackle this class of problems. Section 4 reports empirical results comparing our algorithm with state-of-the-art methods. Various versions of the approach are also compared to analyze the benefits of each of its components. Section 5 discusses the contribution and presents some perspectives for future work.

2 Preliminaries: Problem setting, Related work and Motivations

In this section, we first formally introduce the discrete black-box optimization problem. We then review existing work on multivariate EDAs proposed to tackle such problems. Finally, we discuss the opportunities offered by neural generators in this context, particularly regarding their flexibility in capturing implicit inter-variable dependencies. We also highlight the potential benefits of leveraging random variable orderings for both generation and training under stringent sample-efficiency constraints within the EDA training regime.

2.1 Discrete Black-box Optimization

Let $\mathcal{X} = \mathcal{X}_1 \times \cdots \times \mathcal{X}_n$ be the discrete search space of size n, where each $\mathcal{X}_j$ is a finite set (binary or categorical), and let $f : \mathcal{X} \to \mathbb{R}$ be an objective function accessible only as a black box, i.e., without any structural information (such as convexity or smoothness). A combinatorial optimization (CO) problem is then defined by the pair $(\mathcal{X}, f)$. Without loss of generality, the task is to maximize f: $\max_{x\in\mathcal{X}} f(x)$. In the following, $x = (x_1, \dots, x_n) \in \mathcal{X}$ denotes a candidate solution (not necessarily the best) of the CO problem. X_i denotes the variable associated to $\mathcal{X}_i$, whose value in $\mathcal{X}_i$ is x_i. Various existing solving techniques for black-box CO include Bayesian optimization methods and metaheuristics (local-search-based and population-based approaches), which have been improved by machine learning techniques (Talbi, 2021). More related work on combinatorial optimization is given in Appendix A.

2.2 Multivariate Estimation of Distribution Algorithms

Multivariate EDAs are evolutionary algorithms that solve a CO problem by iteratively building and updating a probabilistic model over the search space $\mathcal{X}$. An EDA with parameters $(\mu, \lambda) \in \mathbb{N}^2$ with $0 < \mu < \lambda$ performs the following steps at each iteration t:

1. Draw a population of λ candidate solutions $x^1, \dots, x^\lambda$ from the model P_t and compute fitness values $f^i = f(x^i)$, for $i = 1, \dots, \lambda$.
2. Select the μ best individuals $\mathcal{S}_t = \{x^{r_i} : i \in [1..\mu]\}$, where $(r_1, \dots, r_\lambda)$ is a permutation of $[1..\lambda]$ such that $f^{r_1} \geq \dots \geq f^{r_\lambda}$, and use $\mathcal{S}_t$ to estimate the updated probabilistic model P_{t+1}.

Following this framework, EDAs mainly differ in how they model the generative distribution P_t used to sample new candidate solutions at each generation t. Some approaches, such as PBIL (Baluja, 1994) or UMDA (Mühlenbein & Paass, 1996), approximate P_t as a product of independent univariate distributions: $P_t(x) = \prod_{i=1}^n P_t^i(X_i = x_i)$, where P_t^i denotes the i-th marginal distribution. While such approaches have proved effective on problems with little or no interaction among variables, they suffer from important limitations: they can at best focus on a single mode of the distribution, fail to capture complex inter-variable relationships (including combinatorial or logical dependencies), and are prone to premature convergence or loss of diversity in multimodal landscapes.

To overcome these limitations, classical multivariate EDAs need to employ more expressive probabilistic models that explicitly capture dependencies between variables from best candidates in S_t at each generation t. In the case of Bayesian networks, dependencies are represented by a directed acyclic graph (DAG) $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, whose set of vertices $\mathcal{V}$ contains all the variables X_j for $j = 1, \dots, n$ and whose directed edges $\mathcal{E}$ represent causality relationships. Hence, at any iteration t of the EDA process, the joint density $P_t(x)$ can be factorized as the product of the densities of each variable conditionally on its parents as $P_t(x) = \prod_{j=1}^n P_t(X_j = x_j | X_{\text{Pa}(j;\mathcal{G}_t)} = x_{\text{Pa}(j;\mathcal{G}_t)})$ (Markov factorization) with $\mathcal{G}_t = (\mathcal{V}, \mathcal{E}_t)$ the considered DAG at iteration, $X_{\text{Pa}(j;\mathcal{G}_t)} = \{X_i \in \mathcal{V} : (X_i, X_j) \in \mathcal{E}_t\}$ the set of the parents of the variable X_j in $\mathcal{G}_t$ and $x_{\text{Pa}(j;\mathcal{G}_t)}$ their corresponding values.

Given a DAG $\mathcal{G}_t$, such a factorization allows to significantly reduce the number of required parameters to approximate P_t. It also permits sampling the variables sequentially according to a topological ordering consistent with the causal dependencies encoded by the graph. However, optimal DAGs are usually unknown at the beginning of the process, and need to be learned efficiently from selected candidates S_t at each generation, together with the parameters of each factor of the Markov factorization (more details on EDAs with DAGs can be found in Appendix A).

2.3 The Case on Neural Estimators

Traditionally, EDAs based on Bayesian networks estimate each component of the Markov factorization by contingency tables reporting counts of all joint realizations of the dependent variables together with the combinations of its parents' values. In this setting, restricting the dependencies of each outcome to a small subset of causal variables is crucial to avoid the exponential growth of complexity with the problem dimension. This limitation has motivated a long line of research on structural learning heuristics, pruning strategies, and regularization techniques designed to control the combinatorial explosion (Echegoyen et al., 2008).

Neural estimators fundamentally alter this picture. In classical EDAs, learning an explicit dependency graph was unavoidable: the sampling model could only be specified once the graph structure had been identified. Neural approaches dispense with this requirement. By parameterizing the joint distribution directly—often through autoregressive factorizations with arbitrary variable orderings (Germain et al., 2015; Uria et al., 2016), or via invertible transformations in flow-based models (Papamakarios et al., 2021)—they sidestep the need to commit to a learned structure at all. However, despite their success in density estimation and generative modeling, such neural approaches have scarcely been explored in the context of multivariate EDAs. To the best of our knowledge, no prior work has applied autoregressive to EDA, nor investigated their interaction with the iterative optimization dynamics. This gap motivates our study.

In practice, fitting a flexible neural density estimator is frequently simpler and more robust than inferring the "correct" graph, especially under the limited and evolving sample regimes typical of EDAs. Following an autoregressive model, we can consider any given factorization using any order of variables. That is, given an arbitrary order σ of the dimensions of the problem, we can write $P(X = x) = \prod_{i=1}^{n} P(x_{\sigma_i}|x_{\sigma<i})$, where x_{σ_i} stands as the value of the i-th dimension of x in the permutation σ and $x_{\sigma<i}$ corresponds to the sequence of values of x with rank lower than i in permutation σ (with $x_{\sigma_{<1}}$ standing as an empty sequence). Given N samples of P, this can be estimated by a neural network P_θ, with parameters θ obtained via maximum likelihood estimation (MLE): $\arg\max_{\theta\in\Theta} \frac{1}{N}\sum_{j=1}^{N}\prod_{i=1}^{n} P_\theta(x^j_{\sigma_i}|x^j_{\sigma<i})$, where $x^1 \dots x^N$ are sampled from the target distribution P. We note that this is true for any given permutation σ. In particular, assuming infinite amounts of data and infinite capacity of the used neural networks, at convergence of the MLE, we get that: $\forall \sigma, \sigma' : P_\theta(X|\sigma) = P_{\theta'}(X|\sigma')$, where θ and θ' are optimal parameters (according to MLE) for permutation σ and σ' respectively. NADE (Uria et al., 2016) exploits this idea by defining ensembles of models, each associated with a different variable ordering, which enables sampling from a more diverse set of outcomes. Yet, to the best of our knowledge, such permutation-based ensembles have never been explored in multivariate EDAs, despite population diversity being a key ingredient for black-box optimization and effective exploration. Beyond sampling, we argue that training a single model across multiple orderings provides an additional benefit: it acts as a form of noise reduction when learning from limited data, as is typically the case in online EDAs. In Appendix E, we show that this mechanism can be interpreted as an information-preserving analogue of dropout, allowing the model to efficiently identify the dominant dependencies between variables while mitigating overfitting to transient fluctuations.

3 Multivariate EDA With Order-Invariant Reinforcement Learning

Our proposed algorithm for discrete black-box problems is a multivariate EDA (see Section 2.2) whose probabilistic model is encoded with a set of neural networks. The construction of a solution of the CO problem is seen as an episodic Markov Decision Process (MDP) with a reinforcement learning algorithm adapted for our setting.

3.1 Deep Reinforcement Learning for EDAs: Setting and Architectures

The EDA framework presented above can be easily casted as a reinforcement learning problem, defined on an MDP $\mathcal{M} = (\mathcal{S}, \mathcal{A}, P, R)$ where $\mathcal{S}$ is a set of states, $\mathcal{A}$ a set of actions, $P(s'|s,a)$ is the transition probability function, $R : \mathcal{S} \to \mathbb{R}$ is the reward function, that assigns a scalar reward depending on reached states in $\mathcal{S}$. In the setting of multivariate EDAs, $\mathcal{S}$ corresponds to incomplete solutions from $\mathcal{X}$ (i.e. $S \equiv \{(\varnothing, 0, \sigma) : \sigma \in \Omega\} \cup \{((x_{\sigma_1} \dots x_{\sigma_k}), k, \sigma) : x \in \mathcal{X}, \sigma \in \Omega, k \in [[1, n]]\}$), with Ω the set of all possible generation orders of a sequence of indices $1 \dots n$, and $\varnothing$ an empty sequence that defines starting states s_0. For a given state $s_k = (x_{\sigma_{\leq k}}, k, \sigma)$, the set of possible actions $\mathcal{A}_k \subseteq \mathcal{A}$ is the domain of the $k+1$-th variable of the permutation σ (i.e., $\mathcal{A}_k \equiv \mathcal{X}_{\sigma_{k+1}}$). Thus, transitions are deterministic: for any triplet (s, a, s'), with $s = (x_{\sigma_{\leq k}}, k, \sigma)$ and $a \in \mathcal{X}_{\sigma_{k+1}}$, $P(s'|s,a)$ is 1 iff $s' = (x'_{\sigma_{\leq k+1}}, k+1, \sigma)$ with $x'_{\sigma_{\leq k}} = x_{\sigma_{\leq k}}$ and $x'_{\sigma_{k+1}} = a$. Finally, rewards are non-zeros for states from $\mathcal{S}$ that correspond to complete solutions of the problem only (i.e., those states that contain full instantiation of $\mathcal{X}$).

In that setting, our goal is to optimize a parameterized stochastic generative policy $\pi_\theta(a_k \in \mathcal{X}_{\sigma_{k+1}} | s_k = (x_{\sigma_{\leq k}}, k, \sigma))$, that defines the probability of taking action a_k in state s_k. For the binary setting where the discrete search space is $\mathcal{X} = \{-1, 1\}^n$, we model this generative policy as a neural logistic regressor as $\pi_\theta(a_k = 1 | s_k = (x_{\sigma_{\leq k}}, k, \sigma)) = \text{sigmoid}(g_{\theta_{dim_\sigma(k)}}(x_{\sigma_{\leq k}}))$, with g_{θ_i} a neural network with parameter $\theta_i \in \mathbb{R}^m$ and $dim_\sigma(k)$ the bijective function that returns the index of the dimension at rank k in permutation σ. An example of generation of solutions with this generative policy is displayed in Figure 1 on the left. For categorical domains $\mathcal{X}_i$, we encode each of their d categories as a one hot vector where $X_{i,j} = 1$ iff the represented category is $j \in [[1, d]]$, -1 otherwise. For these outputs, we consider a softmax over the logits produced by g to produce the corresponding categorical distribution.

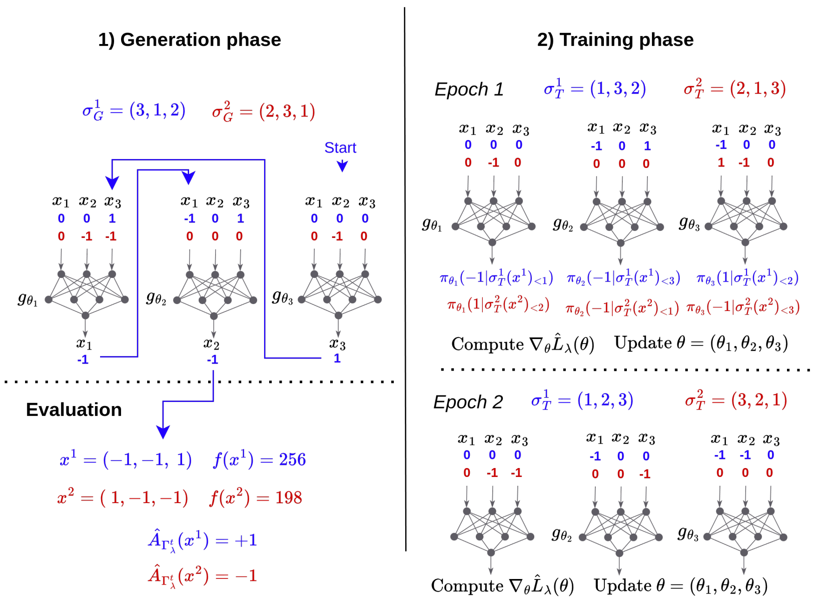

1) Generation phase
$\sigma_G^1 = (3,1,2)$
$\sigma_G^2 = (2,3,1)$
Start
x_1 x_2 x_3
g_{θ_1}
g_{θ_2}
g_{θ_3}
Evaluation
$x^1 = (-1,-1,1)$ $f(x^1) = 256$
$x^2 = (1,-1,-1)$ $f(x^2) = 198$
$\hat{A}_{\Gamma_\lambda^t}(x^1) = +1$
$\hat{A}_{\Gamma_\lambda^t}(x^2) = -1$
2) Training phase
Epoch 1
$\sigma_T^1 = (1,3,2)$
$\sigma_T^2 = (2,1,3)$
$\pi_{\theta_1}(-1|\sigma_T^1(x^1)_{<1})$ $\pi_{\theta_2}(-1|\sigma_T^1(x^1)_{<3})$ $\pi_{\theta_3}(1|\sigma_T^1(x^1)_{<2})$
$\pi_{\theta_1}(1|\sigma_T^2(x^2)_{<2})$ $\pi_{\theta_2}(-1|\sigma_T^2(x^2)_{<1})$ $\pi_{\theta_3}(-1|\sigma_T^2(x^2)_{<3})$
Compute $\nabla_\theta \hat{L}_\lambda(\theta)$ Update $\theta = (\theta_1, \theta_2, \theta_3)$
Epoch 2
$\sigma_T^1 = (1,2,3)$
$\sigma_T^2 = (3,2,1)$
Compute $\nabla_\theta \hat{L}_\lambda(\theta)$ Update $\theta = (\theta_1, \theta_2, \theta_3)$

Figure 1: **Left.** Example of generation at time t of a population Γ_λ^t with $\lambda = 2$ individuals for a maximization problem with $N = 3$. The order of generation of the first individual is indicated with blue arrows. When building it with the MDP and given order $\sigma_G^1 = (3, 1, 2)$, we start with $x_{\sigma_G^1 < 1} = (0, 0, 0)$ given as input to the neural network g_{θ_3} that generates $x_3 = 1$, then $x^1_{\sigma_G^1 < 2} = (0, 0, 1)$ is given as input to g_{θ_1} that generates $x_1 = -1$, and lastly $x^1_{\sigma_G^1 < 3} = (-1, 0, 1)$ is given as input to g_{θ_2} that generates the value x_2 of the last variable and we get the complete solution $(-1, -1, 1)$. When all the individuals of the population are sampled, we pass to the evaluation phase where advantages are computed such that $A_{\Gamma_\lambda^t}(x^i_{best}) = +1$ and $A_{\Gamma_\lambda^t}(x^i_{worse}) = -1$ (see Eq. (5)). **Right.** Training phase during E epochs with the $\lambda = 2$ solutions sampled at this generation. At each epoch, new σ_G orders are sampled for each individual. Conditional probabilities of actions are then computed according to the corresponding causal masks. It allows to compute $\hat{L}_\lambda(\theta)$ (Eq. (8) and to update $\theta = (\theta_1, \theta_2, \theta_3)$ by gradient ascent.

Rather than dealing with neural models specifically dedicated for sequences, such as recurrent networks or Transformers (which are better suited for non structured inputs), we propose to define g as a classical MLP, parameterized with a different set of parameters for each individual output of the problem. For any order of generation σ and any step k, we want to feed g with a fixed-size vector as input. For a given step k of a permutation σ, this is done by modeling the input $x_{\sigma_{<k}}$ as a vector of size n, where each dimension $x_i = 0$ (resp. x_i is a zero vector for the categorical domains) iff $rank_\sigma(i) = dim_\sigma^{-1}(i) \geq k$. During training, this comes down to applying a causal mask to candidate solutions, that masks future of k in permutation σ. Note that, while $\theta \in \mathbb{R}^{n \times m}$ in our architecture, our work can easily be extended by sharing parameters of hidden layers for scaling to very large problems without facing prohibitive training costs, as described in Appendix Q.

3.2 Deep Reinforcement Learning for EDAs: Training

Given the setting stated above, the optimization seeks to maximize the expected global reward over trajectories $\tau = (s_0, a_0, \ldots, s_{n-1}, a_{n-1}, s_n)$: $J(\theta) = \mathbb{E}_{\tau \sim \pi_\theta}[R(\tau)]$, where $R(\tau)$ in our setting corresponds the fitness $f(x)$ computed for the full candidate $x \in \mathcal{X}$ contained in the last state of τ (i.e., $R((s_0, a_0, \ldots, s_n)) = f(x)$, iff $s_n = (x, n, \sigma)$). For a given σ, this is thus equivalent to maximizing $J^\sigma(\theta) = \mathbb{E}_{x \sim \pi_\theta(x|\sigma)}[f(x)]$, where $\pi_\theta(x|\sigma)$ stands for the probability of sampling x as a sequence $x = (x_{\sigma_1}, \ldots, x_{\sigma_n})$ using our generative architecture.[1] Following the policy gradient theorem (Sutton et al., 2000), we get that parameters θ can be obtained using gradient updates defined as

$$\nabla_\theta J^\sigma(\theta) = \mathbb{E}_{x \sim \pi_\theta(x|\sigma)}[f(x) \sum_{k=1}^{n} \nabla_\theta \log \pi_\theta(x_{\sigma_k} | x_{\sigma_{<k}}, \sigma)]. \tag{1}$$

This formulation allows us to sample candidate solutions of the problem from the current distribution $\pi_\theta(x|\sigma)$ (which corresponds to $P_t(x)$ in the EDA framework described in Section 2.2), and then estimate an update of the generative distribution by computing a weighted average of gradients of $\log \pi_\theta(x|\sigma)$, with weights depending on the respective fitness of sampled x (which is the analogue of step 2 from the EDA framework in Section 2.2). However, from updates defined in (1), each sample x can be used for a unique gradient step only, which can reveal as very sample inefficient. Moreover, updates of the policy are strongly dependent on its parametrization, which can lead to hazardous moves that induce catastrophic forgetting when using such neural generators. To improve sample efficiency and stabilize training, the Proximal Policy Optimization (PPO) algorithm (Schulman et al., 2017), following TRPO (Schulman et al., 2015a), optimizes a surrogate objective function that penalizes deviations from a reference policy $\pi_{\theta_{\text{old}}}$, used for sampling, that will be denoted π_{θ^t} at generation t of our EDA. In our setting, the policy gradient update in (1) can be rewritten using importance sampling as an expectation under π_{θ^t}. Approximating the state distribution d^{π_θ} by $d^{\pi_{\theta^t}}$, we obtain (see appendix B for details)

$$\nabla_\theta J^\sigma(\theta) \quad \approx \quad \mathbb{E}_{\pi_{\theta^t}(x|\sigma)} \sum_{k=1}^{n} \frac{\nabla_\theta \pi_\theta(x_{\sigma_k}|x_{\sigma_{<k}}, \sigma)}{\pi_{\theta^t}(x_{\sigma_k}|x_{\sigma_{<k}}, \sigma)} A^{\pi_{\theta^t}}(x_{\sigma_{<k}}, x_{\sigma_k}), \tag{2}$$

where $A^{\pi_{\theta^t}}(x_{\sigma_{<k}}, x_{\sigma_k})$ denotes the expected advantage of setting $X_{\sigma_k} = x_{\sigma_k}$ given $x_{\sigma_{<k}}$, while completing the trajectory with the reference policy. This formulation allows multiple gradient steps for updating the policy (i.e., for obtaining P_{t+1}), given samples obtained using the policy (representing P_t) from the previous iteration t of our EDA RL framework. However, the approximation in (2) (the choice of the KL version of PPO is discussed in section F), which should be understood at the level of expected gradients, introduces an acceptable bias only when π_θ and π_{θ^t} are close (e.g., in KL divergence). Thus, following the KL version of PPO, we consider the maximization of the regularized objective:

$$L^\sigma(\theta) = \mathop{\mathbb{E}}_{\pi_{\theta^t}(x|\sigma)} \sum_{k=1}^{n} \left[\frac{\pi_\theta(x_{\sigma_k}|x_{\sigma_{<k}}, \sigma)}{\pi_{\theta^t}(x_{\sigma_k}|x_{\sigma_{<k}}, \sigma)} A^{\pi_{\theta^t}}(x_{\sigma_{<k}}, x_{\sigma_k}) - \beta D_{\text{KL}}\left(\pi_{\theta^t}(\cdot|x_{\sigma_{<k}}, \sigma) \,\|\, \pi_\theta(\cdot|x_{\sigma_{<k}}, \sigma)\right)\right] \tag{3}$$

where $D_{\text{KL}}(\pi||\pi')$ stands for the Kullback-Leibler (KL) divergence of π from π', and $\beta > 0$ is an adaptive penalty coefficient that controls the strength of the KL regularization. While PPO classically uses critic neural networks to estimate advantages (e.g., using GAE (Schulman et al., 2015b)), we rather take inspiration from the GRPO approach (Shao et al., 2024), specifically dedicated for RL problems with global rewards from finite trajectories without discount, which avoids the need for a critic, by estimating scale-invariant advantages using a normalization of rewards obtained on a population of samples for a same problem.[2] Scale-invariance is particularly desirable in black-box optimization settings, as it enhances robustness to the scaling of objective values (Baluja, 1994; Doerr & Dufay, 2022; Goudet et al., 2025). Given a set of λ candidate solutions $\Gamma_\lambda^t = \{x^i\}_{i=1}^{\lambda}$, each

[1] In the following of this section we consider a fixed arbitrary order σ for every state of the MDP. Using random variations of σ is the subject of the next section.

[2] We compare a baseline that uses a critic to estimate advantages with our GRPO approach in Appendix P.

sampled from $\pi_{\theta^t}(x|\sigma)$, we thus consider at each iteration t of the process the maximization of

$$\hat{L}^{\sigma}_{\lambda}(\theta) = \frac{1}{\lambda} \sum_{x^i \in \Gamma^t_\lambda} \sum_{k=1}^{n} \left[\frac{\pi_\theta(x^i_{\sigma_k}|x^i_{\sigma_{<k}}, \sigma)}{\pi_{\theta^t}(x^i_{\sigma_k}|x^i_{\sigma_{<k}}, \sigma)} \hat{A}_{\Gamma^t_\lambda}(x^i) - \beta D_{\mathrm{KL}}\left(\pi_{\theta^t}(\cdot|x^i_{\sigma_{<k}}, \sigma) \,\|\, \pi_\theta(\cdot|x^i_{\sigma_{<k}}, \sigma)\right) \right], \tag{4}$$

where $A_{\Gamma^t_\lambda}(x)$ is the relative performance of candidate x compared to other solutions from Γ^t_λ. In this paper, we consider advantages computed as

$$A_{\Gamma^t_\lambda}(x) = U\left(\frac{\mathrm{rk}(x, \Gamma^t_\lambda, f)}{\lambda - 1}\right), \tag{5}$$

where U is a non-increasing utility function and $\mathrm{rk}(x^i, \Gamma^t_\lambda, f)$ is the rank of the individual i in the population Γ^t_λ given its fitness $f(x^i)$. Formally, $\mathrm{rk}(x, \Gamma, f) = |\{x' \in \Gamma : f(x') > f(x)\}|$. This advantage formulation, which makes the algorithm invariant under monotone transformation of the fitness function f, is grounded in the Information-Geometric Optimization (IGO) framework (Ollivier et al., 2017). We discuss the connexion of our approach with IGO in Appendix G.

3.3 Order invariant reinforcement learning for EDAs

In the previous section, we introduced a multivariate-RL-EDA, that uses a predetermined arbitrary generation order σ. The aim of this section is to adapt this algorithm for dealing with variations of this generation order, which we claim can strongly benefit for exploration and learning in our black-box optimization setting.

Given a generation order distribution $\xi(\sigma)$, we can consider the expectation $L(\theta) = \mathbb{E}_{\sigma \sim \xi(\sigma)} L^\sigma(\theta)$ in place of using $L^\sigma(\theta)$ with a fixed known order σ. Let for convenience of the following $\sigma(x)_{<k}$ denote a masking (i.e., removing) of any dimension from x whose rank in permutation σ is greater or equal than the one of dimension k (i.e., $\forall i \in [[1, n]], X_i \in \sigma(X)_{<k} \iff rank_\sigma(i) < rank_\sigma(k)$). Using this, we can rewrite the objective (3), as

$$L(\theta) = \mathbb{E}_{\sigma \sim \xi(\sigma)} \mathbb{E}_{\pi_{\theta^t}(x|\sigma)} \sum_{k=1}^{n} \left[\frac{\pi_\theta(x_k|\sigma(x)_{<k})}{\pi_{\theta^t}(x_k|\sigma(x)_{<k})} A^{\pi_{\theta^t}}(\sigma(x)_{<k}, x_k) \right.$$
$$\left. - \beta D_{\mathrm{KL}}\left(\pi_{\theta^t}(\cdot|\sigma(x)_{<k}) \,\|\, \pi_\theta(\cdot|\sigma(x)_{<k})\right) \right]. \tag{6}$$

A notable difference in this writing compared to previous ones is that the inner sum from $k = 1$ to n is taken in the original dimension ordering of the problem, rather than in the generation order. While fully equivalent, this formulation allows us to introduce a second source of variation, specifically dedicated for incentivizing order-invariance of the policy. Let $\xi(\sigma_T|\sigma_G)$ be a conditional distribution that samples a transformation $\sigma_T \in \Omega$ of a given initial permutation $\sigma_G \in \Omega$. We propose to use this transformed permutation σ_T to train the new policy π_θ, given samples from the old policy using the former permutation σ_G used for generation. We get (derivation detailed in section C)

$$L(\theta) = \mathbb{E}_{\substack{\sigma_G \sim \xi(.),\\ \sigma_T \sim \xi(.|\sigma_G)}} \mathbb{E}_{\pi_{\theta^t}(x|\sigma_G)} \sum_{k=1}^{n} \left[\frac{\pi_\theta(x_k|\sigma_T(x)_{<k})}{\pi_{\theta^t}(x_k|\sigma_G(x)_{<k})} A^{\pi_{\theta^t}}(\sigma_G(x)_{<k}, x_k) \right.$$
$$\left. - \beta D_{\mathrm{KL}}\left(\pi_{\theta^t}(\cdot|\sigma_G(x)_{<k}) \,\|\, \pi_\theta(\cdot|\sigma_T(x)_{<k})\right) \right]. \tag{7}$$

As in previous section, we finally consider a Monte-Carlo approximation of this quantity at each iteration, using scale normalized global advantages, given a set of λ i.i.d. candidate solutions associated with their own order of generation $\Gamma^t_\lambda = \{(x^i, \sigma^i_G)\}^\lambda_{i=1}$. For each component i in this set, an order σ^i_G is first sampled from ξ, then x^i is sampled from $\pi_{\theta^t}(.|\sigma_G)$. We get:

$$\hat{L}_\lambda(\theta) = \frac{1}{\lambda} \sum_{(x^i, \sigma^i_G) \in \Gamma^t_\lambda} \mathbb{E}_{\sigma_T \sim \xi(.|\sigma^i_G)} \sum_{k=1}^{n} \left[\frac{\pi_\theta(x^i_k|\sigma_T(x^i)_{<k})}{\pi_{\theta^t}(x^i_k|\sigma^i_G(x^i)_{<k})} \hat{A}_{\Gamma^t_\lambda}(x) \right.$$
$$\left. - \beta D_{\mathrm{KL}}\left(\pi_{\theta^t}(\cdot|\sigma^i_G(x^i)_{<k}) \,\|\, \pi_\theta(\cdot|\sigma_T(x^i)_{<k})\right) \right]. \tag{8}$$

This formulation allows us to experiment various versions of our training process, which we name as:

- (δ, δ')-RL-EDA: uses a fixed arbitrary order for both generation and training (i.e., ξ and $\xi(.|\sigma)$ are both Diracs centered on the original order σ of the problem);
- (δ, σ')-RL-EDA: uses a fixed arbitrary order for generation, but for training $\xi(.|\sigma_G)$ is a uniform distribution;
- (σ, δ')-RL-EDA: uses an identical random order σ_G for both generation and training, with ξ an uniform distribution over Ω and $\xi(.|\sigma_G)$ is a Dirac centered on σ_G.
- (σ, σ')-RL-EDA: uses two sources of noises in the training process. Both the generation order σ_G and the training order σ_T are sampled from a uniform distribution over Ω.

The pseudo-code of our full algorithm, which includes these permutation noises for training, is given in Appendix H (Algorithm 1), and an example of training with these permutation noises is displayed in Figure 1 on the right.

Note that considering varying causal graphs is also possible in this framework, by simply using masks $\sigma(x)_{<k}$ that hide values of non parent variables of x_k in x, in addition to every dimension whose rank in σ is greater or equal than k. We experiment with this structural dropout as a complement or replacement for causal masks for the different versions of the multivariate EDA in Appendices M.1 and M.2. For complementary analysis, we also describe in Appendix I a version called Learned-σ-RL-EDA which uses a Plackett-Luce (PL) distribution (Plackett, 1975) ξ_w^{PL} parametrized by the vector $w \in \mathbb{R}^n$ for both generation and training, trained by gradient descent with the reparametrization trick proposed by (Grover et al., 2019).

4 Experiments

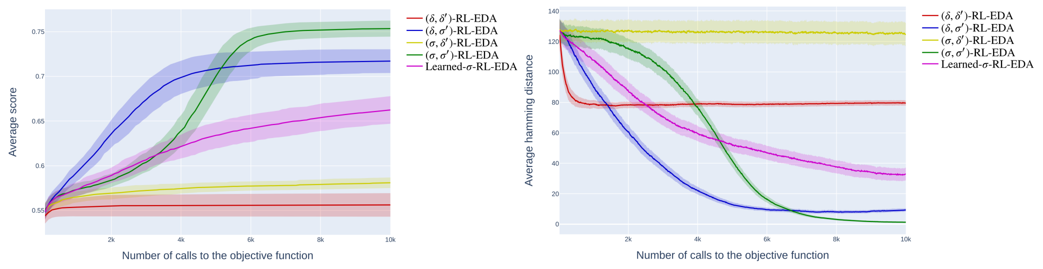
We first examine the following NP-hard problems in this work (seen as black-box CO): the Quadratic unconstrained binary optimization problem (QUBO) (Kochenberger et al., 2014), the pseudo-boolean NK landscape problem (Kauffman & Weinberger, 1989) and its extension with ternary variables called NK3. For each of these problems pb, we generated instances of size $n \in \{64, 128, 256\}$, and for each size, we considered different types K of instances. We generate 10 instances for each tuple (pb, n, K). For each problem instance, we allow a maximum budget of 10,000 objective function evaluations, and we solve it with 10 different restarts. Details regarding the instances and experimental protocol are provided in Appendix J.

4.1 Comparison of the different versions of reinforcement learning multivariate EDA

In this section, we first aim to compare the five different versions of multivariate-RL-EDA presented in Section 3.3: (δ, δ')-RL-EDA, (δ, σ')-RL-EDA, (σ, δ')-RL-EDA, (σ, σ')-RL-EDA and Learned-σ-RL-EDA. The complete hyperparameter configuration of the various versions of the multivariate-RL-EDA, which serves as a baseline for all experiments, is provided in Appendix K. It includes both EDA-specific and GRPO-related parameters, along with implementation and execution details relevant to reproducibility. Here we perform this comparison only for the distribution of instances of the pseudo-boolean NK maximization problem with $N = 256$ and $K = 4$ (moderate roughness). The results displayed here are representative of what we can obtain on the other distributions of instances.

Figure 2a shows the evolution curve of average scores over 100 independent runs for the four different versions (solide lines). The ranges of color around the solid lines correspond to plus or minus one standard deviation from the mean calculated over the 100 runs. Solid lines in Figure 2b corresponds to the evolution of the mean Hamming distance of the individuals of the population from the best solution found during the trajectory. The color range represents the standard deviation of the Hamming distance calculated within the population at each generation, with one standard deviation below and one standard deviation above the average distance. The evolutions of the Mean Hamming distance and standard deviation are averaged over the 100 independent runs.

The different multivariate versions of our EDA exhibit very different behavioral dynamics, even though they are characterized by the same hyperparameters, with the exception of changing sampling distributions of orders, which shows their importance during the sampling and update phases for such a multivariate RL algorithm.



(a) Evolution of the scores (b) Evolution of the distances

Figure 2: X-axis: number of calls to the objective function. Y-axis: Evolution of average scores (a) and average distances (b) obtained by the different variants of multivariate RL EDA for 100 independent runs on instances of the NK problem with $N = 256$ and $K = 4$.

The version (σ, σ')-RL-EDA that uses both uniform distributions of orders for sampling and training converges towards the best scores (green curve). Once the maximum is reached, we see in Figure 2b that the algorithm has converged because the average distance from the best solution encountered on the trajectory is close to 0. The comparison of this green curve with the blue curve of the (δ, σ')-RL-EDA version highlights the contribution of sampling new orders during the EDA generation phase, because it allows to maintain a better diversity of the individuals of the population at each generation and thus allows a better exploration of the search space. It works like an ensembling method where actually different models are used at each generation to produce new solutions. But the main impact is explained when comparing the green curve with the yellow curve of the (σ, δ')-RL-EDA version. It highlights the contribution of sampling new orders during the EDA training phase, which underscores the importance of the specific structural dropout at the input of each network induced by this random sampling of orders. Finally, the purple curves correspond to the version using a learned Placket-Luce distribution of order with a vector w of distribution weights initialized with only ones. The purple curves also show a good evolution of the scores, but the model did not converge with the allocated budget, and the scores are worse than those obtained with the (σ, σ')-RL-EDA version (green curve). This experiment confirms that attempting to extract explicit structures in such an online search process is counterproductive when using neural estimators (at least without additional knowledge about the instance properties), since learning them is at least as difficult as learning neural weights from random orderings, taking advantage of the networks' plasticity to adapt to any ordering. Instead, random resampling of new orderings for both generation and training plays a key role in discovering high-quality solutions, as it promotes exploration and enables a more effective identification of interactions between variables.

4.2 Experimental Validation on Discrete Black-Box Benchmarks

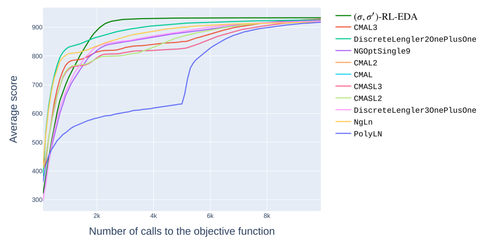
We evaluate the performance of our best version (σ, σ')`-RL-EDA` dentified in the last section against a comprehensive set of 504 algorithms, essentially composed of those available in the `Nevergrad` library (Rapin & Teytaud, 2018).

In version 1.0.12 of the `Nevergrad` library, a total of 542 algorithms were available. We evaluated all of them on the discrete black-box problem QUBO, NK and NK3, with a time budget of one hour per instance. Among these, 500 algorithms successfully produced solutions within the given time limit for pseudo-Boolean problems and 496 for the categorical NK3 problem. This panel includes classic metaheuristic algorithms for black-box optimization (evolutionary and memetic) as well as combinations of solving techniques driven by machine learning (e.g. Adaptive Portfolios). A complete description is provided in Appendix S. In addition to the algorithms already available in `Nevergrad`, we include three well-known **EDAs**: `PBIL` (Baluja, 1994), `MIMIC` (De Bonet et al., 1996), and `BOA` (Pelikan, 2002). For these algorithms, we rely on the publicly available implementation at `https://github.com/e5120/EDAs`, using the default hyperparameter settings. We also incorporate one of the most widely used local search methods for pseudo-Boolean optimization, the *one-flip* **Tabu Search** (hereafter referred to as `Tabu`), which has been employed in many effective metaheuristics in recent years, notably for QUBO or NK pseudo-boolean problems

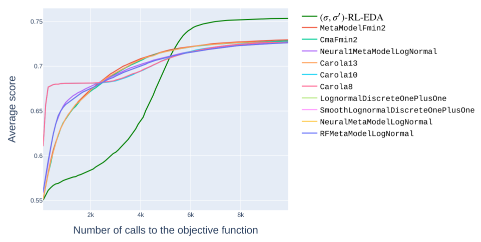
(Glover et al., 2010; Goudet et al., 2024; Samorani et al., 2019; Shi et al., 2017; Wang et al., 2012) (see details in Appendix S).

A detailed presentation of the experimental results can be found in Appendix L. In addition, comprehensive results detailing the performance of all algorithms across the various instance distributions are available in the supplementary material. As shown in Table 4 (see appendix L), the proposed algorithm (σ, σ')-RL-EDA frequently obtains the best performance on larger instances ($n = 128$ and $n = 256$) across the various problems considered in this work and competitive results on the smallest instances ($n = 64$).

Notably, (σ, σ')-RL-EDA performs well on pseudo-Boolean problems QUBO and NK, across a wide range of fitness landscape types—from smooth landscapes (e.g., NK with $K = 1$) to more rugged ones ($K = 8$)—without requiring any change to its hyperparameters, which is rather surprising. As an example, Figure 3 display plots showing the evolution of the best scores (averaged over 100 runs) as a function of the number of objective function evaluations for QUBO instances of size $N = 128$ and type $K = 5$ and NK instances of size $N = 256$ and type $K = 4$. On this plot (σ, σ')-RL-EDA (green curve) is compared against the 10 best-performing other competing algorithms in the set of 504 algorithms. We observe in this plot that our algorithm achieves the best results after 10,000 calls to the objective function. But, it can take time to converge to the best results compared to other algorithms and is therefore dominated when we examine the results after only 1,000 evaluations. This is because (σ, σ')-RL-EDA maintains diversity in the sampled population, precisely to avoid getting stuck too quickly in a low-quality local optimum. A curriculum-based adaptation of the algorithm to accelerate this convergence and cope with a low budget context is described in Appendix O. Furthermore, the adaptation of (σ, σ')-RL-EDA to ternary variables (NK3 instances), also yields very good results using the same hyperparameter configuration, although performance drops are observed for $K = 8$, compared to lower values of K. A more detailed analysis of these under-performances and a way to improve these results is provided in Appendix M.7 (see Figure 14b). Appendix M provides ablation studies and variant analyses to identify the key components that contribute to the effectiveness of (σ, σ')-RL-EDA, including a comparison with input dropout techniques. In Appendix Q, we present the results obtained on large instances of size 1024. The results show that for this instance size, we can obtain good results by adapting the algorithm with parameter sharing between the generators of the n variables. We also compared our method with the competitors on the real neural architecture search public dataset (NAS-Bench-101) (Ying et al., 2019). Our method, with the same hyperparameter configuration as used for the other benchmarks, achieves the best results for small budgets after 1,000 evaluations, but also for large budgets after 10,000 calls to the objective function. Detailed results for this dataset are described in Appendix N.



(a) QUBO instances with $N = 128$ and $K = 5$.



(b) NK instances with $N = 256$ and $K = 4$.

Figure 3: X-axis: number of calls to the objective function. Y-axis: Evolution of average scores.

5 Conclusion

In this work we introduce a novel discrete black-box optimization framework that leverages neural generators of candidate solutions. The model is trained using an original order-invariant reinforcement learning procedure, enhancing sample efficiency. The robustness of our method is supported by extensive empirical evaluation across a diverse set of black-box optimization problems of varying sizes. As future work, we aim to extend this approach to a multi-modal setting, for instance by employing mixtures of distributions, potentially represented through models with attraction–repulsion dynamics.

6 Reproducibility Statement

We provide the source code of our algorithm including instructions on how to launch it in a readme file in the supplementary material.

References

Martin Andersson, Sunith Bandaru, Amos HC Ng, and Anna Syberfeldt. Parameter tuned cma-es on the cec'15 expensive problems. In *2015 IEEE congress on evolutionary computation (CEC)*, pp. 1950–1957. IEEE, 2015.

Charles Audet and Michael Kokkolaras. Blackbox and derivative-free optimization: theory, algorithms and applications. *Optimization and Engineering*, 17(1):1–2, 2016.

Anne Auger and Benjamin Doerr. Theory of randomized search heuristics: Foundations and recent developments. *Theoretical Computer Science*, 412(19):2521–2542, 2011.

Thomas Back. *Evolutionary algorithms in theory and practice: evolution strategies, evolutionary programming, genetic algorithms*. Oxford university press, 1996.

Shumeet Baluja. Population-based incremental learning. a method for integrating genetic search based function optimization and competitive learning. Technical report, 1994.

Eric Benhamou, Jamal Atif, and Rida Laraki. A discrete version of cma-es. *arXiv preprint arXiv:1812.11859*, 2018.

James Bergstra, Rémi Bardenet, Yoshua Bengio, and Balázs Kégl. Algorithms for hyper-parameter optimization. In *Advances in Neural Information Processing Systems*, pp. 2546–2554, 2011.

Eric Brochu, Vlad M Cora, and Nando de Freitas. A tutorial on bayesian optimization of expensive cost functions, with application to active user modeling and hierarchical reinforcement learning. *arXiv preprint arXiv:1012.2599*, 2010.

Emile Contal, David Buffoni, Alexandre Robicquet, and Nicolas Vayatis. Parallel gaussian process optimization with upper confidence bound and pure exploration. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pp. 225–240. Springer, 2013.

Jeremy De Bonet, Charles Isbell, and Paul Viola. Mimic: Finding optima by estimating probability densities. *Advances in neural information processing systems*, 9, 1996.

Benjamin Doerr and Marc Dufay. General univariate estimation-of-distribution algorithms. In *International Conference on Parallel Problem Solving from Nature*, pp. 470–484. Springer, 2022.

Carola Doerr, Furong Ye, Naama Horesh, Hao Wang, Ofer M Shir, and Thomas Bäck. Benchmarking discrete optimization heuristics with iohprofiler. In *Proceedings of the Genetic and Evolutionary Computation Conference Companion*, pp. 1798–1806, 2019.

Carlos Echegoyen, Roberto Santana, Jose A Lozano, and Pedro Larrañaga. The impact of exact probabilistic learning algorithms in edas based on bayesian networks. In *Linkage in Evolutionary Computation*, pp. 109–139. Springer, 2008.

Agoston E Eiben and James E Smith. *Introduction to evolutionary computing*. Springer, 2015.

Michael Emmerich, Kyriakos Giannakoglou, and Boris Naujoks. Single- and multi-objective evolutionary optimization assisted by gaussian random field metamodels. *IEEE Transactions on Evolutionary Computation*, 10(4):421–439, 2006.

Alexander IJ Forrester and Andy J Keane. Recent advances in surrogate-based optimization. *Progress in aerospace sciences*, 45(1-3):50–79, 2009.

Peter I Frazier. Bayesian optimization. In *Recent advances in optimization and modeling of contemporary problems*, pp. 255–278. Informs, 2018.

Mathieu Germain, Karol Gregor, Iain Murray, and Hugo Larochelle. Made: Masked autoencoder for distribution estimation. In *International Conference on Machine Learning*, pp. 881–889. PMLR, 2015.

Fred Glover, Zhipeng Lü, and Jin-Kao Hao. Diversification-driven tabu search for unconstrained binary quadratic problems. *4OR*, 8(3):239–253, 2010.

Javier González, Zhenwen Dai, Philipp Hennig, and Neil D. Lawrence. Batch bayesian optimization via local penalization. In *Proceedings of the 19th International Conference on Artificial Intelligence and Statistics (AISTATS)*, pp. 648–657, 2016.

Olivier Goudet, Adrien Goëffon, and Jin-Kao Hao. A large population island framework for the unconstrained binary quadratic problem. *Computers & Operations Research*, 168:106684, 2024.

Olivier Goudet, Adrien Goëffon, Frédéric Saubion, and Sébastien Verel. Meta-learning of univariate estimation-of-distribution algorithms for pseudo-boolean problems. In *European Conference on Evolutionary Computation in Combinatorial Optimization (Part of EvoStar)*, pp. 84–100. Springer, 2025.

Aditya Grover, Eric Wang, Aaron Zweig, and Stefano Ermon. Stochastic optimization of sorting networks via continuous relaxations. *arXiv preprint arXiv:1903.08850*, 2019.

Nikolaus Hansen. The cma evolution strategy: A tutorial. *arXiv preprint arXiv:1604.00772*, 2016.

Nikolaus Hansen and Andreas Ostermeier. Completely derandomized self-adaptation in evolution strategies. *Evolutionary Computation*, 9(2):159–195, 2001.

Mark Hauschild and Martin Pelikan. An introduction and survey of estimation of distribution algorithms. *Swarm and evolutionary computation*, 1(3):111–128, 2011.

Matthew Hausknecht and Nolan Wagener. Consistent dropout for policy gradient reinforcement learning. *arXiv preprint arXiv:2202.11818*, 2022.

Yaochu Jin. Surrogate-assisted evolutionary computation: Recent advances and future challenges. *Swarm and Evolutionary Computation*, 1(2):61–70, 2011.

Donald R Jones, Matthias Schonlau, and William J Welch. Efficient global optimization of expensive black-box functions. In *Journal of Global optimization*, volume 13, pp. 455–492, 1998.

Kirthevasan Kandasamy, Willie Neiswanger, Jeff Schneider, Barnabás Póczos, and Eric P Xing. Neural architecture search with bayesian optimisation and optimal transport. In *Advances in Neural Information Processing Systems*, volume 31, 2018.

Wathsala Karunarathne, Indu Bala, Dikshit Chauhan, Matthew Roughan, and Lewis Mitchell. Modified cma-es algorithm for multi-modal optimization: incorporating niching strategies and dynamic adaptation mechanism. *arXiv preprint arXiv:2407.00939*, 2024.

Stuart A Kauffman and Edward D Weinberger. The nk model of rugged fitness landscapes and its application to maturation of the immune response. *Journal of theoretical biology*, 141(2): 211–245, 1989.

Minsu Kim, Junyoung Park, and Jinkyoo Park. Sym-nco: Leveraging symmetricity for neural combinatorial optimization. In Sanmi Koyejo, S. Mohamed, A. Agarwal, Danielle Belgrave, K. Cho, and A. Oh (eds.), *Advances in Neural Information Processing Systems 35: Annual Conference on Neural Information Processing Systems 2022, NeurIPS 2022, New Orleans, LA, USA, November 28 - December 9, 2022*, 2022. URL http://papers.nips.cc/paper_files/paper/2022/hash/0cddb777d3441326544e21b67f41bdc8-Abstract-Conference.html.

Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

Gary Kochenberger, Jin-Kao Hao, Fred Glover, Mark Lewis, Zhipeng Lü, Haibo Wang, and Yang Wang. The unconstrained binary quadratic programming problem: a survey. *Journal of combinatorial optimization*, 28(1):58–81, 2014.

Yeong-Dae Kwon, Jinho Choo, Byoungjip Kim, Iljoo Yoon, Youngjune Gwon, and Seungjai Min. POMO: policy optimization with multiple optima for reinforcement learning. In Hugo Larochelle, Marc'Aurelio Ranzato, Raia Hadsell, Maria-Florina Balcan, and Hsuan-Tien Lin (eds.), *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*, 2020. URL `https://proceedings.neurips.cc/paper/2020/hash/f231f2107df69eab0a3862d50018a9b2-Abstract.html`.

Pedro Larranaga. A review on estimation of distribution algorithms: 3. *Estimation of distribution algorithms: a new tool for evolutionary computation*, pp. 57–100, 2002.

Pedro Larrañaga and Concha Bielza. Estimation of distribution algorithms in machine learning: A survey. *IEEE Transactions on Evolutionary Computation*, 28(5):1301–1321, 2024. doi: 10.1109/TEVC.2023.3314105.

Jose A Lozano. *Towards a new evolutionary computation: advances on estimation of distribution algorithms*, volume 192. Springer Science & Business Media, 2006.

Heinz Mühlenbein and Gerhard Paass. From recombination of genes to the estimation of distributions i. binary parameters. In *International conference on parallel problem solving from nature*, pp. 178–187. Springer, 1996.

Yann Ollivier, Léonard Arnold, Anne Auger, and Nikolaus Hansen. Information-geometric optimization algorithms: A unifying picture via invariance principles. *Journal of Machine Learning Research*, 18(18):1–65, 2017.

Ambar Pal, Connor Lane, René Vidal, and Benjamin D Haeffele. On the regularization properties of structured dropout. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 7671–7679, 2020.

Arnaud Pannatier, Evann Courdier, and François Fleuret. σ-gpts: A new approach to autoregressive models. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pp. 143–159. Springer, 2024.

George Papamakarios, Eric Nalisnick, Danilo Jimenez Rezende, Shakir Mohamed, and Balaji Lakshminarayanan. Normalizing flows for probabilistic modeling and inference. *Journal of Machine Learning Research*, 22(57):1–64, 2021.

Martin Pelikan. *Bayesian optimization algorithm: From single level to hierarchy*. University of Illinois at Urbana-Champaign, 2002.

Martin Pelikan. Hierarchical bayesian optimization algorithm. In *Hierarchical Bayesian Optimization Algorithm: Toward a new Generation of Evolutionary Algorithms*, pp. 105–129. Springer, 2005.

Robin L Plackett. The analysis of permutations. *Journal of the Royal Statistical Society Series C: Applied Statistics*, 24(2):193–202, 1975.

Jérémy Rapin and Olivier Teytaud. Nevergrad-a gradient-free optimization platform, 2018.

Michele Samorani, Yang Wang, Zhipeng Lv, and Fred Glover. Clustering-driven evolutionary algorithms: an application of path relinking to the quadratic unconstrained binary optimization problem. *Journal of Heuristics*, 25(4):629–642, 2019.

Roberto Santana. Gray-box optimization and factorized distribution algorithms: where two worlds collide, 2017. URL `https://arxiv.org/abs/1707.03093`.

John Schulman, Sergey Levine, Philipp Moritz, Michael I Jordan, and Pieter Abbeel. Trust region policy optimization. In *Proceedings of the 32nd International Conference on Machine Learning (ICML)*, pp. 1889–1897, 2015a.

John Schulman, Philipp Moritz, Sergey Levine, Michael Jordan, and Pieter Abbeel. High-dimensional continuous control using generalized advantage estimation. *arXiv preprint arXiv:1506.02438*, 2015b.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. In *arXiv preprint arXiv:1707.06347*, 2017. URL `https://arxiv.org/abs/1707.06347`.

Bobak Shahriari, Kevin Swersky, Ziyu Wang, Ryan P Adams, and Nando De Freitas. Taking the human out of the loop: A review of bayesian optimization. *Proceedings of the IEEE*, 104(1): 148–175, 2015.

Siddhartha Shakya. Deum: A framework for an estimation of distribution algorithm based on markov random fields. 2006.

Songqing Shan and G Gary Wang. Survey of modeling and optimization strategies to solve high-dimensional design problems with computationally-expensive black-box functions. *Structural and multidisciplinary optimization*, 41(2):219–241, 2010.

Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, YK Li, Yang Wu, et al. Deepseekmath: Pushing the limits of mathematical reasoning in open language models. *arXiv preprint arXiv:2402.03300*, 2024.

Jialong Shi, Qingfu Zhang, Bilel Derbel, and Arnaud Liefooghe. A parallel tabu search for the unconstrained binary quadratic programming problem. In *2017 IEEE Congress on Evolutionary Computation (CEC)*, pp. 557–564. IEEE, 2017.

Niranjan Srinivas, Andreas Krause, Sham M Kakade, and Matthias W Seeger. Information-theoretic regret bounds for gaussian process optimization in the bandit setting. *IEEE transactions on information theory*, 58(5):3250–3265, 2012.

Richard S Sutton, David A McAllester, Satinder P Singh, and Yishay Mansour. Policy gradient methods for reinforcement learning with function approximation. In *Advances in Neural Information Processing Systems*, volume 12, pp. 1057–1063, 2000.

El-Ghazali Talbi. Machine learning into metaheuristics: A survey and taxonomy. *ACM Comput. Surv.*, 54(6), July 2021. ISSN 0360-0300. doi: 10.1145/3459664. URL `https://doi.org/10.1145/3459664`.

Sara Tari, Sébastien Verel, and Mahmoud Omidvar. Pubo$_\text{i}$: A tunable benchmark with variable importance. In Leslie Pérez Cáceres and Sébastien Verel (eds.), *Evolutionary Computation in Combinatorial Optimization - 22nd European Conference, EvoCOP 2022, Held as Part of EvoStar 2022, Madrid, Spain, April 20-22, 2022, Proceedings*, volume 13222 of *Lecture Notes in Computer Science*, pp. 175–190. Springer, 2022.

Benigno Uria, Marc-Alexandre Côté, Karol Gregor, Iain Murray, and Hugo Larochelle. Neural autoregressive distribution estimation. *Journal of Machine Learning Research*, 17(205):1–37, 2016.

Josu Ceberio Uribe, Benjamin Doerr, Carsten Witt, and Vicente P. Soloviev. Estimation-of-distribution algorithms: Theory and applications (dagstuhl seminar 22182). *Dagstuhl Reports*, 12(5):17–36, 2022. URL `https://doi.org/10.4230/DagRep.12.5.17`.

Xilu Wang, Yaochu Jin, Sebastian Schmitt, and Markus Olhofer. Recent advances in bayesian optimization. *ACM Computing Surveys*, 55(13s):1–36, 2023.

Yang Wang, Zhipeng Lü, Fred Glover, and Jin-Kao Hao. Path relinking for unconstrained binary quadratic programming. *European Journal of Operational Research*, 223(3):595–604, 2012.

Qiang Yang, Wei-Neng Chen, Yun Li, CL Philip Chen, Xiang-Min Xu, and Jun Zhang. Multimodal estimation of distribution algorithms. *IEEE transactions on cybernetics*, 47(3):636–650, 2016.

Chris Ying, Aaron Klein, Eric Christiansen, Esteban Real, Kevin Murphy, and Frank Hutter. Nas-bench-101: Towards reproducible neural architecture search. In *International conference on machine learning*, pp. 7105–7114. PMLR, 2019.

Appendix

A Related methods for solving black-box combinatorial problems

In this appendix, we provide a brief overview of the two principal paradigms that have been developed in the literature for addressing black-box optimization problems: (i) Bayesian optimization (BO) and surrogate-based modeling, and (ii) evolutionary algorithms (EA). We then focus more specifically on Estimation of Distribution Algorithms (EDAs), a subclass of evolutionary algorithms that iteratively use and update a generative model of promising solutions throughout the search process.

Bayesian Optimization: The core idea is to treat the unknown objective f as a random function and place a prior over it, typically using a Gaussian Process (GP). As new evaluations are performed, this prior is updated to form a posterior distribution. The acquisition function—e.g., Expected Improvement (EI), Upper Confidence Bound (UCB), or Probability of Improvement (PI)—guides the search by quantifying the utility of evaluating new candidate solutions (Jones et al., 1998; Srinivas et al., 2012). BO is particularly effective for global optimization under tight evaluation budgets, making it well-suited for expensive black-box problems (Forrester & Keane, 2009; Frazier, 2018; Shahriari et al., 2015). **Limitations :** BO often struggle to scale effectively in high-dimensional discrete domains, particularly when GPs are used as surrogates, due to their computational complexity and modeling assumptions, even if recent advances have extended Bayesian optimization to discrete and structured domains through various adaptations: tree-structured models (Bergstra et al., 2011), relaxations of discrete variables into continuous spaces (Kandasamy et al., 2018), and surrogate models more adapted to categorical or ordinal data with the use of Random Forests (Bergstra et al., 2011) instead of GP. Moreover, these methods are generally based on strong assumptions about the nature of the noise that may appear in the evaluation of the objective function, such as homoscedastic Gaussian noise, which may not hold in real-world settings, thereby compromising the robustness and reliability of the surrogate model (Wang et al., 2023). Another limitation stems from the inherently sequential nature of classical Bayesian optimization, where only one candidate point is evaluated at each iteration. This design can lead to inefficiencies in scenarios where parallel computational resources are available. Although various batch and parallel extensions have been proposed, such as parallel GP-UCB (Contal et al., 2013; González et al., 2016), these approaches often introduce additional computational overhead and require centralized coordination, which can hinder scalability and responsiveness in practical applications.

Evolutionary Algorithms : Metaheuristic approaches (local search, population-based algorithms...) are widely used to solve CO problems, and EAs offer several appealing characteristics. Because they avoid the overhead of building and updating surrogate models, the computational cost per iteration is typically low. EAs also demonstrate robustness to noise, as selection is often based on the ranking of individuals rather than absolute fitness values, making them resilient to stochastic perturbations and invariant under monotonic transformations of the objective. Theoretical convergence results are available for certain classes of EAs, supported by advances in runtime analysis and black-box complexity theory (Auger & Doerr, 2011; Doerr et al., 2019). **Limitations :** EAs may require more function evaluations to identify high-quality solutions compared to model-based approaches for complex problems, which can limit their sample efficiency. Some research, however, has shown that hybrid approaches—combining EAs with surrogate modeling or adaptive sampling strategies—can significantly enhance their effectiveness in scenarios with expensive evaluations (Emmerich et al., 2006; Jin, 2011).

Estimation of distribution Algorithms : Like EAs, EDAs rely on population-based search, but they inherit from BO the notion of modeling structure in the search space, although their modeling goal differs. Instead of modeling the entire objective function, EDAs aim to model only the distribution of promising regions in the fitness landscape, thus avoiding the complexity of full surrogate modeling. This makes EDAs more computationally scalable in high-dimensional or discrete spaces, where standard Gaussian Process-based BO may struggle due to assumptions of smoothness, stationarity, or computational costs of inference (Frazier, 2018; Shan & Wang, 2010). The learning process in EDAs may be as simple as estimating independent univariate marginals, as in the Univariate Marginal Distribution Algorithm (`UMDA`) (Mühlenbein & Paass, 1996), or as sophisticated as constructing full probabilistic graphical models, such as in the Bayesian Optimization Algorithm (`BOA`) (Pelikan, 2002). EDAs still benefit from recent developments (Uribe et al., 2022) that open new possible application domains, for instance, to achieve machine learning tasks (Larrañaga & Bielza, 2024). One of the principal advantages of the modeling strategy of EDAs is its ability to capture variable interactions, an essential feature in epistatic or non-separable problems, where standard EAs often fail. Several EDAs utilize graph structure (DAG) extraction at each generation of the process. The `MIMIC` algorithm (De Bonet et al., 1996) proposes constructing a first-order Markov chain on the variables, classifying them greedily using pairwise mutual information to capture their strongest statistical dependencies. The Bayesian Optimization Algorithm (`BOA`) (Pelikan, 2002) introduces a more expressive probabilistic model using Bayesian networks, allowing it to represent complex, higher-order interactions between variables. The Factorized Distribution Algorithm (`FDA`) (Lozano, 2006; Mühlenbein & Paass, 1996) exploits prior knowledge about the structure of the problem by explicitly incorporating domain-specific decompositions through a predefined factorization of the joint distribution. However, while these approaches can perform well on certain problems, they are fundamentally limited by the exponential growth of computational cost as problem size and dependency complexity increase. In particular, BOA-based methods not only face prohibitive model-construction costs in high-dimensional settings (Hauschild & Pelikan, 2011), but the complexity of learning accurate dependency structures can also hinder effective exploration of the search space. **Limitations :** EDAs exhibit some limitations in terms of premature convergence. Since most EDAs update their probabilistic model solely from the current population, they tend to focus the search around a single promising region, potentially losing diversity and missing other basins of attraction (Hauschild & Pelikan, 2011). To address these limitations, several diversity-preserving or niching-based EDAs have been proposed. For example, the `Multi-CMA-ES` algorithm introduces multiple co-evolving models that repel each other in the search space to maintain diversity and explore multiple optima (Karunarathne et al., 2024). Similar ideas are found in multi-population EDAs or speciation-based approaches (Yang et al., 2016).

A natural limitation is the choice of the distribution model. In the continuous case (i.e. $\mathcal{X} \subseteq \mathbb{R}^n$), a common choice is the multivariate Gaussian distribution, which encodes dependencies via its covariance matrix (e.g. `CMA-ES` (Hansen & Ostermeier, 2001)). In the discrete setting considered here, there is however no direct analogue of the Gaussian. Rather, one instead typically uses probabilistic graphical models, such as Bayesian networks (Echegoyen et al., 2008) or undirected graphical models / Markov networks (e.g. as in `DEUM` (Shakya, 2006)), which model joint dependencies via conditional probability tables or undirected cliques and permit sampling of new candidate vectors. Research on multivariate discrete EDAs has seen a notable decline in recent years because there does not exist the equivalent of the multivariate Gaussian distribution for the discrete space. However, Benhamou et al. (2018) attempts to adapt the `CMA-ES` algorithm to the discrete case, using a multivariate Bernoulli distribution.

B Derivation of the PPO update (2)

While the derivation of (2) is rather straightforward following the proofs in (Schulman et al., 2015a), we detail here its adaptation to our notations and to our undiscounted setting, considering only final rewards, for completeness.

Let us first introduce some classical quantities in reinforcement learning:

- $V^\pi(s)$ is the state value function, which returns the expected cumulative return following policy π from state s. In our setting, this can be defined for any given order σ and any given state $s = (x_{\sigma_{<k}}, k-1, \sigma)$, as:
$$V^\pi(s) = V^{\pi,\sigma}(x_{\sigma_{<k}}) = \mathop{\mathbb{E}}_{\pi_\theta(x_{\sigma_{\geq k}} | x_{\sigma_{<k}}, \sigma)} [f(x)]$$
- $Q^\pi(s, a)$ is the state-action value function, which returns the expected cumulative return from state s, assuming first action in s is a and then subsequent actions are sampled from π. In our setting, this can be defined for any given order σ and any given state $s = (x_{\sigma_{<k}}, k -$

$1, \sigma)$, and any action $a = x_{\sigma_k}$ that specifies the value for X_{σ_k}, as:

$$Q^{\pi}(s,a) = Q^{\pi,\sigma}(x_{\sigma_{<k}}, x_{\sigma_k}) = \mathop{\mathbb{E}}_{\pi_\theta(x_{\sigma_{>k}}|x_{\sigma_{\leq k}},\sigma)}[f(x)]$$

- $A^{\pi}(s,a)$ is the advantage function, defined as:

$$A^{\pi}(s,a) = A^{\pi,\sigma}(x_{\sigma_{<k}}, x_{\sigma_k}) = Q^{\pi,\sigma}(x_{\sigma_{<k}}, x_{\sigma_k}) - V^{\pi,\sigma}(x_{\sigma_{<k}})$$

We are interested in maximizing $J^{\sigma}(\theta) = \mathbb{E}_{\pi_\theta(x|\sigma)}[f(x)]$, while reusing samples from a previous policy to improve sample efficiency and stability.

We start by observing that, given any two policies π_θ and $\pi_{\theta'}$, we have:

$$\arg\max_{\theta} J^{\sigma}(\theta) = \arg\max_{\theta} J^{\sigma}(\theta) - J^{\sigma}(\theta'),$$

since θ does not appear in $J^{\sigma}(\theta')$.

Looking at $J^{\sigma}(\theta) - J^{\sigma}(\theta')$, we get:

$$\begin{aligned}
J^{\sigma}(\theta) - J^{\sigma}(\theta') &= \mathbb{E}_{\pi_\theta(x|\sigma)}[f(x)] - \mathbb{E}_{\pi_{\theta'}(x|\sigma)}[f(x)] && (9)\\
&= \mathbb{E}_{\pi_\theta(x|\sigma)}[f(x)] - V^{\pi_{\theta'},\sigma}(\varnothing) && (10)\\
&= \mathbb{E}_{\pi_\theta(x|\sigma)}\left[f(x) - V^{\pi_{\theta'},\sigma}(\varnothing)\right] && (11)\\
&= \mathbb{E}_{\pi_\theta(x|\sigma)}\left[V^{\pi_{\theta'},\sigma}(x_{\sigma_{\leq n}}) - V^{\pi_{\theta'},\sigma}(\varnothing)\right] && (12)\\
&= \mathbb{E}_{\pi_\theta(x|\sigma)}\left[\sum_{k=1}^{n}\left(V^{\pi_{\theta'},\sigma}(x_{\sigma_{<k+1}}) - V^{\pi_{\theta'},\sigma}(x_{\sigma_{<k}})\right)\right] && (13)\\
&= \mathbb{E}_{\pi_\theta(x|\sigma)}\left[\sum_{k=1}^{n}\left(Q^{\pi_{\theta'},\sigma}(x_{\sigma_{<k}}, x_{\sigma_k}) - V^{\pi_{\theta'},\sigma}(x_{\sigma_{<k}})\right)\right] && (14)\\
&= \mathbb{E}_{\pi_\theta(x|\sigma)}\left[\sum_{k=1}^{n} A^{\pi_{\theta'},\sigma}(x_{\sigma_{<k}}, x_{\sigma_k})\right] && (15)\\
&= \mathbb{E}_{\pi_\theta(x|\sigma)}\sum_{k=1}^{n}\mathbb{E}_{\pi_\theta(x_{\sigma_k}|x_{\sigma_{<k}},\sigma)}\left[A^{\pi_{\theta'},\sigma}(x_{\sigma_{<k}}, x_{\sigma_k})\right] && (16)\\
&= \mathbb{E}_{\pi_\theta(x|\sigma)}\sum_{k=1}^{n}\mathbb{E}_{\pi_{\theta'}(x_{\sigma_k}|x_{\sigma_{<k}},\sigma)}\frac{\pi_\theta(x_{\sigma_k}|x_{\sigma_{<k}},\sigma)}{\pi_{\theta'}(x_{\sigma_k}|x_{\sigma_{<k}},\sigma)}\left[A^{\pi_{\theta'},\sigma}(x_{\sigma_{<k}}, x_{\sigma_k})\right] && (17)
\end{aligned}$$

where $\varnothing$ is the empty sequence (which can also be denoted as the starting point of any sequence $x_{\sigma_{<1}}$). This derivation leverages the fact that in our case, for any sequence x and any policy π, $f(x) = V^{\pi,\sigma}(x_{\sigma_{\leq n}})$ as the sequence is already completed after n steps (we are in a terminal state, as n is the dimension of our combinatorial space $\mathcal{X}$). Also, (13) exploits that every term of the sum telescop except the two extrema that appear in (12), (14) leverages that, following definitions above, for any x and any $0 < k \leq n$, we have: $Q^{\pi_{\theta'},\sigma}(x_{\sigma_{<k}}, x_{\sigma_k}) = V^{\pi_{\theta'},\sigma}(x_{\sigma_{<k+1}})$.

Next, if $\pi_\theta(x|\sigma)$ is sufficiently close to $\pi'_\theta(x|\sigma)$, the idea of TRPO/PPO based approaches is to rather use samples of states from the old policy $\pi_{\theta^t}(x|\sigma)$, rather than the current one. This is done in our case by replacing $\mathbb{E}_{\pi_\theta(x|\sigma)}$ by $\mathbb{E}_{\pi_{\theta^t}(x|\sigma)}$ in (17). We obtain:

$$J^{\sigma}(\theta) - J^{\sigma}(\theta^t) \quad \approx \quad L^{\sigma}_{\theta^t}(\theta) \qquad (18)$$

with

$$\begin{aligned}
L^{\sigma}_{\theta^t}(\theta) &\triangleq \mathbb{E}_{\pi_{\theta^t}(x|\sigma)}\sum_{k=1}^{n}\mathbb{E}_{\pi_{\theta^t}(x_{\sigma_k}|x_{\sigma_{<k}},\sigma)}\frac{\pi_\theta(x_{\sigma_k}|x_{\sigma_{<k}},\sigma)}{\pi_{\theta^t}(x_{\sigma_k}|x_{\sigma_{<k}},\sigma)}\left[A^{\pi_{\theta^t},\sigma}(x_{\sigma_{<k}}, x_{\sigma_k})\right] && (19)\\
&= \mathbb{E}_{\pi_{\theta^t}(x|\sigma)}\sum_{k=1}^{n}\frac{\pi_\theta(x_{\sigma_k}|x_{\sigma_{<k}},\sigma)}{\pi_{\theta^t}(x_{\sigma_k}|x_{\sigma_{<k}},\sigma)}\left[A^{\pi_{\theta^t},\sigma}(x_{\sigma_{<k}}, x_{\sigma_k})\right] && (20)
\end{aligned}$$

Next, we consider $\nabla_\theta L^{\sigma}_{\theta^t}(\theta)$ as a proxy for $\nabla_\theta(J^{\sigma}(\theta) - J^{\sigma}(\theta^t)) = \nabla_\theta J^{\sigma}(\theta)$, which results in (2).

C Derivation of the PPO update with varying generation/training orders

In this section, we check that PPO updates, that we derivated in previous section for the case of an arbitrary fixed generation (and training) order, can be adapted for the case of varying permutations.

For the case where the training order is always the same as the generation one (i.e., $\xi(.|\sigma)$ is a Dirac centered on σ), the derivation of the PPO update is trivial to obtained from (20), as it suffices to take the expectation of $L^{\sigma}_{\theta^t}(\theta)$ depending on distribution $\xi(.)$. The update can be derived by taking the gradient of $L_{\theta^t}(\theta) = \mathbb{E}_{\sigma\sim\xi(\sigma)} L^{\sigma}_{\theta^t}(\theta)$.

Next, we consider the more tricky case, where generation and training orders can be different. For this purpose, looking at $J^{\sigma}(\theta) - J^{\sigma'}(\theta')$, we get:

$$\begin{aligned}
J^{\sigma}(\theta) - J^{\sigma'}(\theta') &= \mathbb{E}_{\pi_\theta(x|\sigma)}[f(x)] - \mathbb{E}_{\pi_{\theta'}(x|\sigma')}[f(x)] && (21)\\
&= \mathbb{E}_{\pi_\theta(x|\sigma)}[f(x)] - V^{\pi_{\theta'},\sigma'}(\varnothing) && (22)\\
&= \mathbb{E}_{\pi_\theta(x|\sigma)}\left[f(x) - V^{\pi_{\theta'},\sigma'}(\varnothing)\right] && (23)\\
&= \mathbb{E}_{\pi_\theta(x|\sigma)}\left[V^{\pi_{\theta'},\sigma'}(\sigma'(x)_{\leq dim_{\sigma'}(n)}) - V^{\pi_{\theta'},\sigma'}(\sigma'(x)_{<dim_{\sigma'}(1)})\right] && (24)\\
&= \mathbb{E}_{\pi_\theta(x|\sigma)}\left[\sum_{k=1}^{n}\Big(V^{\pi_{\theta'},\sigma'}(\sigma'(x)_{<k+1}) - V^{\pi_{\theta'},\sigma'}(\sigma'(x)_{<k})\Big)\right] && (25)\\
&= \mathbb{E}_{\pi_\theta(x|\sigma)}\left[\sum_{k=1}^{n}\Big(Q^{\pi_{\theta'},\sigma'}(\sigma'(x)_{<k}, x_k) - V^{\pi_{\theta'},\sigma'}(\sigma'(x)_{<k})\Big)\right] && (26)\\
&= \mathbb{E}_{\pi_\theta(x|\sigma)}\left[\sum_{k=1}^{n} A^{\pi_{\theta'},\sigma'}(\sigma'(x)_{<k}, x_k)\right] && (27)\\
&= \mathbb{E}_{\pi_\theta(x|\sigma)}\sum_{k=1}^{n}\mathbb{E}_{\pi_\theta(x_k|\sigma(x)_{<k},\sigma)}\left[A^{\pi_{\theta'},\sigma'}(\sigma'(x)_{<k}, x_k)\right] && (28)\\
&= \mathop{\mathbb{E}}_{\pi_\theta(x|\sigma)}\sum_{k=1}^{n}\mathop{\mathbb{E}}_{\pi_{\theta'}(x_k|\sigma'(x)_{<k},\sigma')}\left[\frac{\pi_\theta(x_k|\sigma(x)_{<k},\sigma)}{\pi_{\theta'}(x_k|\sigma'(x)_{<k},\sigma')}A^{\pi_{\theta'},\sigma'}(\sigma'(x)_{<k}, x_k)\right] && (29)
\end{aligned}$$

where we switched to the notation introduced in section 3.3, that is more convenient for dealing with different orders σ and σ'. In particular, this makes that the inner sum from $k = 1$ to n enumerates index from the original problem in $\mathcal{X}$, rather than the generation order from a given permutation. This has an impact on the ordering of advantages functions in (28), but the quantities still telescop, and each advantage is line with the trained transition in (29). We note that importance sampling ratios do not exploit same knowledge, as masks do not apply on same dimensions in the numerator and denominator, but the behavior distribution is still non zero everywhere the training distribution allocates probability mass, which is the main requirement for importance sampling techniques.

Then, given a previous behavior policy π_{θ^t} that sampled solutions with generation order σ', we can train policy π_θ, with training order σ, by considering the following approximator:

$$\begin{aligned}
L^{\sigma,\sigma'}_{\theta^t}(\theta) &\triangleq \mathbb{E}_{\pi_{\theta^t}(x|\sigma')}\sum_{k=1}^{n}\mathbb{E}_{\pi_{\theta^t}(x_k|\sigma'(x)_{<k},\sigma')}\frac{\pi_\theta(x_k|\sigma(x)_{<k},\sigma)}{\pi_{\theta^t}(x_k|\sigma'(x)_{<k},\sigma')}\left[A^{\pi_{\theta^t},\sigma'}(\sigma'(x)_{<k}, x_k)\right]\\
&= \mathbb{E}_{\pi_{\theta^t}(x|\sigma')}\sum_{k=1}^{n}\frac{\pi_\theta(x_k|\sigma(x)_{<k},\sigma)}{\pi_{\theta^t}(x_k|\sigma'(x)_{<k},\sigma')}\left[A^{\pi_{\theta^t},\sigma'}(\sigma'(x)_{<k}, x_k)\right] && (30)
\end{aligned}$$

For any $((\pi_{\theta^t},\sigma'),(\pi_\theta,\sigma))$, we have that: $J^{\sigma}(\theta) - J^{\sigma'}(\theta^t) \approx L^{\sigma,\sigma'}_{\theta^t}(\theta)$ whenever $\pi^t_\theta(.|\sigma')$ remains close to $\pi_\theta(.|\sigma)$.

Finally, we can take $\mathbb{E}_{\sigma\sim\xi(\sigma),\sigma'\sim\xi(\sigma'|\sigma)} L^{\sigma',\sigma}_{\theta^t}(\theta)$ as the maximization objective, with KL regularization constraints that are considered in (8).

D On the Convergence in the Infinite Data and Infinite Capacity Regime

In our approach, we consider at each step of our process the maximization of the quantity (see section 3.3):

$$\hat{L}^t_\lambda(\theta) = \frac{1}{\lambda} \sum_{(x^i,\sigma^i)\in\Gamma^t_\lambda} \mathbb{E}_{\sigma'\sim\xi(\sigma'|\sigma^i)} \sum_{k=1}^{n} \Bigg[\frac{\pi_\theta(x^i_k|\sigma'(x^i)_{<k})}{\pi_{\theta^t}(x^i_k|\sigma^i(x^i)_{<k})} \hat{A}_{\Gamma^t_\lambda}\big(x^{(i)}\big) \\ -\beta D_{\mathrm{KL}}\left(\pi_{\theta^t}(\cdot|\sigma^i(x^i)_{<k}) \,\|\, \pi_\theta(\cdot|\sigma'(x^i)_{<k})\right)\Bigg]. \quad (31)$$

where $\Gamma^t_\lambda = \{x^{(1)}, \ldots, x^{(\lambda)}\}$ is a set of i.i.d. samples from π_{θ^t}, and where $\hat{A}_{\Gamma^t_\lambda}\big(x^{(i)}\big)$ is a ranking function of x_i in the set Γ^t_λ in decreasing order of fitness.

For simplicity of notation, we rewrite this quantity as:

$$\hat{L}^t_\lambda(\theta) = \frac{1}{\lambda}\sum_{i=1}^{\lambda} w_{\theta^t,\theta}\big(x^{(i)},\sigma^{(i)}\big)\, A_{\Gamma^t_\lambda}\big(x^{(i)}\big) + kl_{\theta^t,\theta}\big(x^{(i)},\sigma^{(i)}\big),$$

where:

- $w_{\theta^t,\theta}\big(x^{(i)},\sigma^{(i)}\big) = \mathbb{E}_{\sigma'\sim\xi(\sigma'|\sigma^i)} \sum_{k=1}^{n} \left[\frac{\pi_\theta(x^i_k|\sigma'(x^i)_{<k})}{\pi_{\theta^t}(x^i_k|\sigma^i(x^i)_{<k})}\right]$
- $kl_{\theta^t,\theta}\big(x^{(i)},\sigma^{(i)}\big) = -\beta\mathbb{E}_{\sigma'\sim\xi(\sigma'|\sigma^i)} \sum_{k=1}^{n} \left[D_{\mathrm{KL}}\left(\pi_{\theta^t}(\cdot|\sigma^i(x^i)_{<k}) \,\|\, \pi_\theta(\cdot|\sigma'(x^i)_{<k})\right)\right]$

We first show the following lemma, that states that $\hat{L}^t_\lambda(\theta)$ is an unbiased estimator of:

$$L^t_\lambda(\theta) = \mathbb{E}_\sigma \mathbb{E}_{x\sim\pi_{\theta^t(.|\sigma)}}\Big[w_{\theta^t,\theta}\big(x,\sigma\big)\, \mathbb{E}_{\Gamma^t_\lambda\setminus\{x\}}\big[A_{\Gamma^t_\lambda}(x)\big] + kl_{\theta^t,\theta}\big(x,\sigma\big)\Big], \quad (32)$$

where $\mathbb{E}_{\Gamma^t_\lambda\setminus\{x\}}\big[A_{\Gamma^t_\lambda}(x)\big]$ denotes the expectation of the ranking of x in a set containing $\lambda-1$ other samples from the mixture $\mathbb{E}_\sigma\pi_{\theta^t}(.|\sigma)$:

Lemma 1. $\mathbb{E}\big[\hat{L}^t_\lambda(\theta)\big] = \mathbb{E}_\sigma \mathbb{E}_{x\sim\pi_{\theta^t(.|\sigma)}}\Big[w_{\theta^t,\theta}\big(x,\sigma\big)\, \mathbb{E}_{\Gamma^t_\lambda\setminus\{x\}}\big[A_{\Gamma^t_\lambda}(x)\big] + kl_{\theta^t,\theta}\big(x,\sigma\big)\Big]$

Proof. By the linearity of expectation, we have:

$$\mathbb{E}\big[\hat{L}^t_\lambda(\theta)\big] = \frac{1}{\lambda}\sum_{i=1}^{\lambda} \mathbb{E}\Big[w_{\theta^t,\theta}\big(x^{(i)},\sigma^{(i)}\big)\, A_{\Gamma^t_\lambda}\big(x^{(i)}\big) + kl_{\theta^t,\theta}\big(x^{(i)},\sigma^{(i)}\big)\Big].$$

Then, as all $x^{(i)}$ are i.i.d., each component of the sum owns the same expectation. Thus, by exchangeability, we can say that (arbitrarily taking the first sample $(x^{(1)},\sigma^{(1)})$ from Γ^t_λ as the reference, without loss of generality):

$$\mathbb{E}\big[\hat{L}^t_\lambda(\theta)\big] = \mathbb{E}\Big[w_{\theta^t,\theta}\big(x^{(1)},\sigma^{(1)}\big)\, A_{\Gamma^t_\lambda}\big(x^{(1)}\big) + kl_{\theta^t,\theta}\big(x^{(1)},\sigma^{(1)}\big)\Big].$$

Using the law of total expectation, we obtain:

$$\mathbb{E}\Big[w_{\theta^t,\theta}\big(x^{(1)},\sigma^{(1)}\big)\, A_{\Gamma^t_\lambda}\big(x^{(1)}\big) + kl_{\theta^t,\theta}\big(x^{(1)},\sigma^{(1)}\big)\Big] = \\ \mathbb{E}_{\sigma^{(1)}} \mathbb{E}_{x^{(1)}\sim\pi_{\theta^t}(.|\sigma^{(1)})}\Big[\, w_{\theta^t,\theta}\big(x^{(1)},\sigma^{(1)}\big)\, \mathbb{E}\Big[A_{\Gamma^t_\lambda}\big(x^{(1)}\big) \mid x^{(1)}\Big] + kl_{\theta^t,\theta}\big(x^{(1)},\sigma^{(1)}\big)\Big].$$

Fixing $x^{(1)} = x$ corresponds to considering x as one element of the set Γ^t_λ, and completing it with $\lambda-1$ additional independent draws. Therefore, we have:

$$\mathbb{E}\big[A_{\Gamma^t_\lambda}(x^{(1)}) \mid x^{(1)} = x\big] = \mathbb{E}_{\Gamma^t_\lambda\setminus\{x\}}\big[\,A_{\Gamma^t_\lambda}(x)\,\big].$$

Thus, we finally get:

$$\mathbb{E}\big[\hat{L}^t_\lambda(\theta)\big] = \mathbb{E}_\sigma \mathbb{E}_{x\sim\pi_{\theta^t(.|\sigma)}}\Big[w_{\theta^t,\theta}\big(x,\sigma\big)\, \mathbb{E}_{\Gamma^t_\lambda\setminus\{x\}}\big[A_{\Gamma^t_\lambda}(x)\big] + kl_{\theta^t,\theta}\big(x,\sigma\big)\Big],$$

which concludes the proof and indicates that $\hat{L}^t_\lambda(\theta)$ is an unbiased estimator of $L^t_\lambda(\theta)$. □

Thus, while at each epoch t our algorithm seeks to maximize the stochastic estimator $\hat{L}_\lambda(\theta)$, in expectation it actually aims to optimize the theoretical objective $L^t_\lambda(\theta)$.

Following this, we observe that our surrogate scale-invariant objective $A_{\Gamma^t_\lambda}(x)$ (that we use in (8), in place of the original fitness sore from (7)), can be considered in expectation as a stationary classical reward function at each epoch t, depending only on constant parameters θ_t.

We thus obtain a classical learning problem at each epoch t, where we maximize

$$\pi_\theta(x_k \mid \sigma'(x)_{<k}) \; \frac{\pi_{\theta^t}(x \mid \sigma)}{\pi_{\theta^t}(x_k \mid \sigma(x)_{<k})} \; \mathbb{E}_{\Gamma^t_\lambda \setminus \{x\}}\left[A_{\Gamma^t_\lambda}(x)\right],$$

for any uniformly sampled tuple $(x \in \mathcal{X}, \sigma \in \Omega, \sigma' \in \Omega, k \in [[1,n]])$, under the soft constraint imposed by the KL regularizer. In other words, at each epoch the conditional probability of values for dimension $k \in [[1,n]]$ of solutions likely under $\pi_{\theta^t}(x \mid \sigma)$ is increased (resp. decreased) if they have a positive (resp. negative) expected signed rank among λ samples from $\mathbb{E}_\sigma \pi_{\theta^t}(. \mid \sigma)$. This means that decisions leading to high (resp. low) fitness are reinforced (resp. penalized) at each epoch. As $t \to \infty$, the distribution Γ^t_λ converges asymptotically towards a degenerate set containing a single solution. If λ is infinite, this limiting solution coincides with the global optimum of the problem (i.e., the element $x^\star \in \mathcal{X}$ such that $f(x^\star) = \max_{x \in \mathcal{X}} f(x)$).

E Generation/Training Permutations as Information-Preserving Input Dropout

In section D, we have shown that the quantity we consider in each maximization step is an unbiased estimator of $L^t_\lambda(\theta)$, as defined in (32):

$$L^t_\lambda(\theta) = \mathbb{E}_\sigma \mathbb{E}_{x \sim \pi_{\theta^t(.|\sigma)}} \Big[\mathbb{E}_{\sigma' \sim \xi(\sigma'|\sigma)} \sum_{k=1}^{n} \Big[\frac{\pi_\theta(x_k|\sigma'(x)_{<k})}{\pi_{\theta^t}(x_k|\sigma(x)_{<k})} \; \mathbb{E}_{\Gamma^t_\lambda \setminus \{x\}}\left[A_{\Gamma^t_\lambda}(x)\right] - \beta D_{\mathrm{KL}}\left(\pi_{\theta^t}(\cdot|\sigma(x)_{<k}) \,\|\, \pi_\theta(\cdot|\sigma'(x)_{<k})\right)\Big]\Big], \quad (33)$$

This formulation allows us to distinguish between the two effects of the randomness introduced in the order of generation:

- **Population Diversity**: During first epochs, the neural generators are not prepared for order invariance. Different generation orders σ thus induce different generation distributions $\pi(.|\sigma)$. Uniformly sampling a new σ from Ω for each generation thus implies an higher diversity in the populations. In that cases, any estimation of the reward metric $\mathbb{E}_{\Gamma^t_\lambda \setminus \{x\}}\left[A_{\Gamma^t_\lambda}(x)\right]$ is thus likely to own a greater variance than when using a fixed generation order (especially for low λ), as the variance of a mixture of distributions (i.e., $\mathbb{E}_\sigma \, \pi_{\theta^t}(.,\sigma)$) is always greater or equal than the lowest variance of its components. This allows to better explore in the first steps of the process by introducing more stochasticity in the RL returns. Moreover, this furnishes more diverse samples to the training process, avoiding early collapse on a particular subarea of the search space;
- **Structural Regularization**: Beyond population diversity, the second effect is a form of structural regularization. This arises from presenting, for the same candidate solution x, different contexts at each generation step (i.e., for each neural network g_{θ_k} in our setting). Even when the training order matches the generation order (i.e., when $\xi(\sigma'|\sigma)$ is a Dirac centered at σ), the process encourages the learning of order-invariant generators. In this case, the IS ratios are all equal to 1 at the start of each PPO epoch (with the KL divergence equal to 0). Nevertheless, since each individual processes dimensions in a different order, the generators are encouraged to structure their weights so as to handle arbitrary subsets of variables of any size, ultimately leading to a residual summation structure (see discussion on that point below). However, simply maintaining the same order for training as the one used for generating the training sample is usually not sufficient to efficiently prepare the generator for order-invariance, since a constant order is applied to each training sample

across all iterations of the epoch. The use of a different order for each sample at each iteration of the same epoch (i.e., $\xi(\cdot|\sigma)$ is a uniform distribution in our experiments) provides two benefits. First, it rewards the network for making the same decision under varying contexts, thus facilitating the identification of inter-variable dependencies. Second, it steers the network toward producing, for the same decision, distributions similar to the one used for sampling despite changes in context, through the KL regularizer (which is nonzero even at the first iteration in this setting). All of this benefits sample efficiency, while also promoting generation order invariance and stability through inter-order generalization.

About Residual Structuration In order to further understand the effect of training order permutations on the structuring of a neural network, consider a simple problem of distribution approximation via maximum likelihood estimation (MLE): $\arg\max_\theta \mathbb{E}_p[\log p_\theta(x)]$. Let x be a binary sequence of size n, and let $p_\theta(x)$ be parametrized differently (with parameter θ_i) for each dimension of x, as in the setting of this paper. We specifically focus on the network corresponding to the last dimension of x, i.e., p_{θ_n}.

When optimizing the joint distribution in the original order of the sequence (from dimension 1 to n), p_{θ_n} is always conditioned on all preceding variables, as it predicts the last variable based on the inputs x_1 to x_{n-1}. Given λ samples from p to optimize it via MLE, the gradient updates of p_{θ_n} are computed as an average over λ gradients of the fully informed conditional probability $p_{\theta_n}(x_n \mid x_{<n})$, while some input variables may consist only of noise with respect to the variable being decoded. The optimization process must cope with all these inputs in order to eventually identify true dependencies, despite the presence of potentially significant noise in the input.

Now, let us consider training order permutations σ, which effectively mask every variable x_i whose rank in σ is greater than the rank of x_n (i.e., we set to zero each variable x_i such that $\mathrm{rank}_\sigma(i) > \mathrm{rank}_\sigma(n)$ in the input of p_{θ_i}). The MLE is now given for the variable x_n as:

$$L = \mathbb{E}_p \mathbb{E}_\sigma[\log p_{\theta_n}(x_n|\sigma(x)_{<n})],$$

which, if the distribution of σ is uniform, is equivalent to considering:

$$L = \mathbb{E}_p \Bigg[\frac{(n-1)!}{n!} \log p_{\theta_n}(x_n|\varnothing) + \frac{(n-2)!}{n!} \sum_{i\in[[1,n-1]]} \log p_{\theta_n}(x_n|\{x_i\})$$
$$+ \frac{2(n-3)!}{n!} \sum_{i\in[[1,n-1]]} \sum_{j\in[[1,n-1]], j\neq i} \log p_{\theta_n}(x_n|\{x_i,x_j\}) + \ldots + \frac{(n-1)!}{n!} \log p_{\theta\_n}(x_n|\{x_i\}_{i=1}^{n-1}) \Bigg].$$

or more compactly:

$$L = \mathbb{E}_p \sum_{k=1}^{n} \left[w_k^n \sum_{I_k \in \binom{\{1,\ldots,n-1\}}{k-1}} \log p_{\theta_n}(x_n|\{x_i\}_{i\in I_k}) \right],$$

with $w_k^n = \frac{(k-1)!(n-k)!}{n!}$ the weight of a component depending on the size of its condition (i.e., number of available dimensions for decoding x_n), which in turn can be rewritten as:

$$L = \mathbb{E}_{p(x_n)} \sum_{k=1}^{n} \sum_{I_k \in \binom{\{1,\ldots,n-1\}}{k-1}} \left[w_k^n \mathbb{E}_{p(\{x_i\}_{i\in I_k}|x_n)} \log p_{\theta_n}(x_n|\{x_i\}_{i\in I_k}) \right]$$

From this expansion, we can note a decrease of weights associated with each component of the training problem until $k = n/2$: For any $k < n/2$, $w_{k+1}^n < w_k^n$. This acts on the relative learning speed of the corresponding components, simple dependencies are easier to extract. During optimization, the network thus first learns to encode the marginal probability $p_{\theta_n}(x_n \mid \varnothing)$ for x_n, then incrementally incorporates potential interactions with single variables through $p_{\theta_n}(x_n \mid \{x_i\})$, then with pairs of variables, and so on. As a result, the network naturally develops a form of residual structuring, where outputs are composed by aggregating contributions from different subsets of inputs.

This hierarchical learning process enables the network to more efficiently identify the parent variables that are relevant to the joint distribution, while simultaneously recognizing variables that are unrelated and contribute only noise to $p_{\theta_n}(x_n \mid \sigma(x)_{<n})$. As a result, the network becomes both more robust and sample-efficient, effectively filtering out irrelevant inputs while capturing the essential dependencies.

Order Permutations vs Input Dropout We note that an alternative to permutations is input dropout, whose principle is to randomly mask any feature from the input during training. Similarly to permutation orders, input dropout can be defined as masks that set certain input variables to 0 (or to a null vector in the categorical setting). Here, we consider a mask $m \in \Omega^m$ as a binary $n \times n$ matrix that removes the entry in dimension j for the decision of dimension i if $m_{i,j} = 1$. We denote by $m(x)_k$ the result of applying the dropout mask m to x, using the k-th row of the matrix.

As with permutations, we consider a distribution $\xi^m(.)$ for dropout at generation time, and a distribution $\xi^m(\cdot \mid m)$ for dropout at training time. Given this, our objective in (33) can be naturally extended as:

$$L_\lambda^t(\theta) = \mathbb{E}_{\sigma,m}\mathbb{E}_{x\sim\pi_{\theta^t(.|\sigma,m)}}\Big[\mathbb{E}_{\substack{\sigma'\sim\xi(\sigma'|\sigma^i)\\ m'\sim\xi^m(m'|m)}} \sum_{k=1}^{n}\left[\frac{\pi_\theta(x_k|\sigma'(m'(x)_k)_{<k})}{\pi_{\theta^t}(x_k|\sigma(m(x)_k)_{<k})}\,\mathbb{E}_{\Gamma_\lambda^t\setminus\{x\}}\left[A_{\Gamma_\lambda^t}(x)\right]\right.$$
$$\left. -\beta D_{\mathrm{KL}}\left(\pi_{\theta^t}(\cdot|\sigma(m(x)_k)_{<k}) \,\|\, \pi_\theta(\cdot|\sigma'(m(x)_k)_{<k})\right)\right]\Big], \quad (34)$$

As with permutations, we can consider different distributions for the dropout mask. In this work, we mainly focus on independent Bernoulli distributions for each entry of the mask matrix, controlled by a hyperparameter p. We note in (34) that the dropout mask is applied prior to the causal mask arising from the variable ordering, which allows the combination of both techniques. For the training distribution π_θ, this causal mask can be deactivated by simply implementing σ' as a table that assigns a negative rank to each dimension.

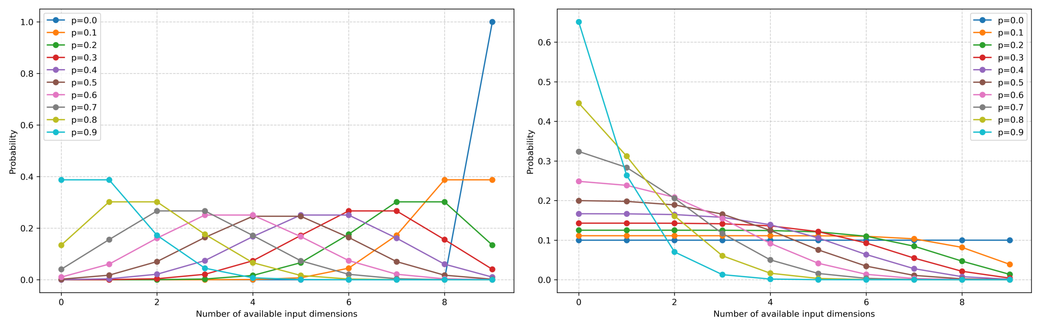
For any configuration, we can compute the probability $P_{mask}(i,j)$ that a given dimension j from the input is masked when decoding variable i. Depending on the setting, we have:

- With the input dropout m only (using Bernoulli parameter p): $P_{mask}^{m_p}(i,j) = p$
- With the causal ordering mask σ only: $P_{mask}^{\sigma}(i,j) = 1 - P(rank_\sigma(j) < rank_\sigma(i)) = 1 - \sum_{r=1}^{n} P(rank_\sigma(i) = r)P(rank_\sigma(j) < rank_\sigma(i)|rank_\sigma(i) = r) = 1 - \frac{1}{n}\sum_{r=1}^{n}\frac{r-1}{n-1} = 1 - \frac{1}{n(n-1)}\sum_{r=0}^{n-1} r = 1 - \frac{n(n-1)/2}{n(n-1)} = 0.5$
- With the input dropout m and causal ordering mask combined: $P_{mask}^{m_p,\sigma}(i,j) = P_{mask}^{m_p}(i,j) + (1 - P_{mask}^{m_p}(i,j)) \times P_{mask}^{\sigma}(i,j) = p + (1-p)0.5 = 0.5 + 0.5p$

Thus, it is possible to set a dropout probability p such that the masking probability of an input for decoding any given dimension is similar to the one induced by random permutations of variable order. However, this equivalence only holds for the marginal distribution over single inputs. To go further, let us consider the distribution $P_{\#\text{available}}(k)$, for $k \in [[0,n]]$, where k denotes the exact number of non-masked inputs available for decoding a given variable i. Depending on the setting, this distribution can differ significantly between permutations and dropout:

- With the input dropout m only: $P_{\#\text{available}}^{m_p}(k) = P^{m_p}(\text{number of non masked dimensions before } n) = \binom{n-1}{k} p^{n-k-1}(1-p)^k$
- With the causal ordering mask σ only: $P_{\#\text{available}}^{\sigma}(k) = P(rank_\sigma(i) = k+1)$
- With the input dropout m and causal ordering mask combined: $P_{\#\text{available}}^{m_p,\sigma}(k) = \sum_{r=k+1}^{n} P(rank_\sigma(i) = r)P^{m_p}(\text{number of non masked dimensions before } r) = \frac{1}{n}\sum_{i=k+1}^{n}\binom{i-1}{k} p^{i-k-1}(1-p)^k = \frac{1}{n}\sum_{i=0}^{n-k-1}\binom{i+k}{k} p^{i}(1-p)^k = \frac{1}{n}\frac{1-I_p(n-k,\,k+1)}{1-p}$, with $I_p(a,b) = \frac{B(p;\,a,b)}{B(a,b)}$ the Regularized incomplete Beta function, $B(p;\,a,b)$ the Incomplete Beta function and $B(a,b)$ the Beta function.

To better illustrate the differences between these settings, Figure 4 shows the distribution of available (non-masked) input variables during neural inference. The x-axis represents k, the number of available inputs, and the y-axis shows the corresponding probability. In both settings - input dropout only (left) and input dropout combined with order permutations under a causal mask (right) - the dropout probability p has a strong impact. Without a causal mask, the distribution is binomial, with mode at $k = \lfloor (n-1)(1-p) \rfloor$. Each variable is independently available with probability $1-p$, but this results in a small chance of observing either very small or very large contexts, which is difficult to control efficiently. Ideally, one would prefer a more evenly spread distribution, providing each variable in diverse contexts. In contrast, when combining input dropout with order permutations under a causal mask (right panel), the distribution becomes more evenly spread across k. This increases the variety of available contexts for each variable during inference, making it easier to learn robust dependencies. Unlike the purely binomial case, each variable can appear in both small and large contexts (for small p values), which improves controllability and ensures that the model sees diverse conditioning patterns. Notably, the case $p = 0$ yields the most uniform distribution of group sizes, enabling more effective structural regularization as discussed above.



(a) Input Dropout Only (b) Input Dropout and Causal Permutations Combined

Figure 4: Probability of having exactly k available (non-masked) input variables during neural inference of the generation probabilities of values for any dimension. Left: input dropout without order permutations. Right: input dropout combined with order permutations.

The effect of input dropout, either alongside or instead of our generation/training order permutations, is evaluated in Section M.1.

Finally, note that using dropout alone cannot be applied for the generation of individuals, since a sampling order must be defined. One option is a predetermined fixed order, combined with a constant causal mask and dropout. This yields a distribution similar to the binomial case above, with k taken among the $i-1$ positions for the i-th variable. However, this approach does not fully exploit structural regularization or population diversity, which would likely require position-dependent parameters. Using varying orders combined with dropout is a potential alternative, but it does not guarantee stable convergence, as input dropout induces information loss during inference. At generation time, this can be detrimental, causing catastrophic forgetting and instability even at the optimum.

In contrast, using permutations of the generation orders without additional dropout is **information-preserving**. For any sampled generation order σ, the joint distribution $\pi_{\theta^t}(\cdot \mid \sigma)$ can fully exploit all dependencies among variables. Moreover, when the generators become fully order-invariant (which is further encouraged by training order permutations through KL regularization across different orderings), we have $\pi_{\theta^t}(\cdot \mid \sigma) = \pi_{\theta^t}(\cdot \mid \sigma')$ for any pair of generation orders $(\sigma, \sigma') \in \Omega^2$, ensuring complete consistency across all orderings.

F ON THE CHOICE OF THE PPO-KL ALGORITHM AS OUR BACKBONE FOR ORDER-INVARIANT RL

As we shown in section E, using random permutations for generation and training in our method can be viewed as a structured dropout of the input features of individuals, which enables various benefits. However, the choice of the KL version of PPO for this purpose is yet to be discussed. This is the focus of this section.

In particular, we can analyze our choices in comparison to findings from (Hausknecht & Wagener, 2022), which also discussed the role of dropout in reinforcement learning and showed that naïvely combining the standard REINFORCE updates with dropout leads to severe instability. Specifically, when the dropout masks differ between trajectory generation and policy updates, the procedure is no longer on-policy, and learning quickly collapses. They investigate PPO in this context, but only the clipped variant. Interestingly, one can observe that the PPO ratio deviates from one even at the first update step (we are no longer on-policy when sampling and training with different masks on layer's inputs). In the clipped version of PPO, this results in most gradients being clipped and therefore prevents meaningful updates. This behavior undermines the intent of clipping—designed to correct occasional overshooting—since here the mechanism blocks learning altogether from the start. To address these issues, the authors propose two strategies for making REINFORCE consistent under dropout: (1) marginalizing over dropout masks, and (2) enforcing identical dropout masks during generation and training (akin to our approach of sampling a permutation during generation and applying the same permutation during training, with σ' drawn from a Dirac distribution). The first strategy is theoretically appealing but practically prohibitive, as even with Monte Carlo approximations using dozens or hundreds of samples, the variance of the estimator overwhelms the learning signal. The second strategy, by contrast, is shown to be more effective and stable.

In our work, we revisit this question from a different angle. While Hausknecht et al. argue that consistency requires using the same dropout mask between rollout and update, we posit that sampling different conditioning patterns at update time can in fact be beneficial. By exposing the policy to multiple conditioning variations from the same rollout, the training process gains additional signal, thereby improving sample efficiency. To make this feasible, we rely on PPO rather than plain REINFORCE. PPO naturally tolerates updates from slightly different policies, which aligns well with our setting where updates need not be fully on-policy. Moreover, we adopt the KL-regularized version of PPO, which avoids the blocking issues observed with the clipped variant: instead of discarding gradients when ratios diverge, the KL penalty smoothly regularizes the policy towards the sampling distribution. This design choice is key to enabling effective training under random permutations.

Importantly, Hausknecht et al. developed their Dropout-Marginalized Gradient in the context of REINFORCE, which forces them to approximate, via Monte Carlo sampling, the exact dropout distribution used during rollout. This requires likelihood normalization over many sampled masks, and thus demands a prohibitively large number of samples to achieve a low-variance estimator. By contrast, in our KL-PPO framework we only need to compute expectations of gradients under the current mask distribution, without approximating the rollout distribution itself. This allows us to train efficiently with as little as a single mask sample per example and iteration, a much lighter procedure in practice.

G CONNECTION WITH NATURAL GRADIENT AND INFORMATION-GEOMETRIC OPTIMIZATION ALGORITHM

The Information-Geometric Optimization (IGO) algorithm (Ollivier et al., 2017) is a natural gradient method that seeks to maximize a quantile-based rewriting of the objective function f.

Let us define $W^f_{\theta^t}$ a monotone rewriting of f at generation t that gives for each individual x^i sampled by the probabilistic model π_{θ^t} for $i = 1, \dots, \lambda$

$$W_{\theta^t}(x^i) = U\left(\frac{\text{rk}(x^i, \Gamma_t)}{\lambda - 1}\right), \tag{35}$$

where U is a non-increasing utility function and $\text{rk}(x^i, \Gamma_t)$ is the rank of the individual i in the population Γ_t given its fitness $f(x^i)$.

For our probabilistic model π_θ with $\theta \in \Theta$, and given a permuation $\sigma \in \Omega$, the IGO flow that defines the trajectory in space Θ to maximize the objective $\mathbb{E}_{x\sim\pi_\theta(x|\sigma)}[W^f_{\theta^t}(x)]$ is given by (see Definition 5 in (Ollivier et al., 2017))

$$\theta^{t+\delta_t} = \theta^t + \delta_t I^{-1}(\theta^t) \sum_{i=1}^{\lambda} W^f_{\theta^t}(x^i) \frac{\nabla \ln \pi_\theta(x^i|\sigma)}{\nabla \theta}\Big|_{\theta=\theta^t}, \tag{36}$$

with x^i for $i = 1, \dots, \lambda$ generated by the model π_{θ^t} at time-step t and $I^{-1}(\theta^t)$ the inverse of the Fisher matrix of π_{θ^t}.

When δt is close to 0, and using Theorem 10 in (Ollivier et al., 2017), (36) can be rewritten as

$$\theta^{t+\delta_t} = \underset{\theta\in\Theta}{\text{argmax}} \left((1 - \delta_t \sum_{i=1}^{\lambda} W^f_{\theta^t}(x^i)) \int \ln \pi_\theta(x|\sigma) \pi_{\theta^t}(dx) + \delta t \sum_{i=1}^{\lambda} W^f_{\theta^t}(x^i) \ln \pi_\theta(x^i|\sigma) \right). \tag{37}$$

When using this framework with our probabilistic model $\pi_\theta(x|\sigma) = \prod_{k=1}^{n} \pi_\theta(x_{\sigma_k}|x_{\sigma<k}, \sigma)$ it gives

$$\theta^{t+\delta_t} = \underset{\theta\in\Theta}{\text{argmax}}[(1 - \delta_t \sum_{i=1}^{\lambda} W^f_{\theta^t}(x^i)) \int \sum_{k=1}^{n} \ln \pi_\theta(x_{\sigma_k}|x_{\sigma<k}, \sigma) \pi_{\theta^t}(dx)$$
$$+ \delta t \sum_{i=1}^{\lambda} \sum_{k=1}^{n} W^f_{\theta^t}(x^i) \ln \pi_\theta(x^i_{\sigma_k}|x^i_{\sigma<k}, \sigma)] \tag{38}$$

As the maximization is on θ we can substract the term $(1 - \delta_t \sum_{i=1}^{\lambda} W^f_{\theta^t}(x^i)) \int \sum_{j=1}^{n} \ln \pi_{\theta^t}(x_{\sigma_k}|x_{\sigma<k}, \sigma) \pi_{\theta^t}(dx)$ that does not depend on θ. Therefore, we have

$$\theta^{t+\delta_t} = \underset{\theta}{\text{argmax}}[\delta t \sum_{i=1}^{\lambda} \sum_{k=1}^{n} W^f_{\theta^t}(x^i) \ln \pi_\theta(x^i_{\sigma_k}|x^i_{\sigma<k}, \sigma)$$
$$+ (\delta_t \sum_{i=1}^{\lambda} W^f_{\theta^t}(x^i) - 1) \sum_{k=1}^{n} \int \ln \frac{\pi_{\theta^t}(x_{\sigma_k}|x_{\sigma<k}, \sigma)}{\pi_\theta(x_{\sigma_k}|x_{\sigma<k}, \sigma)} \pi_{\theta^t}(dx)]. \tag{39}$$

Now using the λ samples to approximate the integral on domain $\mathcal{X}_{\sigma<k}$, and using the fact that all conditional Markov kernels are independent we have for $k = 1, \dots, n$

$$\int \ln \frac{\pi_{\theta^t}(x_{\sigma_k}|x_{\sigma<k}, \sigma)}{\pi_\theta(x_{\sigma_k}|x_{\sigma<k}, \sigma)} \pi_{\theta^t}(dx) \approx \frac{1}{\lambda} \sum_{i=1}^{\lambda} \int \ln \frac{\pi_{\theta^t}(x_{\sigma_k}|x^i_{\sigma<k}, \sigma)}{\pi_\theta(x_{\sigma_k}|x^i_{\sigma<k}, \sigma)} \pi_{\theta^t}(dx_{\sigma_k}). \tag{40}$$

Thus, we have for $k = 1, \dots, n$

$$\int \ln \frac{\pi_{\theta^t}(x_{\sigma_k}|x_{\sigma<k}, \sigma)}{\pi_\theta(x_{\sigma_k}|x_{\sigma<k}, \sigma)} \pi_{\theta^t}(dx) \approx \frac{1}{\lambda} \sum_{i=1}^{\lambda} D_{\text{KL}} \left(\pi_{\theta^t}(\cdot|x^i_{\sigma<k}, \sigma) \parallel \pi_\theta(\cdot|x^i_{\sigma<k}, \sigma) \right) \tag{41}$$

Using (40) and defining $\beta = \frac{1}{\lambda \delta t} - \frac{\sum_{i=1}^{\lambda} W^f_{\theta^t}(x^i)}{\lambda}$, the maximization objective of (39) for the update of the model at each generation becomes

$$L'(\theta) = \frac{1}{\lambda} \sum_{i=1}^{\lambda} \sum_{k=1}^{n} \left[\ln \pi_\theta(x^i_{\sigma_k}|x^i_{\sigma<k}, \sigma) W^f_{\theta^t}(x^i) - \beta D_{\text{KL}} \left(\pi_{\theta^t}(\cdot|x^i_{\sigma<k}, \sigma) \parallel \pi_\theta(\cdot|x^i_{\sigma<k}, \sigma) \right) \right]. \tag{42}$$

The update phase of the algorithm can then be interpreted as the maximization of a weighted log-likelihood over the individuals in the current generation, regularized by a KL divergence term. This regularization penalizes excessive reductions in the entropy of the sampling distribution, thereby maintaining a degree of diversity in the population. By controlling the rate of convergence, this mechanism prevents premature collapse of the distribution onto a single high-performing individual, which could otherwise lead to early stagnation in a local optimum.

It corresponds to the surrogate objective of our GRPO-based framework given by 4 when replacing each term $\ln \pi_\theta(x^i_{\sigma_k}|x^i_{\sigma<k}, \sigma)$ by the ratio importance sampling $\frac{\pi_\theta(x^i_{\sigma_k}|x^i_{\sigma_{<k}},\sigma)}{\pi_{\theta^t}(x^i_{\sigma_k}|x^i_{\sigma_{<k}},\sigma)}$. We empirically observed that maximizing the ratio of importance sampling instead of the log probability gives better results in our context, therefore in the following we stay with the formulation of the objective given by (4) instead of (42).

H Algorithm pseudo-code

In this appendix, we detail the pseudo-code of the multivariate RL EDA with Algorithm 1, which includes the four multivariate RL EDA variants presented in Section 3.3: (δ, δ')-`RL-EDA`, (δ, σ')-`RL-EDA`, (σ, δ')-`RL-EDA` and (σ, σ')-`RL-EDA`.

Until the termination criterion is met, this EDA perform the following steps at each generation t:

1. Draw a population $\Gamma_t = \{(x^i, \sigma^i)\}_{i=1}^{\lambda}$ from the joint distribution $\pi_{\theta^t}(x|\sigma)\xi(\sigma)$.
2. Order the individuals according to their fitness, and compute advantage $\hat{A}_{i,t}$ for each individual.
3. Update the probabilistic model by maximizing during E epochs the objective

$$\hat{L}_\lambda(\theta) = \frac{1}{\lambda} \sum_{(x^i,\sigma^i)\in\Gamma_t} \mathbb{E}_{\sigma'\sim\xi(\sigma'|\sigma^i)} \sum_{k=1}^{n} [\frac{\pi_\theta(x^i_k|\sigma'(x^i)_{<k})}{\pi_{\theta^t}(x^i_k|\sigma^i(x^i)_{<k})}\hat{A}_{i,t} - \beta D_{\mathrm{KL}}\left(\pi_{\theta^t}(\cdot|\sigma^i(x^i)_{<k}) \,\|\, \pi_\theta(\cdot|\sigma'(x^i)_{<k})\right)]. \tag{43}$$

In practice, at each epoch in order to reduce computation time, the expectancy $\mathbb{E}_{\sigma'\sim\xi(\sigma'|\sigma^i)}[.]$ is replaced by an evaluation based on a single sample.

I Multivariate EDA with with learned order

In this appendix, we derive a version of the multivariate EDA learned with PPO, called `Learned-`σ`-RL-EDA` where we model the distribution of order with the Plackett-Luce (PL) distribution (Plackett, 1975) parametrized by the vector of scores $w = (w_1, \ldots, w_n)$ (this distribution is denoted $\xi^{PL}_w(\sigma)$ hereafter) and we use the reparametrization trick proposed by (Grover et al., 2019) to learn w by gradient descent.

I.1 Plackett-Luce distribution

For each $\sigma \in \Omega$, and given $w \in \mathbb{R}^n$ the PL distribution probability mass is given by

$$\xi^{PL}_w(\sigma) = \frac{w_{\sigma(1)}}{Z} \frac{w_{\sigma(2)}}{Z - w_{\sigma(1)}} \cdots \frac{w_{\sigma(n)}}{Z - \sum_{k=1}^{n-1} w_{\sigma(k)}}, \tag{45}$$

with $Z = \sum_{i=1}^{n} w_i$ a normalization constant.

Let $sort : \mathbb{R}^n \to \Omega$ be the operator mapping a n real-valued vector to a permutation σ corresponding to a descending ordering the values of this vector. Let W denote the matrix of absolute pairwise differences of the elements of w such that $W_{ij} = |w_i - w_j|$. As shown by (Grover et al., 2019), the permutation matrix $P_{sort(w)}$ corresponding to $sort(w)$ is given by:

$$P_{\text{sort}(w)}[i,j] = \begin{cases} 1 \text{ if } j = \text{argmax}[(n+1-2i)w - W\mathbb{1}] \\ 0 \text{ otherwise,} \end{cases} \tag{46}$$

Algorithm 1 (σ, σ')-RL-EDA with parameters $\lambda \in \mathbb{N}^*$, $\beta \in \mathbb{R}^+$, utility function U, number of epochs E and functional mechanism g.

```
1: Input: an instance $(\mathcal{X}, f)$, with $\mathcal{X} = \{-1, 1\}^n$, $f : \mathcal{X} \to \mathbb{R}$ and a number of iterations $T$.
2: Randomly initialized the parameters $\theta^0 = (\theta_1^0, \dots, \theta_n^0)$.
3: $x^* \leftarrow \emptyset$ and $f(x^*) \leftarrow -\infty$.
4: for $t = 0, 1, 2, \dots, T-1$ do
5:     for $i = 1, 2, \dots, \lambda$ do
6:         $x^i \leftarrow (0, \dots, 0)$.
7:         Draw a permutation $\sigma^i \sim \xi(\sigma)$.
8:         Generate solution $x^i$ in the order of generation $\sigma^i$:
9:         for $k = 1, 2, \dots, n$ do
10:            $x^i_{\sigma^i(k)} \sim \text{Bernoulli}(\text{sigmoid}(g_{\theta_{\sigma^i(k)}}(x_{\sigma^i < k}))$
11:        end for
12:    end for
13:    for $i = 1, 2, \dots, \lambda$ do
14:        Compute $f(x^i)$.
15:        if $f(x^i) > f(x^*)$ then
16:            $x^* \leftarrow x^i$
17:        end if
18:    end for
19:    for $i = 1, 2, \dots, \lambda$ do
20:        Compute $\hat{A}_{i,t} = U\left(\frac{\text{rk}(x^i)}{\lambda - 1}\right)$.
21:    end for
22:    $\theta \leftarrow \theta^t$
23:    for $e = 1, 2, \dots, E$ do
24:        for $i = 1, 2, \dots, \lambda$ do
25:            $\sigma'^{(i)} \sim \xi(\sigma' | \sigma)$.
26:        end for
27:        Compute
```

$$\hat{L}_\lambda(\theta) = \frac{1}{\lambda} \sum_{(x^i, \sigma^i) \in \Gamma_t} \sum_{k=1}^{n} [\frac{\pi_\theta(x_k^i | \sigma'^{(i)}(x^i)_{<k})}{\pi_{\theta^t}(x_k^i | \sigma^i(x^i)_{<k})} \hat{A}_{i,t} - \beta D_{\text{KL}} \left(\pi_{\theta^t}(\cdot | \sigma^i(x^i)_{<k}) \,\|\, \pi_\theta(\cdot | \sigma'^{(i)}(x^i)_{<k})\right)]. \quad (44)$$

```
28:        Compute $\nabla_\theta \hat{L}_\lambda(\theta)$ and update $\theta$ with gradient ascent.
29:    end for
30:    $\theta^{t+1} \leftarrow \theta$
31: end for
32: Output: the best solution found $x^*$
```

where $\mathbb{1}$ denotes the column vector of all ones.

In practice to sample from $\xi_w(\sigma)$, (Grover et al., 2019) propose a method for sampling from PL distributions with parameters w by sampling for $k = 1, \dots, n$ a noise $\epsilon_k \sim \text{Gumbel}(0, 1)$ with zero mean and unit scale, then by computing $\tilde{w}$ is the vector of perturbed log-scores with entries such that $\tilde{w}_i = \ln w_i + \epsilon_i$, and latsly by applying the sort operator to the perturbed log-scores $\tilde{w}_i$. The resulting order gives a permutation σ sampled from $\xi_w^{PL}(\sigma)$. Indeed (Grover et al., 2019) show that $\mathbb{P}(\tilde{w}_{\sigma(1} \geq \cdots \geq \tilde{w}_{\sigma(n)}) = \xi_w(\sigma)$ (see Proposition 5).

For a vector $\tilde{w}$ of perturbated log-score, the sampled permutation matrices is $P_{sort(\tilde{w})}$ corresponding to permutation $\tilde{\sigma}$, such that $\left[P_{sort(\tilde{w})}\right]_{ij} = 1$ if $i = \tilde{\sigma}(j)$ and 0 otherwise. This permutation matrix allows to compute the adjacency matrix $\tilde{M} = P_{sort(\tilde{w})}^\top B P_{sort(\tilde{w})}$ of the sampling directed acyclic graph (DAG), with B be the strictly upper triangular binary matrix of size $n \times n$, whose entries are defined as $b_{i,j} = 1$ if $j > i$, and $b_{i,j} = 0$ otherwise. Each column vector m_k at position k of $\tilde{M}$ corresponds to the binary causal mask used at step k to mask the entries of g (see Section 3.1).

I.2 Plackett-Luce reparametrization trick

Computing the permutation matrix $P_{sort(\tilde{w})}$ from w is a non differentiable operation due to the use of the argmax function. Therefore, (Grover et al., 2019) propose to replace $P_{sort(\tilde{w})}$ by the continuous relaxation $\widehat{P}_{sort(\tilde{w})}$ using the softmax function instead of the argmax function when gradient computation are required. The i-th row of $\widehat{P}_{sort(w)}$ is given by

$$\widehat{P}_{sort(w)} = \text{softmax}[(n + 1 - 2i)w - W\mathbb{1}/\tau], \tag{47}$$

with $\tau > 0$ a temperature parameter (set at the value of 1 in the following).

I.3 Learned-σ-EDA algorithm

During the sampling phase of `Learned-σ-RL-EDA`, to generate each individual of the population, an order σ^i is first sampled from $\xi_w^{PL}(\sigma)$, then x^i is sampled from $\pi_{\theta^t}(.|\sigma^i)$.

During the update phase of the EDA we maximize following the GRPO objective with respect to (θ, w):

$$\hat{L}_\lambda(\theta, w) = \frac{1}{\lambda} \sum_{(x^i, \sigma^i) \in \Gamma_t} \mathbb{E}_{\sigma' \sim \xi_w^{PL}(\sigma)} \sum_{k=1}^{n} \Big[\frac{\pi_\theta(x_k^i | \sigma'(x^i)_{<k})}{\pi_{\theta^t}(x_k^i | \sigma^i(x^i)_{<k})} \hat{A}_{i,t} \\ - \beta D_{\text{KL}} \left(\pi_{\theta^t}(\cdot | \sigma^i(x^i)_{<k}) \,\|\, \pi_\theta(\cdot | \sigma'(x^i)_{<k}) \right) \Big]. \tag{48}$$

This maximization is done by first order gradient descent using $\nabla_\theta L(\theta, w)$ and $\nabla_w L(\theta, w)$ (computed with the reparametrization trick).

J Synthetic Data Set Generation and experimental protocol

We examine the following NP-hard problems in this work. For each of these problems, we generated instances of size $n \in \{64, 128, 256\}$, and for each size, we considered different types of instances.

The Quadratic unconstrained binary optimization problem (QUBO) aims to find a pseudo-Boolean vector $x = (x_1, \dots, x_n)$ of size n that maximizes the function $f : \{-1, 1\}^n \to \mathbb{R}$ given by $f(x) = x^\top Q x$, where Q is a symmetric real matrix of size $n \times n$. We generate QUBO instances using the PUBO_i generator (Tari et al., 2022), which enables the creation of QUBO problems with controlled structural properties. The parameters of the PUBO_i generator are set to produce six different types K of instances by tuning both the density of the QUBO matrix Q and the relative importance of binary variables, thereby influencing the degree of non-uniformity in Q. We generate QUBO instances using the PUBO_i generator (Tari et al., 2022), which enables the creation of QUBO problems with controlled structural properties.

Formally, the fitness function of each instance of this QUBO problem is defined as $f(x) = \sum_{i=1}^{m} f_i(x_{i_1}, x_{i_2}, x_{i_3}, x_{i_4})$, where each sub-function f_i is a quadratic function randomly selected from the set $\{\varphi_1, \dots, \varphi_4\}$. Each φ_k is designed to have $2k$ symmetric local optima. In PUBO_i, binary variables are divided into two importance classes: important and non-important variables. For each sub-function f_i, the four variables x_{i_j} are selected according to an importance degree parameter d, where the probability of selecting an important variable is proportional to d. An additional importance co-appearance parameter α controls the correlation in the selection of important variables: higher α values increase the likelihood that two important variables co-occur within the same sub-function f_i. The number of sub-functions is given by $m = r \times \frac{n(n-1)}{2}$, where r is a density coefficient controlling the proportion of non-zero entries in Q. For example, with $r = 0.05$ and $r = 0.2$, the density of Q is approximately 16% and 43%, respectively, for uniform instances.

We consider three interaction configurations:

- Uniform random instances when $(d, \alpha) = (1, 1)$, corresponding to no specific important variables, i.e., a fully random QUBO structure.
- Instances with $(d, \alpha) = (10, 1)$, where important variables are 10 times more likely to be selected than non-important variables, but selections are independent.
- Instances with $(d, \alpha) = (10, 1.09)$: the selection of important variables is not independent, and the selection of important variables is concentrated.

Further details on the PUBO_i generator can be found in (Tari et al., 2022). By combining parameters r, degree d of importance of variables and parameter α of co-appearance, we obtain six different types of instance described in Table 1.

Table 1: Parameters of PUBO_i instances.

Type instance K	r	d	α
0	0.05	1	1
1	0.05	10	1
2	0.05	10	1.09
3	0.2	1	1
4	0.2	10	1
5	0.2	10	1.09

The NKD model is a natural extension of the NK model of Kauffman (Kauffman & Weinberger, 1989) to cases where variables can take more than two categorical values. This is a framework for describing fitness lanscapes whose problem size and ruggedness are both parameterizable. The NKD function is defined as $f_{\mathrm{NKD}} : \{0, 1, \dots, D-1\}^n \to [0, 1[$ and takes the same form as NK functions: $f_{\mathrm{NKD}}(x) = \frac{1}{n}\sum_{i=1}^{n} \gamma_i(x_i, x_{l_{i1}}, \dots, x_{l_{iK}})$, except that each subfunction $\gamma_i : \{0, 1, \dots, D-1\}^{K+1} \to [0, 1[$ is defined over categorical variables with D possible values instead of binary ones. We construct instances with $D = 2$, which corresponds to the original pseudo-boolean NK problem, but we also construct instances of a categorical problem called NK3 with $D = 3$. For each variant NK or NK3 of the problem four different types of distribution of instances with $K \in \{1, 2, 4, 8\}$ are built. When $K = 1$, the interaction graph is very sparse and the landscape is smooth; when $K = 8$, the landscape becomes significantly more rugged.

Unless otherwise specified, we treat these problems as black-box problems, meaning that both the objective function and the interaction graph between variables are assumed to be unknown. For each pair (n, K) and for each problem, we generated 10 different instances. For the sake of reproducibility, all these instances are available in the supplementary material. For each problem instance, we allow a maximum budget of 10,000 objective function evaluations. The best solution found since the beginning of the search is recorded every 100 evaluations. For each distribution of instances, defined with the vector of features (pb, n, K) (with pb the problem name, n the instance size and K the type of instance), and for each algorithm, we compute the average performance over 10 distinct instances, each solved with 10 independent restarts using different random seeds. This procedure results in 100 independent runs per algorithm and per instance distribution, from which the evolution of the average score is reported. It is worth noting that, within a given distribution, the best scores obtained across the 10 instances are of comparable magnitude, which justifies averaging them to produce a single representative performance measure.

K MULTIVARIATE EDA HYPERPARAMETER CONFIGURATION AND COMPUTING TIME

In this appendix, we detail the hyperparameter configuration of the multivariate RL EDA presented in Section 3.3, which is used as a baseline for all experiments, and give some details on the complexity and computing time of the proposed approach.

K.1 HYPERPARAMETER CONFIGURATION

The population size is set by default to $\lambda = 10$ accross all benchmark instances. Although fine-tuning this parameter may lead to better performance for specific distributions of problem instances, and may also depend on the instance dimension n, we opt for simplicity and maintain a constant value throughout this work. A sensitivity analysis of this key parameter is presented in Subsection M.4.

By default, each functional mechanism g_{θ_i} for $i = 1, \ldots, n$ is implemented as a feedforward neural network with a single hidden layer of 20 neurons, using the hyperbolic tangent activation function. This choice is particularly advantageous, as it allows the network to approximate both nonlinear and linear relationships when needed. Employing one-hidden-layer neural networks for each variable strikes a practical balance between model expressiveness and computational efficiency, especially given the instance sizes considered in this study. Nevertheless, as discussed in Appendix M.7, we explore alternative configurations—such as linear models and deeper neural networks—which may offer improved performance on more complex tasks, albeit at the cost of increased computational time.

The utility function U used in the advantage calculation of (5) is defined as a linear decreasing function on the interval $[0, 1]$, specifically $U(x) = 1-2x$. Under this definition, the best individual x^i_{best} in the current population, with $\text{rk}(x^i_{best}) = 0$, receives a reward $A_{\Gamma^t_\lambda}(x^i_{best}) = 1$, whereas the worst individual x^i_{worst}, with $\text{rk}(x^i_{best}) = \lambda - 1$, receives $A_{\Gamma^t_\lambda}(x^i_{worse}) = -1$. If λ is odd, the individual with median fitness obtains an advantage of zero. With this choice of U, maximizing (8) assigns the greatest weight to increasing the likelihood of generating the best individual in the population, while simultaneously decreasing the likelihood of generating the worst individual. As a result, the policy is updated so that, in the next generation $t + 1$, it tends to produce individuals that are closer to the best members of generation t, and farther from the worst ones. It is worth noting that a fine-tuned utility function may yield superior performance for specific distributions of problem instances. Prior research has investigated the impact of selecting appropriate utility values or importance weights. For example, in the context of the `CMA-ES` algorithm, (Andersson et al., 2015) showed that adapting these parameters to the distribution of instances can lead to significant performance improvements. Specifically, for smooth landscapes with a single local optimum, a utility function that assigns disproportionately high values to the very best individuals can be advantageous. Conversely, for highly deceptive landscapes, it may be beneficial to assign the highest weights to the worst-performing individuals in the population.

Regarding the coefficient for the KL regularization term, we consistently set $\beta = 1$. A sensitivity analysis of this parameter is presented in Subsection M.5. At each generation, the algorithm is trained for $E = 50$ epochs using the Adam optimizer (Kingma & Ba, 2014) with an initial learning rate 0.001. In practice, to avoid numerical issue in the multivariate RL EDAs, particularly division by zero when evaluating the KL divergence term or the importance sampling ratio, we apply clipping to the probability values of each conditional distribution $\pi_\theta(\cdot|\sigma'(x^i)_{<k})$. Specifically, all probabilities are clipped to lie within the interval $[\epsilon, 1 - \epsilon]$, with $\epsilon = 0.001$. Table 2 summarizes all hyperparameters used in the multivariate RL EDA.

K.2 TIME AND SPACE COMPLEXITY

The overall time complexity of the proposed RL-EDA algorithm can be decomposed into two main components:

1. Solution Generation: For each iteration t of the EDA, a population of size λ is sampled. Each solution has n variables generated sequentially by neural networks with one hidden

Table 2: Hyperparameters settings for (σ, σ')-`RL-EDA`

Parameter	Description	Value
	EDA parameters	
λ	Size of the population	10
L	Number of hidden layers in g	1
n_l	Number of neurons in hidden layer	20
ϵ	Probability threshold coefficient	0.001
	PPO parameters	
U	Utility function	$U(x) = 1 - 2x$
β	KL penalty parameter	1
E	Number of training epoch	50
l_r	Learning rate of Adam optimizer	0.001

layer of size h. The cost per forward pass for one variable is $O(nh)$. For λ solutions with n variables per solution $O(\lambda n^2 h)$.

2. Policy Update (Training) : For E epochs per generation, each epoch recomputes masked inputs and performs gradient updates $O(E\lambda n^2 h)$

Hence the total complexity for T generations is $O(T.E.\lambda.n^2 h)$. Note that a classical BOA (Pelikan, 2002) typically leads to $O(n^3)$.

Concerning space complexity, using the n small NNs in the standard version, we get an $O(n^2 h)$ space complexity, which decreases to $O(nh)$ for the shared-parameter variant (see Appendix Q).

K.3 Computing time

The multivariate RL EDA algorithm is implemented in Python 3.7 with Pytorch 2.5 library for tensor calculation with Cuda 12.4. The source code is available in the supplementary material. It is specifically designed to run on GPU devices.

When using the hyperparameters described in Table 2, the time required to process a single QUBO instance of size $n = 128$, with a budget of 10,000 calls to the objective function—corresponding to 1,000 generations of the algorithm when $\lambda = 10$—is approximately 11.5 minutes on a single Intel(R) Xeon(R) Silver 4208 CPU at 2.10GHz, and 5 minutes on an Nvidia V100 GPU device (including the 10,000 objective function evaluations). The code is also adapted to process batches of multiple instances of the same size in parallel, which greatly benefits from GPU parallelization. In particular, it can process 100 QUBO instances of size $n = 128$, each with a budget of 10,000 objective function calls, in 20 minutes on a single V100 GPU device.

We have also implemented a version of the algorithm with shared parameters in the architecture (see Appendix Q) which scales better in term of CPU/GPU footprints.

Table 3 gives more detail on wall-clock times required to solve QUBO instances of different sizes with a budget of 10,000 evaluations for the standard version (σ, σ')-`RL-EDA` and the version with shared parameters called (σ, σ')-`RL-EDA-share-params`, in comparison with Tabu, BOA EDA and strong Nevergrad baseline CMApara. Times are given in seconds and evaluated for CPU on Xeon(R) Silver 4208 at 2.10GHz and for GPU on an Nvidia V100 GPU device. Note even a time of 2940 seconds to solve a big instance of size $n = 256$ with a budget of 10,000 on a CPU with (σ, σ')-`RL-EDA` is acceptable if we see that it takes actually 0.29 second per solution generated, and for a black box problem such as neural architecture search (see Appendix N), the time required to evaluate a single solution is generally much more costly.

These times are provided for indicative purposes only, as the main criterion used to assess the performance of a black-box algorithm is typically the best score obtained within a limited number of calls to the objective function—a criterion that is precisely retained in our experimental analyses and benchmark comparisons.

Table 3: CPU/GPU wall-clock times required to solve a QUBO instance with a budget of 10,000 calls to the objective function.

Size instance	CPU time (s)	GPU time (s)
Tabu		
64	2	-
128	5	-
256	20	-
CMApara (Nevergrad)		
64	61	-
128	78	-
256	141	-
Multivariate BOA EDA		
64	460	-
128	2100	-
256	9310	-
(σ, σ')-RL-EDA		
64	450	210
128	690	300
256	2940	420
(σ, σ')-RL-EDA-share-params		
64	300	195
128	420	255
256	780	300

L Global Experimental Results

Table 4 presents a selection of these results, comparing (σ, σ')-RL-EDA to the three other EDAs of the same category: PBIL, MIMIC and BOA. The final columns report the performance of the best algorithm among all remaining competitors, including the Nevergrad algorithms and the Tabu algorithm. For each algorithm, we report the average score obtained after 10,000 calls to the objective function, averaged over 100 independant runs. Based on this average score, the algorithms are ranked, and their position among all competitors is indicated.

To facilitate comparison between our proposed algorithm, (σ, σ')-RL-EDA, and the best-performing competing methods, we conducted statistical significance tests. In Table 4, a star next to the results of (σ, σ')-RL-EDA indicates that its average performance over 100 runs is statistically significantly better than that of the best other competing algorithm. Conversely, a star next to a competing algorithm denotes that it significantly outperforms (σ, σ')-RL-EDA on average. Statistical significance is assessed using a two-sample t-test with a p-value threshold of 0.001.

We observe in Table 4 that (σ, σ')-RL-EDA consistently outperforms the other EDAs. Interestingly, among the three competing EDAs, the univariate PBIL algorithm achieves the best results.[3]. This confirms empirical findings previously reported by (Doerr & Dufay, 2022), which suggest that univariate EDAs can sometimes match or even surpass the performance of more complex multivariate EDAs. On possible explanation is that the number of parameters to be learned in multivariate models such as MIMIC and BOA increases rapidly with instance size, potentially slowing convergence compared to the simpler PBIL. Among other competitors, it is worth highlighting the performance of the Tabu algorithm. Despite its simplicity and limited integration in mainstream black-box optimization libraries, it often achieves strong results, particularly on smaller instances.

In addition to the global results table, we also provide plots showing the evolution of the best scores (averaged over 100 runs) as a function of the number of objective function evaluations. In each plot, the curve for (σ, σ')-RL-EDA is always displayed in green and placed first in the legend, for consistency. It is compared against the 10 best-performing competing algorithms, listed in the legend from best to worst.

[3]Since PBIL is designed specifically for pseudo-Boolean optimization, it was not evaluated on NK3 instances involving variables with three categorical values

Instances			Methods										
Pb	n	K	(σ,σ')-RL-EDA		PBIL		MIMIC		BOA		Best method (others)		
			Rank	Score	Rank	Score	Rank	Score	Rank	Score	Name	Rank	Score
QUBO	64	0	34/505	200.8	62/505	199.8	250/505	188.2	268/505	184.6	**Tabu**	**1/505**	**208.4***
QUBO	64	1	82/505	148.8	91/505	147.8	134/505	146.0	140/505	145.4	**CMApara**	**1/505**	**154.3***
QUBO	64	2	115/505	138.1	88/505	139.1	119/505	137.6	154/505	137.4	**DiscreteDE**	**1/505**	**143.4***
QUBO	64	3	80/505	411.2	90/505	410.4	265/505	379.4	267/505	377.5	**Tabu**	**1/505**	**438.1***
QUBO	64	4	114/505	326.1	80/505	329.7	265/505	311.7	276/505	309.7	**CMApara**	**1/505**	**344.2***
QUBO	64	5	77/505	309.4	66/505	310.0	242/505	298.3	261/505	295.9	**CMApara**	**1/505**	**319.3***
QUBO	128	0	**1/505**	**593.7***	66/505	570.8	257/505	504.4	225/505	517.2	Tabu	2/505	588.7
QUBO	128	1	2/505	449.2	21/505	438.3	242/505	408.4	227/505	413.0	**CMApara**	**1/505**	**453.8***
QUBO	128	2	**1/505**	**437.1**	19/505	427.5	238/505	398.9	223/505	403.7	CMAL3	2/505	435.4
QUBO	128	3	**1/505**	**1227.2***	79/505	1177.8	258/505	1034.7	254/505	1046.1	Wiz	2/505	1207.2
QUBO	128	4	2/505	955.4	17/505	934.5	266/505	842.8	254/505	857.3	**CMApara**	**1/505**	**964.9***
QUBO	128	5	**1/505**	**933.3***	54/505	907.6	264/505	817.2	250/505	830.9	CMAL3	2/505	928.6
QUBO	256	0	**1/505**	**1697.7***	46/505	1570.4	199/505	1317.4	99/505	1422.4	NLOPT_LN_PRAXIS	2/505	1607.1
QUBO	256	1	**1/505**	**1367.7***	3/505	1290.5	197/505	1105.2	92/505	1197.0	BigLognormalDiscreteOnePlusOne	2/505	1301.4
QUBO	256	2	**1/505**	**1304.1***	12/505	1230.9	187/505	1073.0	92/505	1154.4	SVMMetaModelLogNormal	2/505	1233.8
QUBO	256	3	**1/505**	**3436.8***	53/505	3208.6	196/505	2650.7	148/505	2854.3	RLSOnePlusOne	2/505	3316.5
QUBO	256	4	**1/505**	**2769.0***	35/505	2597.5	208/505	2219.0	134/505	2391.5	DiscreteLengler2OnePlusOne	2/505	2617.1
QUBO	256	5	**1/505**	**2730.1***	41/505	2557.0	185/505	2206.6	141/505	2349.2	SVM1MetaModelLogNormal	2/505	2605.1
NK	64	1	29/505	0.7103	52/505	0.7096	127/505	0.7050	237/505	0.7008	**CMApara**	**1/505**	**0.7119**
NK	64	2	24/505	0.742	58/505	0.7391	147/505	0.7317	205/505	0.7273	**CMApara**	**1/505**	**0.7459**
NK	64	4	13/505	0.7523	41/505	0.7463	147/505	0.7330	180/505	0.7311	**Tabu**	**1/505**	**0.7657***
NK	64	8	19/505	0.7379	35/505	0.7330	263/505	0.7088	309/505	0.6932	**Tabu**	**1/505**	**0.7602***
NK	128	1	**1/505**	**0.7100**	4/505	0.7061	159/505	0.6958	207/505	0.6941	CMApara	2/505	0.7074
NK	128	2	**1/505**	**0.7375***	2/505	0.7305	141/505	0.7138	139/505	0.7139	CMApara	3/505	0.7304
NK	128	4	**1/505**	**0.7603***	2/505	0.7464	203/505	0.7190	125/505	0.7252	Tabu	3/505	0.7462
NK	128	8	2/505	0.7369	3/505	0.7266	356/505	0.6372	388/505	0.6071	**Tabu**	**1/505**	**0.7429***
NK	256	1	**1/505**	**0.7071***	2/505	0.7014	111/505	0.6810	87/505	0.6869	CMApara	3/505	0.6989
NK	256	2	**1/505**	**0.7364***	2/505	0.7248	98/505	0.7004	60/505	0.7100	MetaModelFmin2	3/505	0.7218
NK	256	4	**1/505**	**0.7534***	2/505	0.7336	104/505	0.7006	189/505	0.6895	MetaModelFmin2	3/505	0.7295
NK	256	8	**1/505**	**0.7232***	2/505	0.7171	385/505	0.5798	390/505	0.5730	LognormalDiscreteOnePlusOne	3/505	0.7166
NK3	64	1	**1/500**	**0.7818***	-	-	71/500	0.7659	116/500	0.7635	DiscreteDE	2/500	0.7772
NK3	64	2	**1/500**	**0.8095**	-	-	8/500	0.7857	74/500	0.7779	Tabu	1/500	0.7995
NK3	64	4	2/500	0.8004	-	-	138/500	0.7622	154/500	0.7570	**Tabu**	**1/500**	**0.8062**
NK3	64	8	63/500	0.7473	-	-	360/500	0.6407	358/500	0.6420	**Tabu**	**1/500**	**0.7855**
NK3	128	1	**1/500**	**0.7876**	-	-	62/500	0.7599	103/500	0.7537	DiscreteLengler3OnePlusOne	1/500	0.7800
NK3	128	2	**1/500**	**0.7986***	-	-	58/500	0.7635	111/500	0.7527	BigLognormalDiscreteOnePlusOne	2/500	0.7820
NK3	128	4	**1/500**	**0.7847***	-	-	124/500	0.7374	130/500	0.7311	Neural1MetaModelLogNormal	2/500	0.7740
NK3	128	8	63/500	0.7373	-	-	377/500	0.5986	345/500	0.6008	**Tabu**	**1/500**	**0.7608***
NK3	256	1	**1/500**	**0.7763***	-	-	55/500	0.7360	62/500	0.7247	NGOpt	1/500	0.7542
NK3	256	2	**1/500**	**0.7801***	-	-	53/500	0.7391	69/500	0.7236	RF1MetaModelLogNormal	2/500	0.7600
NK3	256	4	**1/500**	**0.7615***	-	-	147/500	0.6784	69/500	0.7091	SVM1MetaModelLogNormal	2/500	0.7522
NK3	256	8	43/500	0.7213	-	-	362/500	0.5704	402/500	0.5692	**RLSOnePlusOne**	**1/500**	**0.7362***

Table 4: Global rankings and average scores obtained by (σ, σ')-RL-EDA and the other EDAs (PBIL, MIMIC, and BOA) are reported. The last columns present the ranking and average score of the best-performing method among the 501 additional algorithms considered (496 for NK3 problems). Rankings are computed over all 505 algorithms (500 for NK3 problems) by comparing the best score achieved after 10,000 objective function evaluations, averaged across 100 independent runs. Bold values highlight the best results among all competing methods. A star associated the results obtain by (σ, σ')-RL-EDA indicates that it is significantly better in average (over 100 runs) than the best other competitor. A star associated with a result obtain by an other algorithm indicates that it is significantly better in average (over 100 runs) than (σ, σ')-RL-EDA. A difference on the average scores is said statistically significant according to a t-test with p-value 0.001.

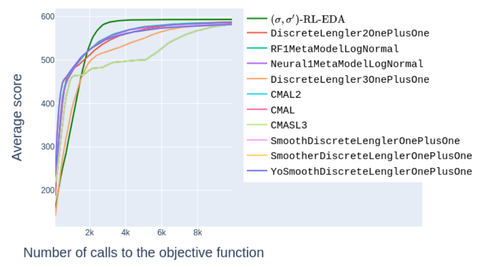
Here, we present these curves only for the different instance types of size $n = 128$ from the pseudo-Boolean QUBO problem (Figure 5) and the categorical NK3 problem (Figure 6).[4] Note that for the QUBO instance distribution with $n = 128$ and $K = 3$ (Figure 5d), the 10 other best algorithms, which are variants of the meta-algorithm NGOpt, exhibit overlapping performance curves. This is because they all selected the same low-level algorithm, DiscreteLenglerOnePlusOne, based on the characteristics of the instance.

When comparing the evolution curves of (σ, σ')-RL-EDA across these two problems, we observe markedly different behaviors. For QUBO problems (Figure 5), (σ, σ')-RL-EDA quickly reaches a good solution and then stagnates for the remainder of the budget. The best scores are typically achieved after approximately 3,000 to 4,000 evaluations, suggesting that the full budget of 10,000 does not benefit (σ, σ')-RL-EDA, but rather favors competing algorithms.

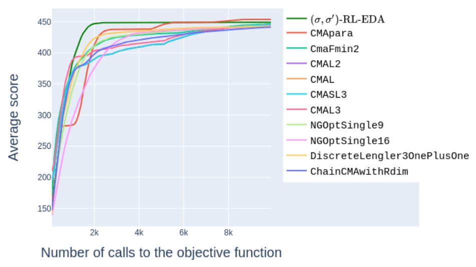
In contrast, for NK3 instances (Figure 6), (σ, σ')-RL-EDA requires significantly more time to converge. The algorithm exhibits an "S"-shaped curve, indicating a delayed learning phase before generating high-quality solutions. This behavior becomes more pronounced as the interaction graph increases (i.e., with higher K values), likely due to the increased difficulty in modeling variable interactions in NK3 compared to QUBO. Notably, for the most complex instances ($K = 8$), (σ, σ')-RL-EDA fails to converge within the allocated budget, explaining its poor

[4] All plots for all instance distributions are available in the supplementary material.

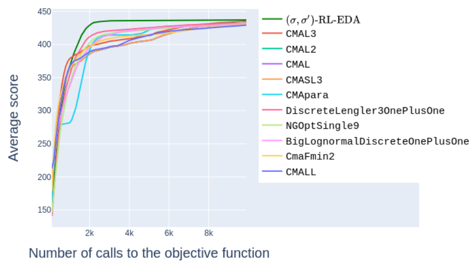
performance reported in Table 4 for this distribution. Meta-algorithms from the `Nevergrad` library that incorporate neural networks (`NeuralMetaModelLogNormal`) or random forests (`NRFMetaModelLogNormal`) achieve good results more rapidly. On the other hand, when (σ, σ')-`RL-EDA` has sufficient time to converge—as in landscapes with $K = 2$ or $k = 4$—it achieves significantly better average scores than its competitors by the end of the search.



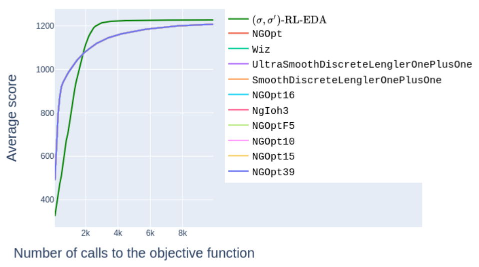
(a) QUBO instances with $n = 128$ and $K = 0$.



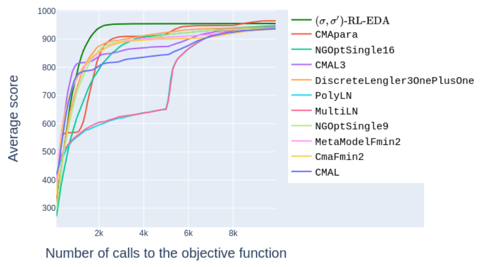
(b) QUBO instances with $n = 128$ and $K = 1$.



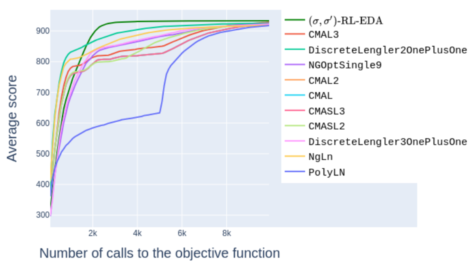
(c) QUBO instances with $n = 128$ and $K = 2$.



(d) QUBO instances with $n = 128$ and $K = 3$.



(e) QUBO instances with $n = 128$ and $K = 4$.



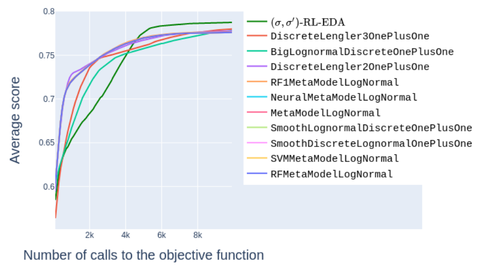
(f) QUBO instances with $n = 128$ and $K = 5$.

Figure 5: Evolution of the average scores w.r.t. the number of calls to the objective function obtained by (σ, σ')-`RL-EDA` and the best 10 other competitors for the different type of QUBO instances with $n = 128$.

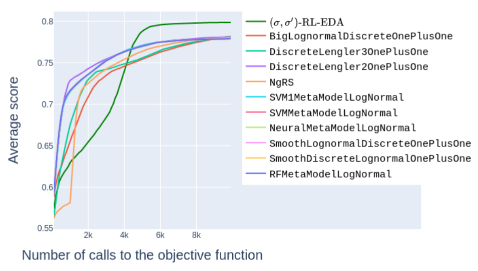
M Ablation studies and sensitivity analyses

In this appendix, we first present two ablation studies aimed at evaluating the impact of the order-invariant reinforcement learning framework used in (σ, σ')-`RL-EDA` (see Section 3.3), which could be partially or totally replaced by naive structural dropout during sampling and/or training.

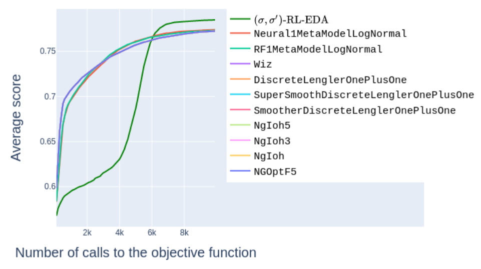
We also investigate the influence of incorporating a known variable interaction graph on the performance of (σ, σ')-`RL-EDA`.



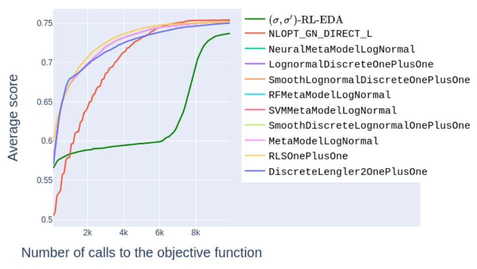
(a) NK3 instances with $n = 128$ and $K = 1$.



(b) NK3 instances with $n = 128$ and $K = 2$.



(c) NK3 instances with $n = 128$ and $K = 4$ and



(d) NK3 instances with $n = 128$ and $K = 8$.

Figure 6: Evolution of the average scores w.r.t. the number of calls to the objective function obtained by (σ, σ')-RL-EDA and the best 10 other competitors for the different type of NK3 instances with $n = 128$.

Furthermore, we conduct a sensitivity analysis of key parameters within the multivariate RL EDA framework, specifically examining the effects of the population size (λ), the KL divergence penalization coefficient (β), various configurations of the g mechanisms employed in the multivariate generative model, and the number of training epochs E at each iteration t of the EDA.

M.1 Impact for using additional structural dropout for generation and training

In this appendix, we aim to test variants of the multivariate RL EDA presented in Section 3.3 ((δ, δ')-RL-EDA, (δ, σ')-RL-EDA, (σ, δ')-RL-EDA, (σ, σ')-RL-EDA), but with additional structural dropout for sampling and training (following the objective (34) combining input dropout and order permutations described in section E).

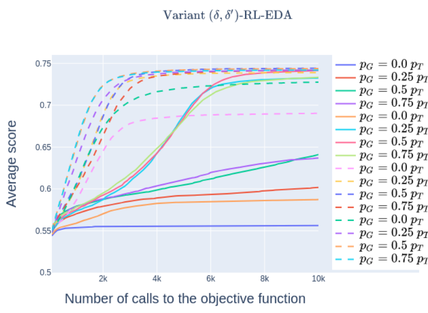
During the generation phase (respectively the training phase) of the EDA, we add a probability $p_G \in \{0.0, 0.25, 0.5, 0.75\}$ (respectively $p_T \in \{0.0, 0.25, 0.5, 0.75\}$) that a parent of a variable in the causal mask is set at the value of zero. Therefore, we test 16 different configurations of structural dropout for each multivariate RL variant.

First, we see in Figure 8a that adding structural dropout during the sampling phase and the training phase can be very beneficial in particular for the variant (δ, δ')-RL-EDA with fix order for both generation and sampling. It helps the model have more diversity during the generation phase of the EDA and to better detect the dependencies between variables during the update phase.

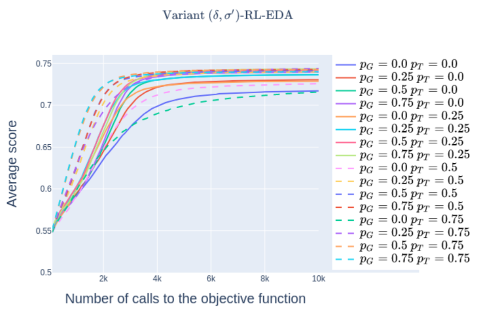
By contrast, adding these structural dropouts for the variant (σ, σ')-RL-EDA in Figure 8d does not improve the results in comparison with the reference version with $p_G = 0.0$ and $p_T = 0.0$ (green solid line), because this version already benefits from structural dropout for sampling and training induced by its double random order sampling process.

Overall, we observe that the reference version $(\sigma, \sigma')-\texttt{RL-EDA}$ without structural dropout performs better with a score of 0.753 in average than all variants across the different combinations of dropout levels used for sampling and training (the best other variant obtains an average score of 0.747). The difference of score is statistically significative according to a t-test with p-value 0.001. It should be noted that it is difficult to obtain an average score higher than 0.006 when the score is already very good for this type of instance. This suggests that the dropout distribution induced by double-order sampling is more advantageous than fine-tuning specific structural dropout values for the generation and update phases of the EDA.

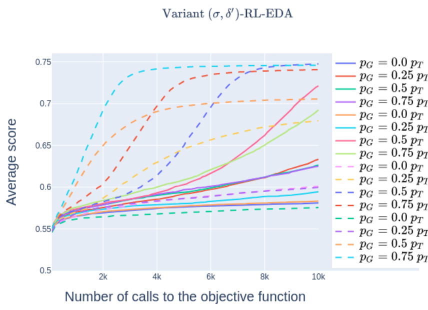
We confirms this results on the large QUBO instances with $N = 256$ and $K = 5$ (see Figure 8. On this distribution of instances our reference variant $(\sigma, \sigma')-\texttt{RL-EDA}$ with $p_G = 0.0$ and $p_T = 0.0$ (green solid line in SubFigure 7d) obtains a score of 2730 in average, while the best other variant $(\sigma, \delta)-\texttt{RL-EDA}$ with dropout ratios $p_G = 0.5$ and $p_T = 0.5$ obtain a score of 2709 in average. The difference of score is statistically significative according to a t-test with p-value 0.1.



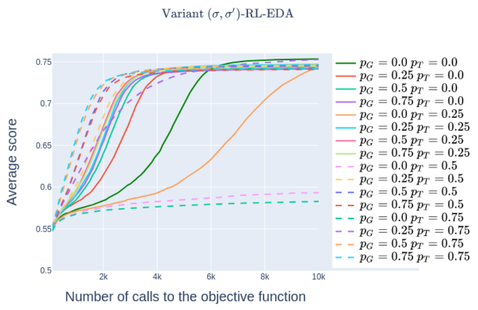
(a) Variant (δ, δ')-RL-EDA



(b) Variant (δ, σ')-RL-EDA



(c) Variant (σ, δ')-RL-EDA



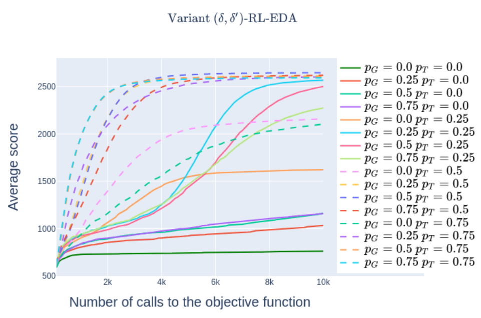
(d) Variant (σ, σ')-RL-EDA

Figure 7: Evolution of the average scores w.r.t. the number of calls to the objective function, obtained by the four different versions of the multivariate RL EDA with additional structural dropout for sampling and training for the instances of the NK landscape problem with $N = 256$ and $K = 4$.

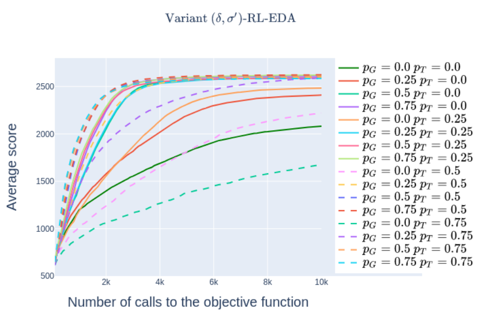
M.2 Impact for using structural dropout instead of causal mask during training

In this appendix, we seek to verify whether the causal used during the EDA training phase can be completely replaced by a structural dropout with a probability $p_T \in \{0.0, 0.25, 0.5, 0.75\}$ for variants with fixed or random orders during generation. These variants without causal mask during training are called $(\delta, p)-\texttt{RL-EDA}$ and $(\sigma, p)-\texttt{RL-EDA}$. We also retain the different structural dropout ratios for generation $p_G \in \{0.0, 0.25, 0.5, 0.75\}$ which is complementary to the mandatory causal mask for generation.

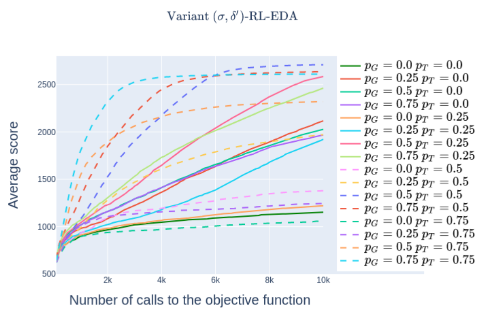
We observe in Figure 9a that the variant $(\delta, p)-\texttt{RL-EDA}$ can obtained at best the same results than the variant $(\delta, \sigma)-\texttt{RL-EDA}$ using fix causal mask during training (see Figure 8a). Symmetrically, the variant $(\sigma, p)-\texttt{RL-EDA}$ obtain also at best the same results than the variant $(\sigma, \delta')-\texttt{RL-EDA}$



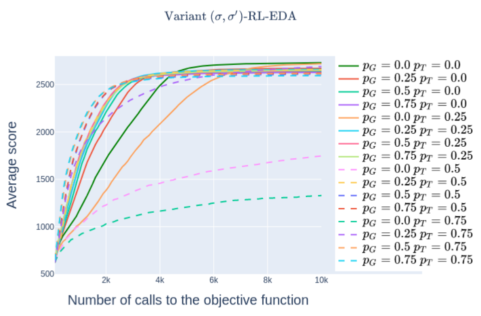
(a) Variant (δ, δ')-RL-EDA



(b) Variant (δ, σ')-RL-EDA



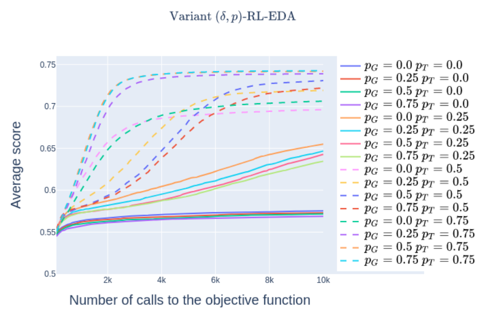
(c) Variant (σ, δ')-RL-EDA



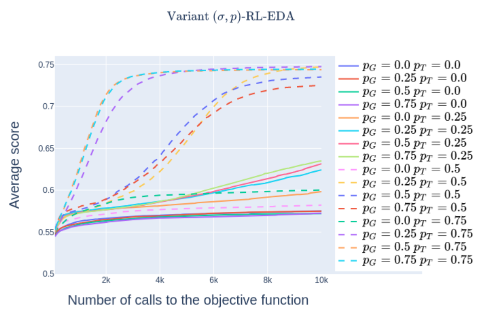
(d) Variant (σ, σ')-RL-EDA

Figure 8: Evolution of the average scores w.r.t. the number of calls to the objective function, obtained by the four different versions of the multivariate RL EDA with additional structural dropout for sampling and training for the instances of the QUBO problem with $N = 256$ and $K = 5$.

(see Figure 8c). However these variants obtain less good results than the reference version (σ, σ')−RL−EDA (green solid line in Figure 8d), which confirm the utility of the specific double uniform distribution of random orders used during the sampling and training phase of the EDA, instead of fine tuned structural dropouts in this context. We confirms this results on the large QUBO instances with $N = 256$ and $K = 5$ (see Figure 10), when comparing the results obtain on these plots with those obtain by the reference version (σ, σ')−RL−EDA on the same distribution of instances (green solid line in Figure 7d).

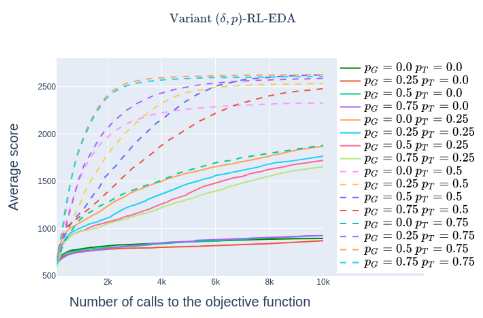


(a) Variant (δ, p)-RL-EDA

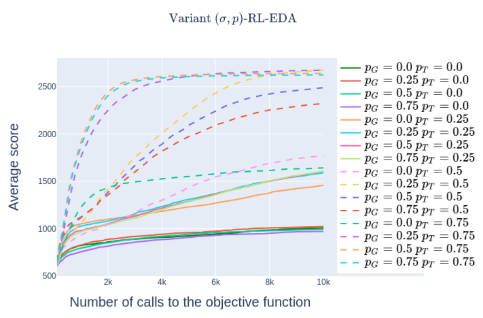


(b) Variant (σ, p)-RL-EDA

Figure 9: Evolution of the average scores w.r.t. the number of calls to the objective function for the variants (δ, p)−RL−EDA and (σ, p)−RL−EDA for the instances of the NK landscape problem with $N = 256$ and $K = 4$.



(a) Variant (δ, p)-RL-EDA



(b) Variant (σ, p)-RL-EDA

Figure 10: Evolution of the average scores w.r.t. the number of calls to the objective function for the variants (δ, p)-`RL-EDA` and (σ, p)-`RL-EDA` for the instances of the QUBO problem with $N = 256$ and $K = 5$.

M.3 Impact of using a known interaction graph between variables

In scenarios where the interaction graph (IG) between variables is assumed to be known—i.e., a gray-box setting (Santana, 2017) —the causal masks used in (σ, σ')-`RL-EDA` can be adapted to respect these structural constraints.

Let A denote the symmetric binary adjacency matrix of the interaction graph, where $a_{i,j} = 1$ indicates that variables X_i and X_j interact in the the evaluation of the objective function f. For example, in the QUBO problem, the objective function is defined as $f(x) = x^\top Qx$, where Q is a symmetric real matrix of size $n \times n$ and coefficients q_{ij}. In this case, the adjacency matrix A is constructed such that $a_{ij} = 1$ if $q_{ij} \neq 0$, and 0 otherwise.

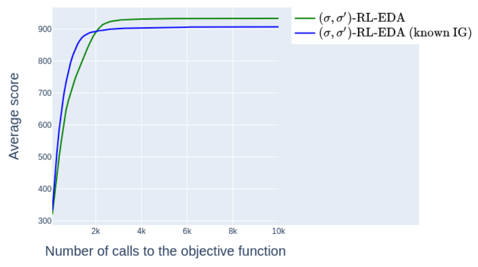
Each causal mask $\sigma(x)_{<k}$ (see Section 3.3) is then adapted to hide values of non adjacent variables in the interaction graph (corresponding to zero coefficients in the adjacency matrix A), in addition to every dimension whose rank in σ is greater or equal than k.

Figure 11 shows the evolution of average scores across 100 independent runs of (σ, σ')-`RL-EDA`, comparing the case with an unknown IG (green curve) to the case with a known IG (blue curve). When comparing the green and blue curves, we observe that providing the interaction graph between variables helps guide the algorithm more effectively at the beginning of the search. Indeed, (σ, σ')-`RL-EDA` with a known IG reaches high-quality solutions more rapidly. However, it is noteworthy that the green curve eventually surpasses the blue one, suggesting that constraining the learning process to the predefined interaction graph may become limiting. Toward the end of the search, generating optimal solutions may benefit from discovering new relationships between variables that are not encoded in the known interaction graph used to compute the objective function. This phenomenon can be attributed to the fact that the learned model of (σ, σ')-`RL-EDA` is not designed to model the full objective function, but rather to approximate the distribution of high-quality solutions within a specific region of the search space.

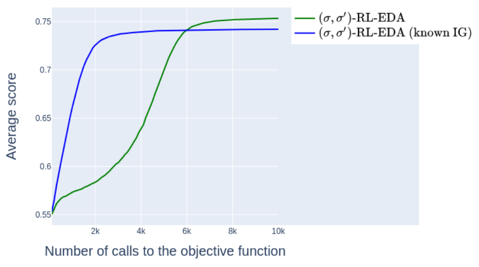
M.4 Sensitivity to the population size

Figure 12 shows the score evolution curves for (σ, σ')-`RL-EDA` with varying population size.

Our analysis reveals that, for the considered instance distributions, a smaller population size tends to promote faster convergence in terms of the number of objective function evaluations. However, this accelerated convergence often comes at the expense of reduced exploration, which can lead the algorithm to suboptimal local solutions. Increasing the population size to $\lambda = 20$ or $\lambda = 50$ improves the average performance previously reported for NK instances with $N = 128$ and $K = 4$ (Figure 12b). In contrast, as shown in Figure 12a, the population size appears to have a negligible impact on performance for QUBO instances.

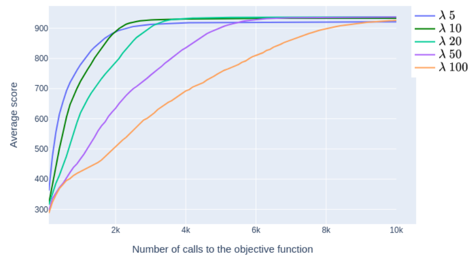


(a) QUBO instances with $n = 128$ and $K = 5$.

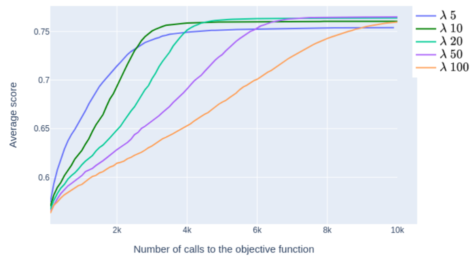


(b) NK instances with $n = 256$ and $K = 4$.

Figure 11: Evolution of the average scores w.r.t. the number of calls to the objective function, obtained by (σ, σ')−RL−EDA with and without known interaction graph.



(a) QUBO instances with $n = 128$ and $K = 5$.

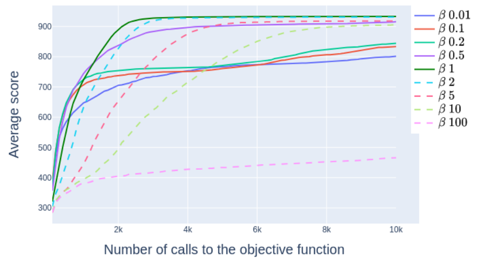


(b) NK instances with $n = 128$ and $K = 4$.

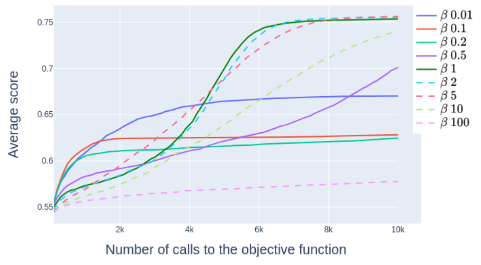
Figure 12: Sensitivity to the population size in (σ, σ')−RL−EDA.

M.5 Sensitivity to the KL penalty coefficient

Figure 13 shows the score evolution curves of (σ, σ')−RL−EDA for different values of the KL penalty coefficient β. By default, this coefficient is set to 1 in (σ, σ')−RL−EDA (green curve). It controls the amplitude of the KL regularization term included in the objective function during the update phase of (σ, σ')−RL−EDA (see Equation 8).



(a) QUBO instances with $n = 128$ and $K = 5$.



(b) NK instances with $n = 256$ and $K = 4$.

Figure 13: Sensitivity to the KL penalty coefficient β in (σ, σ')−RL−EDA.

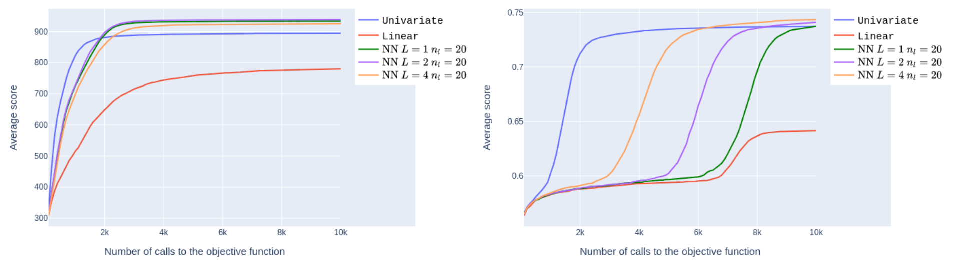
We observe that low values of β lead to faster convergence in terms of objective function evaluations. However, this often results in premature convergence to suboptimal solutions due to insufficient exploration. Conversely, higher values of β help maintain the initial high entropy of the solution distribution for a longer period, thereby promoting broader exploration. Nevertheless, excessively high values—such as $\beta = 100$—can hinder the algorithm's ability to converge toward high-quality solution. These results highlight the critical role of β in balancing exploration and exploitation. For the instance distributions considered and given the evaluation budget, setting β within the range $[1, 5]$ appears to offer a satisfactory trade-off.

M.6 Sensitivity to the logistic regression models used in the Markov Kernels

Figure 14 shows the score evolution of (σ, σ')-RL-EDA for different logistic regression models g used in the generative process of each variable conditioned on the others (see Section 3.1).

The blue curve corresponds to the univariate model, where each variable is generated independently of the others. This model converges the fastest, due to its limited number of parameters. The red curve represents the use of linear logistic regression models. Interestingly, the performance obtained with linear models is even lower than that of the univariate model. This result suggests that it may be preferable to omit interaction modeling entirely rather than attempt to capture complex dependencies using an overly simplistic linear model.

We also evaluate several variants using neural networks of varying depth—specifically with 1, 2, and 4 hidden layers—for each variable. All configurations perform similarly on NK instances with $K = 4$ (Figure 14a), where variable interactions are relatively simple. However, for the more complex categorical NK3 problem with $K = 8$ (Figure 14b), deeper architectures (e.g., the four-hidden-layer model, shown by the orange curve) outperform simpler ones such as the single hidden layer (green curve). This suggests that increased model capacity is beneficial for capturing more complex dependencies. Nevertheless, this improvement comes with increased computational and memory requirements.



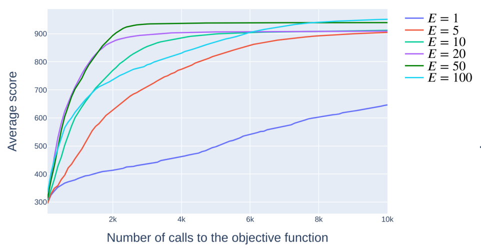
(a) QUBO instances with $n = 128$ and $K = 5$. (b) NK3 instances with $n = 128$ and $K = 8$.

Figure 14: Sensitivity to the logistic regression models used in each conditional generative network of (σ, σ')-RL-EDA. NN corresponds to neural network. L is the number of hidden layer in each neural network and n_l is the number of neurons in each hidden layer.

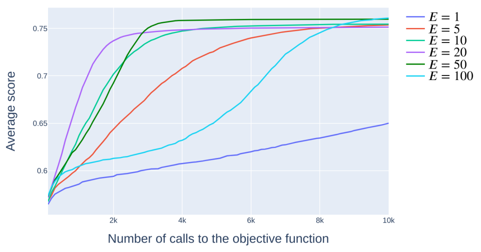
M.7 Sensitivity to the number of training epochs at each generation

Figure 15 shows the score evolution curves of (σ, σ')-RL-EDA for different values of the number of training epochs E (number of permutations) at each iteration t of the RL EDA. By default, this coefficient is set to 50 in (σ, σ')-RL-EDA (green curve).

In Figure 15, we observe that the higher the value of parameter E, the better the long-term results. However, increasing E increases the algorithm's resolution time, which is 5min, 6min, 8min, 11min, 20min, 35min to process 100 hundred instances of size $n = 128$ on a V100 GPU card when E is equal to 1, 5, 10, 20, 50 and 100 respectively.



(a) QUBO instances with $n = 128$ and $K = 5$.



(b) NK instances with $n = 128$ and $K = 4$.

Figure 15: Sensitivity to the number E of training epochs at each generation
.

N Results on NAS-Bench-101 real dataset

The neural architecture search public dataset (Ying et al., 2019) (full NAS-Bench-101 available at `https://github.com/google-research/nasbench`), is a table which maps neural network architectures to their testing metrics. Each architecture is encoded by a set of 26 variables: 21 binary variables and 5 categorical variables, each of which can take one of three values. The resulting search space is therefore $\mathcal{X} = \{0,1\}^{21} \times \{0,1,2\}^{5}$, with $|\mathcal{X}| \approx 510$ million. However, as noted in Ying et al. (2019), a substantial fraction of these configurations correspond to invalid models, which are assigned a testing accuracy of 0 in the dataset. The number of valid architectures, those with an accuracy strictly greater than 0, amounts to 423,000. The goal of this benchmark is to find architectures that have high testing accuracy.

On this benchmark we launch (σ, σ')-RL-EDA with the same hyperparameters used for the synthetic datasets, and compare it with the same competitors as described in the previous subsection, with the same maximum budget of 10,000 calls to the objective function. Figure 16 displays the evolution of the average best accuracy of (σ, σ')-RL-EDA in comparison with the 10 best other baselines, with respect to the number of calls to the objective function. The evolution curves are averaged over 100 independent runs.

The (σ, σ')-RL-EDA algorithm (green line) achieves the best performance both with a budget of 10,000 objective-function evaluations but also under a short budget of only 1,000 evaluations. However the spread with the second best and third methods is not significant at convergence according to a t-test, with only 100 runs. The Tabu search algorithm reaches the second-best performance at the end of the search but performs poorly during the initial phase. This behavior is explained by the large proportion of invalid architectures in the search space: since Tabu Search is a local method, it may struggle to escape extended plateaus of invalid solutions with zero accuracy, making it difficult to reach a valid region of the space. In contrast, evolutionary approaches such as (σ, σ')-RL-EDA, but also MIMIC and the CMA variants, avoid this issue because they sample a diverse set of candidate architectures from the beginning of the search.

O Early-budget behavior and adaptation of the algorithm in this context

Table 5 shows the results of the same experiments as those conducted in Section 4.2 with global results reported in Appendix L, except that the scores are reported after only 1,000 calls to the objective function (i.e., for a small budget) instead of 10,000. In this case, we see that our algorithm (σ, σ')-RL-EDA actually achieves very good results for small binary instances of size 64. However, for more complex instances, larger in size or with more categories, even though it often obtains the best scores after 10,000 steps, as shown in Figure 3 and in Table 4 in Appendix L, it takes longer to converge than other methods, which explains the poor results reported in this Table 5. This is because our EDAs maintain a high degree of diversity in the population at the start of the search, precisely so as not getting stuck too quickly in a local optimum, as we can see in Figure 2. In this regard, we can see that the other EDAs also behave in the same way, as the multivariate EDAs MIMIC and BOA also see their scores deteriorate when the number of iterations is very low. A certain number

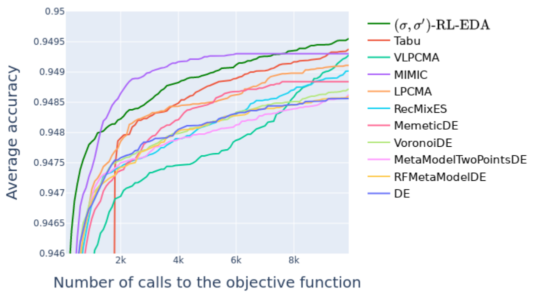


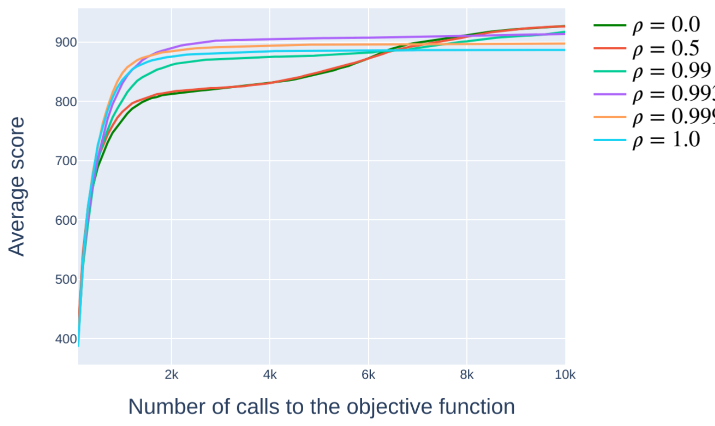
Figure 16: NAS-Bench-101 benchmark. X-axis: number of calls to the objective function. Y-axis: Evolution of the average accuracy of the architectures.

of evaluations are also required for this type of multivariate model in order to properly learn the complex interactions between variables.

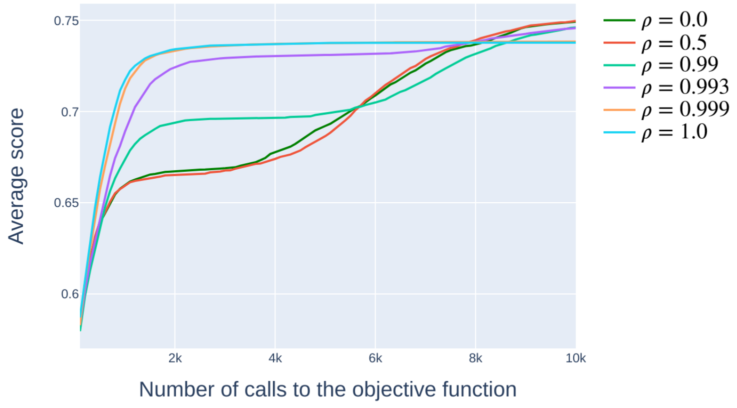
Instances			Methods										
Pb	n	K	(σ, σ')-RL-EDA		PBIL		MIMIC		BOA		Best method (others)		
			Rank	Score	Rank	Score	Rank	Score	Rank	Score	Name	Rank	Score
QUBO	64	0	**1/505**	**195.9***	243/505	159.7	336/505	139.0	362/505	117.9	Carola4	2/505	187.4
QUBO	64	1	**1/505**	**147.7***	127/505	137.3	225/505	129.6	343/505	113.8	LargeCMA	2/505	144.7
QUBO	64	2	**1/505**	**136.6**	123/505	131.3	191/505	127.5	345/505	114.5	LargeCMA	2/505	136.4
QUBO	64	3	**1/505**	**394.8***	275/505	318.9	346/505	271.2	359/505	234.9	Carola4	2/505	378.6
QUBO	64	4	**1/505**	**324.3***	174/505	287.3	306/505	261.1	352/505	230.7	FCarola6	2/505	314.2
QUBO	64	5	**1/505**	**304.3***	197/505	266.7	282/505	252.0	353/505	226.4	NgLglr	2/505	292.5
QUBO	128	0	251/505	354.7	315/505	327.0	355/505	248.0	366/505	224.2	**NLOPT_LN_PRAXIS**	**1/505**	**517.2***
QUBO	128	1	81/505	381.8	246/505	316.7	340/505	266.5	362/505	223.3	**DiscreteLengler2OnePlusOne**	**1/505**	**406.1***
QUBO	128	2	80/505	367.0	246/505	319.7	340/505	269.6	361/505	229.7	**NgIohLn**	**1/505**	**399.3***
QUBO	128	3	249/505	749.54	320/505	661.4	354/505	506.6	368/505	442.7	**NLOPT_LN_PRAXIS**	**1/505**	**1034.5***
QUBO	128	4	98/505	775.9	257/505	645.1	361/505	448.5	254/505	857.3	**DiscreteLengler2OnePlusOne**	**1/505**	**845.8***
QUBO	128	5	179/505	724.3	262/505	629.9	346/505	523.8	365/505	440.8	**DiscreteLengler2OnePlusOne**	**1/505**	**830.8***
QUBO	256	0	359/505	491.5	326/505	623.8	364/505	460.1	369/505	418.6	**Carola1**	**1/505**	**1365.2***
QUBO	256	1	328/505	599.0	278/505	648.7	358/505	485.2	367/505	439.0	**NLOPT_LN_PRAXIS**	**1/505**	**1150.1***
QUBO	256	2	334/505	582.0	359/505	485.1	359/505	485.1	366/505	427.	**NgLglr**	**1/505**	**1083.0***
QUBO	256	3	359/505	992.6	327/505	1262.9	365/505	929.0	368/505	845.3	**NLOPT_LN_PRAXIS**	**1/505**	**2666.7***
QUBO	256	4	334/505	1168.7	271/505	1324.6	358/505	978.8	367/505	856.0	**NLOPT_LN_PRAXIS**	**1/505**	**2280.8***
QUBO	256	5	335/505	1169.0	284/505	1303.5	360/505	977.1	366/505	882.2	**NgLglr**	**1/505**	**2208.7***
NK	64	1	**1/505**	**0.7095***	123/505	0.6876	90/505	0.6953	108/505	0.6914	Neural1MetaModelD	2/505	0.7000
NK	64	2	**1/505**	**0.7378***	162/505	0.6994	141/505	0.7029	200/505	0.6937	LargeDiagCMA	2/505	0.7225
NK	64	4	**1/505**	**0.7341***	308/505	0.6695	334/505	0.6545	342/505	0.6456	CmaFmin2	2/505	0.7187
NK	64	8	335/505	0.6362	355/505	0.6287	359/505	0.6243	363/505	0.6190	**DSsubspace**	**1/505**	**0.7166***
NK	128	1	134/505	0.6658	137/505	0.6616	137/505	0.6616	236/505	0.6447	**RF1MetaModelD**	**1/505**	**0.6883***
NK	128	2	144/505	0.6609	220/505	0.6501	207/505	0.6530	315/505	0.6322	**RF1MetaModelD**	**1/505**	**0.6988**
NK	128	4	316/505	0.6277	318/505	0.6220	342/505	0.6105	355/505	0.6011	**Quad1MetaModelD**	**1/505**	**0.7006***
NK	128	8	359/505	0.5863	353/505	0.5898	361/505	0.5839	360/505	0.5844	**NLOPT_LN_NELDERMEAD**	**1/505**	**0.6978***
NK	256	1	264/505	0.5983	173/505	0.6094	124/505	0.6209	282/505	0.5966	**NLOPT_LN_NELDERMEAD**	**1/505**	**0.6754***
NK	256	2	314/505	0.5923	240/505	0.6071	213/505	0.6103	319/505	0.5901	**LargeDiagCMA**	**1/505**	**0.6809***
NK	256	4	352/505	0.5732	318/505	0.5859	351/505	0.5742	353/505	0.5696	**NLOPT_LN_NELDERMEAD**	**1/505**	**0.6847***
NK	256	8	362/505	0.5598	352	0.5632	364/505	0.5595	401/505	0.5581	**NLOPT_LN_NELDERMEAD**	**1/505**	**0.6760***
NK3	64	1	52/500	0.7228	-	-	71/500	0.7140	45/500	0.7318	**SmallLognormalDiscreteOnePlusOne**	**1/500**	**0.7419***
NK3	64	2	128/500	0.7012	-	-	202/500	0.6804	153/500	0.6934	**NgLglr**	**1/500**	**0.7477***
NK3	64	4	272/500	0.6385	-	-	319/500	0.6252	318/500	0.6252	**NgIohLn**	**1/500**	**0.7358***
NK3	64	8	311/500	0.6201	-	-	335/500	0.6172	320/500	0.6182	**RLSOnePlusOne**	**1/500**	**0.7185***
NK3	128	1	128/500	0.6543	-	-	116/500	0.6689	101/500	0.6846	**DiscreteLengler2OnePlusOne**	**1/500**	**0.7280***
NK3	128	2	159/500	0.6295	-	-	208/500	0.6223	157/500	0.6332	**DiscreteLengler2OnePlusOne**	**1/500**	**0.7285***
NK3	128	4	249/500	0.5946	-	-	271/500	0.5874	268/500	0.5887	**NGOptF5**	**1/500**	**0.7072***
NK3	128	8	286/500	0.5832	-	-	328/500	0.5826	285/500	0.5833	**Carola10**	**1/500**	**0.6918***
NK3	256	1	127/500	0.6143	-	-	117/500	0.6200	110/500	0.6322	**NGOptF5**	**1/500**	**0.7049***
NK3	256	2	212/500	0.5846	-	-	209/500	0.5847	159/500	0.5915	**NGOptF5**	**1/500**	**0.7052***
NK3	256	4	234/500	0.5679	-	-	282/500	0.5621	274/500	0.5633	**Carola1**	**1/500**	**0.6998***
NK3	256	8	314/500	0.5595	-	-	360/500	0.5582	363/500	0.5581	**Cobyla**	**1/500**	**0.6779***

Table 5: Global rankings and average scores obtained by (σ, σ')-RL-EDA and the other EDAs (PBIL, MIMIC, and BOA) are reported. The last columns present the ranking and average score of the best-performing method among the 501 additional algorithms considered (496 for NK3 problems). Rankings are computed over all 505 algorithms (500 for NK3 problems) by comparing the best score achieved **with short budget after 1,000 objective function evaluations**, averaged across 100 independent runs. Bold values highlight the best results among all competing methods. A star associated with the results obtained by (σ, σ')-RL-EDA indicates that it is significantly better in average (over 100 runs) than the best other competitor. A star associated with a result obtained by an other algorithm indicates that it is significantly better in average (over 100 runs) than (σ, σ')-RL-EDA. A difference on the average scores is said statistically significant according to a t-test with p-value 0.001.

Curriculum Adaptation To overcome this problem, we propose adapting the algorithm with a curriculum approach, so that the model is univariate at the start of the search in order to find a good-quality solution more quickly at the beginning, then gradually switches to a multivariate mode. To do this, we revisit the idea of structured dropout introduced in Appendix M.1. At the very beginning of the search, the dropout probabilities p_G^0 and p_T^0 are set to the value of 1, which means that all input variables of all networks are masked, and therefore the model is completely univariate. Then we introduce a coefficient $\rho < 1$ that multiplies these probabilities at each iteration t of the EDA, with the equations $p_G^{t+1} = \rho \times p_G^{t+1}$ and $p_T^{t+1} = \rho \times p_T^{t+1}$, so as to decrease them during the search. When the iteration index t tends towards infinity, p_G^t andp_T^t both tend towards 0, which makes the algorithm return to the standard multivariate model (σ, σ')-$\texttt{RL-EDA}$. In the following figures, we therefore propose a sensitivity analysis for this coefficient ρ for NK and QUBO instances of size 128. We also set $\lambda = 5$ and $E = 100$ instead $\lambda = 10$ and $E = 50$ in order to increase the fast convergence of the algorithm.



(a) QUBO instances with $n = 128$ and $K = 5$.



(b) NK instances with $n = 128$ and $K = 4$.

Figure 17: Sensitivity to the parameter ρ in (σ, σ')-$\texttt{RL-EDA}$.

In Figure 17, we observe that when ρ is close to 1, i.e., when the model is close to the univariate model, the network converges very quickly at the beginning, but converges towards lower scores at the end. Conversely, a lower value of ρ leads to lower scores at the beginning, but these scores end up being higher at the end of the search, because the model learns without loss of information of the context of the full joint distribution, generating higher-quality solutions.

Using a value of $\rho = 0.993$ seems to be a good compromise for obtaining high-quality solutions quickly, as well as good results when the model converges.

However, in order to create an effective version when the budget is limited, and knowing that some of Nevergrad's competitors' algorithms are specially optimized for this purpose, we create a version called Fast-(σ, σ')-RL-EDA with $\rho = 0.999$ and launch it with a budget of 1,000 calls to the objective function. Table 6 shows the results obtained by this variant Fast-(σ, σ')-RL-EDA in comparison with the other methods. We observe on this Table that the version Fast-(σ, σ')-RL-EDA frequently obtains the best results for instances of size $n = 64$ and $n = 128$ for all type of distribution of instances, and results close to those obtain by the best competitors for instances of size 256.

Instances			Methods										
Pb	n	K	Fast-(σ,σ')-RL-EDA		PBIL		MIMIC		BOA		Best method (others)		
			Rank	Score	Rank	Score	Rank	Score	Rank	Score	Name	Rank	Score
QUBO	64	0	**1/505**	**189.5***	243/505	159.7	336/505	139.0	362/505	117.9	Carola4	2/505	187.4
QUBO	64	1	17/505	142.6	127/505	137.3	225/505	129.6	343/505	113.8	**LargeCMA**	**1/505**	**144.7***
QUBO	64	2	21/505	134.3	123/505	131.3	191/505	127.5	345/505	114.5	**LargeCMA**	**1/505**	**136.4***
QUBO	64	3	**1/505**	**396.3***	275/505	318.9	346/505	271.2	359/505	234.9	Carola4	2/505	378.6
QUBO	64	4	**1/505**	**316.5**	174/505	287.3	306/505	261.1	352/505	230.7	FCarola6	2/505	314.2
QUBO	64	5	**1/505**	**297.9***	197/505	266.7	282/505	252.0	353/505	226.4	NgLglr	2/505	292.5
QUBO	128	0	**1/505**	**525.6***	315/505	327.0	355/505	248.0	366/505	224.2	NLOPT_LN_PRAXIS	2/505	517.2
QUBO	128	1	**1/505**	**417.7***	246/505	316.7	340/505	266.5	362/505	223.3	Dis.Lengler2 1+1	2/505	406.1
QUBO	128	2	**1/505**	**402.9***	246/505	319.7	340/505	269.6	361/505	229.7	NgIohLn	2/505	399.3
QUBO	128	3	**1/505**	**1065.3***	320/505	661.4	354/505	506.6	368/505	442.7	NLOPT_LN_PRAXIS	2/505	1034.5
QUBO	128	4	**1/505**	**881.6***	257/505	645.1	361/505	448.5	254/505	857.3	Dis.Lengler2 1+1	2/505	845.8
QUBO	128	5	**1/505**	**847.9***	262/505	629.9	346/505	523.8	365/505	440.8	Dis.Lengler2 1+1	2/505	830.8
QUBO	256	0	49/505	1188.9	326/505	623.8	364/505	460.1	369/505	418.6	**Carola1**	**1/505**	**1365.2***
QUBO	256	1	43/505	1089.5	278/505	648.7	358/505	485.2	367/505	439.0	**NLOPT_LN_PRAXIS**	**1/505**	**1150.1***
QUBO	256	2	31/505	1033.6	359/505	485.1	359/505	485.1	366/505	427.	**NgLglr**	**1/505**	**1083.0***
QUBO	256	3	110/505	2400.6	327/505	1262.9	365/505	929.0	368/505	845.3	**NLOPT_LN_PRAXIS**	**1/505**	**2666.7***
QUBO	256	4	42/505	2233.4	271/505	1324.6	358/505	978.8	367/505	856.0	**NLOPT_LN_PRAXIS**	**1/505**	**2280.8***
QUBO	256	5	42/505	2105.2	284/505	1303.5	360/505	977.1	366/505	882.2	**NgLglr**	**1/505**	**2208.7***
NK	64	1	**1/505**	**0.7051***	123/505	0.6876	90/505	0.6953	108/505	0.6914	Neural1MetaModelD	2/505	0.7000
NK	64	2	**1/505**	**0.7302***	162/505	0.6994	141/505	0.7029	200/505	0.6937	LargeDiagCMA	2/505	0.7225
NK	64	4	**1/505**	**0.7320***	308/505	0.6695	334/505	0.6545	342/505	0.6456	CmaFmin2	2/505	0.7187
NK	64	8	4/505	0.7120	355/505	0.6287	359/505	0.6243	363/505	0.6190	**DSsubspace**	**1/505**	**0.7166**
NK	128	1	**1/505**	**0.6976***	137/505	0.6616	137/505	0.6616	236/505	0.6447	RF1MetaModelD	2/505	0.6883
NK	128	2	**1/505**	**0.7146***	220/505	0.6501	207/505	0.6530	315/505	0.6322	RF1MetaModelD	2/505	0.6988
NK	128	4	**1/505**	**0.7124**	318/505	0.6220	342/505	0.6105	355/505	0.6011	Quad1MetaModelD	2/505	0.7006
NK	128	8	138/505	0.6567	353/505	0.5898	361/505	0.5839	360/505	0.5844	**NLOPT_LN_NELDERMEAD**	**1/505**	**0.6978***
NK	256	1	16/505	0.6694	173/505	0.6094	124/505	0.6209	282/505	0.5966	**NLOPT_LN_NELDERMEAD**	**1/505**	**0.6754***
NK	256	2	40/505	0.6793	240/505	0.6071	213/505	0.6103	319/505	0.5901	**LargeDiagCMA**	**1/505**	**0.6809***
NK	256	4	71/505	0.6565	318/505	0.5859	351/505	0.5742	353/505	0.5696	**NLOPT_LN_NELDERMEAD**	**1/505**	**0.6847***
NK	256	8	164/505	0.6057	352	0.5632	364/505	0.5595	401/505	0.5581	**NLOPT_LN_NELDERMEAD**	**1/505**	**0.6760***
NK3	64	1	**1/500**	**0.7593***	-	-	71/500	0.7140	45/500	0.7318	SmallLognormalDiscreteOnePlusOne	2/500	0.7419
NK3	64	2	**1/500**	**0.7702***	-	-	202/500	0.6804	153/500	0.6934	NgLglr	1/500	0.7477
NK3	64	4	**1/500**	**0.7547***	-	-	319/500	0.6252	318/500	0.6252	NgIohLn	2/500	0.7358
NK3	64	8	2/500	0.7169	-	-	335/500	0.6172	320/500	0.6182	**RLSOnePlusOne**	**1/500**	**0.7185***
NK3	128	1	**1/500**	**0.7482***	-	-	116/500	0.6689	101/500	0.6846	Dis.Lengler2 1+1	2/500	0.7280
NK3	128	2	**1/500**	**0.7457***	-	-	208/500	0.6223	157/500	0.6332	Dis.Lengler2 1+1	2/500	0.7285
NK3	128	4	**1/500**	**0.7277***	-	-	271/500	0.5874	268/500	0.5887	NGOptF5	2/500	0.7072
NK3	128	8	46/500	0.6746	-	-	328/500	0.5826	285/500	0.5833	**Carola10**	**1/500**	**0.6918***
NK3	256	1	28/500	0.6999	-	-	117/500	0.6200	110/500	0.6322	**NGOptF5**	**1/500**	**0.7049***
NK3	256	2	44/500	0.6925	-	-	209/500	0.5847	159/500	0.5915	**NGOptF5**	**1/500**	**0.7052***
NK3	256	4	101/500	0.6571	-	-	282/500	0.5621	274/500	0.5633	**Carola1**	**1/500**	**0.6998***
NK3	256	8	132/500	0.6045	-	-	360/500	0.5582	363/500	0.5581	**Cobyla**	**1/500**	**0.6779***

Table 6: Global rankings and average scores obtained by **Fast-(σ, σ')-RL-EDA** and the other EDAs (PBIL, MIMIC, and BOA) are reported. The last columns present the ranking and average score of the best-performing method among the 501 additional algorithms considered (496 for NK3 problems). Rankings are computed over all 505 algorithms (500 for NK3 problems) by comparing the best score achieved **with short budget after 1,000 objective function evaluations**, averaged across 100 independent runs. Bold values highlight the best results among all competing methods. A star associated the results obtain by Fast-(σ, σ')-RL-EDA indicates that it is significantly better in average (over 100 runs) than the best other competitor. A star associated with a result obtained by an other algorithm indicates that it is significantly better in average (over 100 runs) than Fast-(σ, σ')-RL-EDA. A difference on the average scores is said statistically significant according to a t-test with p-value 0.001.

P Comparison with a variant using a critic neural network

In this appendix, we compare the (σ, σ')-RL-EDA with an alternative version using a critic neural network to compute advantages instead of GRPO advantages described in Section 3.2 and given by (5).

This new variant called (σ, σ')-RL-EDA-Critic uses exactly the same algorithm, expected that advantages for individual i at time step k of the MDP are computed as

$$\hat{A}^{\pi_{\theta^t}}(\sigma^i(x^i)_{<k}, x^i_k) = \alpha(f(x^i) - \hat{V}(\sigma^i(x^i)_{<k}, x^i_k)), \tag{49}$$

with $f(x^i)$ the final score of the complete solution x^i and $\hat{V}(\sigma^i(x^i)_{<k}, x^i_k)$ an estimation of the value of the state $(\sigma^i(x^i)_{<k}, x^i_k)$ given by a critic neural network composed of a set of $(g_{\theta^c_1}, g_{\theta^c_2}, \ldots, g_{\theta^c_n})$ of n neural networks (one for each variable), with exactly the same architecture as the set $(g_{\theta_1}, g_{\theta_2}, \ldots, g_{\theta_n})$ of generative neural network used to build solutions, except that the sigmoid activation function is replaced by an identity function in order to output a value in $\mathbb{R}$. α is a hyperparameter used to adjust the impact of the advantages on the learning process.

At each iteration t of the EDA, at the beginning of the update phase, the n neural networks of the critic are trained in parallel during E epoch to minimize the mean square error between $f(x^i)$ and $\hat{V}(\sigma^i(x^i)_{<k}, x^i_k)$ at time step k and for each individual i.

For each dataset, we evaluated several values of the hyperparameter α from the set $10^{-4}, 10^{-3}, 10^{-2}, 10^{-1}, 1, 10, 100$. The best performance was obtained with $\alpha = 10$ for the NK and NK3 datasets, and with $\alpha = 0.001$ for the QUBO datasets. Table 7 reports the results obtained by the variant incorporating a critic, denoted (σ, σ')-`RL-EDA-Critic`, in comparison with the standard version (σ, σ')-`RL-EDA`. Overall, the standard version outperforms the critic-based variant in most settings.

Furthermore, the critic-based approach exhibits two major drawbacks:

1. it requires nearly twice the computational time required to run the standard version, as the critic must be trained at each generation;
2. it makes the algorithm sensitive to the scale of fitness values, thereby reducing its robustness to the diverse distributions of instances encountered.

Q VARIANT SHARING PARAMETERS OF HIDDEN LAYERS FOR SCALING TO LARGE PROBLEMS

In this appendix, we introduce a variant of the (σ, σ')-`RL-EDA` algorithm using a single MLP $g^n(\theta)$ with 2 hidden layers of 100 neurons and n outputs, instead of the set $(g_{\theta_1}, g_{\theta_2}, \ldots, g_{\theta_n})$ of n MLP, each with a single hidden layer of 20 neurons (see Section 3.1). In g^n_θ, each of the n outputs produces the probability of the value of each variable conditionally on the values of the other variables. This variant, which employs a single MLP denoted g^n_θ, is called (σ, σ')-`RL-EDA-share-params`. All other hyperparameters are identical to those used in the standard version.

Table 8 reports the results obtained by (σ, σ')-`RL-EDA-share-params` in comparison with the standard version (σ, σ')-`RL-EDA`. Overall, the standard version (σ, σ')-`RL-EDA` is generally slightly more effective than (σ, σ')-`RL-EDA-share-params` on these small and medium sized datasets. This suggests that employing one MLP per variable contributes to a more stable learning process during the search for these sizes of instances.

Then, we generated a new larger dataset of 10 NK instances of size $n = 1028$ and $K = 8$, and launched the two variants (σ, σ')-`RL-EDA` and (σ, σ')-`RL-EDA-share-params` on these large instances.

First of all, we notice that the variant sharing parameters scales very well in term of computational time required in comparison with the standard version. It required 1h30 to process the 10 instances of size $n = 1028$ with a budget of 10,000 evaluations for the variant (σ, σ')-`RL-EDA-share-params`, while it required more than 10 hours for the standard version (σ, σ')-`RL-EDA` to perform the same task, which can be explained by the much lower number of parameters to be learned for the version with shared parameters.

Figure 18 shows the evolution of the average results over 100 runs of the two variants (σ, σ')-`RL-EDA` and (σ, σ')-`RL-EDA-share-params` on these 10 large instances with 10 independent restarts on each instance, in comparison with the best other Nevegrad competitors as well as the other EDAs and the Tabu search.

First, we observe that the Tabu method (dotted red line), which was very good for small instances, does not scale at all. This is because the method makes fewer than 10 improvements with a budget of 10,000, since at each step it must evaluate its entire neighborhood of size 1024 to choose which action to perform.

Instances			Methods	
Pb	n	K	(σ,σ')-RL-EDA	(σ,σ')-RL-EDA-critic
QUBO	64	0	**200.8***	195.3
QUBO	64	1	**148.8***	146.7
QUBO	64	2	**138.1***	134.7
QUBO	64	3	**411.2***	407.7
QUBO	64	4	326.1	**326.4**
QUBO	64	5	**309.4***	303.6
QUBO	128	0	**593.7***	560.7
QUBO	128	1	**449.2***	432.5
QUBO	128	2	**437.1***	412.6
QUBO	128	3	**1227.2***	943.2
QUBO	128	4	**955.4***	781.9
QUBO	128	5	**933.3***	768.4
QUBO	256	0	**1697.7***	1119.0
QUBO	256	1	**1367.7***	596.9
QUBO	256	2	**1304.1***	944.8
QUBO	256	3	**3436.8***	1645.9
QUBO	256	4	**2769.0***	1500.4
QUBO	256	5	**2730.1***	1491.6
NK	64	1	**0.7103**	0.7099
NK	64	2	**0.7420**	0.7413
NK	64	4	**0.7523**	0.7495
NK	64	8	0.7379	**0.7420***
NK	128	1	**0.7100**	0.7086
NK	128	2	**0.7375***	0.7355
NK	128	4	**0.7603**	0.7574
NK	128	8	0.7369	**0.7408***
NK	256	1	**0.7071**	0.706
NK	256	2	**0.7364***	0.7349
NK	256	4	**0.7534**	0.7527
NK	256	8	**0.7232***	0.696
NK3	64	1	0.7818	**0.7861***
NK3	64	2	0.8095	**0.8114**
NK3	64	4	0.8004	**0.8016**
NK3	64	8	**0.7473**	0.7416
NK3	128	1	0.7876	**0.7957***
NK3	128	2	0.7986	**0.8101***
NK3	128	4	0.7847	**0.7988***
NK3	128	8	**0.7373***	0.6031
NK3	256	1	0.7763	**0.7811***
NK3	256	2	0.7801	**0.7802**
NK3	256	4	**0.7615***	0.6342
NK3	256	8	**0.7213***	0.5723

Table 7: Average scores obtained by (σ,σ')-RL-EDA and its variant (σ,σ')-RL-EDA-critic. Bold values highlight the best results. A star associated with the results indicates that it is significantly better in average (over 100 runs). A difference on the average scores is said statistically significant according to a t-test with p-value 0.001.

We then notice that our standard version (σ,σ')-RL-EDA also performs very poorly. This is because each generation of variables is produced by a small network with a hidden layer of size 20 that takes the other 1023 variables as input. This becomes too small to properly model the complex interactions between the variables.

However, we can see that the new variant sharing parameters called (σ,σ')-RL-EDA-share-params yields very good results (green dotted line), in comparison with the other best competitors.

Instances			Methods	
Pb	n	K	(σ,σ')-RL-EDA	(σ,σ')-RL-EDA-share-params
QUBO	64	0	**200.8**	198.7
QUBO	64	1	**148.8***	145.9
QUBO	64	2	**138.1**	137.9
QUBO	64	3	411.2	**415.4**
QUBO	64	4	**326.1***	323.6
QUBO	64	5	**309.4**	309.26
QUBO	128	0	**593.7***	584.8
QUBO	128	1	**449.2***	437.9
QUBO	128	2	**437.1***	429.5
QUBO	128	3	**1227.2***	1211.3
QUBO	128	4	**955.4***	944.6
QUBO	128	5	**933.3***	920.9
QUBO	256	0	**1697.7***	1669.6
QUBO	256	1	**1367.7***	1337.4
QUBO	256	2	**1304.1***	1272.9
QUBO	256	3	**3436.8***	3400.9
QUBO	256	4	**2769.0***	2696.9
QUBO	256	5	**2730.1***	2654.6
NK	64	1	0.7103	0.7103
NK	64	2	**0.7420**	0.7402
NK	64	4	**0.7523**	0.7521
NK	64	8	**0.7379**	0.7367
NK	128	1	**0.7100**	0.7094
NK	128	2	**0.7375***	0.7360
NK	128	4	**0.7603**	0.7569
NK	128	8	**0.7369**	0.7331
NK	256	1	**0.7071**	0.7065
NK	256	2	**0.7364***	0.7352
NK	256	4	**0.7534***	0.7478
NK	256	8	0.7232	**0.7243**
NK3	64	1	0.7818	**0.7835***
NK3	64	2	**0.8095**	0.8079
NK3	64	4	**0.8004**	0.7933
NK3	64	8	0.7473	**0.7478**
NK3	128	1	**0.7876**	0.7816
NK3	128	2	**0.7986**	0.7908
NK3	128	4	**0.7847**	0.7715
NK3	128	8	**0.7373***	0.7271
NK3	256	1	**0.7763***	0.7553
NK3	256	2	**0.7801**	0.7601
NK3	256	4	**0.7615***	0.7434
NK3	256	8	**0.7213***	0.7105

Table 8: Average scores obtained by (σ,σ')-RL-EDA and its variant (σ,σ')-RL-EDA-share-params. Bold values highlight the best results. A star associated with the results indicates that it is significantly better in average (over 100 runs). A difference on the average scores is said statistically significant according to a t-test with p-value 0.001.

R ABLATION STUDY: NON AUTO-REGRESSIVE GENERATION (USING GIBBS SAMPLING)

In this appendix, we propose a baseline variant of the algorithm, where the sequential order of generation is replaced by a Gibbs sampling. This is an other way to build an order-invariant RL EDA.

In this case, in order to construct a complete solution x, we start with $x^0 = (x_1^0, x_2^0, \dots, x_n^0) = (0, 0, \dots, 0)$, then at iteration ℓ, given a sample $x^{(\ell)} = (x_1^{(\ell)}, x_2^{(\ell)}, \dots, x_n^{(\ell)})$, to obtain the next sample $x^{(\ell+1)} = (x_1^{(\ell+1)}, x_2^{(\ell+1)}, \dots, x_n^{(\ell+1)})$, we sample each component $x_j^{(\ell+1)}$ conditioned on

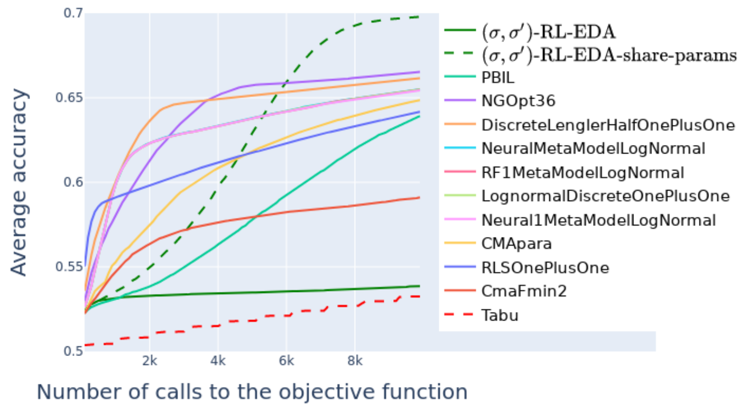


Figure 18: NK instances with $n = 1024$ and $K = 8$.

all other variable values sampled so far, such that $\pi_\theta(x_j^{(\ell+1)} = 1 | x_{-j}^{(\ell)}) = \text{sigmoid}(g_{\theta_j}(x_{-j}^{(\ell)}))$, with $x_{-j}^{(\ell)} = (x_1^{(\ell)}, \ldots, x_{j-1}^{(\ell)}, 0, x_{j+1}^{(\ell)}, \ldots, x_n^{(\ell+1)})$ corresponding to the vector $x_j^{(\ell)}$, but with a 0 in position j. When $\ell \to \infty$, this process allows to obtain a sampling of the joint multivariate distribution. However in practice we restrict this sampling to G iterations.

In this variant without order, during training, the GRPO objective to maximize becomes

$$\hat{L}_\lambda(\theta) = \frac{1}{\lambda} \sum_{(x^i,\sigma^i)\in\Gamma_\lambda^t} \sum_{k=1}^{n} \left[\frac{\pi_\theta(x_k^i | x_{-k}^i)}{\pi_{\theta^t}(x_k^i | x_{-k}^i)} \hat{A}_{\Gamma_\lambda^t}(x) - \beta D_{\text{KL}}\left(\pi_{\theta^t}(\cdot | x_{-k}^i) \,\|\, \pi_\theta(\cdot | x_{-k}^i)\right) \right]. \qquad (50)$$

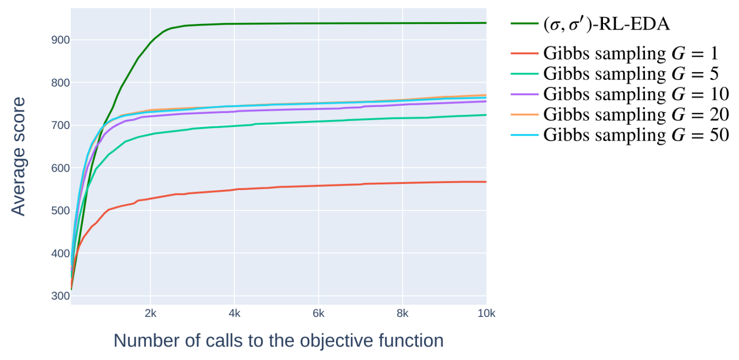
Figure 19 displays the results obtain by this variant with Gibbs sampling for different number G of iterations during sampling. The figure indicates that increasing the value of G leads to improved performance. However, these gains rapidly plateau and remain below those achieved by the standard variant (σ, σ')-RL-EDA, which employs a sequential generation of variables based on randomly sampled permutation masks during both training and inference, which allow to better uncover dependency relationships between variables (see section E for discussions about what can bring samplings of different causal masks at train time, in term of residual structuration of the networks).

Moreover, the Gibbs-sampling version incurs substantially higher computational costs during the variable-generation phase, as it requires multiple re-samplings of each variable. In contrast, the standard approach assigns a value to each variable only once, resulting in significantly lower computational overhead.

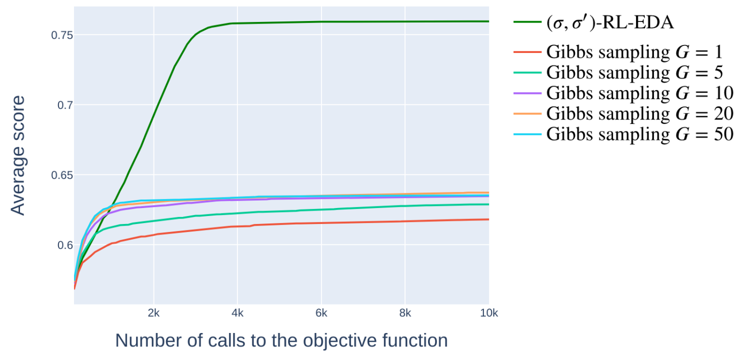
S Nevergrad competing algorithms

It is important to note that some algorithms in the library are primarily designed for continuous optimization—such as various variants of Particle Swarm Optimization (`PSO`) and `CMA-ES`— and are not expected to perform competitively on discrete problems. Nevertheless, `Nevergrad` (Rapin & Teytaud, 2018) also includes a wide range of algorithms specifically tailored for large-scale discrete black-box optimization. The algorithms of the `Nevergrad` library can be grouped into the following categories:

- **Memetic and Genetic Algorithms**, such as `cGA` and `discretememetic`.
- **Discrete** $(1+1)$ **Evolutionary Algorithms**, including variants with adaptive mutation rates like `DiscreteLengler2OnePlusOne` and `FastGADiscreteOnePlusOne`.



(a) QUBO instances with $n = 128$ and $K = 5$.



(b) NK instances with $n = 128$ and $K = 4$.

Figure 19: Evolution of the score of a variant with Gibbs sampling with different numbers of iterations G for the sampling, in comparison with the standard version (σ, σ')-RL-EDA

- **Differential Evolution algorithms**, e.g., DiscreteDE, LhsHSDE.
- **Chaining Algorithms**, which are meta-algorithms applying several baseline algorithms in sequence, such as ChainDEwithLHS30, Carola1, ..., Carola15.
- **Portfolio Algorithms**, including NGOpt, NgIoh, and Wiz, which select low-level algorithms based on problem dimension and budget.
- **Adaptive Portfolio Algorithms**, which test several algorithms during early search phases before selecting one for later stages, e.g., PolyLN.
- **Learning Meta-Models**, which approximate the optimum using supervised models (e.g., random forests, neural networks, SVMs) trained on the best solutions generated by low-level algorithms. Example include RF1MetaModel, Neural1MetaModelOnePlusOne, and SVM1MetaModelD.

Additional Tabu Search Algorithm : Given a solution x, Tabu explores its neighborhood by changing the value of a discrete variable x_j, thereby generating a neighbor x' differing from x in exactly one component. At each iteration, the best eligible neighbor with respect to the objective function f is selected. A move is considered eligible if it is not forbidden by the tabu list, unless it improves upon the best solution found so far. After the value of a variable x_j is changed, it becomes tabu for the next T iterations. In many effective QUBO implementations, T is defined as $\alpha n + R$, where $R \in \{1, \ldots, 10\}$ is a random integer and α is a hyperparameter typically set to 0.1. We retain this configuration in our experiments.

Here we provide the complete list of all competing algorithm of the 1.0.12 Nevergrad library used in the experiments (sorted by name). Detailed documentation and source code of these algorithms are available at https://facebookresearch.github.io/nevergrad.

AdaptiveDiscreteOnePlusOne, AlmostRotationInvariantDE, AlmostRotationInvariantDEAndBigPop, AnisoEMNA, AnisoEMNATBPSA, AnisotropicAdaptiveDiscreteOnePlusOne, ASCMADEthird, AvgHammersleySearch, AvgHammersleySearchPlusMiddlePoint, AvgMetaRecenteringNoHull, AvgRandomSearch, BAR, BAR2, BAR3, BAR4, BFGS, BFGSCMA, BFGSCMAPlus, BigLognormalDiscreteOnePlusOne, BPRotationInvariantDE, Carola1, Carola2, Carola3, Carola4, Carola5, Carola6, Carola7, Carola8, Carola9, Carola10, Carola11, Carola13, Carola14, Carola15, CauchyLHSSearch, CauchyOnePlusOne, CauchyRandomSearch, CauchyScrHammersleySearch, cGA, ChainCMAPowell, ChainCMASQP, ChainCMAwithLHS, ChainCMAwithLHS30, ChainCMAwithLHSdim, ChainCMAwithLHSsqrt, ChainCMAwithMetaRecentering, ChainCMAwithMetaRecentering30, ChainCMAwithMetaRecenteringdim, ChainCMAwithMetaRecenteringsqrt, ChainCMAwithR, ChainCMAwithR30, ChainCMAwithRdim, ChainCMAwithRsqrt, ChainDE, ChainDEwithLHS, ChainDEwithLHS30, ChainDEwithLHSdim, ChainDEwithLHSsqrt, ChainDEwithMetaRecentering, ChainDEwithMetaRecentering30, ChainDEwithMetaRecenteringdim, ChainDEwithMetaRecenteringsqrt, ChainDEwithMetaTuneRecentering, ChainDEwithMetaTuneRecentering30, ChainDEwithMetaTuneRecenteringdim, ChainDEwithMetaTuneRecenteringsqrt, ChainDEwithR, ChainDEwithR30, ChainDEwithRdim, ChainDEwithRsqrt, ChainDiagonalCMAPowell, ChainDSPowell, ChainMetaModelDSSQP, ChainMetaModelPowell, ChainMetaModelSQP, ChainNaiveTBPSACMAPowell, ChainNaiveTBPSAPowell, ChainPSOwithLHS, ChainPSOwithLHS30, ChainPSOwithLHSdim, ChainPSOwithLHSsqrt, ChainPSOwithMetaRecentering, ChainPSOwithMetaRecentering30, ChainPSOwithMetaRecenteringdim, ChainPSOwithMetaRecenteringsqrt, ChainPSOwithR, ChainPSOwithR30, ChainPSOwithRdim, ChainPSOwithRsqrt, ChoiceBase, CLengler, CM, CMA, CMAbounded, CmaFmin2, CMAL, CMAL2, CMAL3, CMALL, CMALn, CMALS, CMALYS, CMandAS2, CMandAS3, CMApara, CMARS, CMASL, CMASL2, CMASL3, CMAsmall, CMAstd, CMAtuning, Cobyla, CSEC, CSEC10, CSEC11, DE, DiagonalCMA, DiscreteBSOOnePlusOne, DiscreteDE, DiscreteDoerrOnePlusOne, DiscreteLengler2OnePlusOne, DiscreteLengler3OnePlusOne, DiscreteLenglerFourthOnePlusOne, DiscreteLenglerHalfOnePlusOne, DiscreteLenglerOnePlusOne, DiscreteLenglerOnePlusOneT, discretememetic, DiscreteNoisy13Splits, DiscreteOnePlusOne, DiscreteOnePlusOneT, DoubleFastGADiscreteOnePlusOne, DoubleFastGAOptimisticNoisyDiscreteOnePlusOne, DS2, DS3p, DS4, DS5, DS6, DS8, DS9, DS14, DSbase, DSproba, DSsubspace, ECMA, EDA, EDCMA, ES, F2SQPCMA, F3SQPCMA, FastGADiscreteOnePlusOne, FastGANoisyDiscreteOnePlusOne, FastGAOptimisticNoisyDiscreteOnePlusOne, FCarola6, FCMA, FCMAp13, FCMAs03, file, ForceMultiCobyla, FSQPCMA, GeneticDE, HaltonSearch, HaltonSearchPlusMiddlePoint, HammersleySearch, HammersleySearchPlusMiddlePoint, HSDE, HugeLognormalDiscreteOnePlusOne, HullAvgMetaRecentering, HullAvgMetaTuneRecentering, HullCenterHullAvgCauchyLHSSearch, HullCenterHullAvgCauchyScrHammersleySearch, HullCenterHullAvgLargeHammersleySearch, HullCenterHullAvgLHSSearch, HullCenterHullAvgRandomSearch, HullCenterHullAvgScrHaltonSearch, HullCenterHullAvgScrHaltonSearchPlusMiddlePoint, HullCenterHullAvgScrHammersleySearch, HullCenterHullAvgScrHammersleySearchPlusMiddlePoint, IsoEMNA, IsoEMNATBPSA, LargeCMA, LargeDiagCMA, LargeHaltonSearch, LBFGSB, LhsDE, LhsHSDE, LHSSearch, LocalBFGS, LogBFGSCMA, LogBFGSCMAPlus, LogMultiBFGS, LogMultiBFGSPlus, LognormalDiscreteOnePlusOne, LogSQPCMA, LogSQPCMAPlus, LPCMA, LPSDE, LQODE, LQOTPDE, LSCMA, LSDE, ManyLN, MaxRecombiningDiscreteLenglerOnePlusOne, MemeticDE, MetaCauchyRecentering, MetaCMA, MetaModel, MetaModelDE, MetaModelDiagonalCMA, MetaModelDSproba, MetaModelFmin2, MetaModelLogNormal, MetaModelOnePlusOne, MetaModelPSO, MetaModelQODE, MetaModelTwoPointsDE, MetaNGOpt10, MetaRecentering, MetaTuneRecentering, MicroCMA, MicroSPSA, MicroSQP, MilliCMA, MiniDE, MiniLhsDE, MiniQrDE, MinRecombiningDiscreteLenglerOnePlusOne, MixDeterministicRL, MixES, MultiBFGS, MultiBFGSPlus, MultiCMA, MultiCobyla, MultiCobylaPlus, MultiDiscrete, MultiDS, MultiLN, MultiScaleCMA, MultiSQP, MultiSQPPlus, MutDE, NaiveAnisoEMNA, NaiveAnisoEMNATBPSA, NaiveIsoEMNA, NaiveIsoEMNATBPSA, NaiveTBPSA, NelderMead, Neural1MetaModel, Neural1MetaModelD, Neural1MetaModelE, Neural1MetaModelLogNormal, NeuralMetaModel, NeuralMetaModelDE, NeuralMetaModelLogNormal, NeuralMetaModelTwoPointsDE, NgDS, NgDS11, NgDS2, NgDS3, NGDSRW, NgIoh, NgIoh2, NgIoh3, NgIoh4, NgIoh5, NgIoh6, NgIoh7, NgIoh8, NgIoh9, NgIoh10, NgIoh11, NgIoh12, NgIoh12b, NgIoh13, NgIoh13b, NgIoh14, NgIoh14b, NgIoh15, NgIoh15b, NgIoh16, NgIoh17, NgIoh18, NgIoh19, NgIoh20, NgIoh21, NgIohLn, NgIohMLn, NgIohRS, NgIohRW2, NgIohTuned, NgLglr, NgLn, NGO, NGOpt, NGOpt10, NGOpt15, NGOpt16, NGOpt36, NGOpt39, NGOpt4, NGOpt8, NGOptBase, NGOptDSBase, NGOptF, NGOptF2, NGOptF3, NGOptF5, NGOptRW, NGOptSingle16, NGOptSingle25, NGOptSingle9, NgRS, NLOPT_GN_CRS2_LM, NLOPT_GN_DIRECT, NLOPT_GN_DIRECT_L, NLOPT_GN_ESCH, NLOPT_GN_ISRES, NLOPT_LN_NELDERMEAD, NLOPT_LN_PRAXIS, NLOPT_LN_SBPLX, Noisy13Splits, NoisyBandit, NoisyDE, NoisyDiscreteOnePlusOne, NoisyOnePlusOne, NoisyRL1, NoisyRL2, NoisyRL3, NonNSGAIIES, OldCMA, OLNDiscreteOnePlusOne, OnePlusLambda, OnePlusOne, OnePointDE, OnePtRecombiningDiscreteLenglerOnePlusOne, OpoDE, OpoTinyDE, OptimisticDiscreteOnePlusOne, OptimisticNoisyOnePlusOne, ORandomSearch, OScrHammersleySearch, ParametrizationDE, ParaPortfolio, pCarola6, PCarola6, PolyCMA, PolyLN, Portfolio, PortfolioDiscreteOnePlusOne, PortfolioDiscreteOnePlusOneT, PortfolioNoisyDiscreteOnePlusOne, PortfolioOptimisticNoisyDiscreteOnePlusOne, Powell, PSO, QNDE, QODE, QOPSO, QORandomSearch, QORealSpacePSO, QOScrHammersleySearch, QOTPDE, QrDE, Quad1MetaModel, Quad1MetaModelD, Quad1MetaModelE, RandomScaleRandomSearch, RandomScaleRandomSearchPlusMiddlePoint, RandomSearch, RandomSearchPlusMiddlePoint, RandRecombiningDiscreteLenglerOnePlusOne, RandRecombiningDiscreteLognormalOnePlusOne, RBFGS, RealSpacePSO, RecES, RecMixES, RecMutDE, RecombiningDiscreteLenglerOnePlusOne, RecombiningDiscreteLognormalOnePlusOne, RecombiningGA, RecombiningOptimisticNoisyDiscreteOnePlusOne, RecombiningPortfolioDiscreteOnePlusOne, RecombiningPortfolioOptimisticNoisyDiscreteOnePlusOne, RescaledCMA, RescaleScrHammersleySearch, RF1MetaModel, RF1MetaModelD, RF1MetaModelE, RF1MetaModelLogNormal, RFMetaModel, RFMetaModelDE, RFMetaModelLogNormal, RFMetaModelOnePlusOne, RFMetaModelPSO, RFMetaModelTwoPointsDE, RLSOnePlusOne, RotatedRecombiningGA, RotatedTwoPointsDE, RotationInvariantDE, RPowell, RSQP, SADiscreteLenglerOnePlusOneExp09, SADiscreteLenglerOnePlusOneExp099, SADiscreteLenglerOnePlusOneExp09Auto, SADiscreteLenglerOnePlusOneLin1, SADiscreteLenglerOnePlusOneLin100, SADiscreteLenglerOnePlusOneLinAuto, SADiscreteOnePlusOneExp09, SADiscreteOnePlusOneExp099, SADiscreteOnePlusOneLin100, ScrHaltonSearch, ScrHaltonSearchPlusMiddlePoint, ScrHammersleySearch, ScrHammersleySearchPlusMiddlePoint, SDiagonalCMA, Shiwa, SmallLognormalDiscreteOnePlusOne, SmoothAdaptiveDiscreteOnePlusOne, SmoothDiscreteLenglerOnePlusOne, SmoothDiscreteLognormalOnePlusOne, SmoothDiscreteOnePlusOne, SmoothElitistRandRecombiningDiscreteLenglerOnePlusOne, SmoothElitistRandRecombiningDiscreteLognormalOnePlusOne, SmoothElitistRecombiningDiscreteLenglerOnePlusOne, SmootherDiscreteLenglerOnePlusOne, SmoothLognormalDiscreteOnePlusOne, SmoothPortfolioDiscreteOnePlusOne, SmoothRecombiningDiscreteLenglerOnePlusOne, SmoothRecombiningPortfolioDiscreteOnePlusOne, SODE, SOPSO, SparseDiscreteOnePlusOne, SparseDoubleFastGADiscreteOnePlusOne, SparseOrNot, SpecialRL, SplitCMA, SplitDE, SplitPSO, SplitQODE, SplitSQOPSO, SplitTwoPointsDE, SPQODE, SPSA, SQOPSO, SQOPSODCMA, SQOPSODCMA20, SQORealSpacePSO, SQP, SQPCMA, SQPCMAPlus, SqrtBFGSCMA, SqrtBFGSCMAPlus, SqrtMultiBFGS, SqrtMultiBFGSPlus, SqrtSQPCMA, SqrtSQPCMAPlus, StupidRandom, SuperSmoothDiscreteLenglerOnePlusOne, SuperSmoothElitistRecombiningDiscreteLenglerOnePlusOne, SuperSmoothRecombiningDiscreteLenglerOnePlusOne, SuperSmoothRecombiningDiscreteLognormalOnePlusOne, SuperSmoothTinyLognormalDiscreteOnePlusOne, SVM1MetaModel, SVM1MetaModelD, SVM1MetaModelE, SVM1MetaModelLogNormal, SVMMetaModel, SVMMetaModelDE, SVMMetaModelLogNormal, SVMMetaModelPSO, SVMMetaModelTwoPointsDE, TBPSA, TEAvgCauchyLHSSearch, TEAvgCauchyScrHammersleySearch, TEAvgLHSSearch, TEAvgRandomSearch, TEAvgScrHammersleySearch, TEAvgScrHammersleySearchPlusMiddlePoint, TinyCMA, TinyLhsDE, TinyLognormalDiscreteOnePlusOne, TinyQODE, TinySPSA, TinySQP, TripleCMA, TripleDiagonalCMA, TripleOnePlusOne, TwoPointsDE, TwoPtRecombiningDiscreteLenglerOnePlusOne, UltraSmoothDiscreteLenglerOnePlusOne, UltraSmoothElitistRecombiningDiscreteLenglerOnePlusOne, UltraSmoothElitistRecombiningDiscreteLognormalOnePlusOne, UltraSmoothRecombiningDiscreteLenglerOnePlusOne, VastDE, VastLengler, VLPCMA, VoronoiDE, Wiz, XLognormalDiscreteOnePlusOne, XSmallLognormalDiscreteOnePlusOne, YoSmoothDiscreteLenglerOnePlusOne, Zero.

T LLM Usage Declaration

During the preparation of this manuscript, we used Large Language Models to assist with text clarity, grammar, and formulation, particularly in polishing the abstract and certain explanatory sentences. The scientific content, experimental design, results, and interpretations were entirely conceived and written by the authors. LLMs were not used to generate original ideas, proofs, or analyses; its contribution was limited to language refinement.