Emergence of new local search algorithms with neuro-evolution

Olivier Goudet, Mohamed Salim Amri Sakhri, Adrien Goëffon, Frédéric Saubion

Emergence of new local search algorithms with neuro-evolution

Olivier Goudet, Mohamed Salim Amri Sakhri,
Adrien Goëffon, and Frédéric Saubion

LERIA, Université d'Angers, 2 Boulevard Lavoisier, 49045 Angers, France
{olivier.goudet,mohamedsalim.amrisakhri,adrien.goeffon,
frederic.saubion}@univ-angers.fr

Abstract. This paper explores a novel approach aimed at overcoming existing challenges in the realm of local search algorithms. The main objective is to better manage information within these algorithms, while retaining simplicity and generality in their core components. Our goal is to equip a neural network with the same information as the basic local search and, after a training phase, use the neural network as the fundamental move component within a straightforward local search process. To assess the efficiency of this approach, we develop an experimental setup centered around NK landscape problems, offering the flexibility to adjust problem size and ruggedness. This approach offers a promising avenue for the emergence of new local search algorithms and the improvement of their problem-solving capabilities for black-box problems.

Keywords: Neuro-evolution, Local search, Black-box optimization, NK landscapes

1 Introduction

Local search (LS) algorithms are commonly used to heuristically solve discrete optimization problems [10]. LS algorithms are usually composed of several components: a search space, a neighborhood relation, an evaluation function, and a selection strategy. The optimization problem instance to be solved can be fully defined by its set of feasible solutions —the decision space— and an objective function that must be optimized. A classic and direct use of local search, when applicable, is to consider the decision space as the search space, the objective function as the evaluation function, and a natural neighborhood relation, defined from an elementary transformation function (move) such as bitflip for pseudo-Boolean problems, or induced by specific operators for permutation problems [22].

Starting from an initial random solution, various components collaborate to drive the search towards optimal solutions. The effectiveness of this search process depends on the complexity of the problem, including factors such as deception and other structural characteristics [2,13,28,29]. The strategy to advance in the search involves selecting neighboring solutions based on their evaluations, using a wide variety of criteria. These criteria can range from simple

ones, such as choosing neighbors with better or the best evaluations, to more intricate approaches involving stochastic methods. The strategy is often derived from metaheuristic frameworks based on local search such as tabu search [6] or iterated local search [15]. The goal is to effectively leverage the available local and partial knowledge of the landscape to identify the most promising search paths that lead to optimal solutions.

Fitness landscape analysis [16] provides the optimization and evolutionary computation community with theoretical and practical tools to examine search landscapes. It allows for the assessment of problem characteristics and the evaluation of the performance of search algorithms. In the context of combinatorial fitness landscapes, these are represented as graphs defined by a discrete search space and a neighborhood relation. A fundamental challenge lies in developing a search algorithm capable of navigating a fitness landscape to reach the highest possible fitness value.

In general, achieving optimal solutions through a straightforward adaptive approach is quite challenging. This difficulty arises from the complex interplay among different parts of the solution, which can lead to locally optimal solutions. These local optima cannot be escaped by intensification or exploitative move strategies. Consequently, the optimization algorithm becomes stuck in a suboptimal state. The primary concern in such optimization processes is to strike a balance between exploiting promising search areas through greedy search methods and diversifying search trajectories by temporarily exploring less promising solutions.

To overcome this challenge, researchers have developed many metaheuristics [23] and even hyperheuristic schemes [20] to mix different strategies. These approaches typically incorporate parameters that allow for precise adjustment of the trade-off between exploration and exploitation. Still, they might not perform efficiently in a black-box context, as many heuristics leverage the unique properties of the given problem to solve it efficiently. Machine learning techniques have been widely used to improve combinatorial optimization solving [24] and to address the optimal configuration of solving algorithms. An approach to algorithm design known as "programming by optimization" (PbO) was introduced by Hoos [9]. This paradigm encourages algorithm developers to adopt and leverage extensive design possibilities that encompass a wide range of algorithmic techniques, to optimize performance for specific categories or groups of problem instances.

In various works, different machine learning approaches have been used either to consider offline adjustment, selection of parameters, or online control of the search process using reinforcement learning (RL) techniques (see [17] for a recent survey). The use of neural networks (NNs) in solving combinatorial optimization problems has been studied for decades [26], starting with the early work of Hopfield [11]. Recent applications of Graph Neural Networks in the context of combinatorial optimization have been proposed to reach optimal solutions or to assist the solving algorithm in proving the optimality of a given solution (see [4] for a recent survey).

In the traveling salesman problem, a GNN can be used to predict the regret associated with adding each edge to the solution to improve the computation of the fitness function of the LS algorithm [12]. Note that NNs are classically used in surrogate model-based optimization [30]. In [21], the authors introduce a GNN into a hybrid genetic search process to solve the vehicle routing problem. The GNN is used to predict the efficiency of search operators and to select them optimally in the solving process. A deep Q-learning approach has recently been proposed to manage the different stages of an LS-based metaheuristic to solve routing and job-shop problems [5]. High-level solving policies can often be managed by reinforcement learning in LS processes [27]. In this paper, our purpose is different, as we focus on building simple search heuristics for black-box problems, rather than scheduling specific operators or parameters. In particular, we assume that the learning process cannot be based on the immediate rewards that are used in RL. This motivates our choice of neuro-evolution.

Objective of the paper This study explores the potential for emerging search strategies to overcome existing challenges. The objective is to change how information is leveraged while retaining simple and generic search components. Considering a basic hill climber algorithm to achieve a baseline search process for solving black-box binary problem instances, our aim is to benefit from machine learning techniques to get new local search heuristics that will be built from basic search information instead of choosing a priori a predefined move heuristic (e.g., always select the best neighbor). Hence, our goal is to provide a NN with the same information as a basic LS and, after training, to use the NN as the basic move component of a simple LS process. To evaluate the efficiency of our approach, we define an experimental setup based on NK landscape problems, which allows us to describe a fitness landscape whose problem size and ruggedness (determining the number of local minima) can be adjusted as parameters.

2 General framework

2.1 Pseudo-Boolean Optimization Problems and Local Search

Let us consider a finite set $\mathcal{X} \subseteq \{0,1\}^N$ of solutions to a specific problem instance. These solutions are tuples of values that must satisfy certain constraints, which may or may not be explicitly provided. We evaluate the quality of these solutions using a pseudo-Boolean objective function $f_{\text{obj}} : \mathcal{X} \to \mathbb{R}$. Therefore, a problem instance can be fully characterized by the pair $(\mathcal{X}, f_{\text{obj}})$. In terms of solving this problem, $\mathcal{X}$ is referred to as the search space.

When solving an optimization problem instance with an LS algorithm, the objective is to identify a solution $x \in \mathcal{X}$ that optimizes the value of $f_{\text{obj}}(x)$. Since we are primarily concerned with maximization problems, let us note that any minimization problem can be reformulated as a maximization problem. In this context, an optimal solution, denoted as $x^* \in \mathcal{X}$, must satisfy the condition that for every $x \in \mathcal{X}$, $f_{\text{obj}}(x) \leq f_{\text{obj}}(x^*)$. While exhaustive search methods, or branch and bound algorithms, guarantee the computation of an optimal solution, this is not the case with LS algorithms. However, computing optimal solutions

within a reasonable time is often infeasible, leading to the use of local search algorithms within a limited budget of evaluations of f_{obj} to approximate near-optimal solutions.

LS algorithms operate within a structured search space, thanks to a fixed-sized neighborhood function denoted as $\mathcal{N} : \mathcal{X} \to 2^{\mathcal{X}}$. This function assigns a set of neighboring solutions $\mathcal{N}(x) \subseteq \mathcal{X}$ for each solution $x \in \mathcal{X}$. To maintain a fundamentally generic approach to LS, we assume that $\mathcal{N}$ is defined using basic flip functions, $flip_i : \mathcal{X} \to \mathcal{X}$, where $i \in [\![1, N]\!]$, and $flip_i(x)$ is equal to x except for the i^{th} element, which is changed from 0 to 1 or vice versa. In this case, $\mathcal{N}(x) = \{flip_i(x) \mid i \in [\![1, N]\!]\}$. Starting from an initial solution, often selected randomly and denoted as x_0, LS constructs a path through the search space based on neighborhood relationships. This path is typically represented as a sequence of solutions bounded by a limit (horizon) H, denoted as $(x_0, x_1, \ldots, x_H)$, where for each $i \in [\![0, H-1]\!]$, $x_{i+1} \in \mathcal{N}(x_i)$. Let us denote $\mathcal{P} = \mathcal{X}^*$ the set of all paths (i.e., the set of all possible sequences built on $\mathcal{X}$).

In addition to the neighborhood function, this sequence of solutions is determined by a strategy, often involving the use of a fitness function f. For example, hill climbing algorithms select the next solution on the path based on a simple criterion: $\forall t \in [\![0, H-1]\!], f(x_t) < f(x_{t+1})$, where $f(x_t)$ represents the fitness evaluation of solution x_t. The process of choosing the next solution on the search path is referred to as a move. We denote $x_{t+1} = x_t \oplus flip_i$ the move that corresponds to $x_{t+1} = flip_i(x_t)$.

2.2 Local Search as an Episodic Tasks Process

According to previous remarks, an LS process can be modeled by a sequence of actions performed on states. To be as exhaustive as possible, these states must be general enough to describe the current solution, as well as the path already explored and future possible moves. Therefore, we may define a set of states as $\mathcal{S} = \mathcal{P} \times \mathcal{X} \times 2^{\mathcal{X}}$. Of course, states of $\mathcal{S}$ cannot be managed by a practical local search algorithm, at least from a memory size point of view. We introduce the notion of observation of a state as a function $o : \mathcal{S} \to \Omega$ where Ω is a domain that corresponds to an abstraction of the real states of search to gather only useful information for the considered local search strategy.

Note that here we are in the context of episodic (discrete states) tasks with a terminal state (the end of the search fixed for instance by a maximal number of moves H). Following a reinforcement learning-based description, a local search process can be encoded by a policy $\pi : \Omega \to \mathcal{A}$ where $\mathcal{A}$ is a set of actions. Here we consider deterministic policies, i.e. for each $o \in \Omega$, there exists one and only one action $a \in \mathcal{A}$, such that $\pi(o) = a$. Note that the policy can be parameterized by a parameter vector θ. The set of all policies is $\Pi = \{\pi_\theta \mid \theta \in \Theta\}$ where Θ is the parameter space.

In our context, we consider actions that are bitflips $flip_i$ defined in Section 2.1. Hence, $\mathcal{A} \supseteq \{flip_i \,|\, i \in [\![1, N]\!]\}$. Note that of course, other actions can be introduced to fit specific strategies.

We consider a transition function $\delta : \mathcal{X} \times \mathcal{A} \rightarrow \mathcal{X}$ such that $\delta(x, a) = x \oplus a$. The transition function turns an action that is defined on the set of observations to a move in the set of solutions. Note that the transition is used to update the current state and compute the next state of the LS process. An LS run can be fully characterized by the search trajectory that it has produced from an initial starting solution.

Given an instance $I = (\mathcal{X}, f_{\text{obj}})$, an initial solution $x_0 \in \mathcal{X}$, a policy π_θ, and a horizon H, a trajectory $T(x_0, \pi_\theta, H, I)$ is a sequence $(x_0, a_0, x_1, a_1, \ldots, x_{H-1}, a_{H-1}, x_H)$ such that $(x_0, \ldots, x_H) \in \mathcal{P}$ is an LS path (the multiset $\{x_0, \ldots, x_H\}$ will be denoted $P(T(x_0, \pi_\theta, H, I))$), $\forall i \in [\![0, H-1]\!], a_i = \pi_\theta(o(x_i))$ and $\forall i \in [\![1, H]\!], x_i = \delta(x_{i-1}, a_{i-1})$. Note that the trajectory is defined with regard to solutions belonging to $\mathcal{X}$, while the policy operates on the space of observations obtained from these solutions. Compared to classic reinforcement schemes, where a reward can be assigned after each action, in our context, the reward will be computed globally for a given trajectory. Note that if we considered only the best-improvement strategy, then the reward could be assigned after each move to assess that the best move has been selected. Unfortunately, such a strategy will only be suitable for simple smooth unimodal problems. Hence we translate this reward as a function $R(x_0, \pi_\theta, H, I) = \max_{x \in P(T(x_0, \pi_\theta, H, I))} f_{\text{obj}}(x)$.

Example 1. In order to illustrate our framework, let us consider a basic hill climber (HC) that uses a simple best-improve strategy using objective function f_{obj} as fitness function. In the following, we only consider LS processes that do not involve memory that records past decisions. Hence, a state of a HC is fully described by a current solution $x \in \mathcal{X}$ and its neighborhood $\mathcal{N}(x)$. An observation will abstract the state as the variation of the fitness function for each possible flip of the values of x, $o(x) = (f_{\text{obj}}(flip_1(x)) - f_{\text{obj}}(x), \ldots, f_{\text{obj}}(flip_N(x)) - f_{\text{obj}}(x)) \in \mathbb{R}^N$ ($\Omega = \mathbb{R}^N$). In a best-improve HC process, the search stops when a local optimum is reached and no improving move can be performed. We consider $\mathcal{A} = \{flip_i \,|\, i \in [\![1, N]\!]\} \cup \{Id\}$, where Id is the identity function on $\mathcal{X}$. Then, we define the policy $\pi_{HC}(o(x)) = \text{argmax}_{a \in \mathcal{A}}(f_{\text{obj}}(\delta(x, a)) - f_{\text{obj}}(x))$. Note that when a local optimum is reached, the identity action will always be selected for the remainder of the process.

Our objective is to maximize the maximum score encountered by the agent during its trajectory, and not the sum of local fitness improvements obtained during its trajectory. We are therefore not in the case of learning a Markov decision process. This is why classical reinforcement learning algorithms such as Q-learning or policy gradient are not applicable in this context.

This justifies our choice of neuro-evolution where the policy parameters will be abstracted by a neural network that will be used to select the suitable action. The parameters of this neural network will be searched by means of an evolutionary algorithm according to a learning process defined below.

2.3 Policy Learning for a Set of Instances

In this paper, we focus on NK landscapes as pseudo-boolean optimization problems. The NK landscape model was introduced to describe binary fitness land-

scapes [14]. The characteristics of these landscapes are determined by two key parameters: N, which represents the dimension (number of variables), and K (where $K < N$), which indicates the average number of dependencies per variable and, in turn, influences the ruggedness of the fitness landscape. An NK problem instance is an optimization problem represented by an NK function. We use random NK functions to create optimization problem instances with adjustable search landscape characteristics. This adjustment will be achieved by varying the parameters (N, K), thus generating diverse search landscapes. Hence, we consider $\mathcal{NK}(N, K)$ as a distribution of instances generated by a random NK function generator.

In order to achieve our policy learning process, we must assess the performance of a policy as $F(\pi_\theta, \mathcal{NK}(N, K), H) = \mathbb{E}_{I \sim \mathcal{NK}(N,K), x_0 \sim \mathcal{X}}[R(x_0, \pi_\theta, H, I)]$ where $x_0 \sim \mathcal{X}$ stands for a uniform selection of an element in $\mathcal{X}$. However, since this expectancy cannot be practically evaluated, we rely on an empirical estimator computed as an average of the score obtained by the policy π_θ for a finite number q of instances $I_1, \ldots, I_q$ sampled from the distribution $\mathcal{NK}(N, K)$ and a finite number r of restarts $x_0^{(1)}, \ldots, x_0^{(r)}$ drawn uniformly in $\mathcal{X}$:

$$\bar{F}(\pi_\theta, \mathcal{NK}(N, K), H) = \frac{1}{qr} \sum_{i=1}^{q} \sum_{j=1}^{r} R(x_0^{(j)}, \pi_\theta, H, I_i). \tag{1}$$

Figure 1 highlights our general learning methodology and the connections between the LS process at the instance solving level and the policy learning task that will be achieved by a neural network presented in the next section.

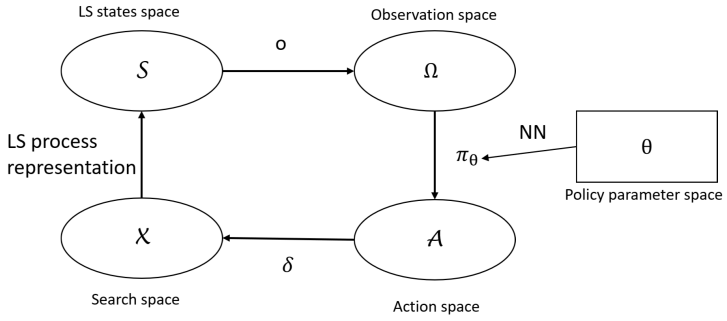

Fig. 1. Global View of the Process

3 Deterministic local search policies for pseudo-boolean optimization

In this paper, our objective is to compare an LS policy learned by neuroevolution, with three different baseline LS algorithms. To ensure a fair compari-

son among all the algorithms, all the different strategies take as input the same vector of observations corresponding to the variation of the objective/fitness function f for each possible flip of the values of x: $o(x) = (f(flip_1(x)) - f(x), \ldots, f(flip_N(x)) - f(x)) \in \mathbb{R}^N$. The set of possible actions available to the different strategies always remains $\mathcal{A} = \{flip_i \,|\, i \in [\![1, N]\!]\}$.

3.1 Neural network local search policy

We introduce a deterministic LS policy $\pi_\theta : \mathbb{R}^N \to \mathcal{A}$, called Neuro-LS, which uses a neural network g_θ, parametrized by a vector of real numbers θ. This neural network takes as input an observation vector $o \in \mathbb{R}^N$ and gives as output a vector $g_\theta(o)$ of size N whose component $g_\theta(o)_i$ corresponds to a preference score associated to each observation o_i. Then, the action a_i corresponding to the highest score $g_\theta(o)_i$ is selected.

Permutation equivariant neural network

A desirable property of this neural network policy is to be *permutation equivariant* with respect to the input vector of observations, which is a property generally entailed by a local search algorithm, in order to make its behavior consistent for solving any type of instance. Formally, an LS algorithm is said *to be invariant to permutations* in the observations if for any permutation σ on $[\![1, N]\!]$, we have $a_{\sigma(i)} = \pi_\theta(o_{\sigma(1)}, o_{\sigma(2)}, \ldots, o_{\sigma(N)})$. As an example, the basic hill climber HC defined with Example 1 in Section 2.2 has this property.

In order to obtain this property for Neuro-LS, g_θ must be function from $\mathbb{R}^N$ to $\mathbb{R}^N$, such that for any permutation σ we have $(g_\theta(o)_{\sigma(1)}, \ldots, g_\theta(o)_{\sigma(N)}) = g_\theta(o_{\sigma(1)}, \ldots, o_{\sigma(N)})$. Such type of permutation equivariant neural network can be obtained by using the deep sets architecture [31].

Each layer of the proposed network combines the treatment of each observation o_i associated to each of the N variables with an additional operation that performs an average of the features across the different variables. This N-averaging operation is independent by permutation of the inputs and allows to *transmit* some general contextual information between the N features vector.

For a vector of observation $o = (o_1, o_2, \ldots, o_N)$ given as input, a permutation equivariant network g_θ with P hidden layers is defined as $g_\theta(o) = \phi_{\theta_P} \circ \phi_{\theta_{P-1}} \circ \cdots \circ \phi_{\theta_0}(o)$, where each ϕ_{θ_j} is a permutation invariant function from $\mathbb{R}^{N \times l_j}$ to $\mathbb{R}^{N \times l_{j+1}}$. The l_j values correspond to the layer sizes. Note that for the first layer $l_0 = 1$ and for the last layer $l_{P+1} = 1$. This network g_θ is shown in Figure 2.

Each layer operation ϕ_{θ_j} with l_j input features and l_{j+1} output features includes a weight matrix $W_j \in \mathbb{R}^{l_j \times l_{j+1}}$ that treats each variable information independently, a variable-mixing weight matrix $\Gamma_j \in \mathbb{R}^{l_j \times l_{j+1}}$ and a bias vector $\beta_j \in \mathbb{R}^l$. Note that the size of these matrices W_j and Γ_j does not depend on the size N of the observation vector, allowing the Neuro-LS strategy to adapt to pseudo-Boolean optimization problems of different sizes.

When taking as input a feature matrix $v = (v_1, \ldots, v_N)$ of size $N \times l_j$, the matrix of weights W_j processes each feature vector v_i associated with each

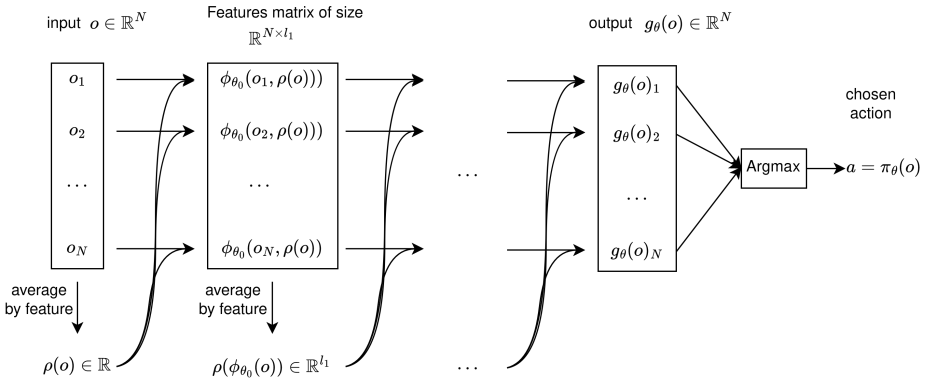

Fig. 2. Neural network policy $\pi_\theta : \Omega \to \mathcal{A}$ using a deep sets network architecture. Given a vector of observation $o \in \mathbb{R}^N$, the neural network outputs a vector $g_\theta(o) \in \mathbb{R}^N$. Then, the action $a = \text{argmax}_{i \in [\![1,N]\!]}\, g_\theta(o)_i$ is returned by π_θ.

variable i independently. Then, the weight matrix Γ_j processes an average vector $\rho(v)$ computed across the different N feature vectors of size l_j and given by $\rho(v) = \frac{1}{N} \sum_{i=1}^{N} v_i$.

Given $v \in \mathbb{R}^{N \times l_j}$, the output of the layer ϕ_{θ_j} is a matrix in $\mathbb{R}^{N \times l_{j+1}}$, which is the concatenation of N output vectors of size l_{j+1}, $\phi_{\theta_j}(v) = (\phi_{\theta_j}(v)_1, \ldots, \phi_{\theta_j}(v)_N)$, where for $1 \leq i \leq N$,

$$\phi_{\theta_j}(v)_i = \eta(\beta_j + x_i W_j + \rho(v)\Gamma_j), \tag{2}$$

with $\eta : \mathbb{R}^{l_j} \to \mathbb{R}^{l_{j+1}}$ the element-wise nonlinear activation map defined by $\eta(z) := (\tanh(z_1), \ldots, \tanh(z_{l_j}))$. We denote $\theta := \{(W_j, \Gamma_j, \beta_j)\}_{j \in [\![0,P]\!]}$, the set of all weight matrices and bias vectors of the neural network.

Neuro-evolution with CMA-ES

The neural network policy π_θ is characterized by a set of parameters denoted as θ. The optimization goal is to maximize the estimated score $\bar{F}(\pi_\theta, \mathcal{NK}(N, K), H)$. This poses a stochastic black-box optimization problem within the real-valued search space $\mathbb{R}^{|\theta|}$. To tackle this problem, we propose to use the covariance matrix adaptation evolution strategy (CMA-ES) [8], which stands out as one of the most powerful evolutionary algorithms for addressing such black box optimization problems [18].

The principle of CMA-ES is to iteratively test new generations of real-valued parameter vectors θ (individuals). Each new generation of parameter vectors is stochastically sampled according to a multivariate normal distribution. The mean and covariance matrix of this distribution are incrementally updated, so as to maximize the likelihood of previously successful candidate solutions. Thanks

to the use of a stepwise-adapted covariance matrix, the algorithm is able to quickly detect correlations between parameters, which is an important advantage when optimizing the (many) parameters of the neural network policy. Another advantage of CMA-ES is that it relies on a ranking mechanism of the estimated scores $\bar{F}$ given by the Equation (1) for the different individuals of the population, rather than on their absolute values, making the algorithm more robust to stochastic noise related to the incertitude on the estimation of the performance score F with a finite number of trajectories.

3.2 Basic local search policies

We compare the neural network policy above with three *basic* local search policies which have been extensively studied in the literature [19,1]. All these strategies take as input the same vector of observation as the neural network policy and return an action $a \in \mathcal{A}$. These three policies are two hill climbers as well as a $(1, \lambda)$-evolution strategy [3] used as a local search [25]. They are made deterministic using a pseudo-random number generator h whose seed is determined with a hash function from the current state x encountered by the LS.

Best improvement hill climber [+jump] ($\mathbf{BHC^+}$). This strategy always selects the action $a_i = flip_i \in \mathcal{A}$ in such a way that $f(a_i(x)) - f(x)$ is maximized, provided there is at least one action a_i that strictly improves the score. If $\forall i,\ f(a_i(x)) - f(x) \leq 0$, then this strategy performs a random jump by choosing a random action $a \in \mathcal{A}$ using the pseudo-random number generator $h(x)$.

First improvement hill climber [+jump] ($\mathbf{FHC^+}$). This strategy iterates through all actions in $\mathcal{A}$ in random order and selects the first action a_i leading to a strictly positive score improvement, i.e. such that $f(a_i(x)) - f(x) > 0$. Similar to the BHC^+ strategy, if $\forall i,\ f(a_i(x)) - f(x) \leq 0$, it performs a random jump.

$(1, \lambda)$-evolution strategy ($(1, \lambda)$-ES). This strategy randomly evaluates λ actions in $\mathcal{A}$ and chooses the one that yields the best score, even if it results in a deteriorating move. λ is a method hyperparameter that will be calibrated to maximize the estimated score $\bar{F}$ for each type of NK landscape instance as detailed in the next section.

4 Computational experiments

The aim of this section is to answer two questions experimentally. The first concerns the performance of Neuro-LS compared to the baseline LS strategies presented in the last subsection for NK landscape problems of different size and ruggedness. The second is to study the emergent strategies discovered by Neuro-LS at the end of its evolutionary process.

First, we discuss the experimental setting. Then, we follow the classical steps of a machine learning workflow: subsection 4.2 describes the Neuro-LS training process; it will allow us to select different emerging strategies for each type of NK landscape on validation sets, which we will then compare with the different baseline LS strategies on test sets (subsection 4.3). Finally, an in-depth analysis of the best emerging strategies discovered by Neuro-LS will be performed in subsection 4.4.

4.1 Experimental settings

In these experiments, we consider independent instances of NK-landscape problems. 12 different scenarios with $N \in \{32, 64, 128\}$ and $K \in \{1, 2, 4, 8\}$ are considered. For each scenario, three different sets of instances are sampled independently from the $\mathcal{NK}(N, K)$ distribution described in Section 2.3: a training set, a validation set and a test set. For the resolution of each instance, given a random starting point $x_0 \in \mathcal{X}$, each LS algorithm performs a trajectory of size $H = 2 \times N$ (iterations) and returns the best solution found during this trajectory. The experiments were performed on a computer equipped with a 12th generation Intel® CoreTM i7-1265U processor and 14.8 GB of RAM.

Neuro-LS is implemented in Python 3.7 with Pytorch 1.4 library.[1] For all experiments with different values N and K, we use the same architecture of the neural network composed of two hidden layers of size 10 and 5, with a total of $|\theta| = 162$ parameters to calibrate. To optimise the weights of the neural network, we used the CMA-ES algorithm of the pycma library [7]. The multivariate normal distribution of CMA-ES is initialized with mean parameter μ (randomly sampled according to a unit normal distribution) and initial standard deviation $\sigma_{\text{init}} = 0.2$.

4.2 Neuro-LS training phase

For each NK-landscape configuration, we run 10 different training processes of Neuro-LS with CMA-ES, with a time limit of two hours, to optimize the empirical score $\bar{F}$ defined by equation (1) computed as an average of the best fitness scores obtained for 50 trajectories ($r = 5$ independent random restarts for each of the $q = 10$ training instances).

Figure 3 displays the results of 10 independent neuro-evolution training processes for the NK landscape instances with $N = 32$ and $K = 8$. At each generation, the 10 learning instances are regenerated to avoid over-fitting, then CMA-ES samples a population of 19 individuals (19 vectors of weights $\theta \in \mathbb{R}^{162}$ of the neural network), and the best Neuro-LS strategy of the population on the training set is evaluated on the 10 instances of the validation set.

The evolution of the average score of Neuro-LS on the training and validation sets are indicated with orange and blue lines in Figure 3. The red line is a

[1] The program source code, benchmark instances and result files are available at the url `https://github.com/Salim-AMRI/NK_Landscape_Project.git`.

reference score. It corresponds to the average score $\bar{F}$ obtained by the BHC$^+$ local search strategy (see Section 3.2) on the same validation set.

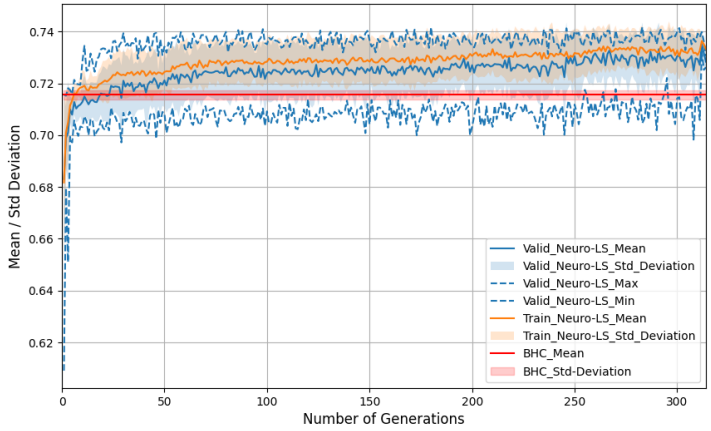

Fig. 3. Evolution of the average score obtained by Neuro-LS on the training and validation sets over the generations of CMA-ES.

First, we find that the validation curve closely follows the training curve in average, even if it is slightly below. This reassuring consistency suggests that our model successfully generalizes the strategy learned in training instances.

Furthermore, upon comparing the validation curves of Neuro-LS and BHC$^+$, we observe that the Neuro-LS curve progresses over generations, and eventually surpasses BHC$^+$ when evaluated on the same set of validation instances. This finding highlights that, once trained, the Neuro-LS method is able to find solutions more efficiently compared to the baseline BHC$^+$ local search for these types of instances.

In Figure 3, the minimum and maximum scores on the validation set obtained during the 10 independent neuro-evolution runs are also indicated with blue dashed lines. The figure illustrates significant variability in the results, highlighting the diversity in the performance of the emerging strategies. Nevertheless, this variability poses no issue in our context, as only the strategy with the best results on the validation set will be selected for the test phase presented in the next subsection.

4.3 Test phase

In this phase, we performed a series of evaluations to assess whether the best Neuro-LS strategies, selected based on the validation set for each configuration

of NK landscape, continue to perform well in new test instances that are independently sampled from the same $\mathcal{NK}(N, K)$ distribution.

Table 1 summarizes the average score obtained by Neuro-LS and the three other competing methods, namely the BHC$^+$, FHC$^+$ and (1, λ)-ES algorithms, presented in section 3.2. The strategy (1, λ)-ES has one hyperparameter λ that we calibrated in the range $[\![1, N]\!]$ on training instances for each (N, K) configuration.

In order to perform a fair comparison between the different strategies, we computed an average estimated score $\bar{F}$ on the same 100 test instances. For each instance we use the same starting point to compute the trajectory produced by each LS strategy.[2]

In Table 1, the best average result obtained among the mentioned methods for each configuration (N, K) of test instances is highlighted in bold. Results underlined indicate significant better results in average compared to all the other strategies (p-value below 0.001), measured with a Student t-test without assuming equal variance.[3]

Instances		Methods			
N	K	BHC$^+$	FHC$^+$	(1, λ)-ES	Neuro-LS
32	1	0.694	0.688	0.695	**0.699**
32	2	0.713	0.709	0.717	**0.721**
32	4	0.717	0.721	0.713	**0.735**
32	8	0.702	0.712	0.707	**0.732**
64	1	0.700	0.693	0.696	**0.702**
64	2	0.712	0.709	0.712	**0.715**
64	4	0.721	0.721	0.714	**0.734**
64	8	0.710	0.719	0.707	**0.737**
128	1	0.698	0.691	0.696	**0.701**
128	2	0.712	0.711	0.710	**0.713**
128	4	0.724	0.723	0.717	**0.728**
128	8	0.711	0.719	0.705	**0.730**

Table 1. Average results on test instances obtained by different local search strategies for different NK landscape configurations.

Table 1 shows that Neuro-LS always obtains better results for all the configurations of NK landscape, but the difference in score is only really significant when $K = 4$ and $K = 8$ for all values of N (except for $N = 128$ and $K = 4$). It

[2] For this evaluation test, we only perform one restart per instance, to avoid any dependency between the different executions that might take place on the same instance. It allows to obtain a distribution of 100 independently and identically distributed scores for each strategy and for each NK configuration.

[3] The normality condition required for this test was first confirmed using a Shapiro statistical test on the empirical distributions of 100 iid scores obtained by each strategy.

means that our strategy becomes more effective than other methods when the landscape is more rugged.

This score improvement compared to the other baseline methods, obtained with the same budget of $H = 2 \times N$ iterations performed on each instance, can be attributed to a more efficient exploration of the search space as K increases (as seen in the next subsection).

4.4 Study of Neuro-LS emerging strategies

The objective here is to analyse in detail the best Neuro-LS strategies that have emerged with neuro-evolution and to understand their decision-making processes for the different types of landscape studied (smooth or rugged).

We have observed two main patterns of emerging strategies:

- For smooth landscapes, when $K = 1$ or $K = 2$, Neuro-LS learns to perform almost all the time a best improvement move, which explains why for these instances, it obtains almost the same score as the BHC^+ strategy (see Table 1).
- For rugged landscapes, when $K = 4$ and $K = 8$, the emerging strategy is much more interesting. Figure 4 displays a representative example of the trajectory performed by Neuro-LS when $N = 64$ and $K = 8$. This figure shows two graphs based on data collected during the resolution of this instance. The graph on the top of this Figure shows the evolution of the fitness reached by Neuro-LS over the $H = 2 \times N = 128$ iterations. The graph below shows at each iteration the number of available actions corresponding to an improvement of the score (in blue), and the rank of the action selected by Neuro-LS, measured in term of fitness improvement (in red). A rank of 1 on this plot indicates that Neuro-LS has chosen a best improvement move, while a rank of 64 indicates a worst deteriorating move (note that the y-axis is inverted, because it is a maximization problem). We observe on this plot that the emerging Neuro-LS strategy has three different successive operating modes:
  1. **Median hill climbing behavior.** When the number of actions associated with a positive improvement of the score, N_a^+, is relatively large ($\sim$ above $N/10$), Neuro-LS does not always choose the best improvement move, but instead a move with a rank approximately equal to $N_a^+/2$. This provides a good compromise between improving the score and avoiding being trapped too quickly in a local optimum.
  2. **Best improvement hill climbing behavior.** When the number of actions corresponding to a positive delta fitness is low ($\sim$ below $N/10$), Neuro-LS often chooses the best move (with rank 1) to quickly converge toward the closest local optimum.
  3. **Jump with worst move.** When there is no more improving move, Neuro-LS does not stagnate, but instead directly chooses to perform the worst possible move (with rank 64). Even if this movement considerably deteriorates the current fitness score, it actually maximizes its long-term

chances of escaping the current local optimum and continually exploring new areas of the search space. Indeed, we observe on this plot that Neuro-LS continuously improves its score with this strategy for this instance. Note that after choosing the worst possible move, it does not choose the best possible move, otherwise it would return to the same local optimum.

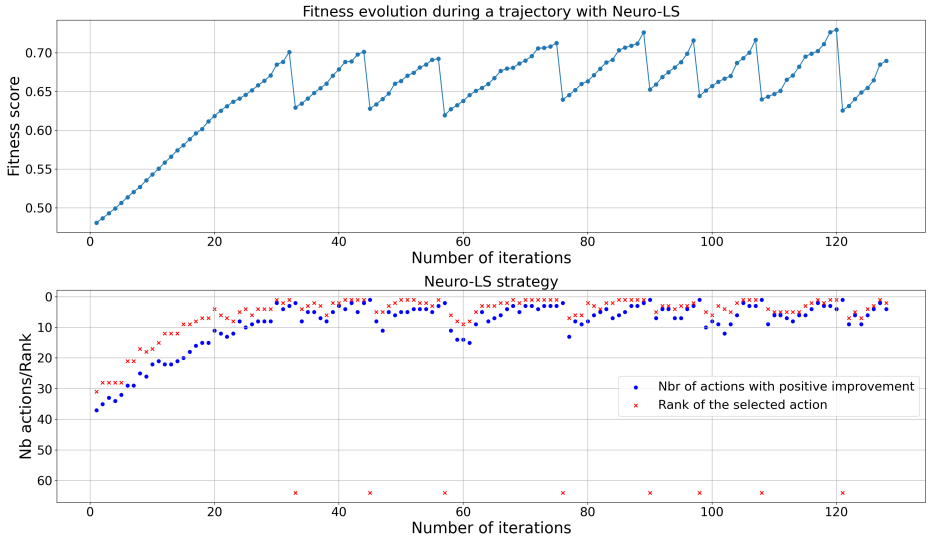

Fig. 4. Fitness evolution curve and strategy used by Neuro-LS for the resolution of a instance with significantly rugged NK landscape ($N = 64$ and $K = 8$).

Conclusion

Our study explores the emergence of new local search algorithms with neuro-evolution. Results on NK landscapes show that different neural network policies are learned, each adapted to the resolution of a particular landscape distribution type (smooth or rugged). Our algorithm is competitive with basic deterministic local search procedures for all the NK landscape types considered in this work. Particularly for rugged landscapes, it can achieve significantly better results with an original emerging strategy, using a worst-case improvement move, which proves very effective in the long run for escaping local optima.

This study outlines avenues for future research on the automatic discovery of more advanced strategies using as input a richer set of observations to make its decision. The proposed framework could also be applied to study the emergence of strategies adapted to other types of combinatorial optimization problems.

Acknowledgment

This work was granted access to the HPC resources of IDRIS (Grant No. AD010611887R1) from GENCI. The authors would like to thank the Pays de la Loire region for its financiel support for the Deep Meta project (Etoiles Montantes en Pays de la Loire). The authors also acknowledge ANR – FRANCE (French National Research Agency) for its financial support of the COMBO project (PRC - AAPG 2023 - Axe E.2 - CE23). We are grateful to the reviewers for their comments.

References

1. Basseur, M., Goëffon, A.: Hill-climbing strategies on various landscapes: an empirical comparison. In: Proceedings of the 15th Annual Conference on Genetic and Evolutionary Computation. pp. 479–486 (2013)
2. Basseur, M., Goëffon, A.: Climbing combinatorial fitness landscapes. Applied Soft Computing **30**, 688–704 (2015)
3. Beyer, H.G.: The theory of evolution strategies. Springer Science & Business Media (2001)
4. Cappart, Q., Chételat, D., Khalil, E.B., Lodi, A., Morris, C., Velickovic, P.: Combinatorial optimization and reasoning with graph neural networks. Journal of Machine Learning Research **24**, 130:1–130:61 (2023)
5. Falkner, J.K., Thyssens, D., Bdeir, A., Schmidt-Thieme, L.: Learning to control local search for combinatorial optimization. In: Amini, M., Canu, S., Fischer, A., Guns, T., Novak, P.K., Tsoumakas, G. (eds.) Machine Learning and Knowledge Discovery in Databases - European Conference, ECML PKDD 2022, Grenoble, France, September 19-23, 2022, Proceedings, Part V. Lecture Notes in Computer Science, vol. 13717, pp. 361–376. Springer (2022)
6. Glover, F.: Tabu search—part i. ORSA Journal on Computing **1**(3), 190–206 (1989)
7. Hansen, N., Akimoto, Y., Baudis, P.: CMA-ES/pycma on Github. Zenodo, DOI:10.5281/zenodo.2559634 (Feb 2019)
8. Hansen, N., Ostermeier, A.: Completely derandomized self-adaptation in evolution strategies. Evolutionary Computation **9**(2), 159–195 (2001)
9. Hoos, H.H.: Programming by optimization. Commun. ACM **55**(2), 70–80 (2012)
10. Hoos, H.H., Stützle, T.: Stochastic local search: Foundations and Applications. Elsevier (2004)
11. Hopfield, J.: Neural networks and physical systems with emergent collective computational abilities. Proceedings of the National Academy of Sciences of the United States of America **79**, 2554–8 (05 1982)
12. Hudson, B., Li, Q., Malencia, M., Prorok, A.: Graph neural network guided local search for the traveling salesperson problem. In: The Tenth International Conference on Learning Representations, ICLR 2022, Virtual Event, April 25-29, 2022. OpenReview.net (2022)
13. Jones, T., Forrest, S., et al.: Fitness distance correlation as a measure of problem difficulty for genetic algorithms. In: ICGA. vol. 95, pp. 184–192 (1995)
14. Kauffman, S.A., Weinberger, E.D.: 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)

15. Lourenço, H.R., Martin, O.C., Stützle, T.: Iterated local search. In: Handbook of Metaheuristics, pp. 320–353. Springer (2003)
16. Malan, K.M.: A survey of advances in landscape analysis for optimisation. Algorithms **14**(2), 40 (2021)
17. Mamaghan, M.K., Mohammadi, M., Meyer, P., Karimi-Mamaghan, A.M., Talbi, E.: Machine learning at the service of meta-heuristics for solving combinatorial optimization problems: A state-of-the-art. European Journal of Operational Research **296**(2), 393–422 (2022)
18. Müller, N., Glasmachers, T.: Challenges in high-dimensional reinforcement learning with evolution strategies. In: Parallel Problem Solving from Nature–PPSN XV: 15th International Conference, Coimbra, Portugal, September 8–12, 2018, Proceedings, Part II 15. pp. 411–423. Springer (2018)
19. Ochoa, G., Verel, S., Tomassini, M.: First-improvement vs. best-improvement local optima networks of NK landscapes. In: International Conference on Parallel Problem Solving from Nature. pp. 104–113. Springer (2010)
20. Özcan, E., Bilgin, B., Korkmaz, E.E.: A comprehensive analysis of hyper-heuristics. Intelligent Data Analysis **12**(1), 3–23 (2008)
21. Santana, Í., Lodi, A., Vidal, T.: Neural networks for local search and crossover in vehicle routing: A possible overkill? In: Ciré, A.A. (ed.) Integration of Constraint Programming, Artificial Intelligence, and Operations Research - 20th International Conference, CPAIOR 2023, Nice, France, May 29 - June 1, 2023, Proceedings. Lecture Notes in Computer Science, vol. 13884, pp. 184–199. Springer (2023)
22. Schiavinotto, T., Stützle, T.: A review of metrics on permutations for search landscape analysis. Computers & Operations Research **34**(10), 3143–3153 (2007)
23. Sörensen, K., Glover, F.: Metaheuristics. Encyclopedia of Operations Research and Management Science **62**, 960–970 (2013)
24. Talbi, E.: Machine learning into metaheuristics: A survey and taxonomy. ACM Computing Surveys **54**(6), 129:1–129:32 (2022)
25. Tari, S., Basseur, M., Goëffon, A.: On the use of (1, λ)-evolution strategy as efficient local search mechanism for discrete optimization: a behavioral analysis. Natural Computing **20**, 345–361 (2021)
26. Trafalis, T.B., Kasap, S.: Neural networks for combinatorial optimization. In: Floudas, C.A., Pardalos, P.M. (eds.) Encyclopedia of Optimization, Second Edition, pp. 2547–2555. Springer (2009)
27. Veerapen, N., Hamadi, Y., Saubion, F.: Using local search with adaptive operator selection to solve the progressive party problem. In: Proceedings of the IEEE Congress on Evolutionary Computation, CEC 2013, Cancun, Mexico, June 20-23, 2013. pp. 554–561. IEEE (2013)
28. Vuculescu, O., Pedersen, M.K., Sherson, J.F., Bergenholtz, C.: Human search in a fitness landscape: How to assess the difficulty of a search problem. Complexity **2020** (2020)
29. Whitley, D.: Mk landscapes, NK landscapes, MAX-kSAT: A proof that the only challenging problems are deceptive. In: Proceedings of the 2015 Annual Conference on Genetic and Evolutionary Computation. pp. 927–934 (2015)
30. Willmes, L., Bäck, T., Jin, Y., Sendhoff, B.: Comparing neural networks and kriging for fitness approximation in evolutionary optimization. In: Proceedings of the IEEE Congress on Evolutionary Computation, CEC 2003, Canberra, Australia, December 8-12, 2003. pp. 663–670. IEEE (2003)
31. Zaheer, M., Kottur, S., Ravanbakhsh, S., Poczos, B., Salakhutdinov, R.R., Smola, A.J.: Deep sets. Advances in Neural Information Processing Systems **30** (2017)