



Surrogate-assisted evolutionary neural architecture search with network embedding

Liang Fan¹ · Handing Wang¹

Received: 23 May 2022 / Accepted: 14 November 2022 / Published online: 5 December 2022
© The Author(s) 2022

Abstract

To accelerate the performance estimation in neural architecture search, recently proposed algorithms adopt surrogate models to predict the performance of neural architectures instead of training the network from scratch. However, it is time-consuming to collect sufficient labeled architectures for surrogate model training. To enhance the capability of surrogate models using a small amount of training data, we propose a surrogate-assisted evolutionary algorithm with network embedding for neural architecture search (SAENAS-NE). Here, an unsupervised learning method is used to generate meaningful representation of each architecture and the architectures with more similar structures are closer in the embedding space, which considerably benefits the training of surrogate models. In addition, a new environmental selection based on a reference population is designed to keep diversity of the population in each generation and an infill criterion for handling the trade-off between convergence and model uncertainty is proposed for re-evaluation. Experimental results on three different NASBench and DARTS search space illustrate that network embedding makes the surrogate model achieve comparable or superior performance. The superiority of our proposed method SAENAS-NE over other state-of-the-art neural architecture algorithm has been verified in the experiments.

Keywords Neural architecture search · Evolutionary algorithm · Surrogate-assisted · Network embedding

Introduction

Deep neural networks (DNNs) have achieved significant success in tackling various tasks such as classification [1], object detection [2], and natural language processing [3]. The effect of a neural network depends on its architecture and network weights. There are several methods used to solve the weight optimization problem, such as Adagrad [4], Adadelta [5], and Adam [6]. However, the development of a new state-of-the-art architecture often requires a vast amount of domain knowledge and repetitive trials. Neural architecture search (NAS) seeks to automate this process and can be formulated as an optimization problem:

$$\max_{a \in A} \text{ACC} \left(W_a^*; \mathcal{D}^{\text{val}} \right);$$

$$s.t. \ W_a^* = \arg \min_{W_a} \mathcal{L} \left(W_a; \mathcal{D}^{\text{trn}} \right), \quad (1)$$

where A denotes the search space of neural architecture, the architecture a is evaluated on the validation set \mathcal{D}^{val} , W_a^* is the weight of a which is trained on the training set \mathcal{D}^{trn} .

According to different optimization methods, NAS can be divided into three categories, namely evolutionary algorithm (EA) based, reinforcement learning (RL) based, and gradient-based methods. In EA-based NAS [7–12], the architectures are regarded as the individuals via encoding scheme, and iteratively evolve for the optimal architecture. RL-based NAS [13, 14] uses the controller to sample new architectures from the pre-defined search space, and trains the networks to obtain the final performance as reward. Unlike EA-based and RL-based methods, learning over a discrete and non-differentiable search space, gradient-based NAS [15, 16] relaxes the search space to be continuous and jointly optimizes the architecture and network weights with gradient descent.

Although the experiments in the previous work [8] show that EAs have a stronger search ability than RL, they suffer from high computational demands because each network in

✉ Handing Wang
hdwang@xidian.edu.cn

Liang Fan
liangfan@stu.xidian.edu.cn

¹ School of Artificial Intelligence, Xidian University,
Xi'an 710071, Shaanxi, China

the population needs to be trained from scratch. For example, the LargeEvo algorithm [7] requires a total of 3150 GPU days to complete one search. To tackle the high computational cost, surrogate-assisted evolutionary algorithms (SAEAs) [17–20] use the trained surrogate models to replace the process of training the network from scratch, i.e., Sun et al. [21] propose an end-to-end performance predictor to accelerate the fitness evaluation in EA, Lu et al. [22] propose adaptive switching, which can adaptively select among four types of surrogate models via cross-validation, Rawal et al. [23] speed up the search process by estimating performance of candidate structures through a trained long short-term memory (LSTM).

Although SAEAs greatly reduce the computational cost of NAS, the surrogate models still require a vast amount of well-trained networks for supervised learning. The reason for this is that supervised learning requires a large amount of labeled samples to extract features. For the classification task on CIFAR-10, Peephole [24] samples over 800 networks as the training data of the surrogate model. To overcome this drawback, there are a few recent methods [25–27] that integrate unsupervised learning into NAS. The embedding of each architecture is obtained through unsupervised representation learning and used to train the surrogate model. Since unsupervised learning can obtain informative representation of each architecture with unlabeled data, the amount of labeled data required for surrogate models training can be reduced using these representations as input to the surrogate models [26].

Unfortunately, these work also have some shortcomings. Arch2vec [25] assumes that the embeddings of architectures follow the Gaussian distribution and reconstructs the input neural architectures using a variational autoencoder, but this assumption is not guaranteed [27]. NASGEM [26] uses an autoencoder to map the architectures to the embedding space, and improves the feature representation by minimizing the reconstruction loss and the similarity loss. However, NASGEM only vectorizes the adjacency matrix of the input neural architecture to an embedding space and ignores the node operations, which is crucial to the performance of the network. Unlike the two previous methods that use autoencoders for unsupervised learning, the work in [27] proposes two self-supervised learning methods to pre-train the architecture embedding part of the surrogate model, namely SS-RL and SS-CCL. SS-RL takes predicting the graph edit distance (GED) between two architectures as a pretext task. SS-CCL is a central contrastive learning algorithm, which uses GED to construct positive sample set and negative sample set. GED is calculated based on position-aware path-based encoding (PAPE), which is a coding strategy that records the position of each operation by assigning each node with a unique index. However, different index orders on the same network architecture may result in completely different encodings. It

causes a large GED between isomorphic network architectures whereas the real GED should be zero.

To reduce the number of labeled data required for the surrogate model, another approach is to use online learning. Online learning actively identifies more new valuable solutions to be evaluated using the expensive fitness function and uses these solutions to update the surrogate model [28]. This process is called infill criterion in SAEAs and it should consider both convergence and uncertainty of each candidate solution [29]. On the one hand, the solutions with better accuracy are selected to update the surrogate model can improve the prediction performance in the promising region of search space. On the other hand, the solutions with greater uncertainty can not only encourage exploration but also improve the effectiveness of the surrogate model in the region of high uncertainty. Nevertheless, some work on NAS so far has not considered on both. The work in [30] obtains the uncertainty estimation by calculating the standard deviation of m predictions and uses independent Thompson sampling (ITS) choose new individuals to update the surrogate model. However, solutions with high uncertainty in ITS do not have high probability of being selected, which is inconsistent with our expectations. Another infill criterion is proposed in [22], which chooses solutions with high diversity on the Pareto front, without considering the uncertainty.

Furthermore, environmental selection is a critical step in SAEAs. It should trade-off exploitation and exploration, namely, the search process can not only exploit the known information to speed up the convergence, but also explore the uncertain regions in the search space. There is some work using clustering to increase the diversity of populations to balance exploitation and exploration, such as [31,32] uses k -means to partition the population and select solutions from each cluster into the next generation. However, the main drawback of k -means algorithm is that the initialization of centroid is not easy to be determined. In addition, the number of solutions in each cluster is unbalanced, which will cause outliers with poor quality to be erroneously retained because they belong to a cluster of their own.

To sum up, the SAEAs for NAS have three open issues: (1) an efficient embedding method is needed to extract the inherent characteristics of neural architecture which can enhance the performance of surrogate model, (2) an infill criterion is needed to efficiently update the surrogate models and take into account both the convergence and uncertainty, (3) a simple and effective environmental selection strategy is needed to trade-off between exploitation and exploration.

In order to address these issues, we propose a novel algorithm for NAS, called surrogate-assisted evolutionary neural architecture search with network embedding (SAENAS-NE). SAENAS-NE uses graph2vec [33] to map each neural architecture to the embedding space. Graph2vec learns representations for entire network by predicting whether a

subgraph exists and networks with more identical subgraphs have more similar representations. It does not require prior assumptions about the distribution of architecture representations, nor does it require a unique index to each operation. Further, to reduce the number of real evaluations required, we use RankNet [34] as surrogate model and design a novel infill criterion that determines new architectures to evaluate. In addition, we propose a novel environmental selection strategy which chooses individuals to form the next population. The main contributions of this paper are the following:

- To enhance the performance of surrogate model with less training data, graph2vec is applied to NAS to obtain the embedding of each architecture, which encourages the neural architectures with similar topologies to cluster together. In addition, different initialization of graph2vec in the process of generating embeddings can get different but similar embeddings. Based on this trait, the surrogate model calculates different embeddings of the same architecture, and obtains the standard deviation of the estimated fitness value as its uncertainty estimation.
- To effectively update the surrogate model, we adopt a novel infill criterion to choose individuals for re-evaluation. The infill criterion takes the estimated value of the individual and the model uncertainty as two objectives and uses nondominated sorting to choose new individuals for real evaluation. It considers both convergence and uncertainty, and gives the updated surrogate model better prediction performance throughout the search process.
- To balance the exploitation and exploration in the optimization process, a novel environmental selection strategy is proposed to keep diversity among the solutions in the obtained population. It clusters all candidate members based on their distances from the reference solution, and selects the best individual in each cluster to enter the next population. It does not require additional centroids, while the number of solutions in each cluster is consistent.

The remainder structure of this paper takes the form of five sections. We first introduce the background of NAS in Section “[Related work](#)”. Section “[Preliminaries: graph2vec](#)” gives a brief review of graph2vec. Then, in Section “[Methodology](#)”, the proposed algorithm, a surrogate-assisted ENAS with network embedding, is described in detail. Experimental settings and results are presented in Section “[Experiments](#)”. Finally, a summary containing conclusions and future work is in Section “[Conclusions](#)”.

Related work

Evolutionary algorithm-based neural architecture search

Over the past years, EAs have gradually become popular in NAS and many popular EA-based optimizers have been employed recently in NAS.

Genetic algorithm (GA) is the most widely used optimizer in EA-based NAS. LargeEvo [7] uses the binary tournament selection for mating and does not require human participation. Genetic CNN [10] uses a fixed-length binary string to represent the architecture, which makes various genetic operations easier. The work in [35] encodes three different building blocks (the convolutional layer, the pooling layer, the full connection layer) into one chromosome and uses crossover and mutation to create offspring. In particular, it designs a method called unit alignment for crossover with variable length encoding. REA [8] removes the oldest individual from population to guarantee the diversity of the population and has achieved the state-of-the-art performance.

Genetic programming (GP) approach is also used for NAS. CGP-CNN [36,37] uses the Cartesian genetic programming (CGP) encoding scheme and adopts the highly functional modules for searching the optimal architectures. GPCNN [38] is the first research which uses tree-based GP to design CNN architectures with a novel crossover operator called partial subtree crossover. AutoML-Zero [39] uses basic mathematical operations as building blocks to design the architectures instead of sophisticated expert-designed layers.

Particle swarm optimization (PSO) is another popular evolutionary algorithm for NAS. Junior et al. [40] present a novel PSO algorithm called psoCNN with a new difference operator and a new velocity operator, and it updates the particle based on the type of layers, independent of the hyperparameters. Wang et al. [41] focus on searching the optimal hyperparameters of dense blocks and performs multi-objective optimization where two objectives are considered: classification accuracy and computational cost.

Apart from the above methods, there are other EAs for NAS. For example, DE-NAS [42] applies canonical differential evolution (DE) to obtain the better architectures after making the discrete or categorical parameters continuous. DeepSwarm [43] uses ant colony optimization (ACO) [44] to search for the optimal architectures through progressive neural architecture search, which explores the full search space using small incremental steps. Sharaf et al. [45] use a firefly algorithm based on the k -nearest neighborhood attraction firefly and produce satisfying solutions.

Encoding schemes

In addition to the optimizer, another factor that affects the search performance is encoding schemes. Recent studies have demonstrated the significant effect of encoding schemes on NAS [30,46]. The encoding schemes can be categorized into two broad categories: the adjacency matrix and path-based encodings [47].

The adjacency matrix encoding represents the network architecture through vectorized adjacency matrix and a list of operation labels of each node [13,15,46]. In addition, some work presents several variants of adjacency matrix encoding, i.e., a categorical-valued variant in [47] where the features are a list of the indices specifying the corresponding edges in the adjacency matrix. In order to address the challenge of discrete coding to the optimization algorithm, the work [46] defines a real value in $[0,1]$ for each edge, rather than just $\{0,1\}$ in the previous work.

Path-based encoding treats the neural architecture as a directed acyclic graph (DAG), and encodes each path of the DAG from the input to the output. Compared to the adjacency matrix encoding, the path-based encoding can reduce the dependency among the features and increase the performance of neural predictors [47]. As far as we know, BANANAS [30] adopts path-based encoding for the first time. The encoding is one-hot vector whose length is $\sum_{i=0}^n q^i = (q^{n+1} - 1)/(q - 1)$, where n is the number of nodes, q is the number of possible operations. Given an architecture, all paths in this architecture are first found out and the feature corresponding to each path is set to 1. Like the adjacency matrix encoding mentioned above, the path-based encoding also has the categorical-valued variant and the continuous-valued variant [47]. In contrast to [27,30] proposes a novel encoding scheme termed PAPE. Different from the previous path encoding, PAPE records the position of each operation in the path, then encodes each path by the position and type of operations, and finally concatenates all path encoding in the order of path length to form the encoding to the architecture.

In addition to the above encoding schemes, different schemes exist in the recent studies. Some work factorizes an architecture into unique blocks and encode each block by its kernel size, expansion rate, number of layers [22,48,49]. Inspired by the gene expression process, a novel encoding scheme called action command encoding (ACEncoding) is proposed in [50]. ACEncoding uses seven action commands which composed of three integers and encodes these action commands into a variable-length sequence and it can represent more complex neural architectures.

Performance estimation strategy

After generating the new architectures, one significant bottleneck of NAS is how to efficiently evaluate them. Early work (e.g., [7,13]) trains all candidate networks from scratch and use the accuracy of corresponding networks as the evaluation value, which is the simplest but very time-consuming performance estimation strategy. To alleviate this problem, many methods for speeding up performance estimation have been proposed. Below we review three different types of accelerated evaluation methods, namely the low fidelity estimation, weight sharing mechanism and surrogate model technique.

Many studies estimate the performance of networks at a low fidelity level. For example, some work trains each network on shorter training time [51,52]; Some work trains the networks on a subset of the training data [53] or lower resolution images [54]. While low fidelity approximation reduces the computational cost, its underestimation of the performance brings a bias to the selection of better architectures [55].

In recent years, there has been an increasing amount of literature that adopt the weight sharing mechanism in one-shot architecture search to reduce the computational cost. SMASH [56] first trains a hypernetwork which can dynamically generate the weights of the networks, and then compares the validation performance of candidate networks with hypernetwork-generated weights. Besides, some work pre-trains the supernet, and then all candidate networks as sub-networks inherit the weight of supernet. One-Shot [57] uses a dynamic dropout rate to randomly zero out a subset of the operations at supernet training time. To alleviate the weight co-adaption problem, SPOS [58] assumes that all architectures are single paths of the supernet and trains the supernet by uniform path sampling. Because of the inherent unfairness in the supernet training, FairNAS [59] trains the supernet with a fairness perspective, namely, it makes each block be activated and updated only once. In Landmark Regularization [60], a regularization term is defined by leveraging a set of stand-alone performance and guide the supernet training. Different from the above methods divide the training hypernetwork and search candidate architecture into two stages, Darts [15] adopts the gradient-based method and optimizes the weights together with the architecture.

As a consequence, the weight sharing mechanism can reduce the evaluation cost. However, some work shows that there are a large gap between supernet predicted accuracies and that of stand-alone model which trained from scratch [15,46]. Therefore, the approach that the expensive evaluation of all candidate architectures replaced by surrogate models gets more and more attention. Some work estimates the performance of candidate architectures by extrapolating the model training curve [23,61,62]. The alternative way to build surrogate models is supporting prediction performance

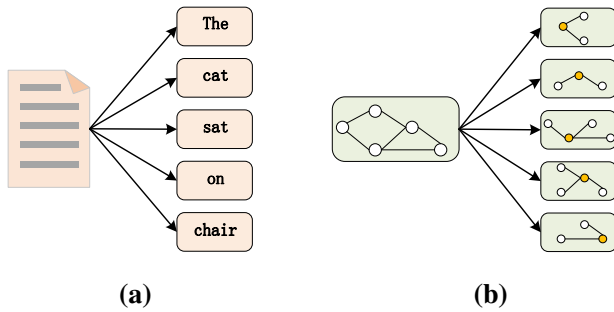


Fig. 1 **a** doc2vec samples 5 words from the document. **b** graph2vec samples 5 rooted subgraphs from the full graph

based on architecture instead of partial learning curves. There are various surrogate models adopted in different NAS methods, such as multilayer perceptron [30], LSTM [63], and random forest [21].

Preliminaries: graph2vec

Graph embedding [64,65] projects graphs into a continuous vector space, which preserves graphs' properties. Graph2vec is a neural embedding approach that learns representations of the graphs. Inspired by doc2vec which is a document embedding method, graph2vec views an entire graph as a document and the rooted subgraphs as words, and then learns the representations of graphs through the doc2vec skip-gram training process [66]. As shown in Fig. 1a, the document composed of words and the graph composed of rooted subgraphs in Figure 1b. Graph, rooted subgraphs and the detailed algorithm are introduced below.

Let $G = (N, E)$ represent a graph, where N as a set of nodes and E as a set of edges. Each node is associated with a label to indicate different node types. Graph2vec represents each graph by a fixed-length features which is trained to predict the rooted subgraphs in the full graph, and the rooted subgraphs are defined as following:

Definition 1 Rooted subgraphs are a specific class of subgraphs. In a given graph G , $sg_n^d = (N_{sg}, E_{sg})$ is a rooted subgraph of degree d around node n , so long as $N_{sg} \in N$, $E_{sg} \in E$ and all the nodes in the rooted subgraphs are reachable in d hops from n .

Figure 1b shows that a graph containing five nodes extracts five rooted subgraphs with $d = 1$. Given a set of graphs, graph2vec learns their embeddings in three steps:

1. extract the rooted subgraphs from all graphs to produce a vocabulary;
2. build the skip-gram model with negative sampling;
3. use the stochastic gradient descent (SGD) optimizer [67] to optimize the parameters.

Algorithm 1 graph2vec

Require: $\mathbb{G} = \{G_1, G_2, \dots, G_n\}$: Set of graphs to be learnt and $G_i = (N, E)$;
 λ : a function that maps a node to its label;
 D : Maximum degree of the rooted subgraphs;
 δ : dimension of embedding;
 e : number of epochs;
 α : learning rate.

```

1: function GRAPH2VEC( $\mathbb{G}, D, \lambda, e, \alpha$ )
2:   Initialize the  $\Phi(G)$ :  $\Phi(G)$  is the embedding of  $G$ 
3:   for  $e = 1$  to  $e$  do
4:      $\mathbb{G} = \text{Shuffle}(\mathbb{G})$ 
5:     for  $G_i$  in  $\mathbb{G}$  do
6:       for  $n \in N_i$  do
7:         for  $d = 0$  to  $D$  do
8:            $sg_n^d = \text{GETROOTEDGRAPH}(n, G_i, d)$ 
9:            $J(\Phi) = -\log \Pr(sg_n^d | \Phi(G))$ 
10:           $\Phi = \Phi - \alpha \frac{\partial J}{\partial \Phi}$ 
11:   return  $\Phi$ 
12: function GETROOTEDGRAPH( $n, G, d$ )
13:   //  $n$  is the root of the subgraph
14:    $sg_n^d = \{\}$ 
15:   if  $d = 0$  then
16:      $sg_n^d = \{\lambda(n)\}$ 
17:   else
18:      $\mathbb{N} = \{n' | (n, n') \in E\}$ 
19:      $M_n^d = \{\text{GETROOTEDGRAPH}(n', G, d-1) | n' \in \mathbb{N}\}$ 
20:      $sg_n^d = sg_n^d \cup \text{GETROOTEDGRAPH}(n, G, d-1) \oplus \text{sort}(M_n^d)$ 
21:   return  $sg_n^d$ 

```

The whole process is elaborated in Algorithm 1.

As described before, we should extract the rooted subgraphs from the graphs (see lines 12–21 of Algorithm 1). The procedure takes the root node n , graph G and the degree d as the inputs and return the rooted subgraphs sg_n^d . For cases where $d = 0$, the root node is returned. Otherwise, we first get the degree $d - 1$ rooted subgraphs of all neighbors of the root node and sort them, then concatenate them with the degree $d - 1$ rooted subgraphs of the root node. Given a sequence of the rooted subgraphs $sg_n^d = \{sg_1, sg_2, \dots, sg_l\}$ extracted from G , we intend to maximize the following log likelihood:

$$\sum_{j=1}^l \log \Pr(sg_j | G). \quad (2)$$

As shown in Fig. 2, in order to maximize Eq. (2), the skip-gram model builds l classifiers. In each classifier, the probability $\Pr(sg_i | G)$ is defined as follows:

$$\Pr(sg_i | G) = \frac{\exp(\Phi(G) \cdot w_{sg_i})}{\sum_{sg \in \mathcal{V}} \exp(\Phi(G) \cdot w_{sg})}, \quad (3)$$

where $\Phi(G)$ is the embedding vector of G , w_{sg} and w_{sg_i} are the network weights corresponding to sg and sg_i , respectively, \mathcal{V} is the vocabulary containing all rooted subgraphs.

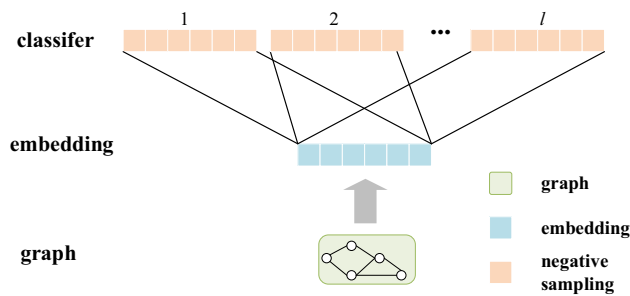


Fig. 2 The skip-gram model: the graph embedding is trained to predict the rooted subgraphs

As can be seen from Eq. (3), it is prohibitively expensive if the whole vocabulary are considered. To alleviate this problem, graph2vec adopts negative sampling which selects a small subset of rooted graphs at random that are not in the target graph to train the model.

At the beginning of training the skip-gram model, the embeddings of all the graphs in \mathbb{G} are randomly initialized and then they are updated with SGD (see lines 3–10 of Algorithm 1). For prediction, the embedding of a new graph is obtained by gradient descent. In this step, in addition to the graph embedding, the rest parameters of the model have been trained and fixed, and the graph embedding is updated through the gradient descent.

Methodology

Overall framework

In this section, the proposed SAENAS-NE using an SAEA to search for the optimal architecture in the network embedding space is given in detail. As shown in Fig. 3, it can be divided into three components: network embedding, surrogate model and EA-based NAS. In SAENAS-NE, we first train the graph2vec model which maps the architectures into the embedding space and then employs RankNet as the surrogate model to guide the search of EA. Particularly, we propose an effective environmental selection and infill criterion in SAENAS-NE. Algorithm 2 lists the framework of the SAENAS-NE.

To begin this algorithm, we use an architecture dataset to train a graph2vec model which maps architectures into the embedding space (line 1). Then, we initialize a training data $Arch$ consisting of N randomly generated architectures, where each architecture is trained. In addition, the training data constitute the initial population P_{FE} and is used to train the surrogate model RankNet (lines 3–4). Finally, we repeat the following steps until the maximum number of real evaluations is reached (lines 5–18).

Algorithm 2 SAENAS-NE

Require: $A = \{a_1, a_2, \dots, a_k\}$: architecture dataset;
 λ : a function that maps a node to its label;
 D : Maximum degree of the rooted subgraphs;
 δ : dimension of embedding;
 e : number of epochs;
 α : learning rate;
 N : size of population;
 r : number of offspring members associated with each reference solution;
 K : number of new architectures are selected from $P_{t_{max}}$;
 FE_{max} : the maximum number of the real evaluations.
Return: the best neural network architecture.

- 1: $g2v \leftarrow \text{GRAPH2VEC}(A, D, \lambda, e, \alpha)$
- 2: Generate an initial training data set $Arch = \{a_1, a_2, \dots, a_N\}$ using random sampling,
- 3: $P_{FE} = Arch, FE = |Arch|$
- 4: Train the surrogate model RankNet by $Arch$
- 5: **while** $FE \leq FE_{max}$ **do**
- 6: $P_t = P_{FE}, t = 1$
- 7: **while** $t < t_{max}$ **do**
- 8: Generate offspring Q_t
- 9: $R_t = P_t \cup Q_t$
- 10: Surrogate Assisted Evaluation
- 11: $P_{t+1} = \text{Environmentalselection}(R_t, N)$
- 12: $t = t + 1$
- 13: Select K individuals using infill criterion from $P_{t_{max}}$ and obtain their accuracy, recorded as P_{infill}
- 14: $FE = FE + |P_{infill}|$
- 15: $Arch = Arch \cup P_{infill}$
- 16: Use $Arch$ to update the surrogate model RankNet
- 17: $R_t = P_{FE} \cup P_{infill}$
- 18: $P_{FE} = \text{Environmentalselection}(R_t, N)$

In each generation, there is a surrogate-assisted evaluation phase where the predicted value of the RankNet will be used to replace the real evaluation of the individual (lines 7–12) and the initial population of surrogate-assisted evaluation is set as P_{FE} . Subsequently, an infill criterion is adopted to select several individuals P_{infill} for real evaluations, that is, to fully train the network corresponding to each individual and obtain their validation accuracy (line 13). After that, these real evaluated individuals are collected in the $Arch$ and the surrogate model is updated (lines 15–16). At last, we select the parent population of the next generation from the mixed population of P_{FE} and P_{infill} (lines 17–18).

In the following subsections, we elaborate on the detail of network embedding, surrogate model, environmental selection and infill sampling.

Network embedding

As described in Section “Preliminaries: graph2vec”, we train the skip-gram model to get the architecture embedding by predicting whether the substructures are present in the architecture and expect that architectures with more identical substructures will be closer to each other in the embedding space. In our work, the substructures are the rooted sub-

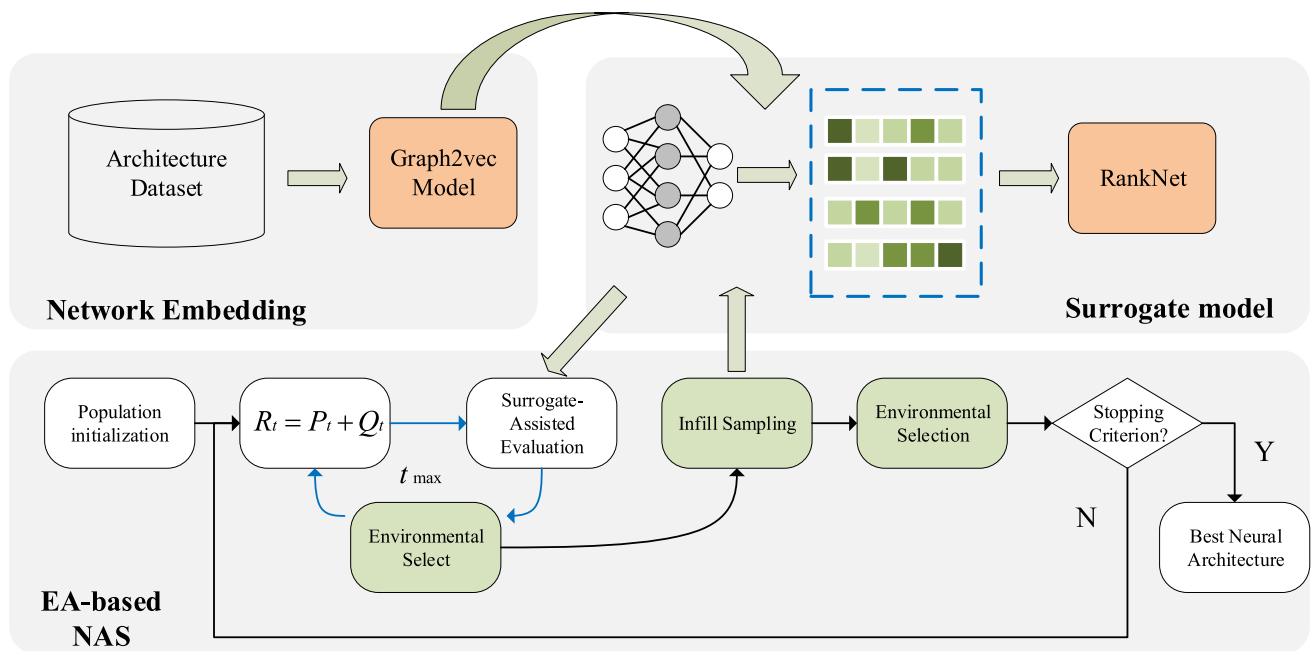


Fig. 3 The overall framework of SAENAS-NE

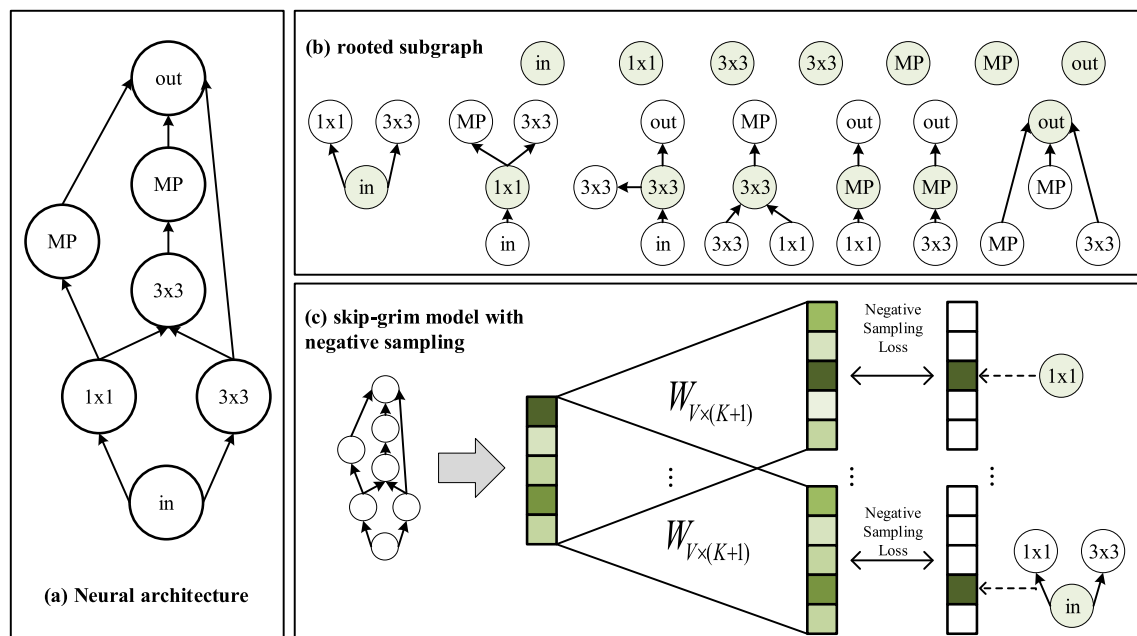


Fig. 4 An example of network embedding

graphs and a classifier will be constructed for each rooted subgraph. Figure 4 provides an example to illustrates how the network is embedded into a fixed-length vector. Figure 4a depicts a neural network in the NASBench-101 search space. The extracted rooted subgraphs with $d = 0$ or 1 are shown in Fig. 4b. In Fig. 4c, we demonstrate the process of network embedding in graph2vec. If we consider the vocabulary that contains all rooted subgraphs, the output and weights of

the classifier are huge. In order to offer a computationally efficient classifier, T. Mikolov et al. [68] propose negative sampling by the following formula:

$$\log \sigma (\Phi(G) \cdot w_{sg_i}) + \sum_{j=1}^k \mathbb{E}_{sg_j \sim P_n(sg)} \left[\log \sigma (-\Phi(G) \cdot w_{sg_j}) \right], \quad (4)$$

where σ is sigmoid function, $P_n(\text{sg})$ is the noise distribution where there are k negative samples¹ of sg_i . Equation (4) is used to replace every $\log \Pr(\text{sg}_i|G)$ in Eq. (2). For each classifier, the meaning expressed by the formula is to distinguish the target rooted subgraph sg_i from the k negative samples which are not included in the graph.

After training the skip-gram model, the model saves the embedding of all graphs $\Phi(G)$ and the weights corresponding to all rooted subgraphs w_{sg} in the training. When we require the embedding of a new architecture, we can fix the weights corresponding to the rooted subgraphs in the skip-gram model, and continuously update the embedding of the new architecture through the gradient descent.

Surrogate model

Unlike previous work [21,30,69] that use regression models as the surrogate model, we use RankNet to evaluate the individual architectures. The advantage of RankNet is that the goal of model training is to directly rank each individual architecture instead of predicting their accuracy. In RankNet, we use feedforward neural network with two hidden layers as the underlying model which maps an input vector $x \in \mathbb{R}^n$ to a real number $s_i = f(x_i)$. For each pair of architectures given, their embedding x_i, x_j is presented to the RankNet, which compute their scores $s_i = f(x_i)$ and $s_j = f(x_j)$. Then we use the sigmoid function to calculate the probability that x_i is better than x_j :

$$P_{ij} = P(x_i \triangleright x_j) = \frac{1}{1 + e^{-(s_i - s_j)}}. \quad (5)$$

Finally, RankNet trains the underlying model by minimizing the following cost function:

$$C = -\bar{P}_{ij} \log P_{ij} - (1 - \bar{P}_{ij}) \log (1 - P_{ij}). \quad (6)$$

where $\bar{P}_{ij} \in \{0, 1\}$ is the target label.

After training the RankNet, we can compute the score of each individual architecture instead of the accuracy as its fitness. To leverage the confidence level of the predictions of surrogate models to perform a global search, the surrogate models need to output an uncertainty estimation for the predictions. In our work, the embedding of the network is continuously updated through the gradient descent in the skip-gram model and different random vector initializations will eventually get different but similar embedding vectors, as shown in surrogate model diagram in Fig. 3. Thus, we can obtain several embedding of network and their predictions, and compute the mean and standard deviation of these

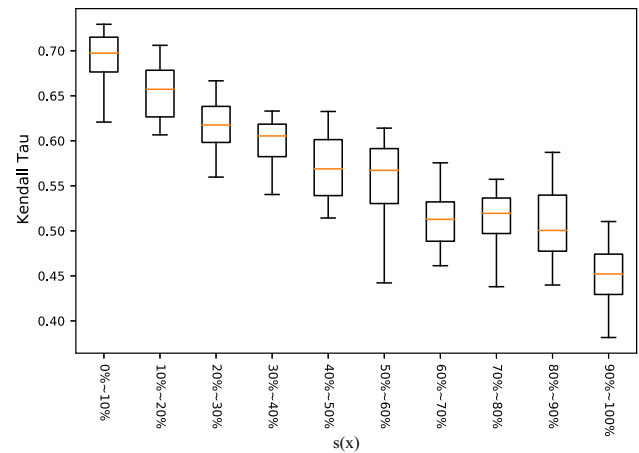


Fig. 5 The Kendall Tau metric under different $s(x)$. The horizontal axis indicates the neural architectures within a certain percentage range after the architectures sorted in ascending order of $s(x)$

predictions as their fitness $\hat{f}(x)$ and uncertainty estimation $s(x)$. In our experiments, we compute the Kendall Tau (τ) [70] metric as the correlation between the predictions rankings and the ground truth. As shown in Fig. 5, we report the change of the Kendall Tau metric of 2000 random networks under different $s(x)$ across 20 different runs. We can see that the Kendall tau metric are reduced by increasing $s(x)$. The lower the Kendall Tau the higher the true uncertainty, thus our method of calculating the uncertainty estimation is effective.

Environmental selection based on reference population

The selection of the individuals with benefits on convergence and diversity for the next population brings improvements to SAEAs. That is, the selected individuals need a high fitness and try to stay as far away as possible between different individuals. In order to solve the problem of centroid initialization and the imbalance of the number of individuals in each cluster, we propose a new method called environmental selection based on reference population. More specifically, the reference population is parent solutions, and candidate solutions with a shorter distance from a reference solution p will be preferentially associated with p unless the number of solutions in the cluster represented by p reaches a pre-specified number.

In order to cluster all candidates, we need to compute distances between children and reference solutions. After obtaining their embeddings, we compute the distance by

$$\theta_{i,j} = 1 - \frac{x_i \cdot x_j}{\|x_i\| \cdot \|x_j\|}, \quad (7)$$

¹ We set k to 5, which is recommended in <https://github.com/RaRe-Technologies/gensim>.

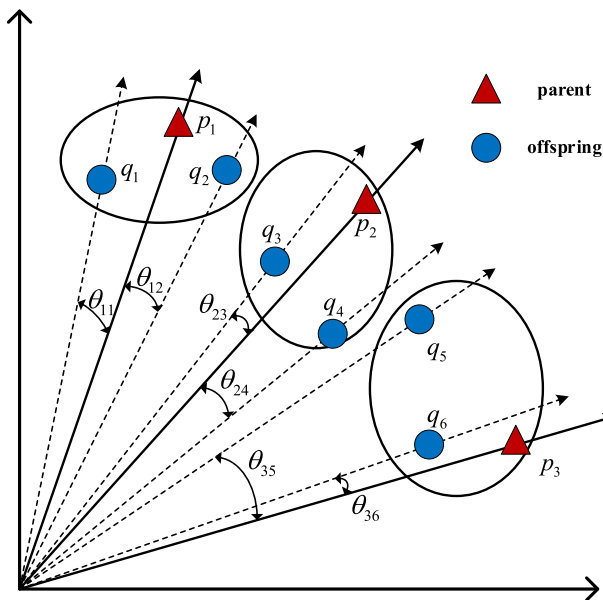


Fig. 6 Using clustering in environmental selection

where x_i and x_j are the embeddings and the second item is the cosine similarity of two vectors. $\theta_{i,j} \in [0, 2]$ and the closer the $\theta_{i,j}$ is to zero, the more similar the two architecture.

There are two environmental selection operators in Fig. 3. One selects the next population P_{t+1} from the mixed population of P_t and Q_t in the phase of surrogate-assisted evaluation, and the other selects the individuals from the parent population P_{FE} and the new evaluated offspring P_{infill} to form the next population in the phase of real evaluation. The two environmental selection operators are similar and their reference population are parent population and the worst k individuals in P_{FE} , respectively.

The details about environmental selection are presented in Algorithm 3. We first cluster the candidates based on the distance between the reference population and the offspring, and then we select the best member in each cluster for next population. Let $\Theta \in R^{N,r \times N}$ be the distance matrix that is build from Eq. (7), N is the number of reference population and r is the number of offspring members associated with each reference solution (the r is 2 in Fig. 6). An element $\theta_{i,j}$ in Θ represents the distance between the i -th reference solution and j -th child. After obtaining Θ , we find the smallest distance from the matrix Θ , suppose it is $\theta_{i,j}$, associate the i -th reference solution and j -th child and set the distance from all reference solutions to the j -th child to infinity. If the number of associated children of the i -th reference solution reaches r , the distance from the reference solution to all children is set to infinity. Repeat this process until all elements of the matrix Θ are infinite.

In the phase of surrogate-assisted evaluation, the r is 6 and in order to prevent individuals which have been real evaluated

Algorithm 3 Environmental Selection Based on Reference Population

Require: P : the reference population;
 Q : the offspring population;
 r : the number of offspring members associated with each reference solution;
Return: pop : the selected individual

```

1: Initial the distance matrix  $\Theta$ 
2:  $n_1 = |P|, n_2 = |Q|$ 
3:  $pop = \Phi, clusters = \Phi$ 
4: for  $p_i$  in  $P_t$  do
5:    $clusters.append([p_i])$ 
6: for  $\theta_{i,j}$  in  $\Theta$  do
7:   Compute  $\theta_{i,j}$  through Eq. (7)
8: while  $\exists \theta_{i,j} \neq \inf$  do
9:   Find the smallest distance  $\theta_{i,j}$ 
10:  Associate the child  $q_j$  with the reference solution  $p_i$ 
11:  Set  $\theta_{w,j} = \inf, \forall w \in [1, n_1]$ 
12:  if  $|clusters[i]| == r + 1$  then
13:    Set  $\theta_{i,w} = \inf, \forall w \in [1, n_2]$ 
14: if the phase is the surrogate-assisted evaluation then
15:   for  $C$  in  $clusters$  do
16:     Delete the individuals which have been real evaluation in  $C$ .
17:   for  $C$  in  $clusters$  do
18:     $p = best(C)$ 
19:     $pop.append(p)$ 
return  $pop$ 

```

from being repeatedly interviewed, these individuals will be deleted from the cluster. In the phase of real evaluation, the r is 1 and the reference population consists of the worst K individuals in P_{EF} . An example of candidate clustering is shown in Fig. 6, $\theta_{3,6}$ is the smallest distance, so associate child q_6 with reference solution p_3 and set $\theta_{i,6} = \inf, \forall i \in [1, 3]$. Then, the children q_1 and q_3 are, respectively, associate with p_1 and p_2 . Similarly, we set $\theta_{i,1} = \theta_{i,3} = \inf, \forall i \in [1, 3]$. Next, q_4 is associated to p_2 , and set $\theta_{i,4} = \inf, \forall i \in [1, 3]$. Particularly, the number of offspring members associated with p_2 reaches $r = 2$, so $\theta_{2,j} = \inf, \forall j \in [1, 6]$. Finally, q_2 and q_5 are also associated with p_1 and p_3 , respectively, in the same way. Because each reference solution has only r children associated with it, q_5 is associated with p_3 even though it is closer to p_2 .

Infill sampling based on multi-objective selection

The approximated fitness $\hat{f}(x)$ and the uncertainty $s(x)$ can measure the new solution's merit [71], in order to consider both for all candidate architectures in infill sampling, inspired by multi-objective optimization [72], we consider $\hat{f}(x)$ and $s(x)$ as two separate objectives and choose K new architectures from $P_{t_{max}}$ by utilizing nondominated sorting. Given two individuals x and y , x dominates y if and only if $\hat{f}(x) > \hat{f}(y), s(x) \geq s(y)$ or $\hat{f}(x) \geq \hat{f}(y), s(x) > s(y)$. The candidate architectures are sorted into several nondomi-

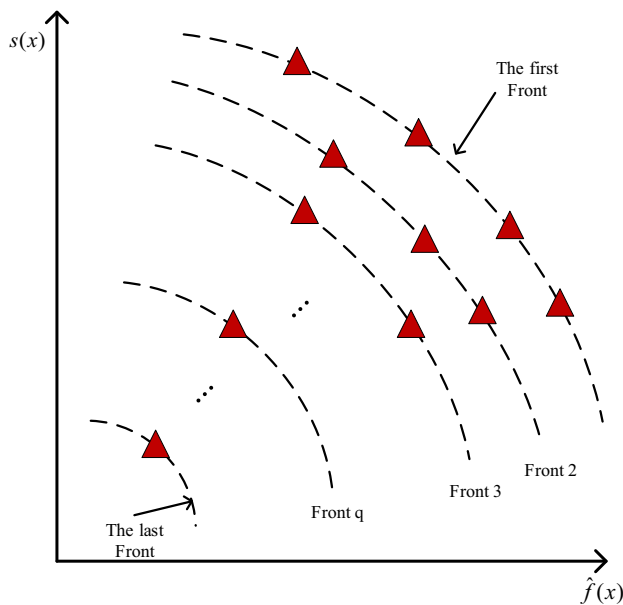


Fig. 7 All nondominated fronts sorted according to the approximated fitness $\hat{f}(x)$ and the uncertainty $s(x)$

nated fronts according to the domination relationships, Fig. 7 illustrates this process.

The candidate architectures have a higher selection priority in the lower rank nondominated front. First, we select the candidates from the front 1. If the size of the front 1 is smaller than K , all members of front 1 will be selected for real evaluation. The remaining members for real evaluations are chosen from subsequent nondominated fronts in order of their ranks. Assume that front q is the last chosen front, that is, if all members in front q are selected, the number of candidates which are real evaluated will be greater than K . In this case, we sort the members in the front q according to the approximated fitness $\hat{f}(x)$, and then select the member with the larger $\hat{f}(x)$ until the number of candidates for real evaluations reaches K .

Since we sort all the members in $P_{t_{\max}}$ by the nondominated sorting according to $\hat{f}(x)$ and $s(x)$, the selected solutions have either a higher approximated fitness, a greater uncertainty, or both. Finally, these selected solutions are used to update the surrogate model. In addition, the infill sampling strategy is implemented after environment selection, thus the diversity of the selected solutions can be guaranteed.

Experiments

In this section, we conduct experiments on three commonly used NASBench search spaces [46,73] and DARTS search space [15]. The experiments consist of three parts. First, the performance of the proposed SAENAS-NE is quantified on

the three commonly used NASBench and DARTS search spaces, and compared to the existing NAS algorithms. Second, the effectiveness of graph2vec is verified by comparing it with the existing network embedding method. Lastly, we perform the ablation experiments to validate the effectiveness of the new environmental selection and infill sampling strategies, and the effectiveness of the hyper-parameter r . All the experiments are conducted using one NVIDIA GTX 2080Ti GPU and one Intel Xeon Gold 4210R CPU.

NASBench search space

NASBench-101, NASBench-201 and NASBench-301 are the commonly used NASBench search space where various NAS algorithms can be compared with each other. Next, we introduce the three NASBench search space, respectively.

- NASBench-101 [46] consists of 423k unique convolutional architectures and all architectures are trained and evaluated three times on CIFAR-10 with different random initializations. For each architecture, its validation accuracies and test accuracies corresponding to the three independent trainings are reported. NASBench-101 builds the architecture by stacking cells and restrict the search space to a cell. The cells are defined by directed acyclic graphs on V nodes, where each node represents the operations and the adjacency matrix represents the connection of different operations. In order to limit the size of the search space, only 3×3 convolution, 1×1 convolution and 3×3 max-pool are allowed to be used and the maximum number of edges is 9.
- NASBench-201 [73] contains 15,625 architectures and each architecture generated by 4 nodes and 5 associated options (zeroize, skip-connect, 1×1 convolution, 3×3 convolution and 3×3 avg-pool). Each node and edge represent the feature map and operation, respectively. NASBench-201 provides the training, validation, and test accuracy on CIFAR-10, CIFAR-100 and ImageNet-16-120.
- NASBench-301 [74] is the first surrogate NAS benchmark which contains 10^{18} architectures. Different from NASBench-101, NASBench-201 and other tabular NAS benchmark, NASBench-301 fits various regression models on CIFAR-10 and provide a predicted accuracy of each architecture. NASBench-301 has the same search space as in DARTS [15], the architectures contain the normal and reduction cell, which is defined as a DAG with 2 input nodes, 4 intermediate nodes, and 1 output node. The nodes represent the feature map and are connected by directed edges representing one of the following 7 operations: separable convolution 3×3 , separable convolution 5×5 , dilated convolution 3×3 , dilated convolution 5×5

5, dilated convolution 3×3 , max-pool 3×3 , avg-pool 3×3 and skip connection.

Peer competitors

The compared algorithms in our experiment are summarized as follows.

- *Random search* is the simplest but competitive baseline for NAS algorithms. It randomly selects n architectures from the search space and uses the architecture with the highest validation accuracy as the final result.
- *REA* [8] adopts EA as the optimizer and introduces an age property to favor the younger individuals in each generation.
- *BANANAS* [30] is a Bayesian optimization algorithm which proposes a path-based encoding scheme for the architectures and uses an ensemble of feedforward neural networks as surrogate model.
- *GP_bayesopt* is another algorithm provided in the work [30] which sets up Bayesian optimization with Gaussian process model and UCB acquisition function.
- *Deep Networks for Global Optimization (DNGO)* [75] performs Bayesian optimization with basis function extracted from the neural network.
- *Bohamiann* is a Bayesian optimization with Hamiltonian Monte Carlo artificial neural networks [30]. It uses a Bayesian neural network as the surrogate model.
- *GCN_predictor* [30] uses Bayesian optimization as its optimizer and a graph convolutional networks as the surrogate model.
- *BONAS* [76] is a Bayesian optimization which uses GCN as a surrogate model to discover the optimal architecture and design a weighted loss focusing on architectures with high performance.
- *Arch2vec-RL* and *Arch2vec-BO* [25]² use reinforcement learning and Bayesian optimization, respectively, as optimizer to search the optimal architecture, and employ the embedding method arch2vec which adopts variational autoencoder to learn the architecture representations.
- *NPENAS-SSRL* and *NPENAS-SSCCL* [27], respectively, use evolutionary algorithms as optimizers and two self-supervised learning model for pre-training the architecture embeddings. One learns the architecture embeddings by introducing the pretext task which predicts the distance between architectures. Another first constructs positive samples and negative samples, and then proposes

a contrastive learning algorithm to learn the architecture embeddings.

Parameter settings

The common parameter settings of all algorithms are the same as in [27,30].

- The size of the architecture embedding is set to 32 for NASBench-101, NASBench-201 and NASBench-301, which is the same as the hidden layer size of GIN layer in [27].
- The maximum number of the real evaluations is set to 150 for NASBench-101, 100 for NASBench-201, and 300 for NASBench-301.

Furthermore, we use the code directly from the open-source repositories³ and the parameter settings are hardly changed. The specific parameter settings in SAENAS-NE are shown as below.

- The maximum degree of the rooted subgraphs is 2. A subgraph of degree 2 can represent up to 5 network layers in a cell, which is sufficient in the search space we use.
- The initial learning rate is 0.025 and the cosine learning rate schedule is adopted.
- The number of epochs for training graph2vec model is 40, which makes the graph2vec train well.
- The number of new architectures (K) are selected from $P_{t_{\max}}$ is set to 10, the same as the value set in [27].
- The size of population is set to 20 for NASBench-101 and NASBench-201, 30 for NASBench-301.

Neural architecture search performance

In order to discuss the behavior of the SAENAS-NE, we compare it with other algorithms described in Section 5.2 on NASBench-101, NASBench-201 and NASBench-301.

Results on NASBench-101

The performance of different NAS algorithms over 200 independent runs on NASBench-101 are reported in Table 1. In Table 1, the top two algorithms in order use real evaluations, namely, the accuracy of each candidate architecture is obtained by training the network from scratch. The middle six algorithms construct surrogate models from the original encoding of the architectures while the bottom five

² The algorithms in the [25] use the same cell for both normal and reduction cell. In our experiments we use the case where the two cells are different, which is more commonly used, and the matrix representation of its architectures refers to [30].

³ <https://github.com/naszilla/naszilla> for Random search, <https://github.com/MSU-MLSys-Lab/arch2vec> for Arch2vec-RL and Arch2vec-BO, <https://github.com/auroua/SSNENAS> for NPENAS-SSRL and NPENAS-SSCCL.

Table 1 The performance of the 12 compared algorithms and SAENAS-NE on NASBench-101

Methods	Search budget	Validation	Test	Architecture embedding
Random Search	150	94.33 \pm 0.12 (+)	93.68 \pm 0.19 (+)	–
REA	150	94.40 \pm 0.18 (+)	93.81 \pm 0.26 (+)	–
BANANAS	150	94.65 \pm 0.14 (+)	94.05 \pm 0.18 (\approx)	Supervised
GP_bayesopt	150	94.67 \pm 0.15 (\approx)	94.05 \pm 0.19 (\approx)	Supervised
DNGO	150	94.54 \pm 0.16 (+)	93.94 \pm 0.20 (+)	Supervised
Bohamiann	150	94.53 \pm 0.17 (+)	93.92 \pm 0.23 (+)	Supervised
GCN_Predictor	150	94.33 \pm 0.11 (+)	93.72 \pm 0.19 (+)	Supervised
BONAS	150	94.63 \pm 0.12 (+)	94.04 \pm 0.17 (\approx)	Supervised
Arch2vec_RL	150	94.35 \pm 0.14 (+)	93.74 \pm 0.17 (+)	Unsupervised
Arch2vec_BO	150	94.44 \pm 0.25 (+)	93.77 \pm 0.24 (+)	Unsupervised
NPENAS-SSRL	150	94.76 \pm 0.19 (–)	94.16 \pm 0.12 (–)	Self-supervised
NPENAS-CCL	150	94.72 \pm 0.17 (\approx)	94.15 \pm 0.14 (–)	Self-supervised
SAENAS-NE	150	94.72 \pm 0.14	94.08 \pm 0.15	Unsupervised
+/ \approx /–		9/2/1	7/3/2	

The bold values is the best results for each instance

The last row of the table is the result of Wilcoxon signed rank test (the significance level is 0.05). The symbols ‘+’, ‘–’, and ‘ \approx ’ indicate that SAENAS-NE is statistically significantly superior to, inferior to, and almost equivalent to the compared algorithm, respectively

algorithms map the architectures into the embedding space through unsupervised or self-supervised algorithms.

Our proposed method SAENAS-NE achieves comparable performance on NASBench-101. The results of Wilcoxon rank-sum (WRS) test show that SAENAS-NE significantly superior to nine algorithms (Random Search, REA, BANANAS, DNGO, Bohamiann, GCN_Predictor, BONAS, Arch2vec_RL, Arch2vec_BO), almost equivalent to two algorithms (GP_bayesopt, NPENAS-CCL), and inferior to NPENAS-SSRL in terms of validation accuracy. According to [69], well-performing neural network does not have a strong correlation between the validation accuracy and the test accuracy on NASBench-101. Thus, both NPENAS-SSRL and NPENAS-CCL outperform SAENAS-NE in test accuracy, which is not exactly the same as its validation accuracy.

Results on NASBench-201

Table 2 shows the results obtained from the 200 independent runs of the 12 compared algorithms and SAENAS-NE on NASBench-201. Our method SAENAS-NE achieves the best performance on both the validation accuracy and test accuracy on CIFAR-10. On CIFAR-100, SAENAS-NE obtains slightly lesser validation accuracy and test accuracy than NPENAS-SSRL and NPENAS-CCL. Specifically, the validation accuracy of SAENAS-NE differs from the accuracy of NPENAS-SSRL and NPENAS-CCL by 0.01% and 0.02%, respectively, and the test accuracy of SAENAS-NE differs by 0.01% and 0.03%, respectively. On ImageNet-16-120, SAENAS-NE is only 0.03% lower than NPENAS-CCL in

validation accuracy while achieves the best result in test accuracy.

In addition, it can be seen from the Table 2 that except the validation accuracy on CIFAR-100, SAENAS-NE maintains the lowest standard deviation in all accuracy, thus it can be proved that our method has good stability on NASBench-201.

Results on NASBench-301

As shown in Table 3, SAENAS-NE achieves the best performance compared with other 11 NAS algorithms. As NPENAS-SSRL [27] is less suitable for a large search space and is not used on DARTS-like search space, it is not compared here.

Except for GP_bayesopt, other compared algorithms are at least 0.09% less accurate than SAENAS-NE. Although GP_bayesopt performs the closest to SAENAS-NE, it is still significantly inferior to SAENAS-NE in WRS test. Figure 8 provides the validation accuracy of the best neural network during the search process of all algorithms. It is apparent from this figure that SAENAS-NE outperforms all methods except GP_bayesopt when the search budget exceeds 50 and outperforms GP_bayesopt when search budget exceeds 200.

Results on the real-world search space

To further demonstrate the effectiveness of our algorithm, we conduct experiments on the DARTS search space. The search space consists of convolutional cells and reduction cells. For each cell, two input nodes and four nodes contain two edges as input form a DAG. The final network is obtained

Table 2 The performance of the 12 compared algorithms and SAENAS-NE on NASBench-201

Methods	Search budget	CIFAR-10		CIFAR-100		ImageNet-16-120	
		Validation	Test	Validation	Test	Validation	Test
Random Search	100	91.07 ± 0.27 (+)	93.82 ± 0.24 (+)	71.44 ± 0.83 (+)	71.40 ± 0.83 (+)	45.37 ± 0.63 (+)	45.36 ± 0.67 (+)
REA	100	91.37 ± 0.25 (+)	94.06 ± 0.29 (+)	72.79 ± 0.69 (+)	72.72 ± 0.72 (+)	46.15 ± 0.45 (+)	45.99 ± 0.51 (+)
BANANAS	100	91.50 ± 0.15 (+)	94.23 ± 0.30 (+)	73.27 ± 0.57 (+)	73.25 ± 0.63 (+)	46.45 ± 0.25 (+)	46.31 ± 0.31 (≈)
GP_bayesopt	100	91.45 ± 0.23 (+)	94.16 ± 0.31 (+)	73.11 ± 0.65 (+)	73.05 ± 0.75 (+)	46.51 ± 0.25 (+)	46.25 ± 0.34 (+)
DNGO	100	91.41 ± 0.17 (+)	94.08 ± 0.26 (+)	72.71 ± 0.66 (+)	72.66 ± 0.67 (+)	46.11 ± 0.44 (+)	46.00 ± 0.48 (+)
Bohamiann	100	91.41 ± 0.18 (+)	94.09 ± 0.26 (+)	72.73 ± 0.64 (+)	72.65 ± 0.66 (+)	46.15 ± 0.45 (+)	46.03 ± 0.48 (+)
GCN_Predictor	100	91.02 ± 0.32 (+)	93.74 ± 0.33 (+)	71.43 ± 0.72 (+)	71.43 ± 0.77 (+)	45.47 ± 0.72 (+)	45.36 ± 0.78 (+)
BONAS	100	91.56 ± 0.10 (≈)	94.32 ± 0.15 (+)	73.32 ± 0.40 (+)	73.30 ± 0.46 (+)	46.56 ± 0.19 (≈)	46.31 ± 0.30 (≈)
Arch2vec_RL	100	91.43 ± 0.28 (+)	94.23 ± 0.26 (+)	73.12 ± 0.66 (+)	73.06 ± 0.87 (+)	46.32 ± 0.20 (+)	46.35 ± 0.35 (≈)
Arch2vec_BO	100	91.51 ± 0.13 (+)	94.31 ± 0.14 (+)	73.38 ± 0.34 (+)	73.44 ± 0.22 (+)	46.33 ± 0.21 (+)	46.35 ± 0.27 (≈)
NPENAS-SSRL	100	91.56 ± 0.14 (+)	94.32 ± 0.19 (+)	73.47 ± 0.22 (-)	73.47 ± 0.30 (-)	46.53 ± 0.33 (+)	45.83 ± 0.60 (+)
NPENAS-CCL	100	91.57 ± 0.13 (+)	94.32 ± 0.19 (+)	73.48 ± 0.15 (-)	73.49 ± 0.23 (-)	46.62 ± 0.34 (-)	45.61 ± 0.41 (+)
SAENAS-NE	100	91.58 ± 0.09	94.34 ± 0.12	73.46 ± 0.18	73.46 ± 0.20	46.59 ± 0.14	46.36 ± 0.26
+/-		11/1/0	12/0/0	10/0/2	10/0/2	10/1/1	7/5/0

The bold values is the best results for each instance

The last row of the table is the result of Wilcoxon signed rank test (the significance level is 0.05). The symbols '+', '-', and '≈' indicate that SAENAS-NE is statistically significantly superior to, inferior to, and almost equivalent to the compared algorithm, respectively

Table 3 The performance of the 11 compared algorithms and SAENAS-NE on NASBench-301

Methods	Search budget	Validation	Architecture embedding
Random Search	300	94.34 ± 0.10 (+)	–
REA	300	94.86 ± 0.11 (+)	–
BANANAS	300	94.89 ± 0.11 (+)	Supervised
GP_bayesopt	300	94.98 ± 0.11 (+)	Supervised
DNGO	300	94.85 ± 0.10 (+)	Supervised
Bohamiann	300	94.85 ± 0.10 (+)	Supervised
GCN_Predictor	300	94.35 ± 0.11 (+)	Supervised
BONAS	300	94.91 ± 0.10 (+)	Supervised
Arch2vec_RL	300	94.48 ± 0.08 (+)	Unsupervised
Arch2vec_BO	300	94.43 ± 0.12 (+)	Unsupervised
NPENAS-CCL	300	94.92 ± 0.12 (+)	Self-supervised
SAENAS-NE	300	95.01 ± 0.07	Unsupervised
+/-/≈		11/0/0	

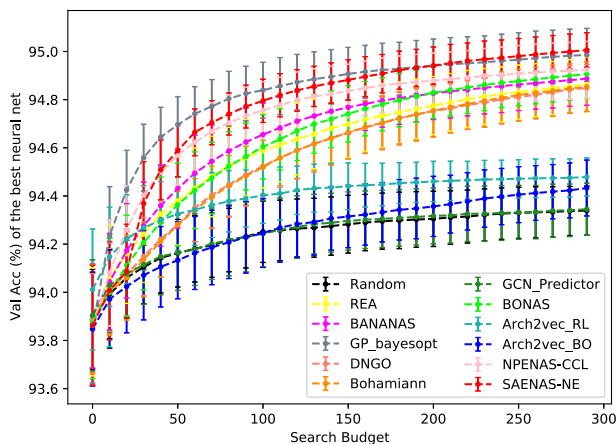
The bold values is the best results for each instance

The last row of the table is the result of Wilcoxon signed rank test (the significance level is 0.05). The symbols '+', '−', and '≈' indicate that SAENAS-NE is statistically significantly superior to, inferior to, and almost equivalent to the compared algorithm, respectively

Table 4 The performance of the 7 compared algorithms and SAENAS-NE on DARTS search space. The test accuracy is averaged over 5 seed

NAS methods	Test Acc (Avg)	Test Acc (Best)	Params (M)	GPU days
Random Search	96.9 ± 0.18	97.29	3.2	4
ENAS [77]	–	97.11	4.6	–
RS WS [78]	97.15 ± 0.08	97.29	2.7	8.7
DARTS [15]	97.24 ± 0.09	–	3.3	5
BANANAS [30]	97.36	97.43	3.6	11.8
arch2vec-RL [25]	97.35 ± 0.05	97.4	3.3	9.5
arch2vec-BO [25]	97.44 ± 0.05	97.52	3.6	10.6
SAENAS-NE	97.48 ± 0.08	97.55	2.9	9

The bold values is the best results for each instance

**Fig. 8** Performance comparison of NAS algorithms on NASBench-301

by stacking these two cells. For simplicity, we search for the same convolutional cells and reduction cells as in [25]. The

implementations of SAENAS-NE on DARTS is available at <https://github.com/HandingWangXDGroup/SAENAS-NE>.

Similar to [25], we set the maximum number of the real evaluations to 100. When an individual needs to be real evaluated, the individual will be decoded into a neural network and trained from scratch for 50 epoch, then the average validation accuracy of the last 5 epochs is used as the real evaluation value of the individual. As shown in Table 4, our algorithm achieves comparable classification accuracy with fewer parameters.

Effects of network embedding and surrogate model

To verify the effectiveness of the graph2vec applied to the network embedding, we compare it with other embedding methods. The comparison is performed on NASBench-101, NASBench-201, NASBench-301 under the search budgets of 20, 50, 100 and 200. To be fair, all the embedding methods use RankNet as the predictor and the prediction performance of the architecture embeddings is used as the metric which

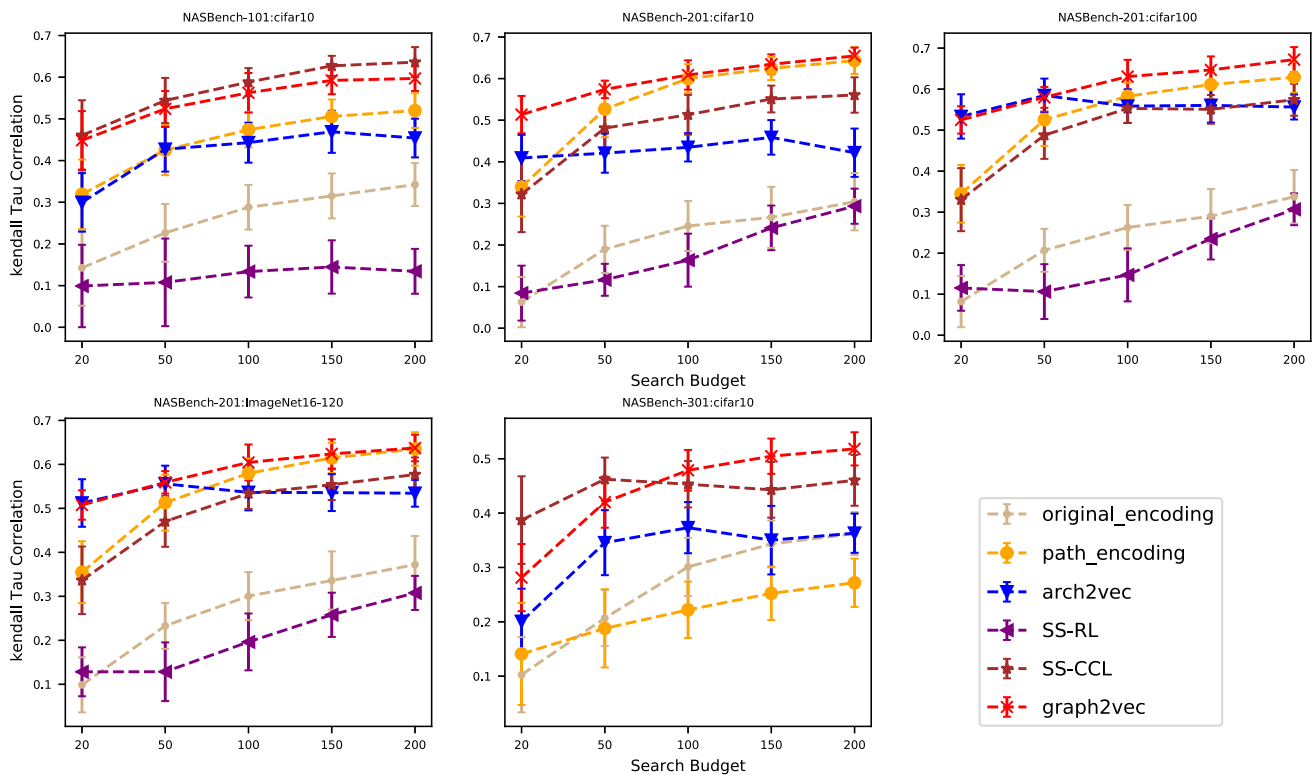


Fig. 9 Predictive performance of embedding methods with different search budget

measures how well the embedding methods. The result with Kendall's Tau coefficient is plotted in Fig. 9.

On the NASBench-101, the performance of graph2vec is slightly inferior to SS-CCL but significantly better than that of other embedding methods, including original encoding, path encoding, arch2vec and SS-RL. SS-RL shows the worst performance. Unlike the work in [27], the weights of the embedding part of the neural network in SS-RL and SS-CCL are not updated after the self-supervised learning is completed, which ensures that architecture embedding is not affected by subsequent supervised training.

On the NASBench-201, graph2vec achieves its best performance in three different dataset, namely CIFAR-10, CIFAR-100 and ImageNet-16-120. Path encoding performs close to graph2vec when the search budget is more than 100, but it is significantly worse than graph2vec when the search budget is less than 100. Arch2vec has similar performance to graph2vec at the search budget of 20 and 50 on CIFAR-100 and ImageNet-16-120, but the gap with graph2vec increases as the search budget increases. Except for path encoding and arch2vec, other embedding methods including original encoding, SS-RL and SS-CCL all have clear performance gaps with graph2vec in all search budget across three different datasets.

On the NASBench-301, graph2vec and SS-CCL have comparable performance and both outperform other embed-

ding methods

(original encoding, path encoding and arch2vec). SS-CCL has better performance when the search budget is 20 and 50, whereas graph2vec achieves the best performance with the search budget exceeds 50. SS-RL is less suitable for the large search space [27], thus we do not compare the performance of SS-RL in NasBench-301.

In summary, graph2vec achieves a superior performance on all three NASBench search spaces. Original encoding and SS-RL have poor performance. A decent performance of path encoding is maintained in three different dataset of NASBench-201 but decreased on NASBench-101 and NASBench-301 with larger search spaces containing more architectures. Arch2vec achieves moderate performance among all embedding methods on the three NASBench search space. SS-CCL shows the best performance on NASBench-101, but the performance decreases on NASBench-201 and NASBench-301.

Effects of environmental selection and infill sampling

To further analyze the behavior of our environmental selection strategy and infill criterion, we set baseline as SAENAS-NE-w/o-S&I, which is SAENAS-NE without the proposed environmental selection and the infill criterion.

Table 5 Performance comparison of three version of SAENAS-NE

Methods	Search budget	Validation (%) AVG	Test (%) STD
SAENAS-NE-w/o-S&I	300	94.96	0.09
SAENAS-NE-w/o-I	300	94.99	0.08
SAENAS-NE	300	95.01	0.07

The bold values is the best results for each instance

Another compared method is SAENAS-NE-w/o-I, which is a version of SAENAS-NE with only the proposed environmental selection strategy. As NASBench-301 is the largest among the three NASBench search space, SAENAS-NE-w/o-S&I, SAENAS-NE-w/o-I and SAENAS-NE are run independently over 200 times, respectively, and compared their validation accuracy on NASBench-301 for analyzing the effect of the proposed environmental selection and infill sampling strategies. The results of performance comparison are summarized in Table 5. As can be seen from the table, SAENAS-NE-w/o-S&I achieves validation accuracy of 94.96%. After adopting the proposed environmental selection strategy, the validation accuracy of SAENAS-NE-w/o reaches 94.99% (0.03% improvement over the baseline). SAENAS-NE contains both the environmental selection and infill criterion, and its validation accuracy reaches 95.01% (0.05% improvement over the baseline). In addition, from the standard deviation of the validation accuracy of the three algorithms, the proposed environmental selection and infill sampling can improve the stability of the algorithm.

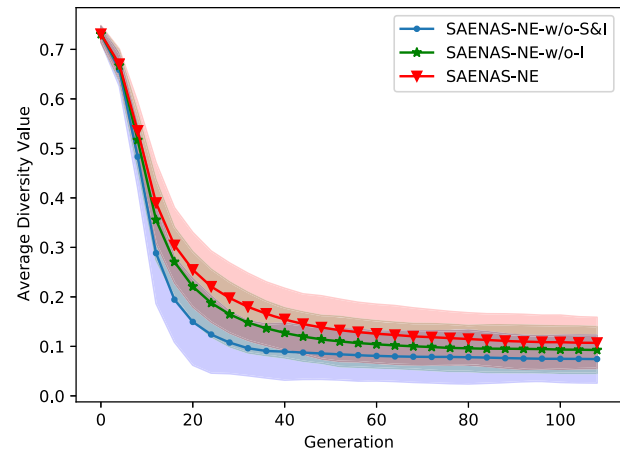
The proposed environmental selection strategy and infill criterion can improve the diversity of the population to avoid local optima. To verify this, the average diversity value (ADV) are defined as a measure of population diversity,

$$ADV(P) = \frac{\sum_{i \neq j} \theta_{i,j}}{N \times (N-1)/2}, \quad (8)$$

where P is the population and $\theta_{i,j}$ is calculated by Eq. (7) to represent the distance between the i -th individual and the j -th individual in the embedding space in the population P . The ADVs of SAENAS-NE-w/o-S&I, SAENAS-NE-w/o-I and SAENAS-NE are presented in Fig. 10. As can be seen from the figure, the proposed environmental selection strategy and infill criterion slow down the deterioration of diversity to increase the possibility of finding a better individual.

Effects of parameter r in the environmental selection

The parameter r is the number of offspring members associated with each reference evaluation in environmental selection. In the phase of real evaluation, the update of the population is a one-to-one competition between the parent individual and the child individual, thus the value of r is fixed to 1. To investigate the influence of the r in the

**Fig. 10** Average diversity value plots for three version of SAENAS-NE under over 200 independent runs on NASBench-301

phase of surrogate-assisted evaluation, we set the parameter $r = 2, 4, 6, 8, 10$ and compare their performance under 20 independent runs on NASBench-301. The change of validation accuracy of the best individual in the population over the search cost for different r are plotted in Fig. 11a. From these results, the best performance on NASBench-301 is achieved when $r = 6$. A larger r means more offspring and more individuals in each cluster, which can enhance the convergence of the optimization. However, it can be seen from Fig. 11b that a larger r makes the population diversity decay more severely. Therefore, $r = 6$ is a suitable value, which can balance the convergence and diversity of the population, and thus obtain better performance.

Conclusions

This paper presents a novel surrogate-assisted evolutionary algorithm with network embedding for neural architecture search, where a graph2vec model is proposed to generate meaningful representation of each architecture and a RankNet model is trained to approximate the true accuracy of the neural network. The graph2vec model enables the vector representations of similar topologically structured architectures to be closer in the embedding space. Furthermore, to enhance the search ability of the algorithm and efficiently update the surrogate model, a new environmental selec-

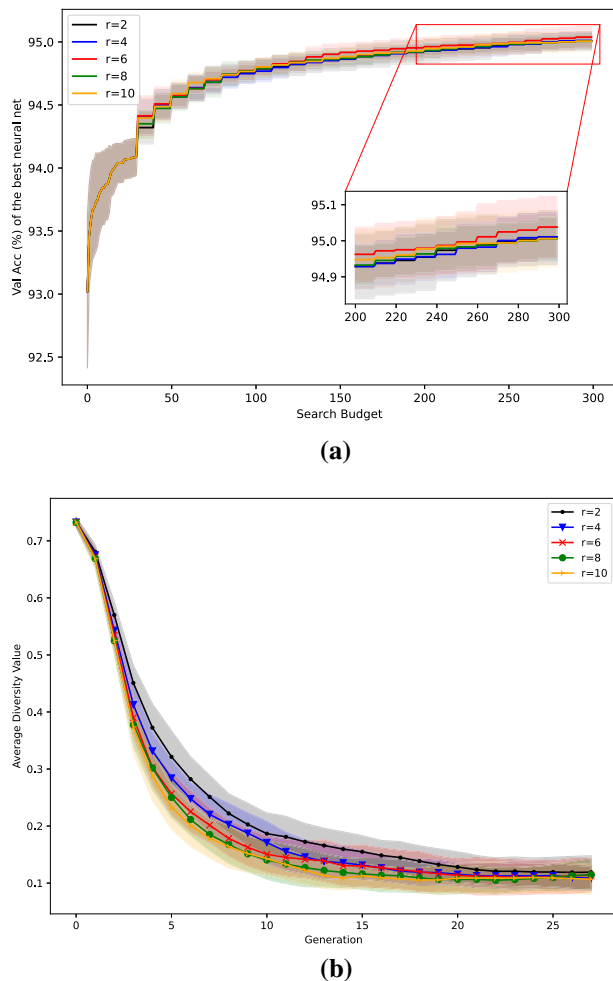


Fig. 11 Effects of parameter r in the environmental selection: **a** performance comparison of different r values. **b** Average diversity value plots for SAENAS-NE with different r values

tion strategy and infill criterion are designed. Our proposed method SAENAS-NE achieves competitive performance on three different NASBench search space. Extensive experiments demonstrate that the proposed embedding method can offer informative representations and improve the accuracy of the surrogate model. In addition, the environmental selection and infill sampling strategy improves the performance of the algorithm by enhancing the diversity of the population and updating the surrogate model efficiently.

For the future work, the generalization ability of SAENAS-NE requires further investigation by applying it on more search spaces or extending it to other tasks including object detection and natural language processing. In addition, more methods to improve the quality of the network embedding are worthy of research, such as contrastive learning [79]. In addition, designing adaptive environmental selection and infill sampling strategies to adapt to different search spaces is also a promising future research direction.

Acknowledgements This work was supported in part by the National Natural Science Foundation of China (No. 61976165).

Data availability statement The data used to support the findings of this study are available from the author upon request.

Open Access This article is licensed under a Creative Commons Attribution 4.0 International License, which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons licence, and indicate if changes were made. The images or other third party material in this article are included in the article's Creative Commons licence, unless indicated otherwise in a credit line to the material. If material is not included in the article's Creative Commons licence and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder. To view a copy of this licence, visit <http://creativecommons.org/licenses/by/4.0/>.

References

1. He K, Zhang X, Ren S, Sun J (2016) Deep residual learning for image recognition. In: Proceedings of the IEEE Conference on computer vision and pattern recognition, pp 770–778
2. Ren S, He K, Girshick R, Sun J (2015) Faster r-cnn: towards real-time object detection with region proposal networks. *Adv Neural Inf Process Syst* 28:91–99
3. Sutskever I, Vinyals O, Le V (2014) Sequence to sequence learning with neural networks. In: Proceedings of the 27th International Conference on Neural Information Processing Systems - Volume 2, NIPS'14, page 3104–3112, Cambridge, MA, USA, 2014. MIT Press
4. Duchi J, Hazan E, Singer Y (2011) Adaptive subgradient methods for online learning and stochastic optimization. *J Mach Learn Res* 12(7):2121–2159
5. Zeiler MD (2012) Adadelta: an adaptive learning rate method. arXiv preprint [arXiv:1212.5701](https://arxiv.org/abs/1212.5701)
6. Kingma DP, Ba J (2014) Adam: a method for stochastic optimization. arXiv preprint [arXiv:1412.6980](https://arxiv.org/abs/1412.6980)
7. Real E, Moore S, Selle A, Saxena S, Suematsu YL, Tan J, Le QV, Kurakin A (2017) Large-scale evolution of image classifiers. In: International Conference on Machine Learning, pp 2902–2911. PMLR
8. Real E, Aggarwal A, Huang Y, Le QV (2019) Regularized evolution for image classifier architecture search. In: Proceedings of the aaai Conference on artificial intelligence, volume 33, pp 4780–4789
9. Sun Y, Xue B, Zhang M, Yen GG (2019) Completely automated cnn architecture design based on blocks. *IEEE Trans Neural Netw Learning Syst* 31(4):1242–1254
10. Xie L, Yuille A (2017) Genetic cnn. In: Proceedings of the IEEE International Conference on computer vision, pp 1379–1388
11. Zhang H, Jin Y, Cheng R, Hao K (2020) Efficient evolutionary search of attention convolutional networks via sampled training and node inheritance. *IEEE Trans Evol Comput* 25(2):371–385
12. Hu S, Cheng R, He C, Lu Z, Wang J, Zhang M (2021) Accelerating multi-objective neural architecture search by random-weight evaluation. *Complex Intell Syst*, pp 1–10
13. Zoph B, Le QV (2016) Neural architecture search with reinforcement learning. arXiv preprint [arXiv:1611.01578](https://arxiv.org/abs/1611.01578)
14. Zhong Z, Yang Z, Deng B, Yan J, Wu W, Shao J, Liu C-L (2020) Efficient block-wise neural network architecture generation. *IEEE Trans Pattern Anal Mach Int* 43(7):2314–2328, 2021

15. Liu H, Simonyan K, Yang Y (2018) Darts: differentiable architecture search. arXiv preprint [arXiv:1806.09055](https://arxiv.org/abs/1806.09055)
16. Xu Y, Xie L, Zhang X, Chen X, Qi G-J, Tian Q, Xiong H (2019) Pc-darts: partial channel connections for memory-efficient architecture search. arXiv preprint [arXiv:1907.05737](https://arxiv.org/abs/1907.05737)
17. Jin Y, Wang H, Chugh T, Guo D, Miettinen K (2018) Data-driven evolutionary optimization: an overview and case studies. *IEEE Trans Evol Comput* 23(3):442–458
18. Zheng N, Wang H, Yuan B (2022) An adaptive model switch-based surrogate-assisted evolutionary algorithm for noisy expensive multi-objective optimization. *Complex Intell Syst*, pp 1–18
19. Haibo Yu, Kang L, Tan Y, Zeng J, Sun C (2021) A multi-model assisted differential evolution algorithm for computationally expensive optimization problems. *Complex Intell Syst* 7(5):2347–2371
20. Wang H, Jin Y, Sun C, Doherty J (2018) Offline data-driven evolutionary optimization using selective surrogate ensembles. *IEEE Trans Evol Comput* 23(2):203–216
21. Sun Y, Wang H, Xue B, Jin Y, Yen Gary G, Zhang M (2019) Surrogate-assisted evolutionary deep learning using an end-to-end random forest-based performance predictor. *IEEE Trans Evol Comput* 24(2):350–364
22. Lu Z, Deb K, Goodman E, Banzhaf W, Boddeti VN (2020) Nsganetv2: evolutionary multi-objective surrogate-assisted neural architecture search. In: *European Conference on computer vision*, pp 35–51. Springer
23. Rawal A, Miikkulainen R (2018) From nodes to networks: evolving recurrent neural networks. arXiv preprint [arXiv:1803.04439](https://arxiv.org/abs/1803.04439)
24. Deng B, Yan J, Lin D (2017) Peephole: predicting network performance before training. arXiv preprint [arXiv:1712.03351](https://arxiv.org/abs/1712.03351)
25. Shen Yan, Zheng Yu, Wei Ao, Xiao Zeng, Mi Zhang (2020) Does unsupervised architecture representation learning help neural architecture search? *Adv Neural Inf Process Syst* 33:12486–12498
26. Cheng H-P, Zhang T, Li S, Yan F, Li M, Chandra V, Li H, Chen Y (2020) Nsgem: neural architecture search via graph embedding method. arXiv preprint [arXiv:2007.04452](https://arxiv.org/abs/2007.04452)
27. Wei C, Tang Y, Niu CNC, Haihong H, Wang Y, Liang J (2021) Self-supervised representation learning for evolutionary neural architecture search. *IEEE Comput Intell Mag* 16(3):33–49
28. Wang H, Jin Y, Doherty J (2017) Committee-based active learning for surrogate-assisted particle swarm optimization of expensive problems. *IEEE Trans Cybern* 47(9):2664–2677
29. Song Z, Wang H, He C, Jin Y (2021) A kriging-assisted two-archive evolutionary algorithm for expensive many-objective optimization. *IEEE Trans Evol Comput* 25:1
30. White C, Neiswanger W, Savani Y (2019) Bananas: Bayesian optimization with neural architectures for neural architecture search. arXiv preprint [arXiv:1910.11858](https://arxiv.org/abs/1910.11858), 1(2)
31. Zhang H, Song S, Zhou A, Gao X-Z (2014) A clustering based multiobjective evolutionary algorithm. In: *2014 IEEE Congress on evolutionary computation (CEC)*, pp 723–730. IEEE
32. Lin Q, Liu S, Wong K-C, Gong M, Coello Carlos A, Jianyong CC, Zhang J (2018) A clustering-based evolutionary algorithm for many-objective optimization problems. *IEEE Trans Evol Comput* 23(3):391–405
33. Narayanan A, Chandramohan M, Venkatesan R, Chen L, Liu Y, Jaiswal S (2017) graph2vec: learning distributed representations of graphs. arXiv preprint [arXiv:1707.05005](https://arxiv.org/abs/1707.05005)
34. Burges C, Shaked T, Renshaw E, Lazier A, Deeds M, Hamilton N, Hullender G (2005) Learning to rank using gradient descent. In: *Proceedings of the 22nd International Conference on Machine learning*, pp 89–96
35. Sun Y, Xue B, Zhang M, Yen GG (2019) Evolving deep convolutional neural networks for image classification. *IEEE Trans Evol Comput* 24(2):394–407
36. Suganuma M, Shirakawa S, Nagao T (2017) A genetic programming approach to designing convolutional neural network architectures. In: *Proceedings of the Genetic and Evolutionary Computation Conference*, pp 497–504
37. Suganuma M, Kobayashi M, Shirakawa S, Nagao T (2020) Evolution of deep convolutional neural networks using cartesian genetic programming. *Evol Comput* 28(1):141–163
38. McGhie A, Xue B, Zhang M (2020) Gpcnn: evolving convolutional neural networks using genetic programming. In: *2020 IEEE Symposium Series on Computational Intelligence (SSCI)*, pp 2684–2691
39. Real E, Liang C, So D, Le Q (2020) Automl-zero: evolving machine learning algorithms from scratch. In: *International Conference on machine learning*, pp 8007–8019. PMLR
40. Junior FEF, Yen GG (2019) Particle swarm optimization of deep neural networks architectures for image classification. *Swarm Evol Comput* 49:62–74
41. Wang B, Sun Y, Xue B, Zhang M (2019) Evolving deep neural networks by multi-objective particle swarm optimization for image classification. In: *Proceedings of the Genetic and Evolutionary Computation Conference*, pp 490–498
42. Awad N, Mallik N, Hutter F (2020) Differential evolution for neural architecture search. arXiv preprint [arXiv:2012.06400](https://arxiv.org/abs/2012.06400)
43. Byla E, Pang W (2019) Deepswarm: optimising convolutional neural networks using swarm intelligence. arXiv preprint [arXiv:1905.07350](https://arxiv.org/abs/1905.07350)
44. Dorigo M, Di Caro G, Gambardella LM (1999) Ant algorithms for discrete optimization. *Artif Life* 5(2):137–172
45. Sharaf AI, Radwan EF (2020) An automated approach for developing a convolutional neural network using a modified firefly algorithm for image classification. In: Nilanjan D (ed) *Applications of firefly algorithm and its variants*. Springer, pages 99–118
46. Ying C, Klein A, Christiansen E, Real E, Murphy K, Hutter F (2019) Nas-bench-101: towards reproducible neural architecture search. In: *International Conference on machine learning*, pp 7105–7114. PMLR
47. White C, Neiswanger W, Nolen S, Savani Y (2020) A study on encodings for neural architecture search. arXiv preprint [arXiv:2007.04965](https://arxiv.org/abs/2007.04965)
48. Tan M, Chen B, Pang R, Vasudevan V, Sandler M, Howard A, Le QV (2019) Mnasnet: Platform-aware neural architecture search for mobile. In: *Proceedings of the IEEE/CVF Conference on computer vision and pattern recognition*, pp 2820–2828
49. Wu B, Dai X, Zhang P, Wang Y, Sun F, Wu Y, Tian Y, Vajda P, Jia Y, Keutzer K (2019) Fbnet: hardware-aware efficient convnet design via differentiable neural architecture search. In: *Proceedings of the IEEE/CVF Conference on computer vision and pattern recognition*, pp 10734–10742
50. Tian Y, Peng S, Yang S, Zhang X, Tan KC (2002) Jin Y (2021) Action command encoding for surrogate-assisted neural architecture search. *IEEE Trans Cognitive Dev Syst* 14(3):1129–1142
51. Zoph B, Vasudevan V, Shlens J, Le QV (2018) Learning transferable architectures for scalable image recognition. In: *Proceedings of the IEEE Conference on computer vision and pattern recognition*, pp 8697–8710
52. Zela A, Klein A, Falkner S, Hutter F (2018) Towards automated deep learning: efficient joint neural architecture and hyperparameter search. arXiv preprint [arXiv:1807.06906](https://arxiv.org/abs/1807.06906)
53. Klein A, Falkner S, Bartels S, Hennig P, Hutter F (2017) Fast Bayesian Optimization of Machine Learning Hyperparameters on Large Datasets. In Aarti Singh and Jerry Zhu, editors, *Proceedings of the 20th International Conference on Artificial Intelligence and Statistics*, volume 54 of *Proceedings of Machine Learning Research*, pages 528–536. PMLR, 20–22 Apr 2017

54. Chrabaszcz P, Loshchilov I, Hutter F (2017) A downsampled variant of imagenet as an alternative to the cifar datasets. [arXiv preprint arXiv:1707.08819](#)
55. Elsken T, Metzen JH, Hutter F (2019) Neural architecture search: a survey. *J Mach Learn Res* 20(1):1997–2017
56. Brock A, Lim T, Ritchie JM, Weston N (2017) Smash: one-shot model architecture search through hypernetworks. [arXiv preprint arXiv:1708.05344](#)
57. Bender G, Kindermans P-J, Zoph B, Vasudevan V, Le Q (2018) Understanding and simplifying one-shot architecture search. In: *International Conference on machine learning*, pp 550–559. PMLR
58. Guo Z, Zhang X, Mu H, Heng W, Liu Z, Wei Y, Sun J (2020) Single path one-shot neural architecture search with uniform sampling. In: *European Conference on computer vision*, pp 544–560. Springer
59. Chu X, Zhang B, Xu R (2021) Fairnas: rethinking evaluation fairness of weight sharing neural architecture search. In: *Proceedings of the IEEE/CVF International Conference on computer vision*, pp 12239–12248
60. Yu K, Ranftl R, Salzmann M (2021) Landmark regularization: ranking guided super-net training in neural architecture search. In: *Proceedings of the IEEE/CVF Conference on computer vision and pattern recognition*, pp 13723–13732
61. Klein A, Falkner S, Springenberg JT, Hutter F (2017) Learning curve prediction with bayesian neural networks. In *International Conference on Learning Representations*
62. Baker B, Gupta O, Raskar R, Naik N (2017) Accelerating neural architecture search using performance prediction. [arXiv preprint arXiv:1705.10823](#)
63. Liu C, Zoph B, Neumann M, Shlens J, Hua W, Li L-J, Fei-Fei L, Yuille A, Huang J, Murphy K (2018) Progressive neural architecture search. In: *Proceedings of the European Conference on computer vision (ECCV)*, pp 19–34
64. Goyal P, Ferrara E (2018) Graph embedding techniques, applications, and performance: a survey. *Knowl-Based Syst* 151:78–94
65. Mohan A, Pramod KV (2021) Temporal network embedding using graph attention network. *Complex Intell Syst*, pp 1–15
66. Mikolov T, Chen K, Corrado G, Dean J (2013) Efficient estimation of word representations in vector space. [arXiv preprint arXiv:1301.3781](#)
67. Bottou L (2010) Large-scale machine learning with stochastic gradient descent. In: *Proceedings of COMPSTAT'2010*, pp 177–186. Springer
68. Mikolov T, Sutskever I, Chen K, Corrado GS, Dean J (2013) Distributed representations of words and phrases and their compositionality. In: Burges CJ, Bottou L, Welling M, Ghahramani Z, Weinberger KQ (eds) *Advances in Neural Information Processing Systems*, Curran Associates, Inc vol 26
69. Wen W, Liu H, Chen Y, Li H, Bender G, Kindermans P-J (2020) Neural predictor for neural architecture search. In: *European Conference on computer vision*, pp 660–676. Springer
70. Kendall MG (1938) A new measure of rank correlation. *Biometrika* 30(1/2):81–93
71. Wang Z, Zhang Q, Ong Y-S, Yao S, Liu H, Luo J (2021) Choose appropriate subproblems for collaborative modeling in expensive multiobjective optimization. *IEEE Trans Cybern* 1–14
72. Deb K, Pratap A, Agarwal S, Meyarivan TAMT (2002) A fast and elitist multiobjective genetic algorithm: Nsga-ii. *IEEE Trans Evol Comput* 6(2):182–197
73. Dong X, Yang Y (2020) Nas-bench-201: extending the scope of reproducible neural architecture search. [arXiv preprint arXiv:2001.00326](#)
74. Siems J, Zimmer L, Zela A, Lukasik J, Keuper M, Hutter F (2020) Nas-bench-301 and the case for surrogate benchmarks for neural architecture search. [arXiv preprint arXiv:2008.09777](#)
75. Snoek J, Rippel O, Swersky K, Kiros R, Satish N, Sundaram N, Patwary M, Prabhat Mr, Adams R (2015) Scalable bayesian optimization using deep neural networks. In: *International Conference on machine learning*, pp 2171–2180. PMLR
76. Shi H, Pi R, Xu H, Li Z, Kwok JT, Zhang T (2019) Multiobjective neural architecture search via predictive network performance optimization. [arXiv:1911.09336](#)
77. Pham H, Guan M, Zoph B, Le Q, Dean J (2018) Efficient neural architecture search via parameters sharing. In: *International Conference on machine learning*, pp 4095–4104. PMLR
78. Li L, Talwalkar A (2020) Random search and reproducibility for neural architecture search. In: *Uncertainty in artificial intelligence*, vol 115. pp 367–377. PMLR
79. Yuan Z, Li G, Wang Z, Sun J, Cheng R (2022) Rl-csl: a combinatorial optimization method using reinforcement learning and contrastive self-supervised learning. *IEEE Trans Emerg Top Comput Intell* 1–15

Publisher's Note Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.