IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Date of publication xxxx 00, 0000, date of current version xxxx 00, 0000.

Digital Object Identifier 10.1109/ACCESS.2017.Doi Number

Image Classification Based on Automatic Neural Architecture Search using Binary Crow Search Algorithm

MOBEEN AHMAD, MUHAMMAD ABDULLAH, HYEONJOON MOON, SEONG JOON YOO, AND DONGIL HAN (Member, IEEE)

Department of Computer Engineering, Sejong University, Seoul 05006, South Korea

Corresponding author: Dongil Han (dihan@sejong.ac.kr)

This work was supported in part by the National Research Foundation of Korea Grant funded by the Korean Government under Grant NRF-2017R1D1A1B03028394, and in part by the Institute of Information and Communications Technology Planning and Evaluation (IITP) Grant funded by the Korean Government (MSIT), Development of AI-Convergence Technologies for Smart City Industry Productivity Innovation, under Grant 2019-0-00136.

ABSTRACT Neural architectures have accelerated the advancement in various domains by enabling automatic pattern detection, image classification, audio recognition, and face recognition etc. However, they are computationally expensive to design and expert knowledge in various domains is required. In this paper, a swarm intelligence algorithm is proposed to search for novel architectures without human intervention that can achieve comparable performance to those of human-designed architectures. This work is inspired by current neural architecture search approaches based on reinforcement learning and genetic algorithm. However, not much attention is paid towards swarm intelligence metaheuristics-based neural architecture search. A framework is proposed for automatically designing neural architectures based on a swarm intelligence metaheuristic: Crow Search Algorithm. First, Crow Search Algorithm is integrated with binary network representation. To make it compatible for Neural Architecture Search, the original distance metric is replaced with hamming distance-based similarity measure. Second, the tuning parameters of Crow Search Algorithm are reduced by replacing the static flight length parameter with our dynamic flight length distribution algorithm. Third, the target selection method (random selection) is replaced by tournament select method. The proposed framework is used to search for architectures on MNIST, CIFAR10, and CIFAR100 datasets and achieved 0.18%, 3.48%, and 15.64% test error, respectively. Furthermore, small-scale transfer experiments are conducted to search architectures for Tiny ImageNet and achieved 34.43% test error. Nonparametric statistical analysis is performed to validate the impact of each modification in improving the quality of search space exploration. The proposed framework has achieved comparable performance with the state-of-the-art approaches, with a comparatively simpler approach and minimum human intervention. The proposed framework can be used to develop completely automated systems for designing architectures for various data-based classification applications.

INDEX TERMS Neural architecture Search, hyperparameter optimization, AutoML, crow search algorithm, metaheuristic, image classification, deep learning

I. INTRODUCTION

Deep learning models have solved various practical problems in a wide range of areas, such as image recognition, speech recognition, reinforcement learning and many more. However, they are hard to design, mainly because of underlying complexities and their inherent dependency on a bunch of hyperparameters. Currently, neural networks are hand-engineered and then tested rigorously with several values for the hyperparameters to get the best performance on a given task. In the early days of machine learning, data features were hand-engineered by experts to identify unique patterns and structures which were then used to train models. With the inception of neural networks, it became possible to let the algorithms decide which features are important for a specific task. Neural networks extract features on different abstraction levels depending upon the network depth. So, it is not wrong to say that neural networks have paved a path

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2020.3031599, IEEE Access

towards automating the machine learning to an extent. Designing a neural network takes expert knowledge such as high-level expertise in mathematics, statistics, and algorithm design. Engineers are required to design an accurate and computationally low-cost architecture for each classification problem. After the design of Neural Network is finalized, engineers iteratively experiment with cumbersome hyperparameter values to tune the model for best possible performance. This practice of finding the optimal hyperparameters for a specific model can be automated with the help of some search algorithms, thus called Hyperparameter Optimization (HPO). Previously, most widely used strategy for HPO was a combination of grid search and manual search [1]. Later Bergstra et al. [2] proposed random search for HPO, which proved to be efficient and more plausible as compared to grid and manual search. However, these approaches are extremely expensive as they have small chance to just stumble upon a set of hyperparameters that will work for a given problem. Also, it will spend a lot of time just wastefully training on under-performing choices. Another technique explored for HPO is Bayesian optimization [3], which outperformed both manual and random search. It builds a probabilistic model of the function mapping from hyperparameter values to the objective evaluated on a held-out validation set. More recently, there are gradient-based approaches for HPO. Maclaurin et al. [4] proposed to compute exact gradients of cross-validation performance with respect to all hyperparameters by demonstrating the applicability of gradient-based HPO to high-dimensional problems. For example, simultaneously optimizing the parameter responsible for weight initialization for each layer, the $L2$ penalty for each parameter in logistic regression, the learning rate for each iteration and each layer in a neural network. Franceschi et al. [5] proposed a method for forward and reverse gradient based HPO. This method uses a similar technique as [4] for reverse mode, following a classical Lagrangian approach used to derive backpropagation algorithm [6]. Furthermore, they propose that the forward-mode procedure is suitable for real-time hyperparameter optimization. Recent studies [7] on gradient based HPO has shown robust performance and have outperformed previous Bayesian optimization techniques. However, hyperparameter optimization alone is not a complete solution for machine learning automation, as it still requires a human-engineered network architecture to begin with.

For a Convolutional Neural Network (CNN), a typical neural network for image classification, it may take a long time to iteratively design, train, test, validate and finalize the model before applying any HPO technique. So, there must be a method to automatically produce neural network architectures. This dates back to 1988 when Fernando et al. [8] proposed self-organizing neural networks (SONN) for the problem of model identification. SONN was a flexible structure capable of adjusting its structure depending upon input data. Neural Architecture Search (NAS) is a domain which specifically aims to solve this problem by employing a technique to generate architectures automatically. NAS methods mainly comprise of (i) search-based, (ii) reinforcement learning-based, or (iii) gradient-based methods to automate the design of Neural networks. Elsken et al. [9] has categorized NAS approaches based on three dimensions namely, (i) search space, (ii) search strategy and (iii) performance estimation strategy. The search space defines what kind of networks are discoverable and directly translates to the architecture's complexity level. Some of the recent state-of-the-art architectures include complex blocks having unique and modern layers. ResNet [10] for instance, consists of Residual block, which implements skip connections which have shown to mitigate gradient vanishing. Then, there are InceptionNet [11], SENet [12], etc. introducing further complex architectures. An architecture that can perform up to the par with such architectures needs to be adequately complex. Such an architecture can only be designed if the search space is complex enough. Usually, search spaces are designed to be as inclusive as possible which causes them to be hyper-dimensional. An efficient yet effective search strategy should be devised to traverse a hyper-dimensional search space. This strategy revolves around the age-old exploration-exploitation dilemma. Finally, there must be a performance estimation strategy to evaluate the discovered architectures. Usually, in the case of NAS, the evaluation strategy consists of some machine learning metrics (validation loss or validation accuracy etc.) as fitness function for search algorithm.

A. SEARCH SPACE

The search space dictates the kind of architectures that can be designed (generated) by the said NAS framework. A search space includes a finite set of networks that can be generated. A search space is defined such that a network $\mathrm{N} \in d$ dimensional search space.

Different kinds of networks can be categorized based on their underlying design-complexity. A rather simple architecture design is a sequential or chain-like architecture. In such networks, layers are connected in a sequential manner such that layer L_i receives input from layer L_{i-1} and sends output to layer L_{i+1} as shown in Figure 1(a). Then, there are some network architectures that are not as simple as chain-like architectures. Most of the modern state-of-the-art architectures have multiple paths as shown in the Figure 1(b). In order to incorporate such modern designs in NAS, the search space needs to be designed with consideration for modern design elements like skip connections. Modern NAS methods use search spaces capable of implementing modern design elements like skip connections, residual or identity blocks, etc. In order to construct a search space which includes such type of multi-path architectures, Genetic CNN [13] has proposed a binary encoding scheme which will be discussed in later section. A search space which includes

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2020.3031599, IEEE Access

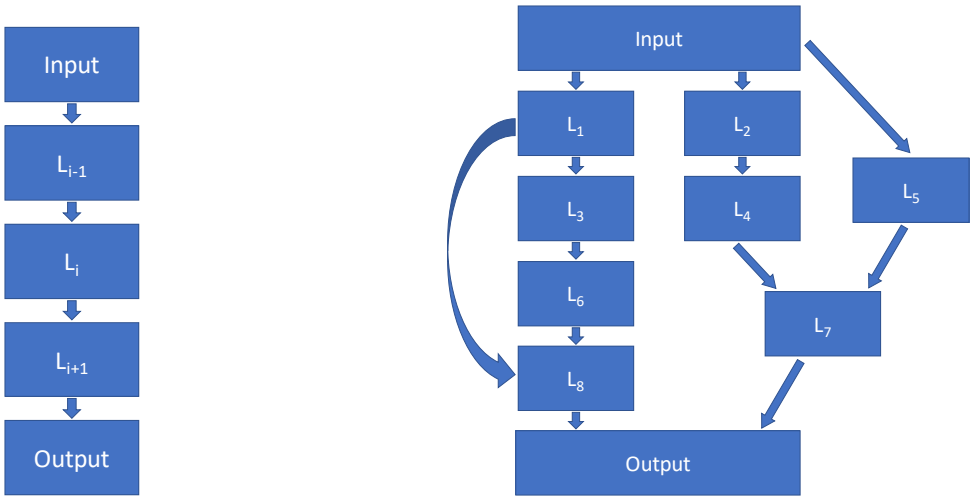

FIGURE 1. (a) **Sequential (chain-like) Architectures. (b) Modern architectures can have multiple paths.**

complex architectures needs to be very complex, as a result such a space will be very large to efficiently search for. One way to reduce search space is to encode some information about what kind of human-made architectures generally perform well. One such example is constructing a meta-architecture by using cells or motifs in a recursive manner [14]. Then the architecture for these cells is searched for. However, this introduces human bias which is a hindrance in automation of machine learning. Besides automation, this will simply divert the human effort from basic architecture design to meta-architecture design.

The intuition behind cell-based search space is that, almost all well-performing human-engineered neural architectures are usually constructed by repeating motifs or blocks. So, instead of searching for full architectures, Zoph et. al [14] proposed NASNet search space, which finds generic cells that can be repeated in series and should also be scalable further to larger datasets. Instead of searching for whole networks, they search for two kinds of cells, (i) normal cell - that preserves the spatial dimensions of input and a (ii) reduction cell - that reduces the spatial dimensionality. Finally, they manually stack these cells in a predefined manner. Reinforcement learning based search method is employed for finding such generic cells. These cells can be constructed of convolution layers, non-linearities, etc. Another approach could be to fix the architecture of motifs and look for meta-architecture. A recent work [15] proposed to find meta-architectures by searching for configurations of fixed architecture blocks such as VGG Block, Residual Block, Convolutional Block, etc. However, as this study is aimed to progress the automation of machine learning, methods which include minimum human intervention are explored. To enable the search algorithms, traverse the search space, the search space needs to be represented in a structured way using a sophisticated encoding scheme.

1) ENCODING SCHEMES

To implement any metaheuristic algorithm, there are mainly two pre-requisites, (i) a representation of the solution domain, (ii) a heuristic or objective function (cost). In this section, representation of the solution domain is discussed. Solution domain can be represented by employing an encoding scheme. Genetic CNN [13] uses binary encoding scheme, where a network structure is represented by a fixed-length binary string. This scheme can be applied to network structures which can be divided into stages e.g. Deep Residual Networks [10] and VGG [16]. Furthermore, in each stage i, there are several numbered nodes where each node corresponds to a convolutional layer. There are two default nodes in each stage i.e. input and output nodes. The input node receives data from previous stage, performs convolution and sends to all the nodes without a predecessor, and output node takes input from all the nodes without any successor and passes on to the next stage. Architecture of a single stage is shown in Figure 2. The intermediate nodes and their underlying connections are represented as a binary encoded string. Genetic CNN explores the search space of binary strings to form a suitable combination of connection between nodes. The connections are only allowed from a lower-numbered node to a higher-numbered node. Recently, Ahmad et al. [15], proposed an encoding scheme based on a search space consisting of fixed blocks most commonly used in modern CNNs e.g. residual block [10], Convolution block etc. They fixed the structure of individual blocks and formulated the search strategy to look for architectures by discovering different configurations or meta-architectures to arrange these blocks. This study follows the work of [13] and designs the search space by dividing network architectures in multiple stages.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2020.3031599, IEEE Access

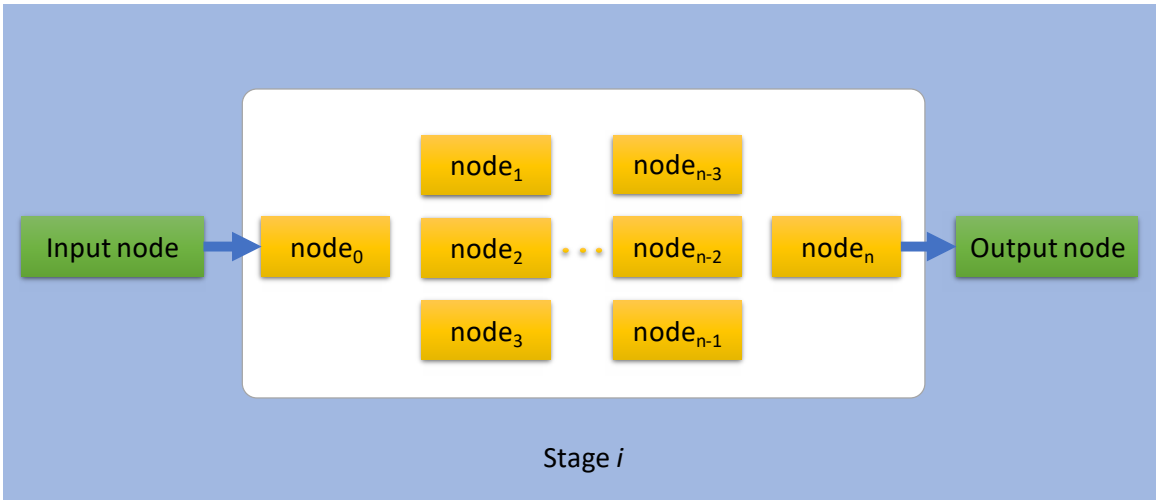

FIGURE 2. **A representation of a stage having $n+1$ number of nodes. Connections among these nodes can be searched by using Crow Search Algorithm or Genetic CNN.**

B. PERFORMANCE ESTIMATION STRATEGY

The performance estimation strategy evaluates the performance of a possible CNN from its design. Performance estimation strategy concerns about objective function to be optimized. In the case of NAS, performance can be estimated by using several machine learning (deep learning) metrics. Most commonly used metrics are training accuracy, validation accuracy, training loss and validation loss. This work is based on optimizing the search strategy using these typical machine learning metrics.

C. SEARCH STRATEGY

A search strategy focuses on maximizing the heuristic function. A search strategy is directly dictated by the search algorithm employed to find the solution. Some of the known search strategies come under the umbrella of evolutionary methods, reinforcement learning (RL) and gradient-based methods.

Evolutionary Algorithms based Neural Architecture Search

Evolutionary search algorithms follow a process inspired by the biological concept of evolution where they try to evolve candidate individuals over several generations using concepts like mutation and cross-over etc. These algorithms have gained attraction for their proven efficiency for solving optimization problems. Some of the evolutionary metaheuristics are discussed in this section. Genetic Algorithm (GA) [17], takes its inspiration from natural process of evolution using basic operations such as mutation and cross-over. With the help of these operations, good performing traits are passed over to the next generations, eventually improving the performance of overall population over a certain number of iterations. Particle Swarm Optimization (PSO) [18] is a swarm intelligence algorithm, where the particles (potential solutions) move in the search space and improve their position iteratively depending on their individual positions as well as swarm's overall position. PSO faces a problem when several objectives are conflicting with each other. Many Objective Particle Swarm Optimization (MOPSO) [19] tried to solve this problem by using a set of reference points dynamically determined depending upon the search process. Harmony Search (HS) [20] is based on the concept of harmony in music, and its main parameters are memory, pitch adjusting and randomization. Differential Evolution [21] is a global numerical optimization metaheuristic based on the mutation operation. Recently, there has been a trend of nature-inspired metaheuristics to solve optimization problems in various domains. Seouza et al. [22] proposed a modified version of crow search algorithm for feature selection where they reduce the continuous search space to discrete search space by restricting the movement of crows to only discrete lcoations. Nowdeh et al. [23] proposed to use matrix moth-flame algorithm for optimal reconfiguration of distribution networks and placement of solar and wind renewable sources. Jahannoosh et al. [24] proposed a new meta-heuristic algorithm for reliable and cos-effective designing of energy systems. Naderipour et al. [25] used grey wolf optimizer algorithm for optimal energy system design. Firefly and harmony search algorithms are also used for optimal power damping [26]. Genetic algorithm is proposed to optimize granular neural network parameters for pattern recognition [27] such as bird swarm optimization [28] for heart-rate classification, firefly algorithm [29] for optimization of modular granular neural networks and grey wolf optimizer [30] for optimizing granular neural networks for human recognition. Sanchez et al. [31] proposed to use particle swarm optimization with its fuzzy dynamic parameter adaptation to design modular granular neural network architectures. In the domain of NAS, Genetic CNN

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2020.3031599, IEEE Access

[13] uses an evolutionary search strategy on a binary encoded search space such that a set of candidate models are initialized in the form of a fixed size binary string, either randomly or by using Bernoulli distribution. Then these candidate individuals undergo genetic crossover by selecting a partner each and produce a child individual. Based on crossover probability and crossover rate child individual inherits a combination of genes from both parent candidates. These individuals may undergo mutation procedure based on mutation probability, which results in randomly flipping the bits in individual's genes in accordance with the mutation rate. Finally, these individuals become candidate for next generation and are evaluated for their fitness for the objective function on a pre-defined dataset. EENA [32] proposes to efficiently evolve populations by modifying crossover and mutation operations of genetic algorithm. Amoeba-Net was the first to outperform human design networks on ImageNet. They apply a modified evolutionary algorithm on NASNet [14] search space. LEMONADE [33] is based on Lamarckian evolution and applies network morphisms operations to produce offspring which help in reducing the training time of individual networks. Furthermore, LEMONADE formulates the NAS as a multi-objective problem which simultaneously minimizes the test error and model size.

Reinforcement Learning based Neural Architecture Search

Among Reinforcement learning based methods, NAS using reinforcement learning [14] and NASNet [34] are popular methods. In [14], it is proposed to use a RNN as controller which can design a string to specify architectures, however, this requires extensive computational power. In order to reduce the required computation, NASNet introduces a new search space which also allows transferability from one dataset to another. They achieve this by limiting the search space to a cell. They search for two cells, namely, normal cell and reduced cell. Normal cell maintains the dimensionality across input and output while reduction cell reduces the dimensionality. Furthermore, PNAS [35] utilizes the same search space and propose a method to progressively search for architectures in increasing order of complexity. Reinforcement learning based methods aim to reduce the search space by focusing on architecture search for small cells or units which can be further repeated based on a meta-architecture. The meta-architecture is designed manually depending upon the dataset. Cell-based architecture search methods help reduce the search space because they only search for cell architecture. This also allows to re-use the cells for different architectures. However, cell-based architecture search methods divert human effort from global architecture search to meta-architecture search and thus cannot substitute fully automated NAS. Reinforcement Learning based methods are computationally demanding even though they have achieved state-of-the-art performances.

Differential Evolution based Neural Architecture Search

Among Differential evolution methods, DARTS is quite notable for its less computational requirement and simplicity. Liu et al. [36] proposed a differentiable architecture search (DARTS) method which can achieve up-to-the-par performance with orders of magnitude less computational resources. This method is also simpler than RL based methods as it does not involve controller. GDAS [37] proposes to use a differentiable architecture sampler and applies it to directed acyclic graphs (DAGs).

In an effort to reduce the search space both RL based methods and DARTS search for cell architectures. However, this study emphasizes on reducing human effort by employing search methods which look for complete architecture, not only a block or cell which has to be arranged and placed in a pre-defined manner. In this way, the problem is formulated to improve the search strategy instead of reducing search space.

Among evolutionary algorithms, swarm optimization algorithms are not yet explored in the domain of NAS. A swarm intelligence algorithm named Crow Search Algorithm (CSA) is proposed by Askarzadeh et al. [38]. CSA is inspired by the methodology used by crows for seeking, hiding their own food, and stealing other's food. CSA replaces concepts like (i) mutation and (ii) crossover with (i) following the better performing candidate (ii) flying to random locations. CSA also incorporates a memory associated with individual crow which also sets apart from other search algorithms. In GA, in every generation new offspring are produced however, in CSA, crows are produced once at the time of initialization. Individuals update their memory as they explore the search space.

CSA resembles some of the previous algorithms e.g. GA, PSO and HS in many aspects. Some are briefly mentioned here. It creates an initial population of seekers to explore the search space. It is also not a greedy algorithm. Unlike GA, CSA includes memory unit to keep track of well-performing solutions found during exploration which is also the case with PSO and HS. In order to keep a balance between exploration and exploitation, CSA uses randomness and gradients [39]. CSA has only 2 decision parameters: *flight length* and *awareness probability* as compared to 4, 3 and 6 decision parameters required for PSO, HS and GA, respectively. This makes it much easier to optimize CSA as compared to other search algorithms.

Among previous evolutionary algorithms as mentioned before GA is well-explored in the NAS domain. Many studies have suggested the use of CSA because of its characteristics such as less parameter settings, easy implementation, and relatively strong development capacity in the search process [40]. It has already been applied to solve several engineering problems. In [38], authors solved 6 constrained engineering problems using CSA and it outperformed Genetic Algorithm and Particle Swarm Optimization. In [41], CSA was applied to power distribution

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2020.3031599, IEEE Access

network to find the optimal position to place the capacitors and their sizes, and experiments show that solutions found by CSA were accurate than other search methods. In computer vision domain CSA is also used by [42] to find the threshold for image segmentation. This helped in avoiding premature convergence and achieving automatic MRI segmentation.

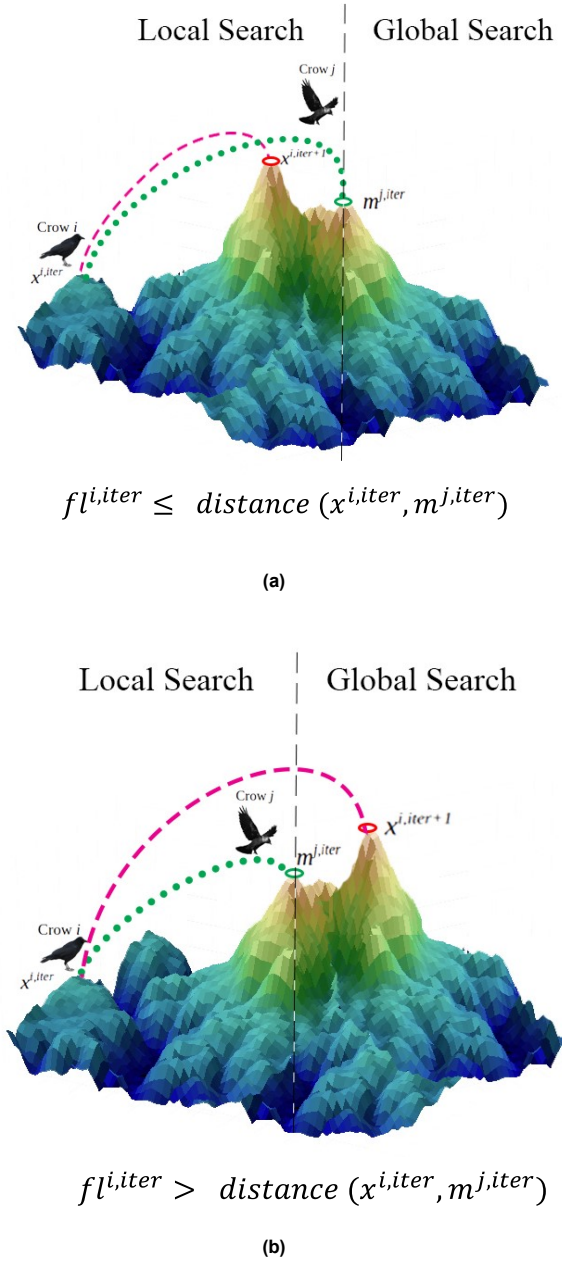

FIGURE 3. (a) If the value of flight length (fl) is selected smaller than the distance between current position $x^{i,iter}$ of crow i and hiding place $m^{j,iter}$ of target crow j i.e. best known location in its memory. In that case the next position of crow i is on the left side of the dash line between $x^{i,iter}$ and $m^{j,iter}$ resulting in Local Search. (b) If the value of fl is selected larger than the distance between current position $x^{i,iter}$ of crow i and memory location $m^{j,iter}$ of target crow j, the next position of crow i is on the right side of the dash line which results in Global Search.

Recently, CSA is also applied for finding input weight of Extreme Learning Machine (ELM) and finding the threshold values for hidden layers [43]. Chiwen et al. [40] proposed and improved CSA based on neighborhood search of non-inferior solution set and applied on pressure vessel design problem and tension-compression spring. Xiaxio et al. [44] proposed an improved CSA based on spiral search mechanism and applied to engineering optimization problems. However, Crow Search Algorithm (CSA) was not explored in NAS domain until recently. In [45], Abdullah et. al proposed Crow-search algorithm for hyperparameter optimization for image classification on MNIST and CIFAR-10 datasets. According to results, CSA outperformed GA with slightly improved results and in a smaller number of total trainings.

This work aims to develop a neural network search framework that is able to find complex architectures without needing any meta-architecture. For this purpose, Crow Search Algorithm (CSA) [38] is implemented on Binary encoded search space proposed by Genetic CNN [13]. The adaptation to binary search space to CSA has various constraints, assumptions, and modifications. Therefore, this new variation of CSA is named Binary CSA to distinguish it from original implementation of CSA. This study suggests applying Binary CSA on top of architecture search paradigm for complete architecture design search. This paper will compare the performance of two nature-inspired algorithms, Firstly, Genetic Algorithm (GA), a well-renowned algorithm based on evolution mechanics where every generation tries to improve individuals. Secondly, Crow Search Algorithm, which is based on lifestyle of crows, where they try to find food by following other crows and memorize their location of finding the food, eventually converging to the best possible location. Furthermore, some enhancements in Binary CSA are introduced for better convergence rate, which are discussed in later sections. In the next subsection crow search algorithm is discussed from the viewpoint of neural architecture search.

D. CROW SEARCH ALGORITHM

Crow Search Algorithm (CSA) is a population-based swarm intelligence algorithm, inspired by intelligent behavior of crows for hiding their food and following other birds to steal their food [38]. Crows watch other birds, observe where the other birds hide their food, and steal it once the owner leaves. The principles of CSA are listed as follows:

- Crows live in the form of flock (group).
- Crows hide their food at a good place.
- Crows memorize their hiding places.
- Crows follow each other to do the thievery.
- Crows use their experience to protect their catch.
- Crows memorize the hiding places other crows.

As crows are thieves themselves, they know well the behavior of a thief and act accordingly to avoid being the victim. When a crow follows another crow there is a

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2020.3031599, IEEE Access

probability of target crow being aware of the fact that it is being followed. This is addressed in algorithm with a parameter named awareness probability AP. If the target crow knows it is being followed then it changes its path to random location instead of going to its hiding place, i.e. the best location from its memory hence, introducing randomness which in turn enhances the exploration of algorithm. If the crow being followed does not know that it is being followed, the crow finally lands to its hidden food location or in case of our algorithm, location of the best solution achieved so far, from its memory. As a result, the other crow will follow it and will land to a nearby location (not exact location). This aids to the exploitation capacity of the search algorithm. In the first scenario, where the crow was not familiar that it is being followed. The landing position of the follower crow depends upon a parameter known as flight length fl. Depending on the fl, follower crow can land before the followed crow's location or farthest from location as depicted in figure 3. If fl is shorter than the distance between current location of the thief crow and the destination of target crow, the thief crow lands before reaching the target crow's food hiding location, hence executing local search as shown in figure 3(a). Whereas, if fl is longer and crow lands farther away, hence resulting in global search as shown in figure 3(b). The local search and global search, both help exploring the solution space by exploiting the experience of the target crow. However, the randomness introduced by awareness probability, leads to the exploration without regarding the experience of target crow. Both of these two parameters provide a good balance between exploration and exploitation.

Formally, Crow Search Algorithm can be described by assuming that there is a d-dimensional environment having N number of crows. The position of i^{th} crow at iteration iter is defined by a vector $x^{i,iter}$ as shown in Eq. (1).

$$x^{i,iter}(i = 1, 2, \dots, N; iter = 1, 2, \dots, iter_{max}) \quad (1)$$

where,

$$X^{i,iter} = \left[x_1^{i,iter}, x_2^{i,iter}, \dots, x_d^{i,iter}\right]$$

Here N is the total number of crows in the flock and, $iter_{max}$ is the maximum number of iterations. The d-dimensional space $X^{i,iter}$ includes all possible locations that can be explored by crow i in iteration $iter$. Crows traverse this d-dimensional $X^{i,iter}$ space by following other crows to find out their hiding location, hence reaching to the best possible solution over multiple iterations. Each crow memorizes only the best location they found during the search of hidden food. The hiding location in the memory of crow i at iteration $iter$ is denoted as $m^{i,iter}$. Now assume that crow j visits its hiding location from its memory $m^{j,iter}$ and crow i decides to steal from crow j, it will try to follow crow j at iteration $iter$. Now, based on the awareness probability of crow j two cases may arise:

Case 1: If crow j is unaware of the fact that it is being followed by crow i, it will keep going towards its hiding location $m^{j,iter}$ (hiding place) and crow i will reach a new location $x^{i,iter+1}$ for next iteration as per Eq. (3).

$$fl^{i,iter} = r^{i,iter} \times fl_{max} \quad (2)$$

$$x^{i,iter+1} = x^{i,iter} + fl^{i,iter} \times (m^{j,iter} - x^{i,iter}) \quad (3)$$

Where, fl_{max} is maximum flight length that a crow can fly. This is a parameter that needs to be assigned a value at the initialization of the search. While $r^{i,iter}$ is a random number which can have a value between 0 and 1. This random number dictates the flight length $fl^{i,iter}$ of crow i at iteration $iter$ as shown in Eq. (2). For instance, if fl_{max} is set to 100, depending on value of $r^{i,iter}$, the value of $fl^{i,iter}$ may be anywhere between 0 to 100. This way, each crow is assigned a different flight length $fl^{i,iter}$ in each iteration. Based on the value of flight length $fl^{i,iter}$ the crow i will reach a location nearby the hiding place of crow j. If $fl^{i,iter}$ has higher value than $diff(m^{j,iter}, x^{i,iter})$ then crow i will move past the hiding place of crow j as shown in figure 3 (a), hence conducting global search. If $fl^{i,iter}$ is smaller than the distance between crow i and crow j then crow i will conduct local search as shown in figure 3 (b).

Case 2: If crow j is aware of the fact that it is being followed by crow i, it will divert its path and go to a random location in space d. In effect, crow i will also be led to a random location, as a result it will explore a new location that may be very far from current area of search, hence increasing exploration. Both cases are expressed in the Eq. (4).

$$x^{i,iter+1} = \begin{cases} x^{i,iter} + fl^{i,iter} . (m^{j,iter} - x^{i,iter}), & ap^{j,iter} \geq AP \\ a\ random\ postion, & otherwise \end{cases} \quad (4)$$

TABLE 1. Pseudo code of original crow search algorithm.

(1). Input: the refence Dataset D, number of iterations T, the number of crows in the flock N, the awareness probability AP, maximum flight length fl_{max}
(2). Initialization: Generate a flock of N crows with randomly assigned locations with memory $mem = current_{location}$
(3). Evaluation: Evaluate all crows for recognition accuracy of the corresponding networks
(4). *while* $iter < iter_{max}$
(5). *for* $i = 1{:}N$
(6). $crow_i = flock(i)$
(7). $crow_j = random(flock)$
(8). $ap^{j,iter} = randrange(100)$
(9). *if* $ap^{j,iter} \geq AP$
(10). $crow_i.loc_{iter+1} = follow(crow_j.mem_{iter}, crow_i.loc_{iter})$
(11). *else*
(12). $crow_i.loc_{iter+1} = generate\_random\_location()$
(13). *for* $i = 1{:}N$
(14). $crow_i = flock(i)$
(15). $crow_i.fitness = eval(crow_i.loc_{iter+1})$
(16). *if* $crow_i.fitness > crow_i.best_{fitness}$
(17). $crow_i.best_{fitness} = crow_i.fitness$
(18). $crow_i.mem_{iter} = crow_i.loc_{iter+1}$
(19). Output: Flock with memory of best locations they explored

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2020.3031599, IEEE Access

Where, AP is the awareness probability defined at the time of initialization and $ap^{j,iter}$ is a random number within the range $(0, AP)$ that represent the awareness score of crow j at iteration $iter$. Whether or not $ap^{j,iter}$ is higher than the AP determines if crow j is aware of being followed or not. The exploration-exploitation trade-off can be tuned using these two parameters i.e. flight length and awareness probability. This following mechanism of crows is further explained in section II and its implementation for specific case of Neural Architecture Search. The pseudo code for original crow search algorithm is shown in table 1.

1) LIMITATIONS OF CROW SEARCH ALGORITHM

This section describes the limitations of employing original crow search algorithm on neural architecture search problem. First of all, previously CSA has been applied to engineering optimization problems where the goal is to find some optimal values for specific parameters. While in a previous work [45], CSA was used to search for four hyperparameters i.e. number of layers, layer width, optimizer and activation function. In such cases various distance formulae can be applied trivially to natural numbered values. However, CSA was not designed to find solutions in complex search spaces such as required by Neural Architecture Search (NAS). Because distance between two solutions as used by CSA cannot be computed directly in case of Neural Architecture Search. If the distance among neural architectures is to be computed, a scheme should be devised to interpret the differences among architectures as distances. As discussed in previous section, a neural architecture can be represented as a binary string using binary encoding scheme. This way the distance between architectures may be considered as a binary string comparison problem. Additionally, new solutions in search space may not be computed using simple arithmetic of CSA as shown in Eq (4).

In order to make this CSA mechanism work for the case of NAS, a new Binary Crow Search Algorithm (BCSA) is proposed to overcome the limitation of CSA. To measure the difference between two architectures being represented by binary strings, first a distance metric needs to be employed which is capable of comparing binary strings. There exist various binary distance metrics such as Levenshtein distance, Longest common subsequence (LCS), Hamming Distance, and Jaro distance. All these metrics have their own string operations and limitations. For instance, Levenshtein distance allows deletion, insertion and substitution, longest common subsequence (LCS distance) allows insertion and deletion, Jaro distance allows only transposition, Damerau-Levenshtein distance allows insertion, deletion, substitution, and the transposition operations, whereas hamming distance allows only substitution. Given the requirement of given task, two strings (architectures) need to be compared such that, compute the difference among them and substitute some bits in a string such that its distance can be reduced as compared to the other string. Levenshtein distance and its variants have the capability to fulfill the said requirement however, a more simplistic approach would be ideal. Therefore, hamming distance is employed to measure distance between binary representations of neural networks which is explained in detail in section II.

Furthermore, as explained in the section I.D. originally CSA uses random selection method for target solution. However, this introduces too much randomness which makes it harder to converge to optimal solutions even over multiple iterations. In BCSA, a selection method based on tournament-selection is proposed which helps in faster convergence to optimal solution. Furthermore, in CSA the maximum range of flight length is provided as algorithm parameter but in case of a binary string the maximum possible changes are equal to the total length of binary string. So, the maximum flight length cannot exceed the length of binary string. Additionally, a constant range of flight length is not an optimal choice, because if a crow is already too close to a target, making a random choice for flight length may lead astray from the possibly optimal solution. Therefore, it is crucial that the choice of flight length is made within an optimal range. Finally, the fixed flight length parameter is replaced by dynamic flight distribution which not only ensures that the flight range remains in optimal range but also eliminates one tunable parameter. A summary of our main contributions to solve all these problems is as follows:

(1). Crow Search Algorithm [38] is proposed to discover complex and novel CNN architectures for the first time.
(2). Binary Crow Search Algorithm is proposed to solve NAS problem in Binary Encoded Search Space.
(3). Target selection method is improved by introducing Tournament Select in baseline implementation of Binary CSA.
(4). Flight Length selection range fl_{max} is computed automatically, hence leaving only one tunable parameter named awareness probability AP, which makes it suitable for automation of neural architecture search problem.
(5). Based on distance of a crow i in iteration $iter$, from target crow j, a scaled range of the Flight Length $fl_{max}^{i,iter}$ is introduced intermediately to improve the convergence rate.
(6). Finally, it is demonstrated that Binary Crow Search algorithm outperforms previous Neural Architecture Search strategies by achieving comparable performance in significantly smaller number of trainings.

All these modifications resulted in a novel algorithm which has all the good qualities of CSA and is compatible with complex search spaces suitable for NAS. The paper presents this new algorithm as Binary Crow Search Algorithm.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2020.3031599, IEEE Access

II. PROPOSED APPROACH

This section presents the Crow Search Algorithm for searching state-of-the-art neural architectures. Initially, experiments are performed on popular datasets such as MNIST [46] and CIFAR10 [47] datasets etc. After initial evaluation on MNIST and CIFAR10 datasets, results are provided on large-scale datasets such as CIFAR100 and Tiny-ImageNet. The authors followed the work of Genetic CNN [13] for network representation such that a binary string is used to represent an architecture as mentioned in section I. Binary Network Representation and basic operations used by Binary Crow Search Algorithm are explained in this section. Technical details, limitations of methods and some examples are also provided in this section.

A. BINARY NETWORK REPRESENTATION

In this work, binary network representation scheme as proposed by [13] is used to represent our search space. The string length depends upon the number of stages S and number of nodes K_n ($n \in \{1,2,\dots,S\}$) in each stage. The authors experimented with two settings, $S = 2$, and $S = 3$, having $(K_1, K_2) = (3, 5)$ and $(K_1, K_2, K_3) = (3, 4, 5)$ nodes respectively. The network shown in figure 4 can be represented by a string consisting of binary numbers such as "$a_{21}a_{31}a_{32}b_{21}b_{31}b_{32}b_{41}b_{42}b_{43}b_{51}b_{52}b_{53}b_{54}$" which can be divided into parts for the sake of clarity. The string is divided into two parts as per the number of stages i.e. "$a_{21}a_{31}a_{32}$" and "$b_{21}b_{31}b_{32}b_{41}b_{42}b_{43}b_{51}b_{52}b_{53}b_{54}$" representing encoding of stage 1 and 2, respectively. In the first stage there are 3 nodes named $A1, A2, A3$ and 5 nodes i.e.

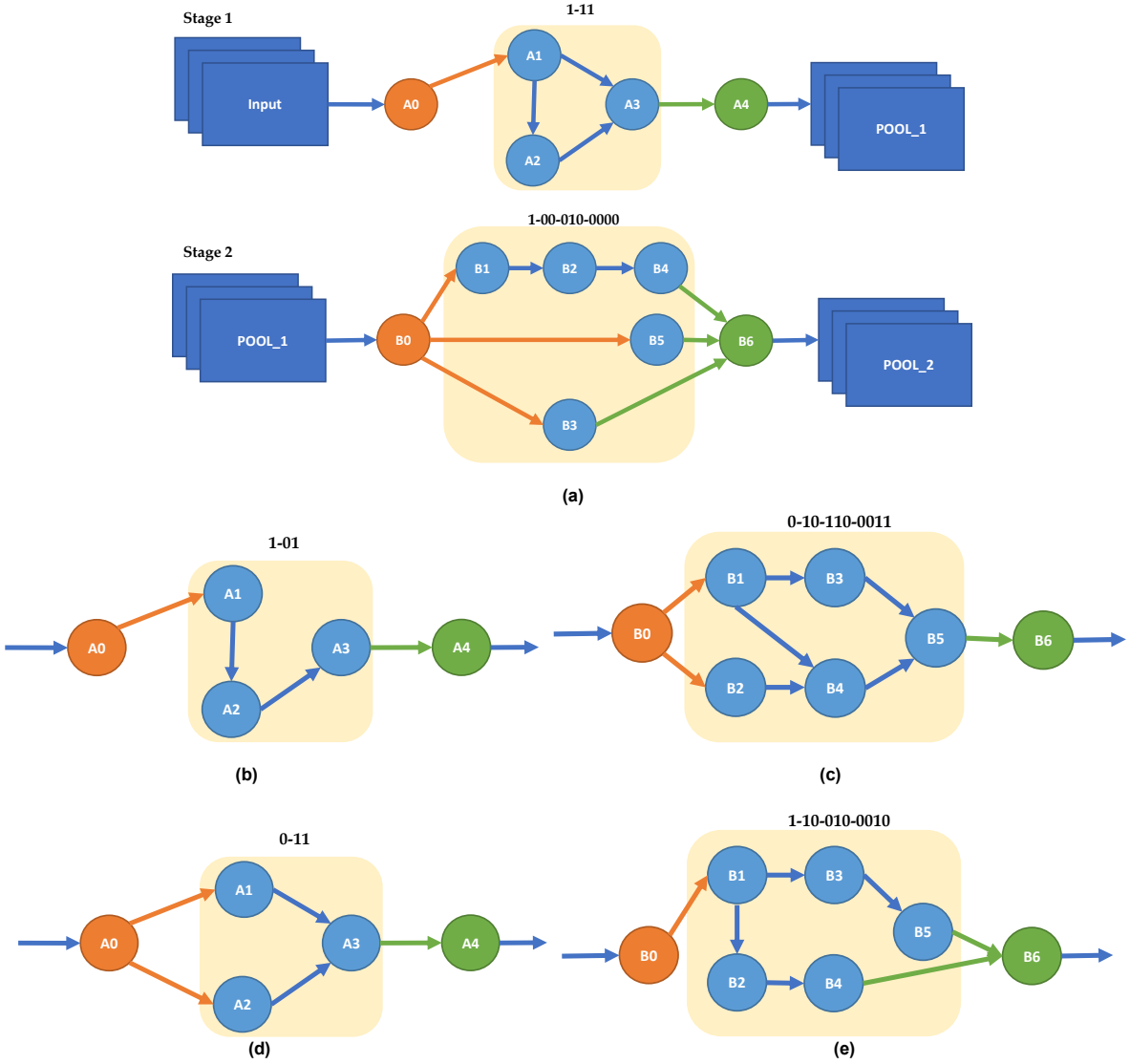

FIGURE 4. (a) A schematic diagram representing two-stage ($S = 2$) network produced from the binary string "1111000100000". First Stage has 3 nodes, whereas second stage has 5 nodes. $(K_1, K_2) = (3, 5)$. (b), (c), (d), and (e) show some example configurations with their respective binary strings. (b) and (c) combine to form the binary string "1010101100011". (d) and (e) combine to form the binary string "0111100100010".

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2020.3031599, IEEE Access

$B1, B2, B3, B5$ in the second stage. Here "$a_{21}a_{31}a_{32}$"can be further split node wise i.e. "$a_{21} - a_{31}a_{32}$". The first bit "a_{21}" of this 3-bit string represents the connection of $A2$ with $A1$, second bit "a_{31}" represents the connection of $A3$ with $A1$ and third bit "a_{32}" represents the connection of $A3$ with $A2$. If a bit is "set", it represents a connection in respective nodes. The first stage of figure 4(a), gets the final string as "1-11". It is to be noted that the indexing of string starts from the second node of the respective stage i.e. $A2$ and $B2$ for stages 1 and 2, respectively. By having a closer look at stage 2 string, it can be seen that $B2$ is connected with $B1$ so "b_{21}" = "1" on first location. The node B3 which is not connected to $B1$ and $B2$ hence, $b_{31} = 0, b_{32} = 0$, respectively. The node $B4$ which gets input from $B2$ but is disconnected with $B1$ and $B3$ hence, $b_{41} = 0, b_{42} = 1, b_{43} = 0$. Finally, $B5$ is not connected to any of the nodes but directly with input node hence the string for node $B5$ comes out to be 0000. The resultant length of the binary string can be calculated by Eq. (5).

$$length\ of\ binary\ string = \sum_{n=1}^{S}\sum_{i=1}^{K_n}(i-1) \tag{5}$$

Where, S is the total number of stages and K_n is the number of nodes in the n^{th} stage. For $S = 2, (K_1, K_2) = (3, 5)$, the string length will be 13 and for $S = 3, (K_1, K_2, K_3) = (3, 4, 5)$, the string length will be 19. Alongside these nodes, there are two default nodes in each stage, i.e. input node and output node. These nodes are fixed by default such that, input node will perform convolution and feed forward to any nodes without predecessor. While output node will receive inputs from all the nodes without successor. As, seen in figure 4, it is possible that search algorithm may come up with different configurations commonly found in state-of-the-art architectures such as skip connections, multiple streams, merging of streams, etc. Depth of each stage may also vary depending upon connections. In this study experiments are conducted with two settings as mentioned above. Furthermore, the number of stages as well as number of nodes in each stage can be modified.

Using aforementioned settings, it is possible to implement many popular architectures such as VGGNet [16], ResNet [48] and DenseNet [49]. However, for fair comparison with Genetic CNN [13], only pooling and convolutional operations are used as nodes.

B. BINARY CROW SEARCH ALGORITHM

This section explains the basic operations performed by the binary crow search algorithm. Some of these operations are briefly explained in section I, as per original algorithm proposed by [38]. Here, the operations are explained for the specific case of Neural Architecture Search domain. Furthermore, some improvements are proposed to the original algorithm as shown in the table 2. These improvements are thoroughly explained in this section. A summary of the differences between CSA and Binary CSA is provided as follows:

TABLE 2. Comparison between Crow Search Algorithm and Binary Crow Search Algorithm

Operations and Parameters	Crow Search Algorithm	Binary Crow Search Algorithm
Distance Formula	Simple subtraction (Not possible for Binary Encoded Solutions)	Binary selection and substitution
Target Selection	Random	Tournament Select
Max Flight length fl_{max}	fl_{max} a parameter of algorithm that needs to be fine-tuned.	fl_{max} is set to be equal to total length of bits in a solution
Number of tuning parameters	2 i.e. Flight Length, Awareness Probability	1 i.e. Awareness Probability
Max Flight length in Iteration *iter*	Constant ($fl_{max}^{i,iter} = fl_{max}$)	$fl_{max}^{i,iter}$ is dynamically computed based on Distance from Target (Eq. (6)).
Flight Length for $crow_i^{iter}$	$fl^{i,iter} = random(fl_{max}^{i,iter})$	$fl^{i,iter} = random(fl_{max}^{i,iter})$

- Introducing tournament select method for faster convergence
- Bound fl_{max} by total length of bits in solution, hence having only algorithm parameter i.e. awareness probability AP
- Dynamic range of fl_{max} hence avoiding large flights when close to the target
- Translation of flight formula into binary selection and substitution operations.

These contributions are explained in detail as follows:

1) INITIALIZATION

Initially a flock of N crows is created where each crow is initialized with a given list of nodes $(K_1, K_2, K_3, .., K_s)$ per stage S. Each stage is then represented by a binary string as explained in previous section. In first iteration the binary string is generated randomly to represent a random location in search space. However, in order to compare the results with fellow algorithms the initial locations may be assigned from a pre-populated list.

2) INITIAL EVALUATION

All crows are then evaluated by decoding the binary network representation and creating the corresponding neural networks as shown in figure 4. Recognition score on a given dataset is used as primary evaluation criteria.

Memory of each crow represents the location of the best performing architecture in the search space. In first iteration since there is no prior performance data, the current location is considered the best location and assigned to the crow's memory. The recognition score is also stored as the best achieved performance.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2020.3031599, IEEE Access

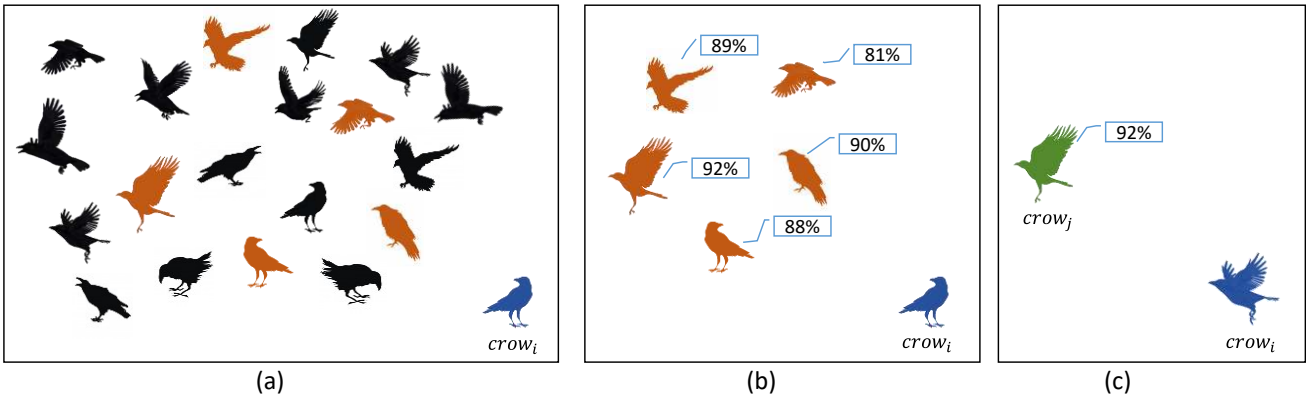

FIGURE 5. Tournament Select illustration for selecting target crow. (a) shows a flock of 20 crows in iteration $iter$. (b) a pool of 5 crows selected randomly from flock for $crow_i$ in iteration $iter$. (c) best performing crow is selected as target $crow_j$ from pool set.

3) TARGET SELECTION

In the original CSA, in each iteration, for each $crow_i$ in the flock, a target $crow_j$ is randomly selected to follow. This induces too much randomness and higher converging time. However, our aim is to gradually improve the performance of the crow in every iteration. Therefore, the selection process is modified to aid in achieving our goal. Instead of selecting the $crow_j$ randomly, it should be selected such that it leads to convergence. One approach could be selecting the best performing individual in every iteration. However, this can cause CSA to converge to sub-optimal solution. So, the best performing crow cannot be selected naively from the flock to be followed by each crow, because it will lead them all to converge in a local region in an iteration. Therefore, tournament select procedure was followed where a small subset of flock is selected randomly for each crow. Among these randomly selected crows, the individual with best performance is selected as target crow i.e. $crow_j$. Tournament select method is performed for every individual in the flock once per iteration as shown in figure 5.

4) FOLLOWING

Once a target $crow_j$ is selected for a given $crow_i$ in an iteration. There might be two cases as explained in section I. In one case the target may be aware that it is being followed, while in other case it may not be aware. To simulate this phenomenon, the algorithm is initialized with an awareness probability AP. At the time when a crow is following its target, a random number is generated in the range of 1 to 100. If that number is smaller than the awareness probability, the target ($crow_j$) is considered to be aware of being followed by the $crow_i$. Otherwise if that number is greater than or equal to the awareness probability the target ($crow_j$) is considered to be unaware of being followed by the $crow_i$.

In the first case the target crow tries to mislead the following crow by going to a random location in search space. Therefore, the final location of the crow i that is following the target is also a random location in search space, which may be generated randomly just like it was done at the time of initialization.

In the second case the following operation is carried out as per Eq. (4). The distance on the current location of $crow_i$ and memory $m^{j,iter}$ of the target $crow_j$ is computed as hamming distance $ham_{iter}^{i,j}$ in binary strings as shown in figure 6(a). It measures the number of bitwise substitutions required to match both strings.

The set of different bits are represented as $diff_{iter}^{i,j}$. Each substitution in these different bits makes it one step closer to the target string. The total number of substitutions done in each flight is defined by the flight length $fl^{i,iter}$. As shown in Figure 6(b), 3 bits were substituted, hence resulting in new location $x^{i,iter+1}$, 3 steps closer to the target. If the flight length smaller than the hamming distance the $fl^{i,iter}$ number of bits are randomly selected from the different bits $diff_{iter}^{i,j}$ for substitution. This will result in local search as explained earlier in section 2, figure 3. However, if flight length is greater than the hamming distance, extra bits will be selected randomly from whole binary string in addition to the different bits. This will cause excessive substitution and may result in final location $x^{i,iter+1}$ to be even farther than the target's destination $m^{j,iter}$ as shown in figure 6 (c).

Originally crow search algorithm is initialized with the maximum allowed flight length fl_{max}. However, in case of binary network representation the maximum flight length fl_{max} can only mean maximum number of changes possible, that is equal to the total length of binary string. Also, in the original crow search algorithm, the flight length $fl^{i,iter}$ of a crow i in iteration $iter$ is selected within the range of 1 to fl_{max}. However, there is a huge probability of the flight length $fl^{i,iter}$ to be very big even when the target is very close. The red line in figure 7 shows the flight length of original CSA. Using the fixed range of flight length $(1: fl_{max})$ will make the agent to go far from solution as soon as it comes near to convergence. Therefore, a method is proposed to scale the maximum flight length fl_{max} based on the hamming distance between a crow and its target in

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2020.3031599, IEEE Access

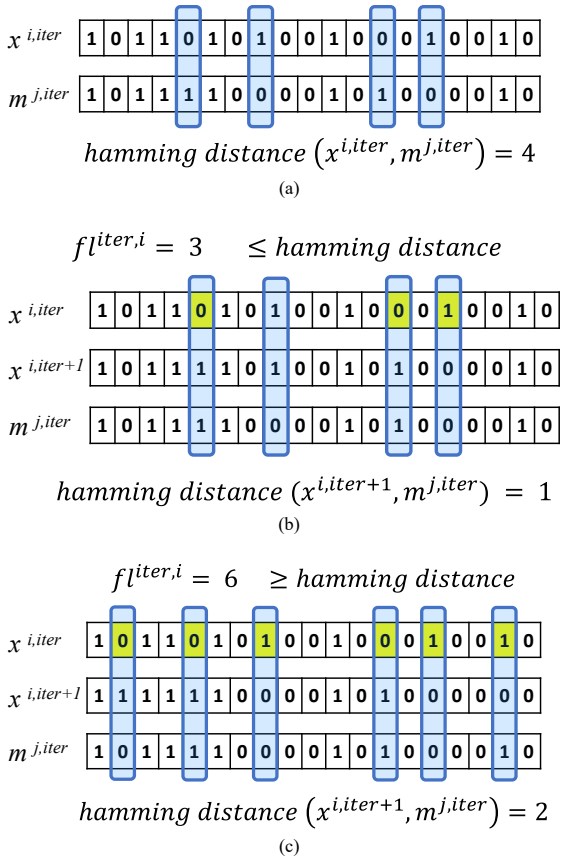

FIGURE 6. **(a) Flight distance is calculated as hamming distance between binary strings. (b) If flight length $fl^{i,iter}$ is smaller than the hamming distance, the final location is not too far from origin thus resulting in local search. (c) If flight length $fl^{i,iter}$ is bigger than hamming distance than final location maybe even farther than the target hence, resulting in global search.**

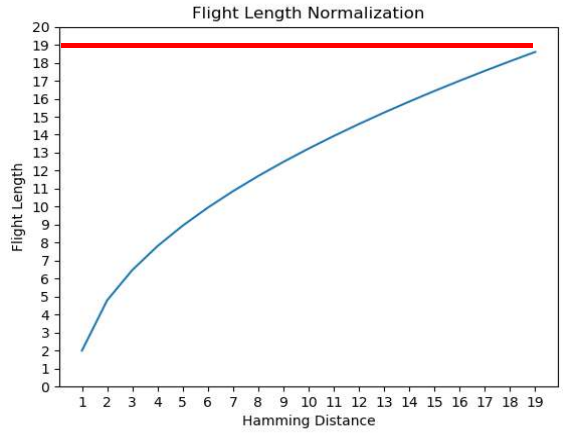

FIGURE 7. **Scaling of maximum flight length fl_{max} into $fl_{max}^{i,iter}$ based on hamming distance using Eq. (6). Red line shows the maximum flight length fl_{max} based on original CSA while blue line shows the scaled distribution of $fl_{max}^{i,iter}$.**

TABLE 3. **Pseudo code of Binary crow search algorithm.**

(1). Input: the refence Dataset D, number of iterations T, the number of crows in the flock N, the awareness probability AP, $tournamentSize$
(2). Initialization: Generate a flock of N crows with randomly assigned locations with memory $mem = current\_location$
(3). Evaluation: Evaluate all crows for recognition accuracy of the corresponding networks
(4). $while\ iter < iter_{max}$
(5). $\quad for\ i = 1: N$
(6). $\quad\quad crow_i = flock(i)$
(7). $\quad\quad crow_j = tournamentSelect(rand(flock, tournamentSize))$
(8). $\quad\quad ap^{j,iter} = randrange(100)$
(9). $\quad\quad if\ ap^{j,iter} \geq AP$
(10). $\quad\quad\quad crow_i.loc_{iter+1} = follow(crow_j.mem_{iter}, crow_i.loc_{iter})$
(11). $\quad\quad else$
(12). $\quad\quad\quad crow_i.loc_{iter+1} = random\_location()$
(13). $\quad for\ i = 1: N$
(14). $\quad\quad crow_i = flock(i)$
(15). $\quad\quad crow_i.fitness = eval(crow_i.loc_{iter+1})$
(16). $\quad\quad if\ crow_i.fitness > crow_i.best_{fitness}$
(17). $\quad\quad\quad crow_i.best_{fitness} = crow_i.fitness$
(18). $\quad\quad\quad crow_i.mem_{iter} = crow_i.loc_{iter+1}$
(19). Output: Flock with memory of best locations they explored

TABLE 4. **Pseudo code of $follow(crow_j.mem_{iter}, crow_i.loc_{iter})$ method used by Binary crow search algorithm.**

(1). $diff_{iter}^{i,j} = compare(crow_j.mem_{iter}, crow_i.loc_{iter})$
(2). $ham_{iter}^{i,j} = len(diff_{iter}^{i,j})$
(3). $fl_{max} = len(crow_i.loc_{iter})$
(4). $fl_{max}^{i,iter} = \sqrt{fl_{max} \times ham_{iter}^{i,j} - k}$ — Eq. (6)
(5). $fl^{i,iter} = random(range(fl_{max}^{i,iter}))$
(6). $if\ fl^{i,iter} > ham_{iter}^{i,j}:$
(7). $\quad for\ count = 1: fl^{i,iter} - ham_{iter}^{i,j}$
(8). $\quad\quad extra\_mile = random(indexes(crow_i.loc_{iter}))$
(9). $\quad\quad diff_{iter}^{i,j}.append(extra\_mile)$
(10). $elif\ fl^{i,iter} < ham_{iter}^{i,j}:$
(11). $\quad for\ count = 1: ham_{iter}^{i,j} - fl^{i,iter}$
(12). $\quad\quad extra\_mile = random(diff_{iter}^{i,j})$
(13). $\quad\quad diff_{iter}^{i,j}.remove(extra\_mile)$
(14). $ham_{iter}^{i,j} = len(diff_{iter}^{i,j})$
(15). $assert(fl^{i,iter} == ham_{iter}^{i,j})$
(16). $crow_i.loc_{iter+1} = [\,]$
(17). $for\ index\ in\ indexes(crow_i.loc_{iter}):$
(18). $\quad if\ index\ in\ diff_{iter}^{i,j}:$
(19). $\quad\quad crow_i.loc_{iter+1}.append(crow_j.mem_{iter}[index])$
(20). $\quad else:$
(21). $\quad\quad crow_i.loc_{iter+1}.append(crow_i.loc_{iter}[index])$
(22). $return\ \ crow_i.loc_{iter+1}$

current iteration. Therefore, the maximum flight length $fl_{max}^{i,iter}$ allowed for crow i in iteration $iter$ can be computed as shown in Eq. (6).

$$fl_{max}^{i,iter} = \sqrt{fl_{max} \times Ham(x^{i,iter}, m^{j,iter}) - k} \quad (6)$$

5) EVALUATION

Similar to the initial evaluation, corresponding locations for all crows are decoded and used to build and compile the neural network models. These networks are trained on given dataset and their evaluation score is used as the fitness of the crow on current location.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2020.3031599, IEEE Access

6) MEMORY UPDATE

The recognition score achieved for each crow in current iteration is compared with their respective best performance achieved so far. If a crow's current performance is better than its prior best achieved performance, then its memory is assigned the current location of the crow. Best performance of the crow is also updated by its current recognition score.

7) ITERATE

Steps from 4 to 7 are repeated until the last iteration. After final iteration, a flock of crows is obtained which have explored the search space and memorized the location representing the top performing neural network architecture. The crow with highest achieved recognition scores has the final binary encoded solution in its memory.

Table 3 shows the pseudocode for Binary Crow Search Algorithm. The pseudo code for the $follow()$ method used by binary crow search algorithm is represented in table 4. Furthermore, flowchart of BCSA is provided in the appendix, figure 14.

III. EXPERIMENTS

Training is performed on a cluster of 10 computers (clients) with GTX 1080 Ti, such that the search algorithm runs on the server and clients are responsible for training and evaluation. Server passes the binary string representation to each client which is then decoded into a CNN architecture. After training and evaluation are done on the clients, the results are sent back to the server. Based on these evaluation results, server performs Binary CSA operations and computes new binary strings which are sent to the clients for next iteration.

Results are compared with Genetic CNN [13] and for fair comparison the same initial population is used for both methods. Furthermore, the authors experimented with two-stage $S = 2, (K_1, K_2) = (3, 5)$ and three-stage $S = 3, (K_1, K_2, K_3) = (3, 4, 5)$ network representations for popular image classification datasets including MNIST, CIFAR10, CIFAR100 and Tiny-ImageNet. MNIST is a well-known handwritten optical character recognition dataset containing 10 classes, each class representing one decimal number. There are total 60,000 training images and 10,000 test images. CIFAR10 and CIFAR100 are popular image classification datasets. CIFAR10 contains 10 object classes with 6000 images per class. Out of 60,000 total images, 50,000 are used for training while 10,000 are used as test images. CIFAR100 contains 100 classes of common objects with 600 images per class. Out of total 60,000 images, 50,000 are used as training and 10,000 are used as test images. Tiny-ImageNet contains 200 image classes, with a training split of 100,000 images, validation split of 10,000 images and test split of 10,000 images. For MNIST, CIFAR10 and CIFAR100, 10% of training images are used as validation split. Small-scale datasets are used to evaluate our algorithm as it will be very time-consuming to evaluate search algorithms on large datasets. The number of filters and kernel sizes are also fixed to match the scope of experiment in Genetic CNN [13]. For instance, for MNIST, the number of filters is fixed to 32 and 64 whereas, for CIFAR10 and CIFAR100, 32, 64 and 128 (for three stage architectures) are used. The kernel of size (3, 3) is used in all experiments. The dense units are also fixed as 512, 1024, 2048 and 4096 for MNIST, CIFAR10, CIFAR100 and Tiny-ImageNet experiments. However, these hyperparameters may also be encoded in the search space and then searched using Binary CSA as demonstrated in [37]. Furthermore, the ablation experiments are performed to study the impact of tournament select method over random selection and our proposed dynamic flight length distribution $fl_{max}^{i,iter}$ (Eq. 6) over static flight length fl_{max} as used in original CSA.

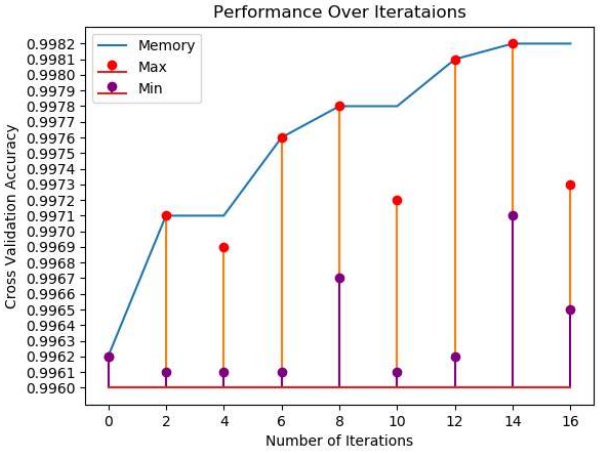

FIGURE 8. MNIST results using Binary Crow Search Algorithm using two-stage representation $S = 2, (K_1, K_2) = (3, 5)$. The orange and purple bars here represent the difference of baseline accuracy with maximum achieve accuracy in an iteration and minimum accuracy in an iteration, respectively. While the blue line shows the progress of best achieve performance over the experiment.

A. MNIST EXPERIMENTS

In the first phase of experiments, the proposed approach is validated on MNIST dataset. The two-stage $S = 2, (K_1, K_2) = (3, 5)$ binary representation is used with 3 and 5 nodes for stage 1 and 2, respectively. Results are shown in table 5 and figure 8.

TABLE 5. Recognition accuracy on the MNIST Dataset (test split). Settings used are $S = 2$, where $(K_1, K_2) = (3, 5)$

Iteration	Memory	Max	Min	Average	Standard Deviation
0	0.9962	0.9962	0.9962	0.9962	0.0003
2	0.9971	0.9971	0.9961	0.9970	0.0002
4	0.9971	0.9969	0.9961	0.9965	0.0003
6	0.9976	0.9976	0.9961	0.9966	0.0003
8	0.9978	0.9978	0.9967	0.9976	0.0003
10	0.9978	0.9972	0.9961	0.9968	0.0003
12	0.9981	0.9981	0.9962	0.9975	0.0002
14	0.9982	0.9982	0.9971	0.9976	0.0003
16	0.9982	0.9973	0.9965	0.9971	0.0003

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2020.3031599, IEEE Access

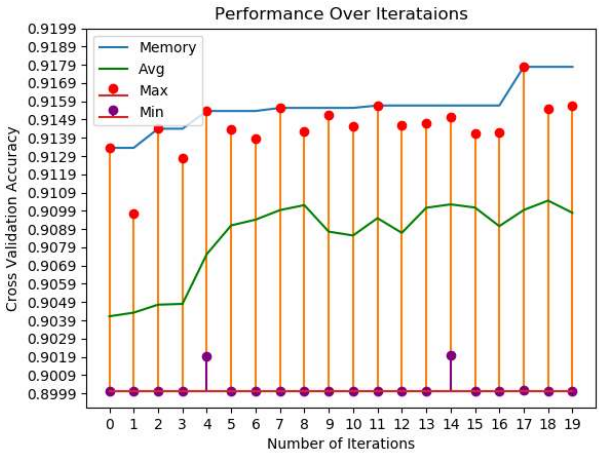

FIGURE 9. CIFAR10 results using Binary Crow Search Algorithm using two-stage representation $S = 2, (K_1, K_2) = (3, 5)$. The orange and purple bars here represent the difference of baseline accuracy with maximum achieve accuracy in an iteration and minimum accuracy in an iteration, respectively. The blue line shows the progress of best achieve performance over the experiment. While green line shows that progress of average performance of each iteration.

TABLE 6. Recognition accuracy on the CIFAR10 Dataset (test split). Settings used are $S = 2$, where $(K_1, K_2,) = (3, 5)$

Iteration	Memory	Max	Min	Average	Standard Deviation
0	0.9134	0.9134	0.9000	0.9041	0.0038
1	0.9134	0.9098	0.9000	0.9043	0.0037
2	0.9144	0.9144	0.9000	0.9048	0.0046
3	0.9144	0.9128	0.9000	0.9048	0.0044
4	0.9154	0.9154	0.9019	0.9075	0.0039
5	0.9154	0.9144	0.9000	0.9091	0.0048
6	0.9154	0.9138	0.9000	0.9094	0.0044
7	0.9155	0.9155	0.9000	0.9099	0.0049
8	0.9155	0.9142	0.9000	0.9102	0.004
9	0.9155	0.9152	0.9000	0.9088	0.005
10	0.9155	0.9145	0.9000	0.9086	0.0052
11	0.9157	0.9157	0.9000	0.9095	0.0052
12	0.9157	0.9146	0.9000	0.9087	0.005
13	0.9157	0.9147	0.9000	0.9101	0.0046
14	0.9157	0.9151	0.9020	0.9103	0.0037
15	0.9157	0.9141	0.9000	0.9101	0.0042
16	0.9157	0.9142	0.9000	0.9091	0.0046
17	0.9178	0.9178	0.9001	0.9099	0.004
18	0.9178	0.9155	0.9000	0.9105	0.0044
19	0.9178	0.9157	0.9000	0.9098	0.0042

B. CIFAR10 EXPERIMENTS

For CIFAR10, experiments were performed with two different settings with $S = 2$ *and* $S = 3$. For 2-stage experiment, the number of nodes per stage were identical to the MNIST experiments i.e. $(K_1, K_2) = (3, 5)$. Results for two-stage experiment are shown in figure 9 and table 6. Furthermore, experiments are conducted with three stage networks i.e. $S = 3$ and number of nodes as $(K_1, K_2, K_3) = (3, 4, 5)$ for stages 1, 2, and 3, respectively. The experiments with Tournament Select and impact of Flight Length are discussed further in subsection C and D.

C. TOURNAMENT SELECT

In order to select the target crow j, the original CSA algorithm randomly selects a crow from entire population, and it is assigned to a crow i.

However, as the experiments are conducted with different configurations, it is noted that this favors to the exploration and reduces the exploitation capability of CSA, hence convergence time increases. For target selection, tournament select method is used which is described in section II.B.3.and figure 5. The tournament select method is configured such that, a pool of 5 crows is randomly selected from the entire population and among them the best performing individual is selected as target crow i.e. crow j. Now, crow i, will follow crow j and perform all the Binary CSA operations. This intuitively introduces a balance between exploration and exploitation such that it keeps randomness along with prioritizing well-performing individuals. Figure 12 (c) shows the search results of Binary CSA performed with tournament select method. When compared with vanilla Binary CSA (figure 12(b)), it shows improvement in the form of early convergence as well as improved accuracy for final solution. This improvement can be credited to slight improvement in exploitation, due to tournament-based target selection.

D. DYNAMIC FLIGHT LENGTH DISTRIBUTION

As discussed earlier, the original implementation of CSA uses a fixed range of flight length fl_{max}, and each crow in each iteration choses a flight length $fl^{i,iter}$ within this range. While our proposed method introduces a dynamic distribution for range of flight length $fl_{max}^{i,iter}$ shown in Eq. (6). The 3-stage experiments are conducted with these two configurations. Empirical results have shown an improvement in accuracy and decrease in convergence time as shown in the figure 12 (d). Binary CSA along with dynamic flight length distribution have outperformed genetic algorithm, vanilla Binary CSA and Binary CSA with Tournament Select as shown in figure 12 and figure 13.

To further analyze the results, in-depth data is recorded about crow travel history during the complete run. The distance between the follower crow i and target crow j is measured at every iteration $iter$ and then computed the distance they actually travelled as shown in figure 10.

This analysis showed that if the range of flight length is fixed, the crow i may fly a very long distance even when it is already very near to the target crow hence, missing the optimal solution. In an ideal scenario, the chosen flight length for a crow i should not be too long when the distance between follower and target crow j is small. Otherwise it will

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2020.3031599, IEEE Access

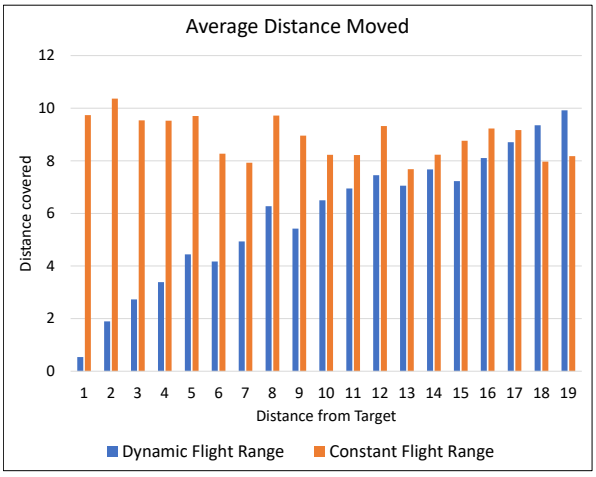

FIGURE 10. Comparison between constant range of flight length and dynamic range of flight length distribution (Eq. 6) of actual distance moved by $crow_i$ against the distance from target $crow_j$ on CIFAR10 using two-stage configuration $S = 3, (K_1, K_2, K_3) = (3, 4, 5)$.

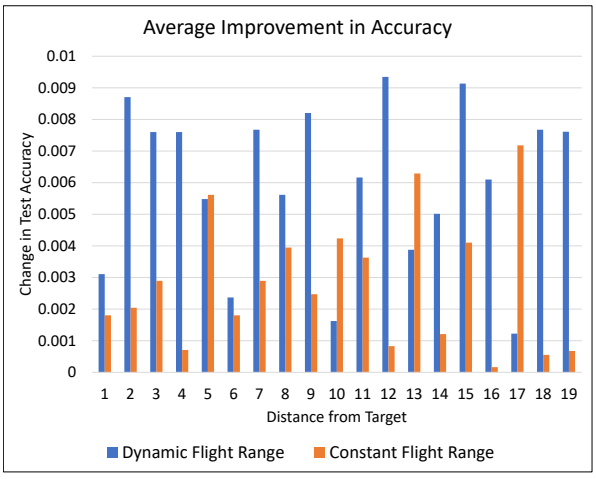

FIGURE 11. Impact of the choice of Flight Range on average improvement in test accuracy using constant flight length and dynamic flight length (Eq. 6).

just keep bouncing between locations instead of converging to an optimal solution.

The impact of both choices for range of flight length on the test accuracy is also recorded. For this purpose, the overall average improvement in accuracy was computed for all crows in all iterations at various distances from their targets. Figure 11 shows improvement in accuracy along the y-axis for each choice of flight range at given distances. It is evident from figure 10 and 11 that for a given distance from the target, a crow may choose a different flight length based on the selected range of flight lengths. Eventually, they may land on different solutions and their results could be quite different. It may be concluded that employing dynamic range of flight length as per Eq. (6) has enhanced the performance of Crow Search Algorithm.

IV. RESULTS AND DISCUSSION

The proposed approach is tested on MNIST, CIFAR and Tiny ImageNet datasets. On the MNIST dataset, our algorithm was able to find the best possible architecture in 15 iterations. In the first iteration, the maximum performing architecture achieved 99.62% accuracy. The maximum accuracy achieved on the 15th iteration was 99.82%. Although, experiments were conducted for more iterations, but CSA could not find any better architecture after 15th iteration as shown in figure 8. This is also evident in the Figure 8, that binary CSA is not greedy search like GA and PSO. When it finds a good architecture, it still explores other possible solutions that may have low performance but because of memory module, it remembers the best-found architectures and does not diverge while exploring. Table 5 shows that in every iteration, CSA is keeping track of best-found architectures in memory while it keeps exploring the search space. As it finds better performing architectures, memory is updated duly.

In the case of CIFAR10, the results are presented for two experiments. One with 2-stage architecture space i.e. $\boldsymbol{S = 2, (K_1, K_2) = (3,\ 5)}$, which is identical to the settings used for MNIST. Cross validation accuracy is shown in figure 9. Second with 3-stage architecture space i.e. $\boldsymbol{S = 3, (K_1, K_{2,}, K_3) = (3,\ 4,\ 5)}$. In the 2-stage experiments, the convergence is achieved in the 18th iteration as shown in figure 9. The architectures found in the first iteration had satisfactory performance. The best-performing architecture in the first iteration achieved 91.34% accuracy while the minimum was at 90% accuracy which stayed same throughout the experiment except 4th and 15th iteration.

These results show that better-performing architectures may not exist in 2-stage search space. To verify this conclusion, further experiments were conducted for 50 iterations, but CSA did not find better performing architectures (locations). 2-stage experiment results for CIFAR10 are summarized in table 6 and 3-stage experiment results in table 7. To solve this problem, the search space is increased by using three-stage architecture space which allowed us to generate further deeper architectures. Experiment results for 3-stage configuration are shown in figure 12. It can be seen that the accuracy improved significantly by increasing the depth of the search space. Figure 12 (a) shows the cross-validation accuracy of 93.75% achieved by applying genetic algorithm on the CIFAR10 dataset while keeping the same configuration as Binary Crow Search Algorithm. The binary CSA outperformed genetic algorithm significantly, in terms of higher accuracy and faster convergence. It is shown in Figure 12 (b), that Binary CSA surpassed the GA in 26th iteration and achieved final architecture with cross-validation accuracy of 94.88%. Extensive experiments were conducted with tournament select method and dynamic flight length distribution to verify that both modification work well with each other. Figure 12 (c) shows even faster convergence due to better selection of more promising targets to be followed. The Binary CSA with Tournament Select found an

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2020.3031599, IEEE Access

architecture that outperformed the final solution found by GA only in 13th iteration whereas, it found an architecture that outperformed the solution by vanilla Binary CSA only in 28th iteration. The final solution found by Binary CSA with Tournament Select achieved 96.25% accuracy. As discussed in the paper, our proposed distribution of flight length ensures that every flight made in the direction of target finds a solution in the vicinity of the target solution. This addition in the algorithm resulted in even faster convergence. The final version of Binary CSA that uses both tournament select and dynamic flight length outperformed GA only in 7th iteration, while it outperformed vanilla Binary CSA

Table 7. Comparison results on CIFAR-10

Algorithm	Test Error	Evaluation Time (GPU Days)	Model Size
Genetic CNN [13]	6.25	16.6	156 M
CNAS [50]	4.23	1	**2.95 M**
LEMONADE II [33]	3.50	56	3.98 M
Darts random [36]	3.49	-	3.16 M
Darts [36]	**2.83**	4	3.4 M
Binary Crow Search Algorithm (Ours)	5.12	6.41	8 M
Binary CSA with Tournament Select (Ours)	3.75	5.16	8.8 M
Binary CSA with Tournament Select and Dynamic Flight Length Distribution (Ours)	**3.48**	**3**	**8 M**

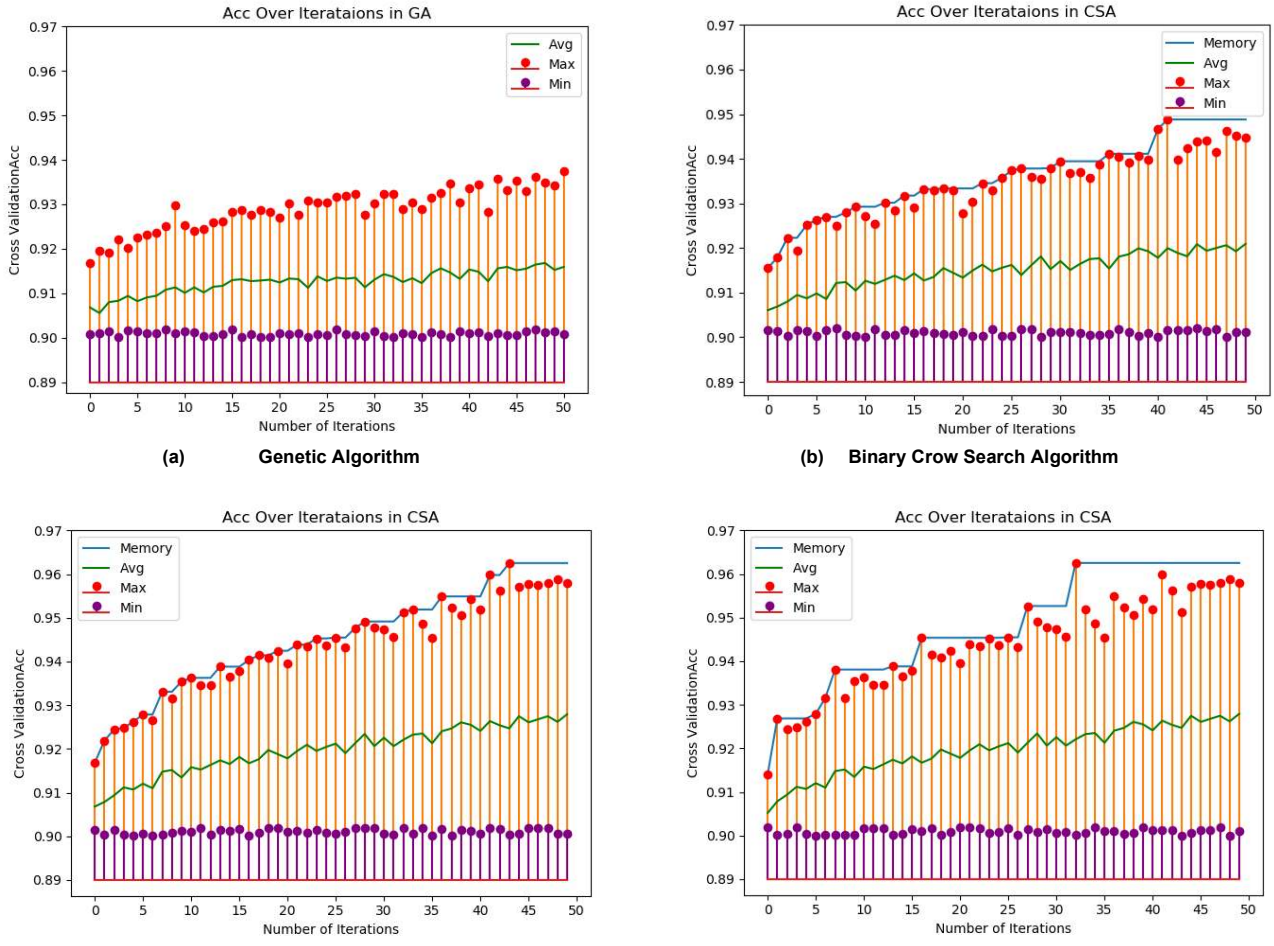

FIGURE 12. Comparison of cross-validation accuracy on CIFAR10 using (a) Genetic Algorithm (b) Binary CSA (c) Binary CSA with Tournament Select and (d) Binary CSA with Tournament Select and Dynamic Flight Distribution. All above used three-stage configuration $S = 3$, where the number of nodes per stage are $(K_1, K_2, K_3) = (3, 4, 5)$. The orange and purple bars here represent the difference of baseline accuracy with maximum achieve accuracy in an iteration and minimum accuracy in an iteration, respectively. The blue line shows the progress of best achieve performance over the experiment. While green line shows that progress of average performance of each iteration.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2020.3031599, IEEE Access

in 27th iteration and outperformed the Binary CSA with Tournament Select only in 33rd iteration by achieving 96.52% accuracy and sustained this performance until the final iteration. Comparison results of different versions of Binary CSA along with Genetic Algorithm are summarized in table 8 and figure 13, such that the iteration number at which each algorithm surpasses the highest achieved accuracy of rest of the algorithms is mentioned. Comparison results of some previous NAS methods on CIFAR10 are presented in the table 7. While table 8 shows the results based on the number of iterations each method took to outperform other variations. Binary CSA has shown better performance in terms of test error as compared to previous methods. However, CNAS [50] has achieved similar performance with less number of parameters. Binary CSA can achieve better results in the terms of a smaller number of parameters as well if applied to a more efficient search space. For now, search space is the bottleneck of our algorithm as it is not possible to find an architecture if it does not exist in the search space.

Search results on CIFAR-100 are presented in table 9, where Binary CSA has outperformed previous algorithms by a significant margin however, the model size of architecture searched by Binary CSA is approximately 3 times larger than the one found by CNAS which again is the limitation on the end of search space. Table 10 presents medium-scale transfer experiments. For medium-scale transfer experiments on Tiny ImageNet, the BCSA population is initialized with the 20 best performing individuals found in the last iteration of CIFAR100 search experiments. This helped in saving many GPU hours. Searching for architectures on small datasets and then instead of re-using them as previous methods have done, it is proposed to initiate the population using already searched top-performing architectures. The results are comparable to state-of-the-art however, the proposed approach does not involve any meta-architecture as required by other state-of-the-art methods. Finally, the test error rate results are presented on all the datasets as shown in table 11. In the future, Binary CSA can be integrated with a more sophisticated search space to generate more efficient architectures.

Such a system can be implemented to provide completely automated AI solutions for various applications such as automatic AI system training from data collection by users of mobile applications. The data may belong to a wide range of applications such as plant disease classification, accidental car damage attribution, used furniture and appliances condition evaluation etc.

Table 8. Comparison results of Genetic CNN along with different versions of Binary CSA based on iterations to outperform the preceding algorithm.

Algorithm (Iter – Max ACC)	GA (50 - 93.75 %)	Vanilla BCSA (41 - 94.88 %)	BCSA-TS (44 - 96.25 %)
Vanilla BCSA	26 – 93.78 %	41 – 94.88 %	-
BCSA with TS	13 – 93.88 %	28 – 94.92 %	44 – 96.25 %
BCSA with DFL	**07 – 93.81 %**	**27 – 95.26 %**	**33 – 96.52 %**

Table 9. Comparison results on CIFAR-100

Algorithm	Test Error	Evaluation Time (GPU Days)	Model Size
Genetic CNN [13] (transferred from CIFAR10)	25.12	-	156 M
CNAS [50]	22.24	1	3.67 M
Darts [36]	23.22	12	3.03 M
AmoebaNet-BC [51]	15.80	3150	34.9 M
Large-scale Evolution [52]	23.70	2600	40.4 M
NASNet-A [34]	16.03	1800	50.9 M
PNAS [35]	17.63	225	3.2 M
NAONet [53]	**14.75**	200	128 M
Neuro-Cell-based Evolution [54]	21.74	1	5.3 M
GDAS(FRC) [37]	18.13	**0.17**	**2.5 M**
EENA [32] (transferred from CIFAR-10)	17.71	-	8.49 M
Binary CSA with Tournament Select and Dynamic Flight Length Distribution (Ours)	**15.64**	**4.166**	**10 M**

Table 10. Comparison results on Tiny ImageNet

Algorithm	Test Error	Evaluation Time (GPU Days)	Model Size
CNAS [50]	36	3.5	3.67 M
Darts [36]	38.6	3.75	**3.03 M**
Binary CSA with Tournament Select and Dynamic Flight Length Distribution (Ours)+	**34.43**	3	13 M

+ top performing architectures on CIFAR100 were used to populate the first generation

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2020.3031599, IEEE Access

Table 11. Classification error rate for Binary Crow Search Algorithm on different datasets

Dataset	Architecture	Test Error (%)	Evaluation Time (GPU Days)	Model Size
MNIST*	'S_1': '100', 'S_2': '010000'	0.18	0.8	8 M
CIFAR-10†	'S_1': '010', 'S_2': '011010', 'S_3': '1000000010'	3.48	3	8 M
CIFAR-100†	'S_1': '101', 'S_2': '001100', 'S_3': '0111010001'	15.64	4.166	10 M
Tiny-ImageNet†+	'S_1': '111', 'S_2': '100111', 'S_3': '1010100110'	34.43	3	13 M

* 2-stage network representation, † 3-stage network representation, + top performing architectures on CIFAR100 were used to populate the first generation

The central system for each application may automatically construct a deep learning model that suites the data provided by users e.g. labeled pictures of leaves, car scratches and damaged furniture. Hence no technical knowledge of machine learning and artificial intelligence will be required to deploy each time a new application is required. The binary crow search algorithm will automatically find a deep learning model that best suites the data-based application.

V. RESULT ANALYSIS

Our proposed approach was able to find comparable architectures to other NAS methods. However, there is one thing to be noticed that the size of models found by our approach is larger than some of the NAS approaches. It is to be noted that the number of parameters (model size) or the type of model that can be produced is solely dependent upon the search space design. As far as the comparison of parameters is concerned, our approach uses the same search space as used by Genetic CNN. Models found by BCSA have significantly small number of parameters, i.e. the model found for MNIST data has only 8 million parameters whereas Genetic CNN achieves the best accuracy with 156 million parameters, similarly for CIFAR10, CIFAR100 and Tiny ImageNet, BCSA outperforms Genetic CNN in terms of accuracy, model size and faster convergence to optimal solution. As for the other NAS methods, such as Darts and CNAS, our method achieves slightly better accuracy, but the model size is larger. Figure 13 presents the amount of trainings required by one approach to outperform the other approaches. It is clear that, BCSA with flight length distribution algorithm along with tournament select significantly outperforms BCSA with original flight length and target selection methods. Moreover, to validate the performance of BCSA and its variants, a statistical analysis is provided in the next section.

1. EFFECT OF TOURNAMENT SELECT AND DYNAMIC FLIGHT LENGTH

In order to validate the results, several experiments were performed to conduct statistical analysis about the improvement in positive changes of the fitness for candidate solutions in an experiment compared to another. For this purpose, difference in improvement is computed for each individual/crow in each generation/iteration in different experiments as shown in table 12.

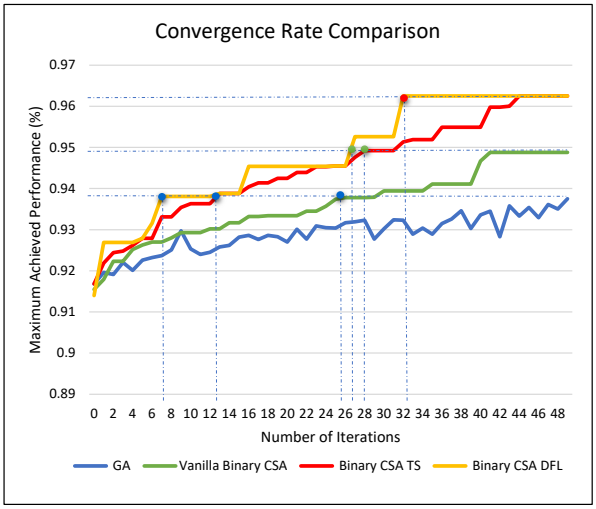

FIGURE 13. Comparison of convergence rate of different versions of Binary CSA along with GA. The colored dot represents the iteration number at which an algorithm outperformed other algorithms. For example, a blue dot on green line represents the iteration at which vanilla CSA outperformed the best accuracy of GA. It can be seen that Binary CSA DFL found GA equivalent architecture in 7 iterations which shows a significant improvement in convergence rate.

Then each experiment is compared to another and counted the solutions explored in the search space with better and worst fitness in a pairwise manner to roughly estimate the effects of choosing each variation over another according to the statistical results. The overall effect of choosing each algorithm and its operations is summarized in table 13. In order to verify the significance of each action, i.e. BCSA is significantly better than GA, Tournament Selection is better than using Random Selection and, Dynamic Flight Length improves the performance of BCSA, two tailed Wilcoxon Signed Rank tests were performed, a nonparametric statistical analysis on our experiment results. The null and alternative hypotheses for these tests are:

Test 1: GA vs BCSA Vanilla

$\boldsymbol{H_0}$: The new solutions computed using follow operation of BCSA do not show significant improvement compared to overall exploration done using crossover and mutation operations of GA.

$\boldsymbol{H_1}$: BCSA's follow operation demonstrates significant improvement for overall explored search space locations.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2020.3031599, IEEE Access

Table 12. Improvement in fitness of each individual in the final iteration ($iter = 20$) of each experiment.

Individual / Crow	GA	BCSA (Vanilla)	BCSA (TS)	BCSA (Dynamic FL)
0	0.0096	-0.0016	-0.003	0.0038
1	0.0072	0.0011	0.004	0.003
2	0.0044	-0.0005	0.0069	-0.0006
3	-0.0038	0.0012	-0.0008	0.001
4	-0.0106	-0.0007	0.003	0.0002
5	-0.0033	0.0009	0.0067	0.002
6	0.0121	0.0015	0.0075	0.001
7	0.0027	0.0014	0.0063	0.0018
8	-0.0009	0.0005	0.0119	0.0044
9	-0.0003	0.0006	0	0.0024
10	-0.0017	0.0008	-0.002	-0.001
11	0.0004	-0.0008	0.0025	0.0064
12	0.0135	0.0005	0.0066	0.0012
13	0.0041	0.0025	0.0079	0.0014
14	0.0042	0.0004	0.0032	0.0046
15	0.0027	0.0022	-0.0007	-0.0002
16	-0.0023	-0.0003	0.0027	-0.0006
17	-0.0095	0.0003	-0.0028	0.0002
18	-0.0066	-0.0007	0.0041	0.0038
19	-0.0049	-0.0012	0.0033	-0.0004

Table 13. Effects of choosing BCSA and its each variation against GA and each other in terms of total count of Better and Worst changes in the fitness of all individuals/crows in all the generations/iteration of the experiment.

Comparison Pair	Better	Worst
GA – BCSA (Vanilla)	236 (59.0 %)	164 (41.0 %)
BCSA Vanilla – BCSA TS	244 (61.0 %)	156 (39.0 %)
BCSA TS – BCSA Dynamic FL	246 (61.5 %)	154 (38.5 %)

Table 14. Wilcoxon Signed Rank test results show that in each comparison pair the latter introduces significant improvement in the performance. Here W+ is the total rank score of the positive changes, W- is the total rank score of the negative changes.

Tests	$W+$	$W-$	Z_{stat}	p-value	Result
1	46316	-33884	2.686	0.0491	Reject H_0
2	48165	-32035	3.485	0.0471	Reject H_0
3	48373	-31827	3.575	0.0487	Reject H_0

Test 2: Random Select vs Tournament Select

H_0: Target selection using Tournament Select in BCSA has no impact on the likeability of reaching a better location as compared to the Random Select.
H_1: Target selection using Tournament Select in BCSA significantly improves the likeability of reaching a better location as compared to using Random Select.

Test 3: Fixed Flight Length vs Dynamic Flight Length

H_0: Choosing a range of flight length dynamically based on the distance from target has no impact on the likeability of reaching a better location as compared to a fixed range of flight length.
H_1: Choosing a range of flight length dynamically based on the distance from target significantly improves the likeability of reaching a better location as compared to using a fixed range of flight length.

The results for these tests are shown in table 14. Z_{stat} is computed using large-sample approximation formula Eq. (7) for Wilcoxon Signed Ranked Test. The p-value is calculated using the normal approximation. The null hypothesis H_0 is rejected if $Z_{stat} > 1.96$ and p-value is less than $\alpha = 0.05$.

$$Z_{stat} = \frac{W - \frac{n'(n'+1)}{4}}{\sqrt{\frac{n'(n'+1)(2n'+1)}{24}}} \qquad (7)$$

VI. CONCLUSION

The study presented in this paper suggests the use of Binary Crow Search Algorithm for Neural Architecture Search. In this study, it is shown that Binary CSA based neural architecture search can achieve comparable accuracy in significantly smaller number of trainings. Furthermore, statistical analysis is performed using Wilcoxon signed rank test and the performance of BCSA with GA and variants of BCSA is compared. The results of Wilcoxon signed rank test prove that the improvement introduced by BCSA and its variants is significantly better than other alternatives as shown in table 14. Current NAS approaches do not use any domain knowledge for finding optimal solution. A search method which exploits domain knowledge will intuitively perform better than a blind method which does not use any domain knowledge. In future, this problem can be addressed by introducing guided search methods instead of blind search. The algorithm responsible for searching for neural networks should understand the impact of hyperparameters, layers, blocks, different types of activation functions, and architectural choices prior to training and evaluating them. Moreover, NAS can be formulated as a multi-objective optimization problem, which can minimize error and model size simultaneously. BCSA can be used with other more sophisticated search spaces such as NASNet or DAG based search spaces. Another multi-objective scheme can be devised which can search for cell architecture as well as, meta-architecture (currently manually designed in cell-based search spaces) which will help reduce the need for manual interventions.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2020.3031599, IEEE Access

APPENDIX

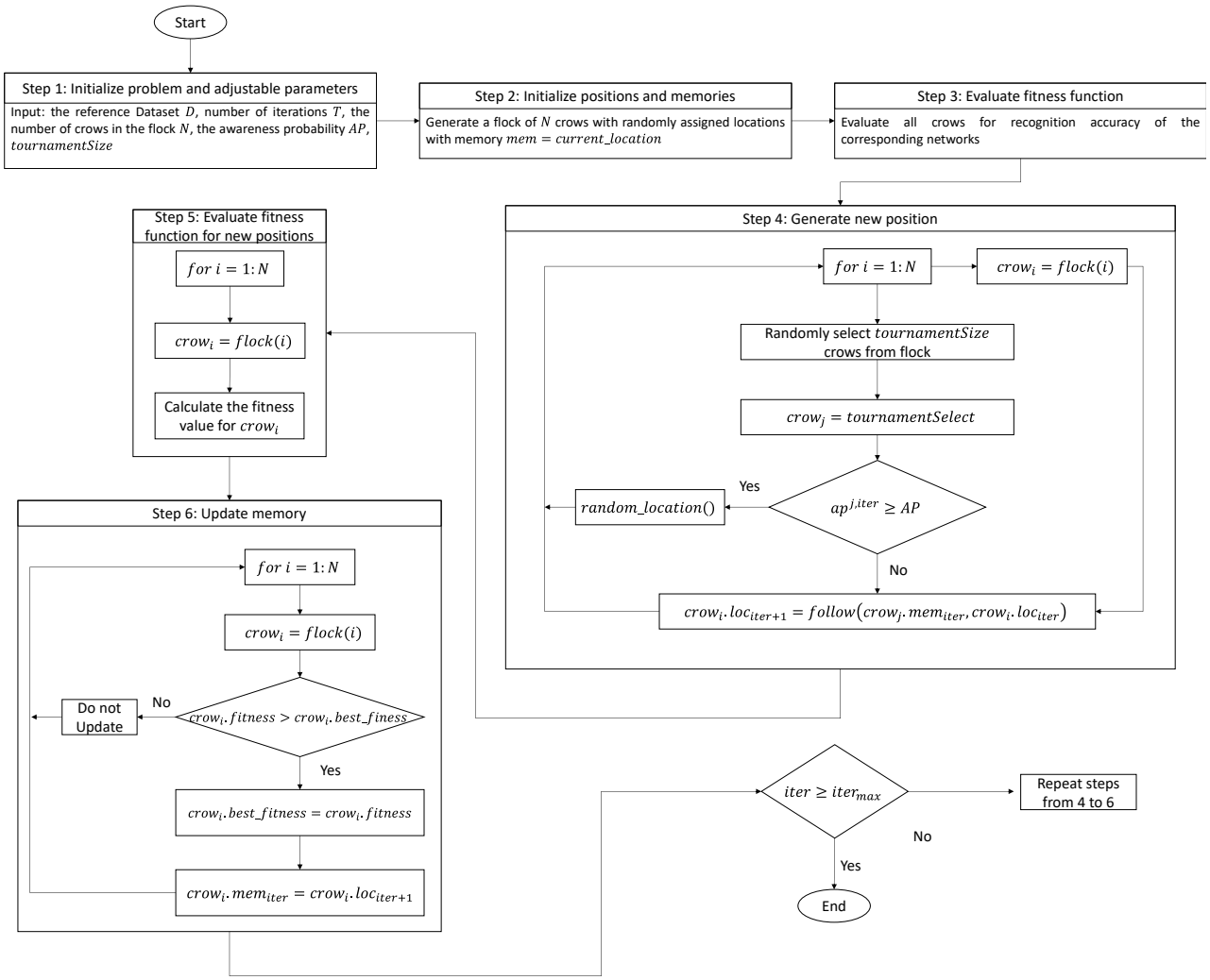

Figure 14. Flowchart of proposed Binary Crow Search Algorithm

REFERENCES

[1] H. Larochelle, A. Courville, and J. Bergstra, "An empirical evaluation of deep architectures on problems with many factors of variation Unsupervised Learning of Speech Representations View project Optical Neural Network View project," *dl.acm.org*, vol. 227, pp. 473–480, 2007.

[2] J. Bergstra and Y. Bengio, "Random search for hyper-parameter optimization," *J. Mach. Learn. Res.*, vol. 13, pp. 281–305, 2012.

[3] J. Snoek, H. Larochelle, and R. P. Adams, "Practical Bayesian Optimization of Machine Learning Algorithms," in *Proceedings of the 26th International Conference on Advances in Neural Information Processing Systems*, 2012, pp. 2960–2968.

[4] D. Maclaurin, D. Duvenaud, and R. P. Adams, "Gradient-based Hyperparameter Optimization through Reversible Learning," in *Proceedings of the 32nd International Conference on Machine Learning*, 2015, pp. 2113–2122.

[5] L. Franceschi, M. Donini, P. Frasconi, and M. Pontil, "Forward and reverse gradient-based hyperparameter optimization," *34th Int. Conf. Mach. Learn. ICML 2017*, vol. 3, pp. 1903–1913, 2017.

[6] Y. Lecun, "A Theoretical Framework for Back-Propagation," in *Proceedings of the 1988 Connectionist Models Summer School, CMU*, 1988, pp. 21–28.

[7] F. Pedregosa, "Hyperparameter optimization with approximate gradient," *33rd Int. Conf. Mach. Learn. ICML 2016*, vol. 2, pp. 1150–1159, Feb. 2016.

[8] M. FernandoTenorio and W.-T. Lee, "Self Organizing Neural Networks for the Identification Problem," in *Proceedings of Advances in Neural Information Processing Systems 1*, 1988, pp. 57–64.

[9] T. Elsken, J. H. Metzen, and F. Hutter, "Neural Architecture Search: A Survey," *J. Mach. Learn. Res.*, vol. 20, no. 55, pp. 1–21, Aug. 2019.

[10] K. He, X. Zhang, S. Ren, and J. Sun, "Deep Residual Learning for Image Recognition," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2016, pp. 770–778.

[11] C. Szegedy *et al.*, "Going deeper with convolutions," in *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2015, pp. 1–9.

[12] J. Hu, L. Shen, and G. Sun, "Squeeze-and-Excitation Networks," in *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 2017, pp. 7132–7141.

[13] L. Xie and A. Yuille, "Genetic CNN," in *Proceedings of the IEEE International Conference on Computer Vision*, 2017, pp. 1388–1397.

[14] B. Zoph and Q. V. Le, "Neural architecture search with reinforcement learning," in *5th International Conference on*

Learning Representations, ICLR 2017 - Conference Track Proceedings, 2017.

[15] M. Ahmad, M. Abdullah, and D. Han, "A Novel Encoding Scheme for Complex Neural Architecture Search," in *34th International Technical Conference on Circuits/Systems, Computers and Communications, ITC-CSCC*, 2019.

[16] K. Simonyan, "Very Deep Convolutional Networks for Large-Scale Image Recognition.," in *Proceedings of International Conference on Learning Representations. (ICLR)*, 2015, pp. 1–14.

[17] J. E. Beasley and P. C. Chu, "A genetic algorithm for the set covering problem," *Eur. J. Oper. Res.*, vol. 94, no. 2, pp. 392–404, Oct. 1996.

[18] Y. Zhang, S. Wang, and G. Ji, "A Comprehensive Survey on Particle Swarm Optimization Algorithm and Its Applications," 2015.

[19] E. M. N. Figueiredo, T. B. Ludermir, and C. J. A. Bastos-Filho, "Many Objective Particle Swarm Optimization," *Inf. Sci. (Ny).*, vol. 374, pp. 115–134, Dec. 2016.

[20] X. S. Yang, "Harmony search as a metaheuristic algorithm," *Studies in Computational Intelligence*, vol. 191. Springer, Berlin, Heidelberg, pp. 1–14, 2009.

[21] K. Price, R. Storn, and J. Lampinen, *Differential evolution: a practical approach to global optimization*. Springer Science & Business Media, 2006.

[22] R. C. T. De Souza, L. D. S. Coelho, C. A. De MacEdo, and J. Pierezan, "A V-Shaped Binary Crow Search Algorithm for Feature Selection," in *2018 IEEE Congress on Evolutionary Computation, CEC 2018 - Proceedings*, 2018.

[23] A. Jafar-Nowdeh *et al.*, "Meta-heuristic matrix moth–flame algorithm for optimal reconfiguration of distribution networks and placement of solar and wind renewable sources considering reliability," *Environ. Technol. Innov.*, vol. 20, p. 101118, Nov. 2020.

[24] M. Jahannoosh, S. A. Nowdeh, A. Naderipour, H. Kamyab, I. F. Davoudkhani, and J. J. Klemeš, "New hybrid meta-heuristic algorithm for reliable and cost-effective designing of photovoltaic/wind/fuel cell energy system considering load interruption probability," *J. Clean. Prod.*, vol. 278, p. 123406, Jan. 2021.

[25] A. Naderipour, Z. Abdul-Malek, M. Zahedi Vahid, Z. Mirzaei Seifabad, M. Hajivand, and S. Arabi-Nowdeh, "Optimal, Reliable and Cost-Effective Framework of Photovoltaic-Wind-Battery Energy System Design Considering Outage Concept Using Grey Wolf Optimizer Algorithm - Case Study for Iran," *IEEE Access*, vol. 7, pp. 182611–182623, 2019.

[26] A. Naderipour, Z. Abdul-Malek, V. K. Ramachandaramurthy, M. R. Miveh, M. J. H. Moghaddam, and J. M. Guerrero, "Optimal SSSC-based power damping inter-area oscillations using firefly and harmony search algorithms," *Sci. Rep.*, vol. 10, no. 1, p. 12176, Dec. 2020.

[27] P. Melin and D. Sánchez, "Multi-objective optimization for modular granular neural networks applied to pattern recognition," *Inf. Sci. (Ny).*, vol. 460–461, pp. 594–610, Sep. 2018.

[28] I. Miramontes, J. Guzman, P. Melin, and G. Prado-Arechiga, "Optimal Design of Interval Type-2 Fuzzy Heart Rate Level Classification Systems Using the Bird Swarm Algorithm," *Algorithms*, vol. 11, no. 12, p. 206, Dec. 2018.

[29] D. Sánchez, P. Melin, and O. Castillo, "Optimization of modular granular neural networks using a firefly algorithm for human recognition," *Eng. Appl. Artif. Intell.*, vol. 64, pp. 172–186, Sep. 2017.

[30] D. Sánchez, P. Melin, and O. Castillo, "A grey Wolf optimizer for modular granular neural networks for human recognition," *Comput. Intell. Neurosci.*, vol. 2017, 2017.

[31] D. Sánchez, P. Melin, and O. Castillo, "Comparison of particle swarm optimization variants with fuzzy dynamic parameter adaptation for modular granular neural networks for human recognition," *J. Intell. Fuzzy Syst.*, vol. 38, no. 3, pp. 3229–3252, Jan. 2020.

[32] H. Zhu, Z. An, C. Yang, K. Xu, E. Zhao, and Y. Xu, "EENA: Efficient evolution of neural architecture," in *Proceedings - 2019 International Conference on Computer Vision Workshop, ICCVW 2019*, 2019, pp. 1891–1899.

[33] T. Elsken, J. H. Metzen, and F. Hutter, "Efficient Multi-objective Neural Architecture Search via Lamarckian Evolution," in *7th International Conference on Learning Representations, ICLR* , 2019.

[34] B. Zoph, V. Vasudevan, J. Shlens, and Q. V. Le, "Learning Transferable Architectures for Scalable Image Recognition," in *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 2018, pp. 8697–8710.

[35] C. Liu *et al.*, "Progressive Neural Architecture Search," in *Proceedings of the European Conference on Computer Vision*, 2018, pp. 19–34.

[36] H. Liu, K. S. Deepmind, and Y. Yang, "DARTS: Differentiable Architecture Search," in *Proceedings of International Conference on Learning Representations 2019*, 2019, pp. 1–12.

[37] X. Dong and Y. Yang, "Searching for A Robust Neural Architecture in Four GPU Hours," in *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019.

[38] A. Askarzadeh, "A novel metaheuristic method for solving constrained engineering optimization problems: Crow search algorithm," *Comput. Struct.*, vol. 169, pp. 1–12, Jun. 2016.

[39] G. I. Sayed, A. E. Hassanien, and A. T. Azar, "Feature selection via a novel chaotic crow search algorithm," *Neural Comput. Appl.*, 2019.

[40] C. Qu and Y. Fu, "Crow search algorithm based on neighborhood search of non-inferior solution set," *IEEE Access*, vol. 7, pp. 52871–52895, 2019.

[41] A. Askarzadeh, "Capacitor placement in distribution systems for power loss reduction and voltage improvement: A new methodology," *IET Gener. Transm. Distrib.*, vol. 10, no. 14, pp. 3631–3638, Nov. 2016.

[42] D. Oliva, S. Hinojosa, E. Cuevas, G. Pajares, O. Avalos, and J. Gálvez, "Cross entropy based thresholding for magnetic resonance brain images using Crow Search Algorithm," *Expert Syst. Appl.*, vol. 79, pp. 164–180, Aug. 2017.

[43] D. Liu *et al.*, "ELM evaluation model of regional groundwater quality based on the crow search algorithm," *Ecol. Indic.*, vol. 81, pp. 302–314, Oct. 2017.

[44] X. Han, Q. Xu, L. Yue, Y. Dong, G. Xie, and X. Xu, "An Improved Crow Search Algorithm Based on Spiral Search Mechanism for Solving Numerical and Engineering Optimization Problems," *IEEE Access*, pp. 1–1, Mar. 2020.

[45] M. Abdullah, M. Ahmad, and D. Han, "Neural Architecture Search using Crow Search Algorithm," in *34th International Technical Conference on Circuits/Systems, Computers and Communications, ITC-CSCC*, 2019.

[46] Y. Lecun, L. Eon Bottou, Y. Bengio, and H. Patrick, "Gradient-Based Learning Applied to Document Recognition," in *Proceedings of the IEEE 86(11)*, 1998, pp. 2278–2324.

[47] A. Krizhevsky, "Learning Multiple Layers of Features from Tiny Images," Toronto, 2009.

[48] F. Wang *et al.*, "Residual Attention Network for Image Classification," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017, no. 1, pp. 3156–3164.

[49] G. Huang, Z. Liu, K. Q. Weinberger, and L. van der Maaten, "Densely Connected Convolutional Networks," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017.

[50] Y. Weng, T. Zhou, L. Liu, and C. Xia, "Automatic Convolutional Neural Architecture Search for Image Classification under Different Scenes," *IEEE Access*, vol. 7, pp. 38495–38506, 2019.

[51] E. Real, A. Aggarwal, Y. Huang, and Q. V. Le, "Regularized Evolution for Image Classifier Architecture Search," in *Proceedings of the AAAI Conference on Artificial Intelligence*, 2019, vol. 33, no. 01, pp. 4780–4789.

[52] E. Real *et al.*, "Large-Scale Evolution of Image Classifiers," in

Proceedings of the 34th international conference on machine learning, 2017, pp. 2902–2911.

[53] R. Luo, F. Tian, T. Qin, E. Chen, and T.-Y. Liu, "Neural Architecture Optimization," in *Advances in neural information processing systems (NIPS)*, 2018, pp. 7816–7827.

[54] M. Wistuba, "Deep learning architecture search by neuro-cell-based evolution with function-preserving mutations," in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 2019, vol. 11052 LNAI, pp. 243–258.