BACKPROPAGATION-FREE 4D CONTINUOUS ANT-BASED NEURAL TOPOLOGY SEARCH

A PREPRINT

AbdElRahman ElSaid *
elsaida@uncw.edu

Karl Ricanek * ricanekk@uncw.edu
Zeming Lyu † zl7069@rit.edu
Alexander Ororbia † ago@cs.rit.edu
Travis Desell † tjdvse@rit.edu

July 24, 2025

ABSTRACT

Continuous Ant-based Topology Search (CANTS) is a previously introduced novel nature-inspired neural architecture search (NAS) algorithm that is based on ant colony optimization (ACO). CANTS utilizes a continuous search space to indirectly-encode a neural architecture search space. Synthetic ant agents explore CANTS' continuous search space based on the density and distribution of pheromones, strongly inspired by how ants move in the real world. This continuous search space allows CANTS to automate the design of artificial neural networks (ANNs) of any size, removing a key limitation inherent to many current NAS algorithms that must operate within structures of a size that is predetermined by the user. This work expands CANTS by adding a fourth dimension to its search space representing potential neural synaptic weights. Adding this extra dimension allows CANTS agents to optimize both the architecture as well as the weights of an ANN without applying backpropagation (BP), which leads to a significant reduction in the time consumed in the optimization process: at least an average of 96% less time consumption with very competitive optimization performance, if not better. The experiments of this study - using real-world data - demonstrate that the ***BP-Free CANTS*** algorithm exhibits highly competitive performance compared to both CANTS and ANTS while requiring significantly less operation time.

1 Introduction

Hand-crafting artificial neural network architectures has been an obstacle in the advancement of machine learning (ML) as it is time-consuming, prone to trial-and-error, and requires significant domain expertise from model architects [43]. Exacerbating the problem, even slight alterations to problem-specific meta-parameters or topological features can lead to degradation in a model's performance [15, 2]. This leads to the need for problem-specific model-optimization, however, the optimization of deep neural networks (DNN), with potentially millions of structural elements and a large number of hyperparameters, is considered an NP hard problem and limited by computational resources [1]. Optimal architectures cannot be directly obtained by applying a continuous function because unlike the optimization of neural network parameters (weights) using a loss-function, an explicit function to measure the architecture-optimization is not available [23], due to the combinatorial and non-differentiable nature of architecture possibilities. Therefore, a number of meta-heuristic based neural architecture search (NAS) [14, 22, 34, 39, 24, 43] and neuroevolution (NE) [36, 5] methods have been developed to automate the process of ANN design. Recently, nature-inspired neural architecture search (NI-NAS) algorithms have shown increasing promise, including the Artificial Bee Colony (ABC) [19], Bat [41], Firefly [40], and Cuckoo Search [21] algorithms.

*Department of Computer Science, University of North Carolina Wilmington
†Software Engineering Department, Rochester Institute of Technology

Ant colony optimization (ACO) - originally introduced as a graph optimization method [10] - has proven itself to be a successful NI-NAS strategy. The reason ACO operates as a strongly feasible NAS method is rooted in the concept of its application as a graph optimization method, providing impressive results [26, 3, 9, 25, 10, 7, 8, 33]. As the structure of neural networks themselves are in essence directed graphs, this makes ACO well-suited towards the problem of NAS.

Initially, ACO for NAS was limited to small Jordan and Elman RNN neural structures [6] or was used to select the network inputs [27]. ACO was also used for optimizing the synaptic connections (weights) within RNN memory cell structures [11] and then later expanded to optimize entire RNN architectures within an algorithmic framework called Ant-based Neural Topology Search (ANTS) [13].

ANTS utilizes a massively connected neural structure as a discrete structural search space. Although the approach has shown success, the main critique for the strategy is the limitation of discreteness of the search space. Other NAS methods are mostly evolutionary-based; instead of operating within fixed bounds, they are constructive and continually add and removing nodes/edges during the evolutionary process [37, 28, 31]. Although that constructive NAS methods do not suffer from the limitations of a discreet search space -like ANTS - they are still more prone to falling into (early/poor) local minima or consume considerable time to evolve structures.

This work introduces a novel ant colony inspired algorithm, the *BackPropagation-Free Continuous Ant-based Neural Topology Search* (BP-Free CANTS), which utilizes a continuous search domain that flexibly facilitates the design of ANNs of any size to address the aforementioned challenges. Synthetic continuous ant (*cant*) agents roam explore the search space exploiting on the density and distribution of pheromone signals, emulating how ants swarm in the real world. The paths resulting from the agents exploration are used to construct nodes and edges of the RNN architectures. BP-Free CANTS (as well as the original CANTS algorithm [12], which we will call BP-CANTS in this work for clarity) is a distributed, asynchronous algorithm, which facilitates scalable usage of high performance computing (HPC) resources, and also utilizes communal intelligence to reduce the amount of training required for candidate evolved networks.

This work compares BP-Free CANTS to state-of-the-art benchmark NAS designing strategies for applied on RNN time series data prediction: ANTS [13] and BP-CANTS [12]. In addition to relaxing the requirement for pre-determined architecture bounds, BP-Free CANTS is shown to yield results that improve upon ANTS and BP-CANTS at a significantly lower computational cost. The BP-Free CANTS algorithm also provides an advancement to the field of NI-NAS is that it is able to efficiently both select for high performing architectures while at the same time finding well performing weights *without backpropagation*. While ACO has been applied to continuous domain problems before [35, 20, 38, 18, 4], to the authors' knowledge, BP-FREE CANTS is one of the first to simulate and apply the movements of ants through a unbounded 4D continuous space to search for optimal neural parameters and architectures. BP-Free CANTS has the capacity to optimize the neural topology and the neural synaptic parameters without applying the time consuming backpropagation and gradient-descent process required for memetic algorithms such as CANTS or other NE-based methods, further reducing required computational power and time consumption. BP-FREE CANTS also provides means for its cants agents to evolve during the optimization process, which enhances the chances of reaching better performing models, while requiring fewer hyperparameters that are associated with cants' behavior.

2 Methodology

Figure 1 presents a abstract-level illustration for how the BP-Free CANTS algorithm works. An asynchronous, distributed "work-stealing" strategy is applied (see pseudo-code in Algorithm 1) for scalable use on HPC clusters[3]. When receiving a request from a worker process (assignee), the manager (assignor) process generates new architectures. When requesting a new architecture, worker reports the fitness of the architecture it trained and evaluated. A fixed population of the best RNNs discovered by the workers is maintained by the manager process. The manager process also rewards the

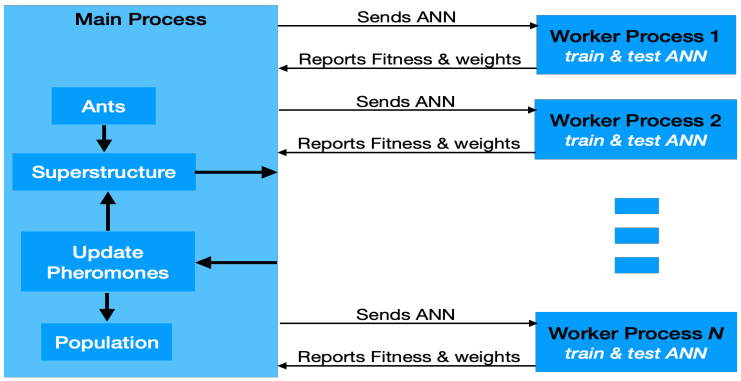


Figure 2: *The CANTS asynchronous design.*

[3]Source code is implemented in Python and is offered as an open-source project on https://github.com/a-elsaid/CANTS_public.git

Algorithm 1: Continuous Ant-guided Neural Topology Search Algorithm

procedure $WorkAssignor$
- ▷ Build 4D space with:
- ▷ inputs at $y_{axis} = 0$
- ▷ output at $y_{axis} = 1$
- ▷ z_{axis}: recurrent time steps
- ▷ x_{axis}: x component of neuron position
- $space \leftarrow$ **new** $SearchSpace$
- **for** $i \leftarrow 1 \dots optimization\_rounds$ **do**
  - $nn \leftarrow Swarm()$
  - $to\_assignee(nn_{new}, assignee.id)$
  - $nn, fitness \leftarrow fitness\_from\_assignee()$
  - **if** $nn\_fit < population.worst\_member$ **then**
    - $population.pop(population.worst\_member)$
    - $population.join(nn)$
    - $DopositePheromone(nn)$
  - **for** $cant \leftarrow 1 \dots no\_cants$ **do**
    - $cant.evovle(fitness)$

procedure $Assignee$
- $nn \leftarrow from\_assignor()$
- $fit \leftarrow train\_evaluate\_nn(nn)$
- $to\_assignor(nn, fit)$

procedure $Swarm$
- **for** $cant \leftarrow 1 \dots no\_cants$ **do**
  - $TakePath(cant)$
- ▷ Cluster paths vertices using DBscan
- $segments \leftarrow PathsDBS(cants)$
- ▷ Construct architecture from segments
- $rnn \leftarrow BuildRNN(segments)$ **return** rnn

procedure $TakePath(cant)$
- ▷ Input chosen discretely
- $PickInput(cant)$
- **while** $cant.y_{curr} < 0.99$ **do**
  - $r \leftarrow rand_{uni}(0, total\_pheromone - 1)$
  - $cant.level_{curr} \leftarrow cant.go\_up$
  - **if** $r > ant.explore\_intuition$ **or** $space[cant.level_{curr}]$ not $Empty$ **then**
    - $spot \leftarrow PutFootPrint(cant.radius_{sense})$
    - $cant.path.add(spot)$
    - $space.add(spot)$
  - **else**
    - $spot \leftarrow locateCenterOfMass(cant.pos_{curr}, cant.radius_{sense})$
    - **if** *spot* not in $space[cant.level_{curr}]$ **then**
      - $cant\_path.add(spot)$
- $PickOutput(cant)$

procedure $cant.Evolve(fitness)$
  $B = cant.behavior_{curr}$
  if $fitness < behaviors.worst$ **then**
    $B_{new} \leftarrow (cant.rate_{explore\_exploit}, cant.radius_{sense}, cant.r1, cant.r2)$
    $behaviors.join(B_{new})$
  if $cant.best\_bahaviors < 10$ **or** $random_{uni}(0,1) < \sigma_{mutation}$ **then**
    $B \leftarrow Mutate(B)$
  else
    $B \leftarrow CrossOver(B, cant.behavior_{best\_1}, cant.behavior_{best\_2})$

procedure $Mutate(behavior)$
  $behavior.rate_{explore\_exploit} \leftarrow random(0,1)$
  $behavior.rate_{sense} \leftarrow random(0,1)$;
  $behavior.r1 \leftarrow random_{uni}(-1,1)$
  $behavior.r2 \leftarrow random_{uni}(-1,1)$

procedure $CrossOver(behavior, behavior1, behavior2)$
  $behavior \leftarrow \big((behavior2 - behavoir1) \times random(0,1)\big) + behavior1$

procedure $PickInput(cant)$
  ▷ Probabilistically with pheromones density
  $total\_pheromone \leftarrow$ **sum**$(inputs.pheromones)$
  $r \leftarrow random_{uni}(0, total\_pheromone - 1)$
  $cant.input \leftarrow 0$
  while $r > 0$ **do**:
    if $r < inputs.pheromones[cant.input]$ **then**
      $cant.input \leftarrow +1$
      break
    else
      $r \leftarrow r - inputs.pheromones[cant.input]$
      $cant.input \mathrel{+}= 1$

procedure $PickOutput(cant)$
  ▷ Probabilistically with pheromones density
  $total\_pheromone \leftarrow$ **sum**$(outputs.pheromones)$
  $r \leftarrow random_{uni}(0, pheromone\_sum - 1)$
  $ant.input \leftarrow 0$
  while $r > 0$ **do**:
    if $r < outputs.pheromones[cant.output]$ **then**
      $cant.output \leftarrow 1$
      break
    else
      $r \leftarrow r - outputs.pheromones[cant.output]$
      $cant.output \mathrel{+}= 1$

procedure $PathsDBS(cants)$
  for $cant \leftarrow 1 \ldots cants\_count$ **do**
    for $spot \leftarrow 1 \ldots cant\_path$ **do**
      $segments[cant].insert(PickSpot(spot))$
  return $segments$

procedure $PickSpot(spot)$
  $[node, cluster_{spots}] \leftarrow PathDBS(spot, space[point.level])$
  $node.edges_{fan_out}.weight \leftarrow (AverageWeights(cluster_{spots}))$
  $space.add(node)$ **return** $node$

procedure $DepositPheromone(nn)$
  for each $node \in nn.nodes$ **do**
    $space[node].pheromone \mathrel{+}= const$
    $space[node].weight \leftarrow Average(node.weight, space[node].weight)$
    if $space[node].pheromone > THRESHOLD$ **then**
      $space[node].pheromone = PHEROMONE$

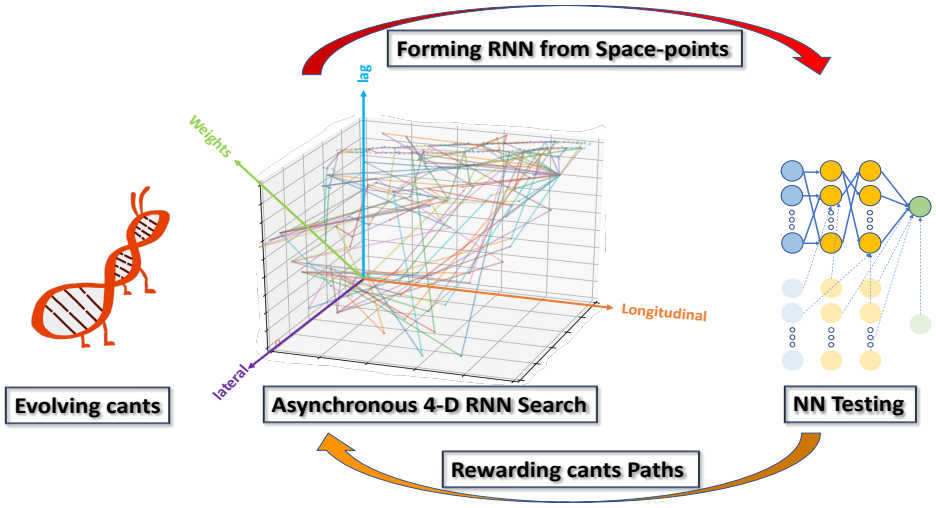


Figure 1: *The BP-free CANTS schematic graph.*

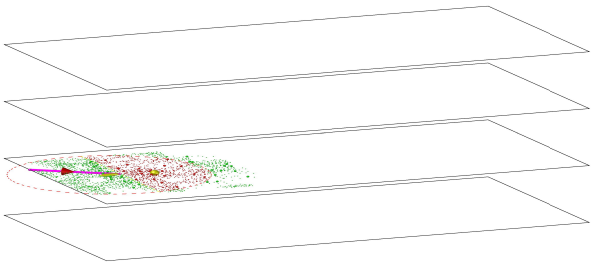
paths that the ant agents took to generate the
RNN architecture by depositing pheromones
in the continuous search space it manages. This computation design allows workers to evaluate and report the performance of the a single generated architecture at a time, without blocking waiting on results from other workers, offering a naturally load-balanced algorithm. Additionally, this allows BP-Free CANTS increased scalability over synchronous parallel evolution strategies because it can support a number of worker processes greater than the population size. The population managed by the manager process houses the best fitness (mean squared error on validation data) of the discovered RNNs reported by workers. This population is held at a fixed size and updated as in steady-state evolutionary algorithms, where the worst member of the population is removed and replaced any newly discovered RNN that performs better.

Candidate neural architectures are sampled from a search space comprised of stacked 2D continuous planes, where each 2D plane represents a particular time step t (see Figure 3a), and crucially in BP-Free CANTS, the fourth dimension of a given point in the search space represents the weight value of that point. The input features at each time step, represented as the input nodes, are uniformly distributed at the zero point of one of the axes of the search space. A synthetic continuous ant agent (or *cant*) discretely selects an input node position to start its path to an output node. The cant then moves on its path through the continuous space based on the current density and distribution of pheromone traces. On a given plane (lag-level) in the search space, a cant can only move forward toward the side of the outputs, or jump up to any lag-level above, but it is not permitted to moving to lower lag-levels in the stack, or to move backwards on the same lag-level. As paths (recurrent connections) moving up the stack represent passing information from neurons at previous time step to neurons at a future time step, the reverse would dictates passing unavailable future information to a previous time step of the RNN, which is not possible. Although cants are restricted to move only forward on a given plane, they are allowed to move backwards when they do a jump to a higher lag-level since recurrent connections can feed information from an architecture layer-level to lower layer-level that has a higher lag-level. This enforced (overall) forward movement on the planes (through layer-level) and upward on the stack (through lag-levels) ensures the continuous progress of cants towards the outputs, and alleviates the cycles in the search space.

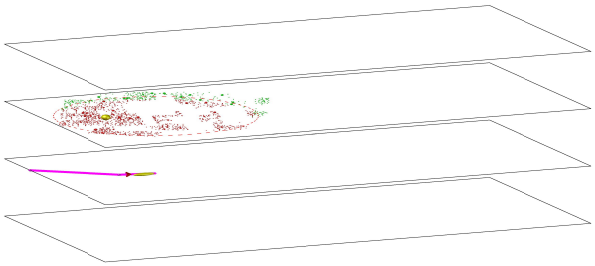
Figure 3 shows scenarios of cants movement through the search space starting from an input to an output. The shown movements illustrate cants exploration of new spots in the search space, cants exploitation of previously searched areas, attracted by previous pheromone deposits, and how cant-paths are translated into a final candidate architecture (refer to [12] for more details).

Cant Agent Input Node and Layer Selection: Each level in the search space has a level-selection pheromone value, p_l, where l is the level. These are initialized to $p_l = 2 * l$ where the top lag-level for the current time step is $l = 1$, the next lag-level for the first time lag is $l = 2$ and so on. A cant selects its starting lag-level according to the probability of starting at lag-level l under $P(l) = \frac{p_l}{\Sigma_{l=1}^{L} p_l}$, where L is the total number of lag-levels. This scheme encourages cants to start at lower lag-levels of the stack at the beginning of the search. After selecting a lag-level, the cant selects its input node in a similar fashion, based on the pheromones for each input node location on that lag-level. When a

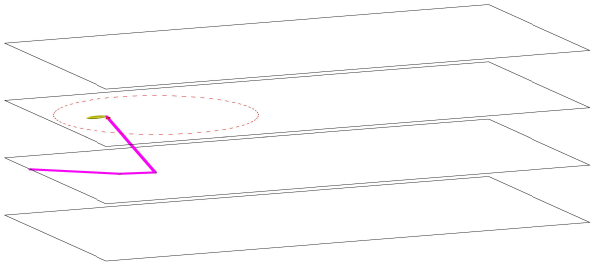
(a) *cant picking* 1^{st} *spot (explore move)*



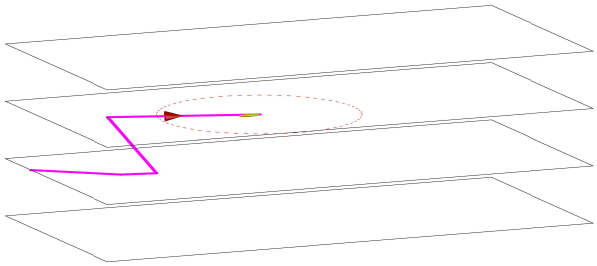
(b) *cant picking a spot (exploit move)*



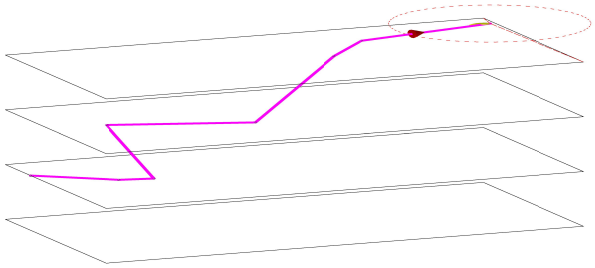
(c) *cant picking a spot in higher level*



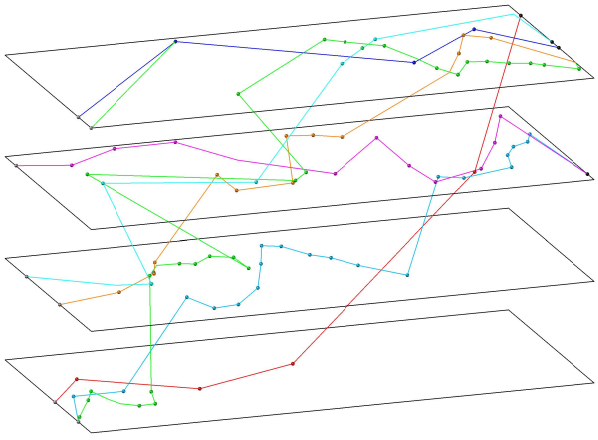
(d) *moving to the new spot*



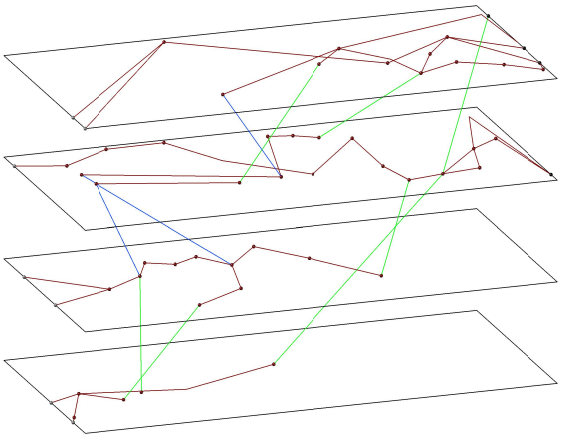
(e) *another spot*



(f) *picking an output*

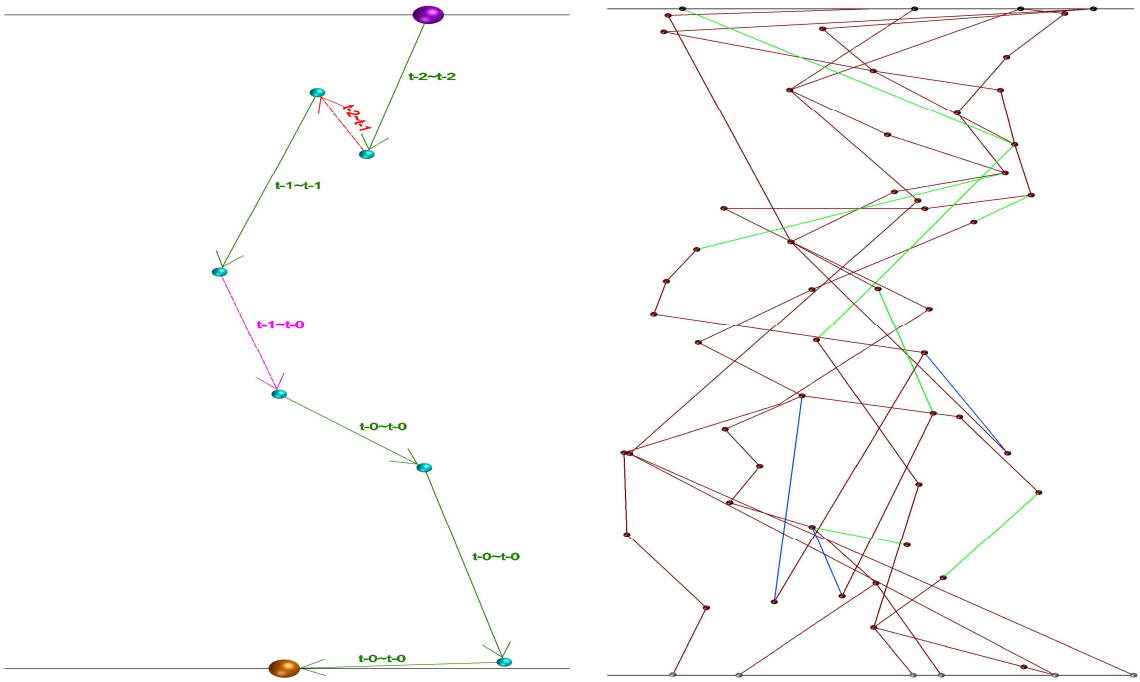


(g) *cants paths*



(h) *cants path after condensation*

Figure 3: ***CANTS paths & Architecture Building***



(i) *2D projection of a cant path* (j) *2D projection of cants paths*

Figure 3: ***CANTS paths & Architecture Building (continued):***

candidate RNN is inserted into the population, the lag-level pheromones for each lag-level, utilized by that RNN, are incremented.

Cant Agent Movement: The movement of cants in the search space is a balance between exploitation and exploration, which is reached by imitating real-world ants, who follows clues to communicate information about a target. When a cant takes a step, it first decides if it will climb to higher lag-level in the planes stack, which is done similarly to how they select the initial lag-level with levels options limited to only the higher planes up the stack. When the cant determines if the next move will be a climb or not, it will make a choice to whether the move will be an exploration or exploitation step. The movement type (exploration or exploitation) is decided by the cant based on its exploration/exploitation parameter, which is initially generated using a uniform random distribution, and then later evolves through the generation of neural architectures, as one the agent's behaviors (see Section *Cants Evolution* below).

In exploitation moves, *i.e.*, following pheromone traces/clues, a cant will first sense the pheromone traces around, that is lying within its sensing radius, ρ. If the move is decided to be on the same lag-level, the cant will ignore previously-deposited pheromone that is behind it, otherwise, the whole pheromone distribution inside the sensing radius will be considered by the cant to determine the next location in its path. The center-of-mass of the pheromone distribution inside the sensing radius is calculated by the cant as the location of the next step in its path. Cants' path points are saved by the their candidate generated RNN for later pheromone deposit, for future communication with other cants, if the RNN made it to the population of best discovered RNNs. Applying the center-of-mass in cants selection to their next move makes the placement of pheromone at individual points a peripherally effective factor in cant-to-cant communication compared to the concentration of pheromone in a region in the space, which resembles real-ants foraging in nature.

When an exploration move is decided by a cant, it will pick a random point lying within its sensing radius for its next location. The exploration search for a new point will start after the cant decides if it will stay on the same lag-level or jump to a higher level. If it is moving on the same level, the cant will generate a random number between $[0, 1]$ as an angle bisector, or between $[-1, 1]$ if jumping to a higher level. An angle is then calculated using the bisector: $\theta = angle\_bisect \times \pi$, which will be the direction of the cant's next coordinate move using these formulae:

$$x_{new} \leftarrow x_{old} + \rho * cos(\theta); \qquad y_{new} \leftarrow y_{old} + \rho * sin(\theta)$$

Cants Evolution: Individual CANTS agents, cants, are born/initialized with uniformly distributed random exploration/exploitation and sensing-range coefficients. In addition, two more parameters are introduced to control the

speed at which a cant will move (for one step) by controlling the size of the sensing radius using this 2^{nd} degree polynomial formula:

$$sense\_range_{precieved} = max\Big(sense\_radius - \big(y^2 \times r_1 + y \times r_2\big), 0.1\Big)$$

where r_1 and r_2 parameters control the size of the single cant's perceived sensing range based on how far the cant is from the input node(s) (y spacial distance from inputs). The bigger the sensing range, the more the cant is able to conceptually move longer steps at a time, and the smaller the sensing rage, the smaller the steps the cant can do, and yet, the lowest range value is set at 0.1 to guarantee that the agent will not completely loose its pheromone sense and halt in position. The polynomial aspect of the formula used will eventually control the pattern of the cant movement speed: *a)* slower at the begging and faster as it approaches the outputs, or *b)* slower at the begging, speeds up in the middle, and slows as it approaches the outputs. Figure 4 illustrates those patterns.

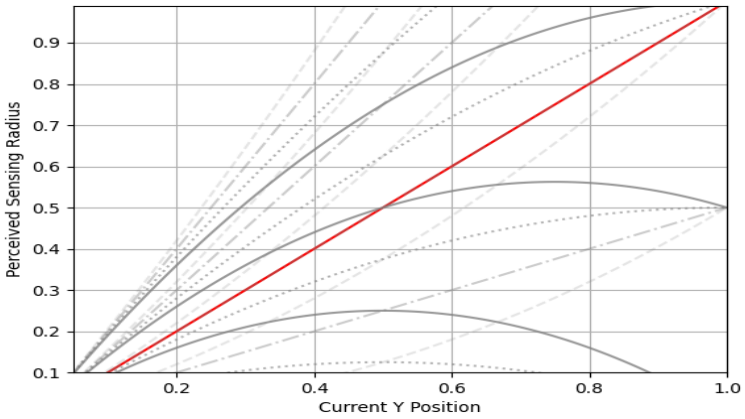


Figure 4: *Potential cant speed pattern based on its y position, r_1, and r_2 values.*

Although these cants parameters are uniquely generated for the individual cants, they evolve between the CANTS iterations using a genetic algorithm that evolves those parameters through mutation and cross-over operations, based on the best performing generated NN (see $cant.Evolve()$ in Algorithm 1).

Incorporating the above behavioral traits provides more flexibility to the optimization agents themselves and devising this evolution mechanism makes those parameters self-tune during evolution instead of being treated as extra hyper-parameters.

Condensing Cant Paths to RNN Nodes: The points on different cants paths can be in a spacial proximity that can be also mapped to neural proximity, thus, represent a type of redundancy. To avoid having extraneous neurons in the neural space, the points on the cants' paths are clustered using the DBSCAN algorithm [16], condensing the resulting clusters to centriods. After that, the points of neighborhood inside a given cluster are represented by the centriod of the cluster, and the centriod is translated to a neuron in the generated neural architecture (see Figures 3g and 3h). The node types are picked by a pheromone-based discrete local search, as is done in the discrete space ANTS algorithm. Each of these node types at the selected point will have their own pheromone values that drive probabilistic selection.

Communal Weight Sharing: To avoid retraining the neural parameters of newly-generated RNN from scratch, BP-Cants uses a communal-weight-sharing strategy is deployed to offer an advanced starting point to the generated RNNs, exploiting parametric values of previously trained RNNs. (refer to [12] for more details). In BP-Free CANTS, the generated RNNs are only tested without BP training, thus, the communal weight sharing is carried throughout the optimization process without requiring any updates from the generated RNNs.

Pheromone Deposit: When a candidate RNN wins a place in the population, the manager process increments the pheromone values of the architecture's corresponding space centroids by a constant value as a reward. Since they contribute to the creation of centroids by their pheromone values, the points in proximity to centroids, i.e. the cluster of points used to generate the centroid, are also rewarded by increasing their pheromone levels by a fraction of the same constant value, depending on their distance from the centroid. The space points' pheromone values is capped at a maximum threshold to avoid having points of super-attraction to cants, causing the optimization to prematurely conclude.

Pheromone Volatility: Pheromones regularly decay through the process iterations, regardless of the RNNs' performance reported by workers. The decay occurs by a constant value and, after a predetermined minimum threshold, the point vanishes from the search space. Wiping out points with faint pheromone traces allows the search space to git rid of pheromones tiny residual that might impede cant-to-cant communication as well as dragging the overall optimization process. This pheromone degradation also prevents the algorithm from getting prematurely stuck in local minima.

Time Complexity of the Algorithms: While the time complexity of the various ANTS algorithms is important, it should be noted that the time spent generating recurrent neural networks with these algorithms is orders of magnitude less than the time to evaluate the fitness of the generated neural networks (which requires training and validation for ANTS and BP-CANTS, and only validation for BP-Free CANTS). This is highlighted in Figure 7.
For ANTS and BP-CANTS, each generated recurrent neural network (which may have tens of thousands of trainable parameters) needs to be trained using backpropagation through time for a number of epochs. As all the ANTS algorithms use an asynchronous distributed/parallel execution strategy, training potentially hundreds of RNNs simultaneously on distributed compute nodes while the master process which generates new RNNs spends most of its time waiting for more work requests from the workers (it may take fractions of a second to generate a network, however, training them and then evaluating them on the validation dataset can take multiple minutes, even for few number of epochs). For BP-Free CANTS, the forward pass of the network over the validation data to evaluate the network without backpropagation is still orders of magnitude faster than the time for the algorithm to generate the networks.
The time complexity of BP-Free CANTS is O(P log P), where P is the number of pheromone points in the search space. This is due to the complexity being bound by the DBSCAN operation on the pheromones in the search space (all other operations, e.g., network construction are linear based on the clusters generated by DBSCAN). New pheromone values are added every time cant agents move through the search space. These pheromone values degrade over time, and are removed when they fall below a given threshold. Unfortunately, due to the stochastic nature of the algorithm it is not possible to compute an upper bound on the maximum number of pheromones possible. That being said, given our experimental results presented in Figure 7 and discussed in Section 3.2.2, RNN generation time did not appear to grow in an unbounded manner in our experiments. A potential area of future work (if this proves to be a computational bottleneck) would be to add in strict limits to the possible number of pheromones present in the search space at any time, by for example removing the oldest pheromones or combining clustered pheromone traces.

3 Results

This work compares BP-Free CANTS to the previous state-of-the-art ANTS and BP-CANTS algorithms on three real world datasets related to power systems. All three methods were used to perform time series data prediction for a parameter, which have been used as benchmarks in prior work. For the coal plant data, note that net plant heat rate was used from the coal plant's boiler. Experiments were also performed to investigate the effect the number of cants.

Computing Environment The results for ANTS, BP-CANTS), and BP-Free CANTS were obtained by scheduling the experiment on Rochester Institute of Technology's high performance computing cluster with 64 Intel® Xeon® Gold 6150 CPUs, each with 36 cores and 375 GB RAM (total 2304 cores and 24 TB of RAM). The experiments used 3 nodes (108 cores) and also took approximately 4 weeks to complete the experiments.

Datasets The dataset used, which is derived from coal-fired power plant sensor readings, has been previously made publicly available on the EXAMM repository to encourage further study in time series data prediction and reproducibility [4]. The dataset comes from measurements collected from 12 burners of a coal-fired power plant.

The dataset is multivariate and non-seasonal, with 12 input variables (potentially dependent). These time series are very long, with the burner data separated into 7000 time step chunks. The dataset is sampled and separated into a training set of 1875 steps and a test set of 625 steps (per minute recordings).

3.1 On the Number of Cant Agents

An experiment was conducted to determine the effect that the number of cant agents would have on the performance of the BP-Free CANTS algorithm. The experiment focused on the net plant heat rate feature from the coal-fired power plant boilers dataset. The number of ants that were simulated and evaluated were 5, 10, 15, 25, 35, and 50.

3.2 Algorithm Benchmark Comparisons

To compare the three different NAS strategies, each experiment was repeated 10 times (trials) to facilitate a statistical comparison and the algorithms were set to the following amounts of computational processing: the ANTS and BP-CANTS (the memetic algorithms) were simulated over 1000 total steps with 30 epochs of local backpropagation applied (to tune candidate RNNs), whereas for the proposed non-memetic BP-Free CANTS, 3000 steps were simulated without any backpropagation. BP-free CANTS was given more optimization iterations because it is faster to finish (as will be discussed later in the section), while BP-CANTS and ANTS consume more time since they perform BP epochs

[4] https://github.com/travisdesell/exact/tree/master/datasets/

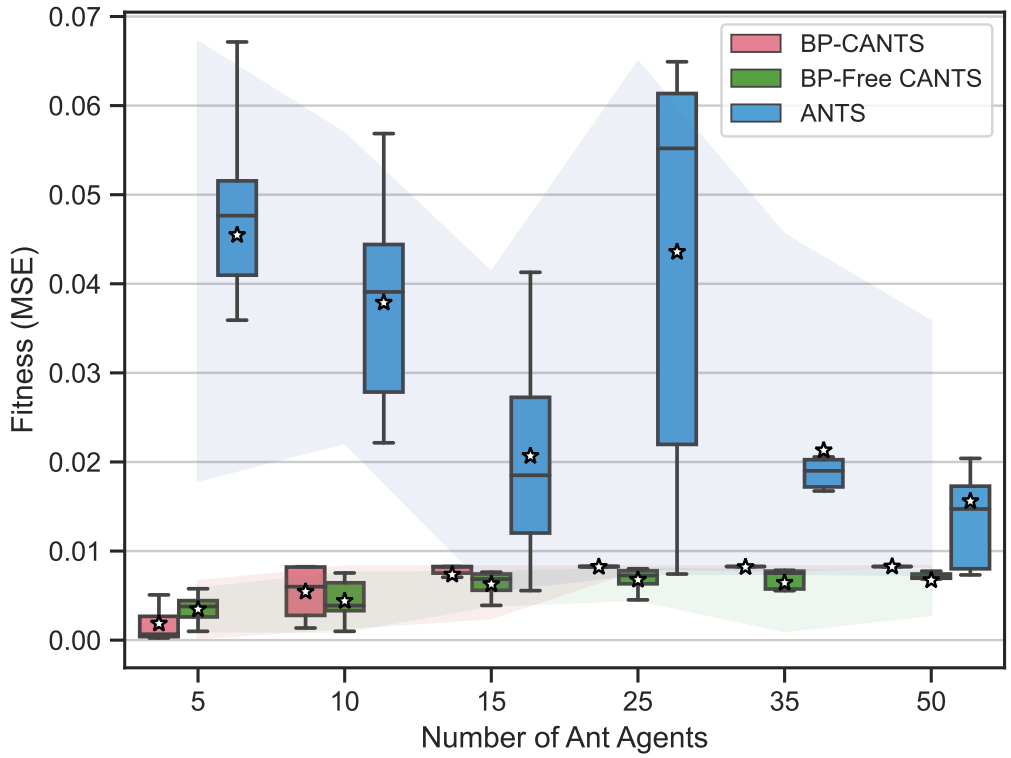


Figure 5: ***BP-free CANTS Performance (as a function of the number of agents/particles) of BP-Free CANTS compared to CANTS and ANTS. Note that a lower value of the fitness/loss is better (given that optimization is focused on minimization).***

	Average Fitness (MSE)		
No. Agents	**BP-Free CANTS**	**BP CANTS**	**ANTS**
5	0.0035	0.0018	0.0455
10	0.0044	0.0058	0.0379
15	0.0063	0.0073	0.0207
25	0.0068	0.0082	0.0436
35	0.0064	0.0082	0.0216
50	0.0067	0.0083	0.0156

Table 1: ***Average Fitness***

per round/iterations of their respective optimization process. For both BP and BP-Free CANTS, the sensing radii of the cant agents and exploration instinct values were generated uniformly via $\sim U(0.01, 0.98)$ when the cants were created/initialized, initial pheromone values were set to 1 and the maximum was kept at 10 with a pheromone decay rate set to 0.05. For the DBSCAN module, clustering distance was set to 0.05 with a minimum point value of 2. The population size used was 10. ANTS, BP-CANTS, and BP-Free CANTS had a maximum recurrent depth of 5 and the predictions were made over a forecasting horizon of 1. The generated RNNs were each allowed 30 epochs of back-propagation for local search/fine-tuning for the memetic algorithms ANTS and BP-CANTS. ANTS utilized the hyper-parameters previously reported to yield best results [31, 13].

3.2.1 Performance Benchmarking

The results shown in Figure 5 - which compare the performance of
BP-Free CANTS, BP-CANTS, and ANTS as described in the experimental settings above - are measurements of the range of mean squared error (MSE) of each algorithm's best-found RNNs. The figure uses a box-plot to illustrate the minimum, maximum, median (line in interquartile range), and average (star markers) of the different trails of

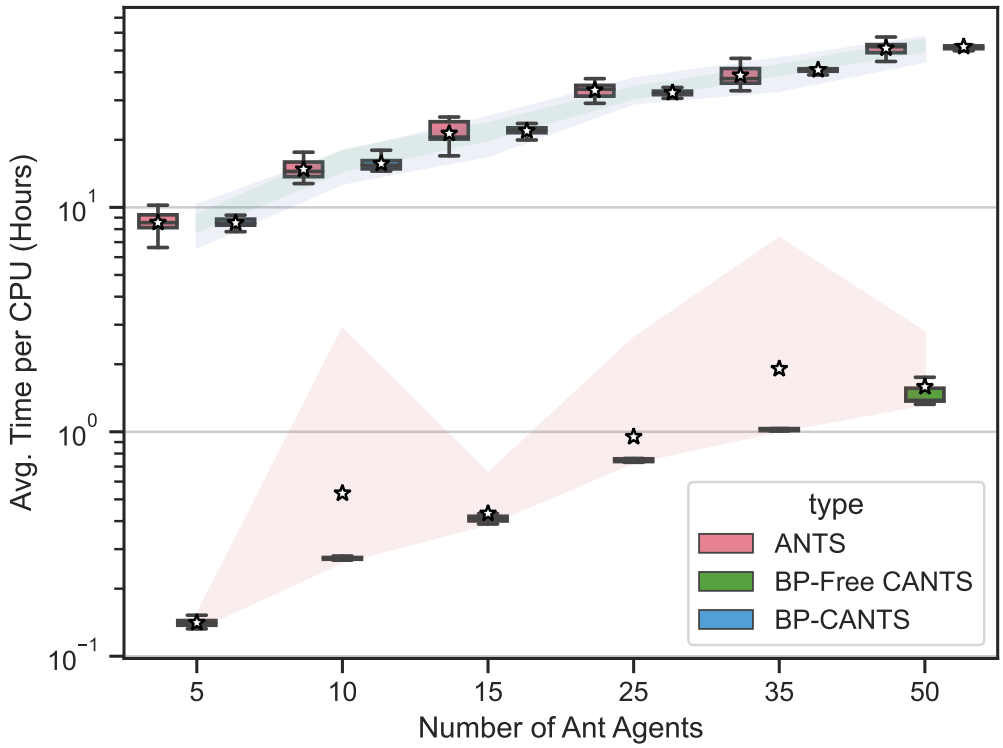


Figure 6: ***Time consumed by the BP-free CANTS , CANTS, and ANTS algorithms (lower/closer to zero is better). Note that the y-axis is a logarithmic scale, which shows BP-Free CANTS operating orders of magnitude faster than BP-CANTS and ANTS.***

the experiments, without showing the outliers. The shadow-shapes bound the area between the maximum and the minimum values of the fitness within the different trails of experiments (but with the outliers counted). Table 1 depicts the average MSE for the 3 methods, showing that BP-Free CANTS is at the lead in the vast majority of the experiments.

Desirably, both BP-CANTS and the proposed BP-Free CANTS - outperformed ANTS in all of the experiment trails. The mean and the average of the ANTS results over the different number of ants/agents seemed to consistently improve from 5 ants to 15 ants, but then some less accurate results started to show up/appear when 25 ants were used. Nevertheless, the ANTS results continued to improve as the trend continued past 35 and 50 ants. Generally, the ANTS algorithm's performance exhibits improved MSE as more agents were used.

While BP-CANTS slightly outperformed BP-free CANTS with 5 cants, the latter exhibited consistently better performance than the former for all other numbers of agents. The saturation of the performance curve of CANTS against the number of cants indicates that for the dataset used, the optimum number of optimization agents was with the fewest number of cants 5, however BP-Free CANTS was finding still finding comparative best newtworks at 35 and 50 cants. Note that BP-Free CANTS obtained these good-quality results (with respect to fitness/loss) using a higher number of optimization iterations but crucially at a significantly lower computational time cost (see next section).

3.2.2 Time Benchmarking

Figure 6 illustrates the average time consumed by a CPU in the experiments done for each optimization method (ANTS, BP-CANTS, BP-Free CANTS) across the different trails of the experiments. As observed in the figure, there is a significant gap between the time periods taken by BP-Free CANTS and both ANTS and BP-CANTS. Not only was the performance of BP-Free CANTS significantly better than CANTS for most of the used agent counts, even with the increased evolution operations undertaken by BP-CANTS to evolve its agents during the simulated optimization iterations, it still finished orders of magnitude faster than ANTS and BP-CANTS because it did not have to apply BP during its iterations. The total operation time of CANTS (with BP) and ANTS were notably similar to each other, however this is to be expected given both were allowed to perform the same number of optimization iterations and BP epochs. Table 2 depicts the large gap between the BP-Free CANTS average consumed time at different number of

agents utilization, and the BP-CANTS and ANTS, which both use backprobagation in their optimization process. On average, BP-Free CANTS is faster than and BP-CANTS and ANTS by 98.5%, and 96.1% respectively.

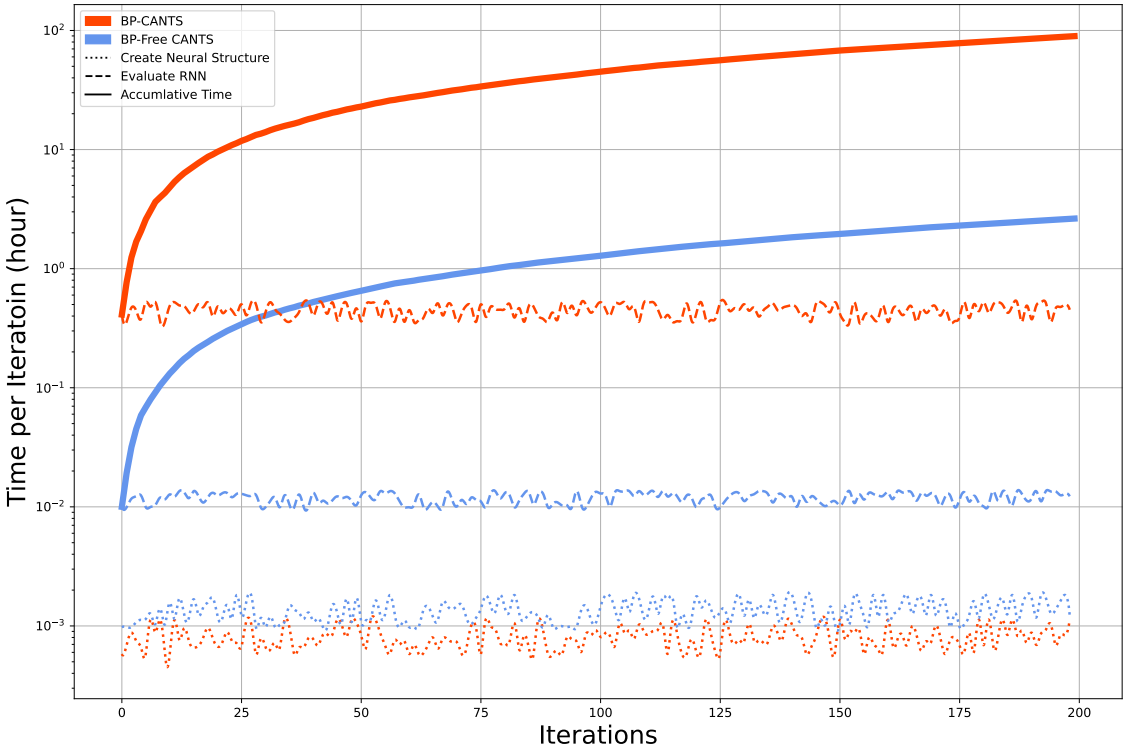


Figure 7: ***Time Consumed by BP-Free CANTS vs. Time Consumed by BP-CANTS:*** *Time consumed to generate neural structures (dotted lines) is about the same for both methods. BP-CANTS consumes more evaluation time (dashed lines) because they use BP. Adding the accumulative generation and evaluation time gives the solid curves for both BP-CANTS and BP-Free CANTS.*

Figure 7 illustrates the time consumption-difference of BP-Free CANTS and BP-CANTS. The two methods were allowed to generate 200 neural structures using 8 CPUs and 35 agents. The figure shows the time consumed to generate the neural structure, where the two methods timing were very close with BP-Free CANTS taking a bit more time to process due to the extra 4^{th} dimension. Observably, the evaluation time of BP-CANTS was an order of magnitude higher than that of the BP-Free CANTS because the former used BP, while the latter does not. The figure also shows the accumulative time of neural-structure-generation and the RNN-evaluation of the two methods as the iterations progress. The great difference between the accumulative time of BP-CANTS and BP-Free originates from the time consumed by BP. The slope of the cumulative time is higher at early iterations compared to later ones because the search space is fairly empty (no pheromone deposits) at the beginning of the iterations, but as iterations increase, the search space is gradually filled, causing the agents to create more neurons, which increases the evaluation time. The curves then saturates a bit because of the pheromone-evaporation effect, which maintains the pheromone traces at about constant level.

4 Discussion and Future Work

This work proposes the backpropagation-free continuous ant-based neural topology search (BP-Free CANTS), a novel, nature-inspired, and non-memetic optimization method that employs a 4D continuous space to conduct unbounded neural architecture search (NAS). CANTS provides a unique strategy to overcome the key limitations of constructive neuro-evolutionary strategies (which are often prematurely trapped at local minimas) as well as other NAS strategies bonded by search space limits . Moreover, BP-Free CANTS expands over previous ACO-centric approaches by crucially mapping the search space of the synaptic weights of the evolved neural networks as a fourth dimension in a 4D space, unifying both the search for optimal neural structure and synaptic weight values.

Experimental evaluations were carried on BP-Free CANTS to validate its performance in automatically designing recurrent neural networks (RNNs) used in time-series predictions, using a real-world dataset in the power systems

No. Agents	Average Time (hrs) BP-Free CANTS	BP CANTS	ANTS
5	0.14	7.75	7.70
10	0.53	14.24	13.29
15	0.43	19.94	19.21
25	0.95	23.57	26.58
35	1.91	26.08	27.13
50	1.43	33.08	35.82

Table 2: ***Average Time Consumed***

domain. We compared our procedure to a state-of-the-art meta-heuristic optimization approach, ANTS (a discrete space ant colony NAS algorithm) as well as the powerful memetic/backprop-centric algorithm CANTS (BP-CANTS). Our results demonstrate that BP-Free CANTS improves over ANTS and BP-CANTS while running substantially and significantly faster.

This study presents important steps in generalizing ant colony optimization algorithms to complex, continuous search spaces, specifically for unbounded graph optimization problems (with NAS as the key target application), opening up a number of promising avenues for future work. In particular, for BP-Free CANTS, while the search space is continuous in each two-dimensional plane (or time step) of our temporal stack, there is still the (maximum) number of discrete levels that a user must specify. Therefore, a promising extension of our algorithm would be to make the search space continuous across all three dimensions, removing this parameter entirely, allowing pheromone placements to guide the depth of the recurrent connections. This could have implications for discrete-event, continuous-time RNN models [30], which attempt to tackle a broader, more complex set of sequence modeling problems. Additionally, BP-Free CANTS can provide a well performing solution to the design and training of networks which have activation functions or other components for which derivatives cannot be calculated (which preclude them from utilizing backpropagation), such as models with sampled activities (e.g., Bernoulli distribution) or even discrete ones such as spiking neural networks. A potential disadvantage of BP-Free CANTS is that if enough computational resources are available to offer BP-based NAS methods a significant number of training epochs (without worry about cost or emissions), the accuracy of the resulting models will be higher than of BP-Free Cants. Nevertheless, the computational cost of BP is always a burden in NAS methods, and a good reason to seek faster alternatives" [29, 42, 32, 44]. Finally, and potentially the most interesting, ACO algorithms including the one designed in this work, generally focus on utilizing only one single colony. According to myrmecologists[5], it would prove fruitful to view and design synthetic ant colonies as living organisms themselves, with ants as their living "cells" [17], potentially offering a flexible, scalable simulation framework for modeling how ant agents might achieve more complex tasks related to overall survival (i.e., a colony of ant colonies). Expanding this algorithm to other domains, such as the automated design of convolutional neural networks (for computer vision) or to other types of recurrent temporal networks, such as those used for natural language processing, would further demonstrate the broad applicability of this nature-inspired approach.

5 Acknowledgements

The experiments carried out in this work were facilitated by the computational resources and support of:

- The NSF Advanced Cyberinfrastructure Coordination Ecosystem: Services & Support (ACCESS - formally XSEDE)[6].
- The Research Computing at the Rochester Institute of Technology[7].

References

[1] BACANIN, N., BEZDAN, T., TUBA, E., STRUMBERGER, I., AND TUBA, M. Optimizing convolutional neural network hyperparameters by enhanced swarm intelligence metaheuristics. *Algorithms 13*, 3 (2020).

[5] Myrmecology: The branch of entomology focusing on the scientific study of ants.

[6] Grant number: CIS220035

[7] Rochester Institute of Technology. https://doi.org/10.34788/0S3G-QD15

[2] BARNA, G., AND KASKI, K. *Choosing optimal network structure*. Springer Netherlands, Dordrecht, 1990, pp. 890–893.

[3] BIANCHI, L., GAMBARDELLA, L. M., AND DORIGO, M. An ant colony optimization approach to the probabilistic traveling salesman problem. In *International Conference on Parallel Problem Solving from Nature* (2002), Springer, pp. 883–892.

[4] BILCHEV, G., AND PARMEE, I. C. The ant colony metaphor for searching continuous design spaces. In *AISB workshop on evolutionary computing* (1995), Springer, pp. 25–39.

[5] DARWISH, A., HASSANIEN, A. E., AND DAS, S. A survey of swarm and evolutionary computing approaches for deep learning. *Artificial Intelligence Review 53*, 3 (2020), 1767–1812.

[6] DESELL, T., CLACHAR, S., HIGGINS, J., AND WILD, B. Evolving deep recurrent neural networks using ant colony optimization. In *Evolutionary Computation in Combinatorial Optimization* (Cham, 2015), G. Ochoa and F. Chicano, Eds., Springer International Publishing, pp. 86–98.

[7] DORIGO, M. Optimization, learning and natural algorithms. *PhD Thesis, Politecnico di Milano* (1992).

[8] DORIGO, M., BIRATTARI, M., AND STUTZLE, T. Ant colony optimization. *IEEE computational intelligence magazine 1*, 4 (2006), 28–39.

[9] DORIGO, M., AND GAMBARDELLA, L. M. Ant colony system: a cooperative learning approach to the traveling salesman problem. *IEEE Transactions on evolutionary computation 1*, 1 (1997), 53–66.

[10] DORIGO, M., MANIEZZO, V., AND COLORNI, A. Ant system: optimization by a colony of cooperating agents. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics) 26*, 1 (1996), 29–41.

[11] ELSAID, A., EL JAMIY, F., HIGGINS, J., WILD, B., AND DESELL, T. Optimizing long short-term memory recurrent neural networks using ant colony optimization to predict turbine engine vibration. *Applied Soft Computing 73* (2018), 969–991.

[12] ELSAID, A., KARNS, J., LYU, Z., ORORBIA, A. G., AND DESELL, T. Continuous ant-based neural topology search. In *International Conference on the Applications of Evolutionary Computation (Part of EvoStar)* (2021), Springer, pp. 291–306.

[13] ELSAID, A., ORORBIA, A. G., AND DESELL, T. J. Ant-based neural topology search (ants) for optimizing recurrent networks. In *International Conference on the Applications of Evolutionary Computation (Part of EvoStar)* (2020), Springer, pp. 626–641.

[14] ELSKEN, T., METZEN, J. H., AND HUTTER, F. Neural architecture search: A survey. *arXiv preprint arXiv:1808.05377* (2018).

[15] ERKAYMAZ, O., ÖZER, M., AND YUMUŞAK, N. Impact of small-world topology on the performance of a feed-forward artificial neural network based on 2 different real-life problems. *Turkish Journal of Electrical Engineering & Computer Sciences 22*, 3 (2014), 708–718.

[16] ESTER, M., KRIEGEL, H.-P., SANDER, J., XU, X., ET AL. A density-based algorithm for discovering clusters in large spatial databases with noise. In *Kdd* (1996), vol. 96, pp. 226–231.

[17] GORDON, D. M. *Ant encounters: interaction networks and colony behavior*, vol. 1. Princeton University Press, 2010.

[18] GUPTA, H., AND GHOSH, B. Transistor size optimization in digital circuits using ant colony optimization for continuous domain. *International Journal of Circuit Theory and Applications 42*, 6 (2014), 642–658.

[19] HORNG, M.-H. Fine-tuning parameters of deep belief networks using artificial bee colony algorithm. *DEStech Transactions on Computer Science and Engineering* (2017).

[20] KUHN, L. D. Ant colony optimization for continuous spaces. *Computer Science and Computer Engineering Undergraduate Honors Theses (35)* (2002).

[21] LEKE, C., NDJIONGUE, A. R., TWALA, B., AND MARWALA, T. A deep learning-cuckoo search method for missing data estimation in high-dimensional datasets. In *International Conference on Swarm Intelligence* (2017), Springer, pp. 561–572.

[22] LIU, H., SIMONYAN, K., AND YANG, Y. Darts: Differentiable architecture search. *arXiv preprint arXiv:1806.09055* (2018).

[23] LIU, Y., SUN, Y., XUE, B., ZHANG, M., YEN, G. G., AND TAN, K. C. A survey on evolutionary neural architecture search. *IEEE transactions on neural networks and learning systems* (2021).

[24] LUO, R., TIAN, F., QIN, T., CHEN, E., AND LIU, T.-Y. Neural architecture optimization. In *Advances in neural information processing systems* (2018), pp. 7816–7827.

[25] M. DORIGO AND L. M. GAMBARDELLA. Ant colonies for the travelling sales man problem. *BioSystems, 43(2):73–81* (1997).

[26] MANFRIN, M., BIRATTARI, M., STÜTZLE, T., AND DORIGO, M. Parallel ant colony optimization for the traveling salesman problem. In *International Workshop on Ant Colony Optimization and Swarm Intelligence* (2006), Springer, pp. 224–234.

[27] MAVROVOUNIOTIS, M., AND YANG, S. Evolving neural networks using ant colony optimization with pheromone trail limits. In *Computational Intelligence (UKCI), 2013 13th UK Workshop on* (2013), IEEE, pp. 16–23.

[28] MIIKKULAINEN, R., LIANG, J., MEYERSON, E., RAWAL, A., FINK, D., FRANCON, O., RAJU, B., SHAHRZAD, H., NAVRUZYAN, A., DUFFY, N., ET AL. Evolving deep neural networks. In *Artificial Intelligence in the Age of Neural Networks and Brain Computing*. Elsevier, 2019, pp. 293–312.

[29] MILLER, W. T., GLANZ, F. H., AND KRAFT, L. G. Cmac: An associative neural network alternative to backpropagation. *Proceedings of the IEEE 78*, 10 (1990), 1561–1567.

[30] MOZER, M. C., KAZAKOV, D., AND LINDSEY, R. V. Discrete event, continuous time rnns. *arXiv preprint arXiv:1710.04110* (2017).

[31] ORORBIA, A., ELSAID, A., AND DESELL, T. Investigating recurrent neural network memory structures using neuro-evolution. In *Proceedings of the Genetic and Evolutionary Computation Conference* (New York, NY, USA, 2019), GECCO '19, ACM, pp. 446–455.

[32] PAOLA, J., AND SCHOWENGERDT, R. A detailed comparison of backpropagation neural network and maximum-likelihood classifiers for urban land use classification. *IEEE Transactions on Geoscience and Remote Sensing 33*, 4 (1995), 981–996.

[33] PEAKE, J., AMOS, M., YIAPANIS, P., AND LLOYD, H. Scaling techniques for parallel ant colony optimization on large problem instances. In *Proceedings of the Genetic and Evolutionary Computation Conference* (2019), pp. 47–54.

[34] PHAM, H., GUAN, M. Y., ZOPH, B., LE, Q. V., AND DEAN, J. Efficient neural architecture search via parameter sharing. *arXiv preprint arXiv:1802.03268* (2018).

[35] SOCHA, K., AND DORIGO, M. Ant colony optimization for continuous domains. *European journal of operational research 185*, 3 (2008), 1155–1173.

[36] STANLEY, K. O., CLUNE, J., LEHMAN, J., AND MIIKKULAINEN, R. Designing neural networks through neuroevolution. *Nature Machine Intelligence 1*, 1 (2019), 24–35.

[37] STANLEY, K. O., AND MIIKKULAINEN, R. Evolving neural networks through augmenting topologies. *Evolutionary computation 10*, 2 (2002), 99–127.

[38] XIAO, J., AND LI, L. A hybrid ant colony optimization for continuous domains. *Expert Systems with Applications 38*, 9 (2011), 11072–11077.

[39] XIE, S., ZHENG, H., LIU, C., AND LIN, L. Snas: stochastic neural architecture search. *arXiv preprint arXiv:1812.09926* (2018).

[40] YANG, X.-S. *Nature-inspired metaheuristic algorithms*. Luniver press, 2010.

[41] YANG, X.-S. A new metaheuristic bat-inspired algorithm. In *Nature inspired cooperative strategies for optimization (NICSO 2010)*. Springer, 2010, pp. 65–74.

[42] YU, C.-C., AND LIU, B.-D. A backpropagation algorithm with adaptive learning rate and momentum coefficient. In *Proceedings of the 2002 International Joint Conference on Neural Networks. IJCNN'02 (Cat. No.02CH37290)* (2002), vol. 2, pp. 1218–1223 vol.2.

[43] ZOPH, B., AND LE, Q. V. Neural architecture search with reinforcement learning. *arXiv preprint arXiv:1611.01578* (2016).

[44] ÖRKCÜ, H. H., AND BAL, H. Comparing performances of backpropagation and genetic algorithms in the data classification. *Expert Systems with Applications 38*, 4 (2011), 3703–3709.