Continuous Ant-Based Neural Topology Search*

AbdElRahman ElSaid, Joshua Karns, Zimeng Lyu,
Alexander G. Ororbia, and Travis Desell

Rochester Institute of Technology, Rochester, NY 14623, USA
aae8800@rit.edu, josh@mail.rit.edu, zimenglyu@mail.rit.edu,
ago@cs.rit.edu, tjdvse@rit.edu

Abstract. This work introduces a novel, nature-inspired neural architecture search (NAS) algorithm based on ant colony optimization, Continuous Ant-based Neural Topology Search (CANTS), which utilizes synthetic ants that move over a continuous search space based on the density and distribution of pheromones, strongly inspired by how ants move in the real world. The paths taken by the ant agents through the search space are utilized to construct artificial neural networks (ANNs). This continuous search space allows CANTS to automate the design of ANNs of any size, removing a key limitation inherent to many current NAS algorithms that must operate within structures of a size predetermined by the user. CANTS employs a distributed asynchronous strategy which allows it to scale to large-scale high performance computing resources, works with a variety of recurrent memory cell structures, and uses of a communal weight sharing strategy to reduce training time. The proposed procedure is evaluated on three real-world, time series prediction problems in the field of power systems and compared to two state-of-the-art algorithms. Results show that CANTS is able to provide improved or competitive results on all of these problems while also being easier to use, requiring half the number of user-specified hyper-parameters.

Keywords: Ant Colony Optimization · Artificial Neural Network · Neural Architecture Search

1 Introduction

Manually optimizing artificial neural network (ANN) structures has been an obstacle to the advancement of machine learning given that it is significantly time-consuming and requires a considerable level of domain expertise [1]. ANN structures are typically chosen based their reputation in existent literature or based on knowledge shared across the machine learning community. However, changing even a few problem-specific meta-parameters can lead to poor generalization upon committing to a specific topology [2, 3]. To address these challenges, a number of neural architecture search (NAS) [1,4–8] and neuroevolution

* This material is based upon work supported by the U.S. Department of Energy, Office of Science, Office of Advanced Combustion Systems under Award Number #FE0031547 and by the Federal Aviation Administration and MITRE Corporation under the National General Aviation Flight Information Database (NGAFID) award.

(NE) [9, 10] algorithms have been developed to automate the process of ANN design. More recently, nature-inspired neural architecture search (NI-NAS) algorithms have shown increasing promise, including the Artificial Bee Colony (ABC) [11], Bat [12], Firefly [13], and Cuckoo Search [14] algorithms.

Ant colony optimization (ACO) [15] is another successful NI-NAS strategy that has been shown to be particularly powerful when automating the design of recurrent neural networks (RNNs). Originally, ACO for NAS was limited to small structures based on Jordan and Elman RNNs [16] or was used as a process for reducing the number of network inputs [17]. Later work proposed generalizations of ACO for optimizing the synaptic connections within RNN memory cell structures [18] and even entire RNN architectures in an algorithmic framework called Ant-based Neural Topology Search (ANTS) [19]. In the ANTS process, ants traverse a single massively-connected "superstructure", searching for optimal RNN sub-networks which connect RNN nodes both in terms of structure, *i.e.*, all possible feed forward connections, and in time, *i.e.*, all possible recurrent synapses that span many different time delays. This approach shares similarity to NAS methods in ANN cell and architecture design [5–7, 20–24], which operate within a limited search space, generating cells or architectures with a pre-determined maximum number of nodes and edges [4].

Most NE methods, instead of operating within fixed bounds, are constructive, continually adding and removing nodes and edges during the evolutionary process (*e.g.*, NEAT [25], CoDeepNEAT [26] and EXAMM [27]). Other strategies involve generative encoding, such as HyperNEAT [28], where a generative network is evolved, which can then be used to create architectures and assign values to their synaptic weights. Nonetheless, these approaches still require manually specifying or constraining the size or scale of the generated architecture in terms of the number of layers and nodes.

In general, constructive NAS methods often suffer from getting stuck in (early) local minima or take considerable computation time to evolve structures that are sufficiently large in order to effectively address the task at hand, especially for large-scale deep learning problems. Alternately, having to prespecify bounds for the space of possible NAS-selected architectures can lead to poorly performing or suboptimal networks if the bounds are incorrect, requiring many runs of varying bound values. In order to address these challenges, this work introduces the novel ACO-inspired algorithm, *Continuous Ant-based Neural Topology Search (CANTS)*, which utilizes a continuous search domain that flexibly allows for the design of ANNs of any size. Synthetic continuous ant (*cant*) agents move through this search space based on the density and distribution of pheromone signals, which emulates how ants swarm in the real world, and the paths resulting from their exploration are used to construct RNN architectures. CANTS 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. The procedure further allows for the selection of recurrent

nodes from a suite of simple neurons and complex memory cells used in modern RNNs: Δ-RNN units [29], GRUs [30], LSTMs [31], MGUs [32], UGRNNs [33].

CANTS is compared to state-of-the-art benchmark algorithms used in designing RNNs for time series data prediction: ANTS [19] and EXAMM [27]. In addition to eliminating the requirement for pre-specified architecture bounds, CANTS is shown to yield results that improve upon or are competitive to ANTS and EXAMM while reducing the number of user specified hyperparameters from 16 in both EXAMM and ANTS down to 8 in CANTS. CANTS also provides an advancement to the field of ant colony optimization as it is the first algorithm capable of optimizing complex graph structures without requiring a predefined (super)structure to operate within. While ACO has been applied to continuous domain problems before [34–38], to the authors' knowledge, our algorithm is the first to simulate and apply the movements of ants through a continuous space to design unbounded graph structures.

2 Methodology

The CANTS procedure (see high-level pseudo-code in Algorithm 1) employs an asynchronous, distributed "work-stealing" strategy to allow for scalable execution on HPC systems. The work generation process maintains a population of the best-found RNN architectures and repeatedly generates candidate RNNs whenever the worker processes request them. This strategy allows workers to complete the training of the generated RNNs at whatever speed they are capable of, yielding an algorithm that is naturally load-balanced. Unlike synchronous parallel evolutionary strategies, CANTS scales up to any number of available processors, supporting population sizes that are independent of processor availability. When the resulting fitness (mean squared error over validation data) of candidate RNNs is reported to the work generator process, if the candidate RNN is better than the worst RNN in the population, then the worst RNN is removed and the candidate is added. Note that the saved pheromone placement points for the candidate are incremented in the continuous search space.

Candidate RNNs are synthesized using a search space of stacked 2D continuous planes, where each 2D plane represents a particular time step t (see Figure 1a). The input nodes for each time step are uniformly distributed at the input edge of the search space. A synthetic continuous ant agent (or *cant*) picks one of the discrete input node positions to start at and then moves through the continuous space based on the current density and distribution of other pheromone placements. Cants are allowed to move forward on the level they are on and can move up to any plane above it. They are restricted from moving down the stack – while connections moving up the stack represent passing information from a previous time step to a future time step, the reverse would require passing unknown future data to a previous time step of the RNN which is not possible. While ants only move forward on a given plane, they are permitted to move backward when moving to a plane higher on the stack since many RNNs have recurrent connections that feed into earlier nodes in the network. This enforced

Algorithm 1. Continuous Ant-guided Neural Topology Search Algorithm

procedure *WorkGenerator*
▷ Construct search space with inputs at y=0 and output at y=1
▷ Recurrent time steps is the spaces's z axis
$search\_space$ = **new** *SearchSpace*
for $i \leftarrow 1 \dots max\_iteration$ **do**
$nn_{new} \leftarrow AntsSwarm()$
$send\_to\_worker(nn_{new}, worker.id)$
$nn_{new}, fit \leftarrow receive\_fit\_from\_worker()$
if $nn\_fitness < worst\_population\_member$ **then**
$population.pop(worst\_population\_member)$
$population.add(nn_{new})$
$RewardPoints(nn_{new})$
procedure *Worker*
$receive\_from\_master(nn)$
$fitness \leftarrow train\_test\_nn(nn)$
$send\_fitness\_to\_master(nn, fitness)$
procedure *AntsSwarm*
▷ Ants choose input in discrete fashion
for $ant \leftarrow 1 \dots no\_ants$ **do**
$CreatePath(ant)$
▷ Use DBscan to cluster ants paths points
$segments \leftarrow DBscanPaths(ants)$
▷ Create RNN from segments
$rnn_{new} \leftarrow CreateRNN(segments)$ **return** rnn_{new}
procedure $CreatePath(ant)$
▷ Choose input in discrete fashion
$ChooseInput(ant)$
▷ Create a path starting from the input
while $ant.current\_y < 0.99$ **do**
$r \leftarrow$ **uniform_random**$(0, pheromone\_sum - 1)$
$ant.current\_level \leftarrow ant.climb$
if $r > ant.exploration\_instinct$ or $search\_space[ant.current\_level]$ is not *Empty* **then**
$point \leftarrow CreateNewPoint(ant.search\_radius)$
$ant.path.insert(point)$
$search\_space.insert(point)$
else
$point \leftarrow FindCenterOfMass(ant.current\_position, ant.search\_radius)$
if *point* not in $search\_space[ant.level]$ **then**
$ant\_path.insert(point)$
▷ Choose Output in discrete fashion
$ChooseOutput(ant)$

```
procedure ChooseInput(ant)
    ▷ Select input probabilistically according to pheromones
    pheromone_sum ← sum(pheromones.input)
    r ← uniform_random(0, pheromone_sum − 1)
    ant.input ← 0
    while r > 0 do:
        if r < pheromones.input[ant.input] then
            ant.input ← 1
            break
        else
            r ← r − pheromones.input[ant.input]
            ant.input ← ant.input + 1
procedure ChooseOutput(ant)
    ▷ Select input probabilistically according to pheromones
    pheromone_sum ← sum(pheromones.output)
    r ← uniform_random(0, pheromone_sum − 1)
    ant.input ← 0
    while r > 0 do:
        if r < pheromones.input[ant.output] then
            ant.output ← 1
            break
        else
            r ← r − pheromones.output[ant.output]
            ant.output ← ant.output + 1
procedure DBscanPaths(ants)
    for ant ← 1 . . . num_ants do
        for point ← 1 . . . ant_path do
            segments[ant].insert(PickPoint(point))
    return segments
procedure PickPoint(point)
    [node, points_cluster] ← DBscane(point, search_space[point.level])
    node.out_edges_weights.insert(AvrgWeights(points_cluster))
    search_space.insert(node) return node
procedure RewardPoints(rnn)
    for each node ∈ rnn.nodes do
        search_space[node].pheromone += constant
        search_space[node].weight ← average_weight(node.weight, search_space[node].weight)
        if search_space[node].pheromone > PHEROMONE_THRESHOLD
then
            search_space[node].pheromone = PHEROMONE_THRESHOLD
```

upward and (overall) forward movement ensures that cants continue to progress towards outputs and do not needlessly circle around in the search space. Figures 1 shows examples of how cants move from an input edge of the search space to the output edge, how cants explore new regions in the search space, how cants exploit previously searched areas via attraction to deposited pheromones, and how cant paths through the space are translated into a final candidate RNN.

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 level for the current time step is $l = 1$, the next level for the first time lag is $l = 2$ and so on. A cant selects its starting level according to the probability of starting at level l as $P(l) = \frac{p_l}{\Sigma_{l=1}^{L} p_l}$, where L is the total number of levels. This scheme encourages cants to start at lower levels of the stack at the beginning of the search. After selecting a level, the cant selects its input node in a similar fashion, based on the pheromones for each input node location on that level. When a candidate RNN is inserted into the population, the level pheromones for each level, utilized by that RNN, are incremented.

Cant Agent Movement: To balance exploration with exploitation, cants behave similarly to real-world ants by following communication clues to reach to targets. When a cant moves, it first decides if it will climb up to a higher (stack) level. This is done in the same manner as selecting its initial layer, except that it only selects between its current level and higher ones. After deciding if it will climb or not, the agent will then decide if it will explore or exploit. Cants randomly choose to exploit at a percentage equal to an exploitation parameter, ϵ.

When a cant decides to exploit and follow pheromone traces, *i.e.*, clues, it will start sensing the pheromone points around it, given a sensing radius, ρ. If the cant is staying on the same level, it will only consider deposited pheromones that are in front of it (*i.e.*, closer to the output nodes), otherwise, it will consider all the pheromones that are inside its sensing radius on the level it is moving to. The cant then calculates the center of mass of the pheromones in this region using the point in the space it will move to. This point is then saved by the candidate RNN (as a point to potentially increment pheromone values) if the RNN is later to be inserted into the RNN population. Since cants consider the center of mass of the pheromone values, the individual points of pheromone values are not the effective factor in cant-to-cant communication. Rather, it is the concentration of the pheromone in a region of the space that more closely aligns with how real ants move in nature.

When a cant instead decides that it will explore, it instead selects a random point that lies within the range of their sensing radius to move to. Once a cant decides if it is climbing or staying in the same level, it will generate an angle bisector that is either a random number between $[0, 1]$ if the current and next point are on the same level or $[-1, 1]$ if the current and next points are on different levels. This angle bisector is used to calculate the angle of the next movement of the cant: $\theta = angle\_bisect * PI$. The movement angle is then subsequently used to calculate the next x and y coordinates of the next position

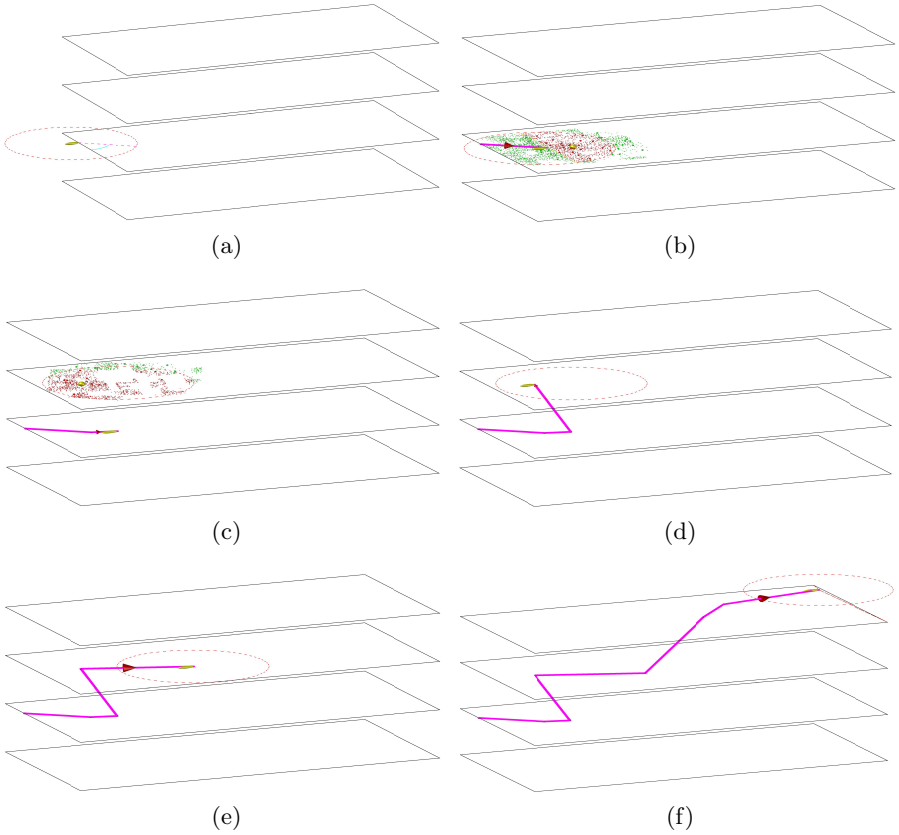
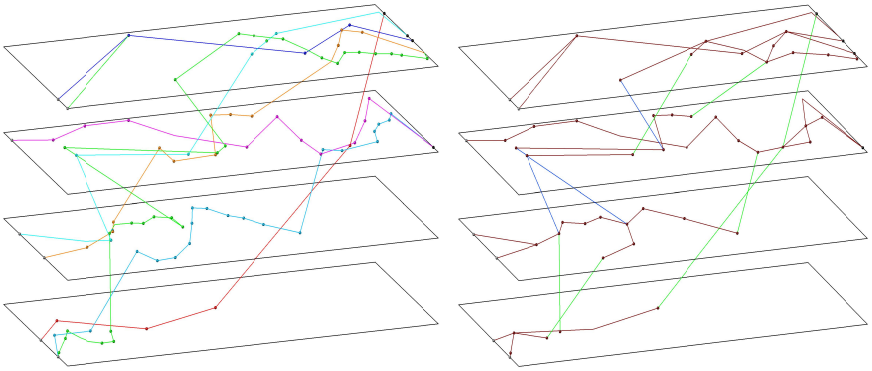
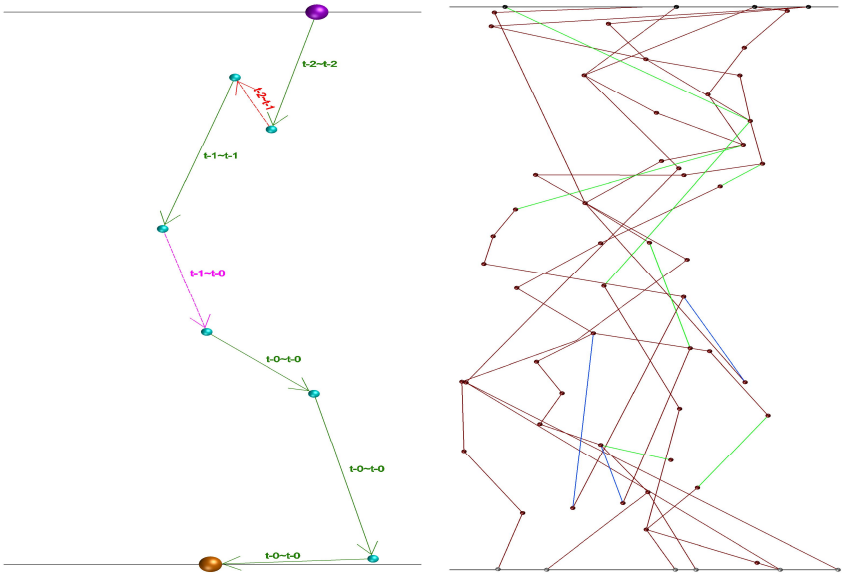


Fig. 1. Cant path selection and network construction: (a) After an cant picks a layer to start with and an input node, it decides if it will move to a new random point (exploration), or follow pheromone traces (exploitation). If the the former, the cant will randomly pick a forward angle between 0° and 180° and move in that direction equal to its red sensing radius. (b) When the cant wants to use pheromone traces to determine its new point, it will first sense the the pheromone traces within its sensing radius. The example cant did not change its layer, so the cant will only consider the pheromone traces in front of it and not move backwards. The ant will then calculate the center of mass of the pheromone traces within its sensing radius and then move to the center of their mass (gold sphere). (c) When the cant moves to a level above it and decides that it will use exploitation, it will consider the pheromone traces in its sensing range in all directions, which lie between the angles 0° and 360°. This way, the cant can move backwards when jumping from a layer to another, which makes a recurrent connection that goes back between hidden layers. (d) The cant moves upward to the higher level. (e) The cant will move to a new point by exploration. (f) After a series of upward and forward moves by either exploration or exploitation, when the cant has output nodes within its sensing radius, it will stop the continuous search and select an output node based on its discrete pheromone values. If there is only one output node, then the cant will directly connect its last point to the output.



(g) (h)

Fig. 1. Cant path selection and network construction (continued): (g) Several cants make their path from an input to an output. (h) The cants' nodes on each level are then condensed (clustered) based on their density using DBSCAN.



(i) (j)

Fig. 1. Cant path selection and network construction (continued): (i) The cant picked its input point, starting at level t_{-2}, picked a node at t_{-2} (green edge), picked a node at t_{-1} (red backward recurrent edge), picked a node at t_{-1} (green edge), picked a node at t_0 (magenta forward recurrent edge), picked a node at t_0 (green edge), picked a node at t_0 (green edge), and finally picked an output node at t_0 (green edge). (j) The final network is the final result of clustering the nodes and defining the connections between nodes in the same layer as red edges, and the connection between nodes and between layers as green forward recurrent edges or blue backward recurrent edges. The flow moves from the gray inputs at the bottom to the black outputs at the top.

of the cant: $x_{new} \leftarrow x_{old} + \rho * cos(\theta)$, $y_{new} \leftarrow y_{old} + \rho * sin(\theta)$. These points are also saved for potential future pheromone modification.

Condensing Cant Paths to RNN Nodes: After cants choose the points in their paths from the inputs to the outputs, the points in the search space are clustered using the DBSCAN algorithm [39] to condense those points to centroids. The points of the segments of the cants' paths are then shifted to the centroids that they belong to in the search space and those new points become the nodes of the generated RNN architecture (see Figures 1g and 1h). The node types are picked by a pheromone-based discrete local search, as is done in the discrete space ANTS. Each of these node types at the selected point will have their own pheromone values that drive probabilistic selection.

Communal Weight Sharing: In order to avoid having to retrain every newly-generated RNN from scratch, a communal weight sharing method has been implemented to allow generated RNNs to start with values similar to those of previously generated and trained RNNs. The centroid points (*i.e.*, the RNN node points in the continuous space) in CANTS retain the weights of all the outgoing edges from those nodes. Each newly-created centroid is assigned a weight value which is passed to the edges of the generated RNN. In the case where a centroid did not have any previously created centroid in its cluster, randomly initialized weights are assigned to those outgoing edges either uniformly at random between -0.5 and 0.5, or via the Kaiming [40] or Xavier [41] strategies. If there were previously-created centroids in the clustering region, the weight values assigned to the generated RNN nodes are the average of the weights of those existing centroids. The weights of a centroid are updated after an RNN is trained by calculating the averages of the original centroid weight values and all the weights of the outgoing edges of the corresponding node (after training). The updated weights can then be used to initialize new centroid weights when they lie in their cluster when DBSCAN is applied in the following iteration.

Pheromone Volatility: Pheromone decay happens on a regular basis after each iteration of optimization regardless of the performance of the generated RNN(s). The pheromones decay by a constant value and after a specific minimum threshold the point is removed from the search space. By letting points vanish, the search space removes tiny residual pheromones which might provide distraction to cant-to-cant communication as well as slow down the overall algorithm.

Pheromone Incrementation: For each successful candidate RNN, *i.e.*, each RNN that performs at least better than the worst in the population, the corresponding centroids for its RNN nodes in the search space are rewarded by increasing their pheromone values by a constant value. The values of the pheromones have a maximum limit to avoid becoming overly attractive points to the cants, which could result in premature convergence.

3 Results

This work compares CANTS to the state-of-the-art ANTS and EXAMM algorithms on three real world datasets related to power systems. All three methods were used to perform time series data prediction for different parameters, which have been used as benchmarks in prior work. Main flame intensity was used as the prediction parameter from the coal plant's burner, net plant heat rate was used from the coal plant's boiler, and average power output was used from the wind turbines. Experiments were also performed to investigate the effect of CANTS hyper-parameters: the number of cants and cant sensing radii, ϵ.

Computing Environment The results for ANTS, CANTS, and EXAMM 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). Each ANTS experiment utilized 15 nodes (540 cores), taking approximately 30 days to complete all the experiments. CANTS experiments used 5 nodes (180 cores), taking 7 days to finish all the experiments. EXAMM experiments also used 5 nodes (180 cores) and also took approximately 7 days to complete the experiments.

Datasets The datasets used, which are derived from coal-fired power plant and wind turbine data, have been previously made publicly available on the EXAMM repository to encourage further study in time series data prediction and reproducibility[1]. The first dataset comes from measurements collected from 12 burners of a coal-fired power plant as well as its boiler parameters and the second dataset comes from wind turbine engine data from the years 2013 to 2020, collected and made available by ENGIE's La Haute Borne open data windfarm[2].

All of the datasets are multivariate and non-seasonal, with 12 (burner), 48 (boiler), and 78 (wind turbine) input variables (potentially dependent). These time series are very long, with the burner data separated into 7000 time step chunks – one for training and one for testing (per minute recordings). The boiler dataset is separated into a training set of 850 steps and test set of 211 steps (per hour recordings). The wind turbine dataset is separated into a training set of $190,974$ steps and test set of $37,514$ steps (each step taken every 10 minutes).

3.1 Number of Cant Agents

An experiment was conducted to determine the effect that the number of cant agents has on the performance of CANTS. The experiment focused on the net plant heat rate feature from the coal-fired power plant dataset. The number of ants evaluated were 10, 30, 60, 100, 150, and 210. The results, shown in Figure 2, show that, as the number of cants are increased, the performance increases until

[1] https://github.com/travisdesell/exact/tree/master/datasets/
[2] https://opendata-renewables.engie.com

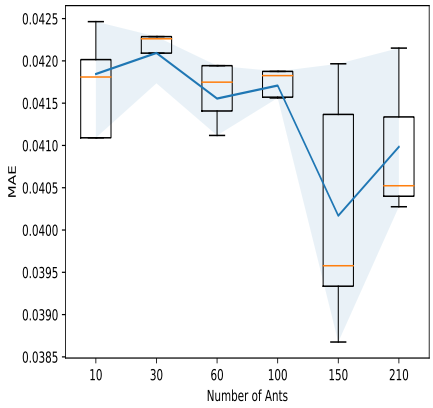


Fig. 2. CANTS w/ varying # of agents.

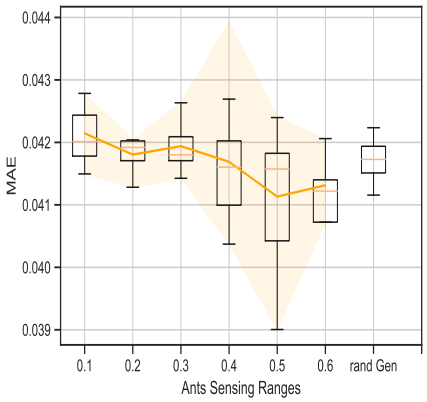


Fig. 3. CANTS w/ different sensing radii.

150 cants are used and then a decline is observed. This shows that the number of cant agents is an important hyper-parameter and requires tuning, potentially exhibiting "sweet spots" that, if uncovered, provide strong results.

3.2 Cant Agent Sensing Radius

We next investigated the effect that the sensing radii (range) of the cant agents had on algorithm performance. Figure 3 shows that a sensing radius of 0.5 obtained better performance compared to the 0.1, 0.2, 0.3, 0.4, and 0.6 sensing radii values tested. We also evaluated the effect that using a randomly generated sensing radius per cant agent would have. For these, ϵ was randomly initialized (uniformly) via $\sim U(0.01, 0.98)$. Ultimately, we discovered that the sensing radius of 0.5 still provided the best results.

3.3 Algorithm Benchmark Comparisons

To compare the three different NAS strategies, each experiment was repeated 10 times (trials) for statistical comparison and all algorithms were set to generate 2000 RNNs per trial. For 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, initial pheromone values were 1 and the maximum was kept at 10 with a pheromone decay rate set to 0.05. For the DBSCAN module, clustering distance was 0.05 with a minimum point value of 2 – runs with these settings were done using 30 and 150 ants. CANTS and ANTS used a population of size 20 while EXAMM used 4 islands, each with a population of 10. ANTS, CANTS, and EXAMM all had a maximum recurrent depth of 5 and the predictions were made over a forecasting horizon of 1. The generated RNNs were each allowed 40 epochs of back-propagation for local fine-tuning (since all algorithms

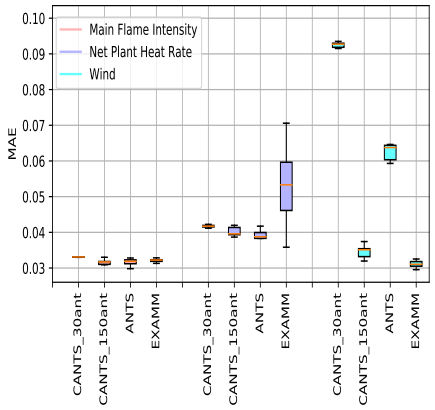


Fig. 4. Mean Average Error (MAE) ranges of best-found RNNs from each method.

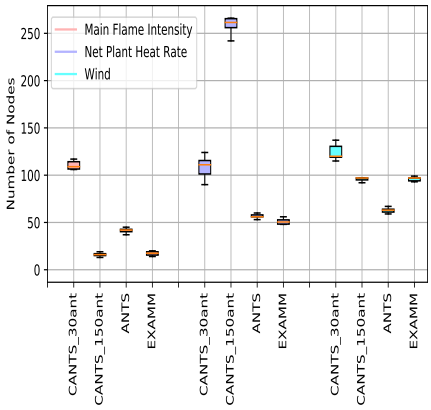


Fig. 5. Number of nodes in the best found RNNs from each method.

are mmetic). ANTS and EXAMM utilized the hyper-parameters previously reported to yield best results [19, 27].

The results shown in Figure 4, which compare CANTS, ANTS and EXAMM in the three experiments described above (over the three datasets), report the range of mean average error (MAE) of each algorithm's best-found RNNs. While EXAMM outperformed CANTS with 30 ants, CANTS with 150 ants had a better performance than EXAMM and ANTS. CANTS was competitive with ANTS on the net plant heat rate predictions and outperformed EXAMM on this dataset. CANTS also outperformed ANTS on the wind energy dataset yet could not beat EXAMM. Potential reasons for this could be that the complexity/size of this dataset is greater and that the task is simply more difficult which results in a potentially larger search space. As CANTS allows for potentially unbounded network sizes, its search space is significantly larger than either that of ANTS or EXANM. Though ANTS outperformed CANTS on the wind dataset, CANTS is still a good competitor, especially since it has less hyper-parameters (8) to tune compared to both ANTS and EXAMM (both require at least 16). While all these reasons may be valid, the size of the search space is likely the biggest challenge. Further evidence of this is provided in Figures 5, 6, 7, present the number of structural elements (nodes, edges, and recurrent edges, respectively) of the best-found RNN architectures using the different algorithms. The CANTS runs with 150 ants resulted in significantly more complex architectures for many of the problems, which may be an indication that CANTS can evolve better performing structure if provided more optimization iterations.

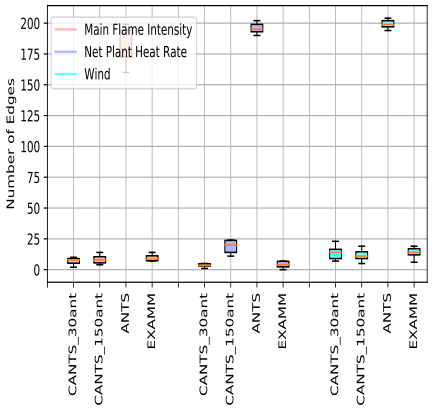


Fig. 6. Number of edges in the best-found RNNs from each algorithm.

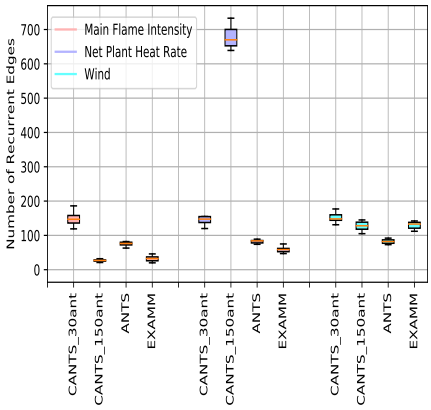


Fig. 7. Number of recurrent edges in the best-found RNNs each algorithm.

4 Discussion and Future Work

This work introduces continuous ant-based neural topology search (CANTS), a novel nature-inspired optimization algorithm that utilizes a continuous search space to conduct unbounded neural architecture search (NAS). This approach provides a unique strategy to overcome key limitations of constructive neuro-evolutionary strategies (which often prematurely get stuck at finding smaller, less performant architectures) as well as other neural architecture search strategies that require users to carefully specify the bounds limiting the neural architecture size. CANTS was experimentally evaluated for the automated design of recurrent neural networks (RNNs) to make time series predictions across three challenging real-world data sets in the power systems domain. We compared it to two state-of-the-art algorithms, ANTS (a discrete space ant colony NAS algorithm) and EXAMM (a constructive neuro-evolution algorithm). CANTS is shown to improve on or be competitive with these strategies, while also being simpler to use and tune, only requiring 8 hyper-parameters as opposed to the 16 hyper-parameters of the other two strategies.

This study presents some initial work generalizing ant colony algorithms to complex, continuous search spaces, specifically for unbounded graph optimization problems (with NAS as a target application), opening up a number of promising avenues for future work. In particular, while the search space is continuous in each two-dimensional plane (or time step) of our temporal stack, there is still the number of discrete levels that a user must specify. Therefore, a promising extension of the algorithm would be to make the search space continuous across all three dimensions, removing this parameter, and allowing pheromone placements to guide the depth of recurrent connections. This could have implications for discrete-event, continuous-time RNN models [42], which attempt to

tackle a broader, more complex set of sequence modeling problems. Finally, and potentially the most interesting, is the fact that the exploitation parameter, ϵ, and the sensing radius, ρ, for each synthetic ant agent in our algorithm was held fixed (or in some cases randomly initialized) for the duration of each CANTS search. However, the ants could instead be treated as complex agents that evolve with time, learning the best exploitation and sensing parameters for the task search spaces they are applied to. This could provide far greater flexibility to the CANTS framework. Expanding this algorithm to other domains, such as the automated design of convolutional neural networks (for computer vision) or to other types of RNNs, such as those used for natural language processing, could further demonstrate the potentially broad applicability of this nature-inspired approach.

Acknowledgements

Most of the computation of this research was done on the high performance computing clusters of Research Computing at Rochester Institute of Technology [43]. We would like to thank the Research Computing team for their assistance and the support they generously offered to ensure that the heavy computation this study required was available.

References

1. Zoph, B., Le, Q.V.: Neural architecture search with reinforcement learning. arXiv preprint arXiv:1611.01578 (2016)
2. Erkaymaz, O., Özer, M., 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), 708–718 (2014)
3. Barna, G., Kaski, K.: Choosing optimal network structure, pp. 890–893. Springer Netherlands, Dordrecht (1990), https://doi.org/10.1007/978-94-009-0643-3_122
4. Elsken, T., Metzen, J.H., Hutter, F.: Neural architecture search: A survey. arXiv preprint arXiv:1808.05377 (2018)
5. Liu, H., Simonyan, K., Yang, Y.: Darts: Differentiable architecture search. arXiv preprint arXiv:1806.09055 (2018)
6. Pham, H., Guan, M.Y., Zoph, B., Le, Q.V., Dean, J.: Efficient neural architecture search via parameter sharing. arXiv preprint arXiv:1802.03268 (2018)
7. Xie, S., Zheng, H., Liu, C., Lin, L.: Snas: stochastic neural architecture search. arXiv preprint arXiv:1812.09926 (2018)
8. Luo, R., Tian, F., Qin, T., Chen, E., Liu, T.Y.: Neural architecture optimization. In: Advances in neural information processing systems. pp. 7816–7827 (2018)
9. Stanley, K.O., Clune, J., Lehman, J., Miikkulainen, R.: Designing neural networks through neuroevolution. Nature Machine Intelligence 1(1), 24–35 (2019)
10. Darwish, A., Hassanien, A.E., Das, S.: A survey of swarm and evolutionary computing approaches for deep learning. Artificial Intelligence Review 53(3), 1767–1812 (2020)

11. Horng, M.H.: Fine-tuning parameters of deep belief networks using artificial bee colony algorithm. DEStech Transactions on Computer Science and Engineering (2017)
12. Yang, X.S.: A new metaheuristic bat-inspired algorithm. In: Nature inspired cooperative strategies for optimization (NICSO 2010), pp. 65–74. Springer (2010)
13. Yang, X.S.: Nature-inspired metaheuristic algorithms. Luniver press (2010)
14. Leke, C., Ndjiongue, A.R., Twala, B., Marwala, T.: A deep learning-cuckoo search method for missing data estimation in high-dimensional datasets. In: International Conference on Swarm Intelligence. pp. 561–572. Springer (2017)
15. Dorigo, M., Maniezzo, V., Colorni, A.: Ant system: optimization by a colony of cooperating agents. IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics) 26(1), 29–41 (1996)
16. Desell, T., Clachar, S., Higgins, J., Wild, B.: Evolving deep recurrent neural networks using ant colony optimization. In: Ochoa, G., Chicano, F. (eds.) Evolutionary Computation in Combinatorial Optimization. pp. 86–98. Springer International Publishing, Cham (2015)
17. Mavrovouniotis, M., Yang, S.: Evolving neural networks using ant colony optimization with pheromone trail limits. In: Computational Intelligence (UKCI), 2013 13th UK Workshop on. pp. 16–23. IEEE (2013)
18. ElSaid, A., El Jamiy, F., Higgins, J., Wild, B., Desell, T.: Optimizing long short-term memory recurrent neural networks using ant colony optimization to predict turbine engine vibration. Applied Soft Computing 73, 969–991 (2018)
19. ElSaid, A., Ororbia, A.G., 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). pp. 626–641. Springer (2020)
20. Cai, H., Zhu, L., Han, S.: Proxylessnas: Direct neural architecture search on target task and hardware. arXiv preprint arXiv:1812.00332 (2018)
21. Guo, Z., Zhang, X., Mu, H., Heng, W., Liu, Z., Wei, Y., Sun, J.: Single path one-shot neural architecture search with uniform sampling. In: European Conference on Computer Vision. pp. 544–560. Springer (2020)
22. Bender, G., Kindermans, P.J., Zoph, B., Vasudevan, V., Le, Q.: Understanding and simplifying one-shot architecture search. In: International Conference on Machine Learning. pp. 550–559 (2018)
23. Dong, X., Yang, Y.: One-shot neural architecture search via self-evaluated template network. In: Proceedings of the IEEE International Conference on Computer Vision. pp. 3681–3690 (2019)
24. Zhao, Y., Wang, L., Tian, Y., Fonseca, R., Guo, T.: Few-shot neural architecture search. arXiv preprint arXiv:2006.06863 (2020)
25. Stanley, K.O., Miikkulainen, R.: Evolving neural networks through augmenting topologies. Evolutionary computation 10(2), 99–127 (2002)
26. 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, pp. 293–312. Elsevier (2019)
27. Ororbia, A., ElSaid, A., Desell, T.: Investigating recurrent neural network memory structures using neuro-evolution. In: Proceedings of the Genetic and Evolutionary Computation Conference. pp. 446–455. GECCO '19, ACM, New York, NY, USA (2019), http://doi.acm.org/10.1145/3321707.3321795
28. Stanley, K.O., D'Ambrosio, D.B., Gauci, J.: A hypercube-based encoding for evolving large-scale neural networks. Artificial life 15(2), 185–212 (2009)

29. Ororbia II, A.G., Mikolov, T., Reitter, D.: Learning simpler language models with the differential state framework. Neural Computation 0(0), 1–26 (2017), https://doi.org/10.1162/neco_a_01017, pMID: 28957029
30. Chung, J., Gulcehre, C., Cho, K., Bengio, Y.: Empirical evaluation of gated recurrent neural networks on sequence modeling. arXiv preprint arXiv:1412.3555 (2014)
31. Hochreiter, S., Schmidhuber, J.: Long short-term memory. Neural computation 9(8), 1735–1780 (1997)
32. Zhou, G.B., Wu, J., Zhang, C.L., Zhou, Z.H.: Minimal gated unit for recurrent neural networks. International Journal of Automation and Computing 13(3), 226–234 (2016)
33. Collins, J., Sohl-Dickstein, J., Sussillo, D.: Capacity and trainability in recurrent neural networks. arXiv preprint arXiv:1611.09913 (2016)
34. Socha, K., Dorigo, M.: Ant colony optimization for continuous domains. European journal of operational research 185(3), 1155–1173 (2008)
35. Kuhn, L.D.: Ant colony optimization for continuous spaces. Computer Science and Computer Engineering Undergraduate Honors Theses (35) (2002)
36. Xiao, J., Li, L.: A hybrid ant colony optimization for continuous domains. Expert Systems with Applications 38(9), 11072–11077 (2011)
37. Gupta, H., Ghosh, B.: Transistor size optimization in digital circuits using ant colony optimization for continuous domain. International Journal of Circuit Theory and Applications 42(6), 642–658 (2014)
38. Bilchev, G., Parmee, I.C.: The ant colony metaphor for searching continuous design spaces. In: AISB workshop on evolutionary computing. pp. 25–39. Springer (1995)
39. 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. vol. 96, pp. 226–231 (1996)
40. He, K., Zhang, X., Ren, S., Sun, J.: Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In: Proceedings of the IEEE international conference on computer vision. pp. 1026–1034 (2015)
41. Glorot, X., Bengio, Y.: Understanding the difficulty of training deep feedforward neural networks. In: Proceedings of the thirteenth international conference on artificial intelligence and statistics. pp. 249–256 (2010)
42. Mozer, M.C., Kazakov, D., Lindsey, R.V.: Discrete event, continuous time rnns. arXiv preprint arXiv:1710.04110 (2017)
43. Rochester Institute of Technology: Research computing services (2019), https://www.rit.edu/researchcomputing/