Ant-based Neural Topology Search (ANTS) for Optimizing Recurrent Networks

AbdElRahman ElSaid and Alexander G. Ororbia** and Travis J. Desell**

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

Abstract. Hand-crafting effective and efficient structures for recurrent neural networks (RNNs) is a difficult, expensive, and time-consuming process. To address this challenge, we propose a novel neuro-evolution algorithm based on ant colony optimization (ACO), called Ant-based Neural Topology Search (ANTS), for directly optimizing RNN topologies. The procedure selects from multiple modern recurrent cell types such as Δ-RNN, GRU, LSTM, MGU and UGRNN cells, as well as recurrent connections which may span multiple layers and/or steps of time. In order to introduce an inductive bias that encourages the formation of sparser synaptic connectivity patterns, we investigate several variations of the core algorithm. We do so primarily by formulating different functions that drive the underlying pheromone simulation process (which mimic L1 and L2 regularization in standard machine learning) as well as by introducing ant agents with specialized roles (inspired by how real ant colonies operate), i.e., *explorer ants* that construct the initial feed forward structure and *social ants* which select nodes from the feed forward connections to subsequently craft recurrent memory structures. We also incorporate communal intelligence, where best weights are shared by the ant colony for weight initialization, reducing the number of backpropagation epochs required to locally train candidate RNNs, speeding up the neuro-evolution process. Our results demonstrate that the sparser RNNs evolved by ANTS significantly outperform traditional one and two layer architectures consisting of modern memory cells, as well as the well-known NEAT algorithm. Furthermore, we improve upon prior state-of-the-art results on the time series dataset utilized in our experiments.

Keywords: Ant Colony Optimization · Swarm Intelligence · Neuro-Evolution · Recurrent Neural Networks · Time Series Data Prediction.

1 Introduction

Given their success across a wide swath of pattern recognition tasks, artificial neural networks (ANNs) have become a popular tool to use when attempting to

**Indicates equal advising.

solve data-driven problems. However, in order to solve increasingly more complicated problems, neural architectures are becoming vastly more complex. Increasing the complexity of an ANN entails having to operate with more layers of neural processing elements required, most of which are usually wider and more densely-connected, greatly complicating the model design process. The resulting increase in complexity introduces new challenges and complications when fitting these ANN models to actual data. These problems are further compounded when ANNs are meant to process temporal data, entailing recurrent connections which can span varying periods of time. As a result, crafting performant ANNs becomes expensive and incredibly difficult for engineers, highlighting a grand challenge facing the domain of machine learning – the automation of ANN architecture design, which includes selecting the form of the underlying synaptic topology as well as the values of the weights themselves. The key to this automation might lie in developing optimization procedures that can effectively explore the vast, combinatorial search space of possible topological structures that could be constructed from a large set of neuronal units and the wide variety of synaptic connectivity patterns that relate them to one another.

Recent interest in automated architecture search has resulted in many proposed ideas related to deep feed forward and convolutional networks, including those based on nature-inspired metaheuristics [1]. However, few, if any, have focused on the far more difficult problem of optimizing recurrent neural networks (RNNs) aimed at processing temporal, sequential data such as time series, i.e., automated RNN design.

This study addresses the challenge of automated RNN design by developing a novel ANN topology optimizer based on concepts from artificial evolution and ant colony optimization (ACO). Specifically, we propose an algorithm called Ant-based Neural Topology Search (ANTS), which automatically constructs and optimizes the topology of RNNs, with a focus on time series data prediction. We further experiment with variations of our ANTS method in the following ways:

- In order to encourage the discovery of more sparsely-connected neural topologies, we investigate different schemes for dynamically modifying the pheromone traces deposited by ant agents that compose the swarm. Specifically, we introduce functions for introducing regularization into the overall optimization, slowly clearing out densely-connected synaptic areas by depriving poorly performing weights/edges of pheromone accumulation.
- We incorporate and analyze various weight initialization schemes and find that a strategy incorporating communal intelligence where best found weights are shared by the colony is highly effective.
- Inspired by the role-specialization that ants operate under within the context of real-world ant colonies, we extend ANTS to utilize different specialized ant agents to modularize the underlying synaptic connectivity construction process, which we find greatly improves solutions found by our metaheuristic.

Experimentally, we validate our proposed nature-inspired metaheuristic on an open-access real-world time series data set collected form a coal-fired power

plant. A rigorous ablation study of the ANTS algorithm is conducted by analyzing the candidate network topologies it finds. A total over 1600 experiments with varying heuristics and hyperparameters were performed, which entailed training $32,000,000$ different RNNs. Our results indicate that ANTS is able to build well performing, arbitrary RNN structures with connections that span both structure and time using both simple and complex memory cells. More importantly, ANTS is shown to significantly outperform the well-known neuro-evolutionary algorithm, NEAT [2], as well as the state-of-the-art evolutionary optimizer, EXAMM [3], which held the prior best results on this data set. Our ANTS source code is open source and freely availiable on our GitHub repository*.

2 Related Work

With respect to neuroevolution of recurrent network topologies, a great deal of work already exists, ranging from stochastic alteration of the topology as in dropout [4] to something more sophisticated like that in the original NEAT [2] and its more modern incarnate HyperNEAT [5]. Other proposed approaches include EPNet [6], EANT [7], GeNet[8], CoDeepNEAT [9], and EXACT [10]. EXACT was extended to evolve RNNs that used LSTM memory cells (named EXALT) and shown to perform quite well on time-series prediction problems [11]. Later, the algorithm, retitled EXAMM, was generalized to evolve networks consisting of a library of recurrent memory cells [3]. These previously proposed ideas center around the use of a genetic algorithm [12], where optimization is inspired by approaches that draw from the evolution of organisms, of either Darwinian and/or Lamarckian nature.

Nonetheless, very few studies in the body of work described above consider ant colony optimization (ACO) [13] as the central optimizer for network topology, and even fewer in general focus on exploring how to evolve complex temporal models like the RNN, with EXALT and EXAMM being exceptions. Of the few that have investigated ACO, most existing work has used it to strictly optimize feed forward networks and, even in that case, have dominantly focused on either initializing the weights of the connections [14], or on reducing the dimension of the input vector solution space [15]. One notable effort that has used ACO for RNN optimization in some form is [16], which used ACO to optimize smaller neural network structures based on Elman recurrent networks [17].

This paper contributes to the domain of nature-inspired neural network topology optimization by proposing a novel metaheuristic for evolving the full structure of an RNN as opposed to prior studies that have applied the technique as only a partial component of the optimization process [18] or in smaller Elman RNN topologies with limited recurrent connectivity [16]. Furthermore, our algorithm is capable of utilizing the same full suite of recurrent memory cells as the state-of-the-art evolutionary algorithm EXAMM (LSTM, GRU, MGU, UGRNN, and Δ-RNN cells). To the best of our knowledge, we are the first to

*https://github.com/travisdesell/exact/tree/adding_ant_colony

propose an ACO-based approach to automate RNN design, offering a powerful procedure that combines concepts of both neuro-evolution and ant colony metaheuristic optimization.

3 Ant-based Neural Topology Search (ANTS)

ANTS handles the optimization of ANN structures using a multi-agent system, where each agent (an ant) treats the ANN as graph structure, considering neuronal processing elements (PEs) as the nodes and the synaptic weights that connect PEs as the edges. In order to design the operations that these agents perform as well as the manner in which they traverse the ANN graph, we may appeal to the metaphor of ants and the collective they holistically form, *i.e.*, the ant colony. As a result, the agents will function based on simplifications of myrmecological principles, such as the mechanics of ant-to-ant social interaction.

At a high level, in ANTS, the individual ant agents operate on a single massively connected "superstructure", which contains all possible ways that PEs may connect with each other both in terms of structure, *i.e.*, all possible feed forward pathways that start from the input/sensory PEs and end at the output/actuator PEs, and time, *i.e.*, all possible recurrent connections that span many multiple time delays[†]. In our implementation, ants choose to move over connections between nodes (or neurons), probabilistically and as a function of a simulated chemical known as the "pheromone", which is placed on connections by ants based on how well they have been utilized to generate candidate RNNs.

ANTS was developed as an asynchronous parallel system for use on high performance computing resources, which has a master process that maintains the colony information and worker processes to (locally) train the RNNs. This parallel implementation is asynchronous, the master process generates new RNNs as needed for worker processes (which operate on separate, dedicated CPU or GPU resources) and updates colony information and pheromones as trained RNN results are returned. This results in a naturally load balanced algorithm with high scalability. From the overall superstructure, which the ant agents exclusively operate on, RNN subnetworks are extracted (as dictated by the current pheromone trace network available at the current simulation time step, which yields a map of nodes and connecting synapses, both recurrent and feed forward, visited by the ant agents) and sent to worker processes. The worker processes train the extracteed RNNs locally with only a few epochs of back-propagation through time (BPTT). After a particular worker is done locally training a RNN subnetwork, the candidate's weight values and cost (fitness) function (measure on a validation subset of data) are communicated back to the swarm and superstructure (housed in the master process), adjusting the pheromone trace network and affecting future ant agent traversal behavior.

[†]Note that this superstructure is more connected than a standard fully connected neural network – each layer is also fully connected to each other layer as well, allowing for forward and backward layer skipping connections, with additional recurrent connections between node pairs for each time skip allowed.

Within the framework of ANTS, we investigate variations of its various underlying mechanisms. These include the use of communal intelligence by sharing the best weights found among the colony, allowing ant agents to also select from multiple memory cell types as opposed to operating exclusively with simple neurons, introducing specialized ants that have different graph traversal strategies, and constraining ant movement and manipulating the pheromone evaporation function in order to encourage the discovery of sparse RNN topologies. One particularly crucial element in our ANTS procedure is the introduction of different ant agent types or species, which is inspired by how real ants specialize to act according to specific roles to serve the needs of the colony [19]. Specifically, we consider designing ant agents that serve specific roles in constructing parts of candidate RNN subnetworks – some ants exclusively traverse feed forward synaptic pathways while others only explore recurrent synaptic pathways.

3.1 Communal Weight Sharing

Randomly initializing edges and recurrent edges' weights each time a new RNN is generated requires local tuning (via BPTT) for many epochs for the RNN to reach suitable generalization error, as they do not make use of any information gained by prior trained RNN candidates. It has been shown that the resuse of prior trained weights (*i.e.*, epigenetic or Lamarckian weight initialization) can significantly speed up the neuro-evolution process and result in better performing, smaller ANNs in general [20]. To apply similar use of prior knowledge, ANTS utilizes a communal weight sharing strategy. Each edge in the ant swarm's connectivity super-structure also tracks a weight value in addition to its pheromone value. These weights are randomly initialized uniformly $U(-0.5, 0.5)$. Each time a generated RNN performs well, the weights of its best performance, as measured on a validation data subset, are used to update the shared weight values in the swarm's super-structure.

Formally, we define phi (Φ) as a function of the population's best and worse evaluated RNN fitness, where W_{colony_i} is the colony's edge weight, W_{RNN_i} is the corresponding neural network's edge weight, fit_{pop_best} is the population's best fitness, and fit_{pop_worst} is the population's worst fitness. Weight initialization then proceeds as follows:

$$x = \frac{fit_{new} - fit_{pop\_best}}{fit_{pop\_worst} - fit_{pop\_best}} \tag{1a}$$

$$\Phi(fit_{new}) = min\Bigg(max\Big((1-x), 0\Big), 1\Bigg) \tag{1b}$$

$$W_{colony_i} = \Phi W_{RNN_i} + (1-\Phi)W_{colony_i}. \tag{1c}$$

With respect to the function Φ, we investigated two variations. The first variant, as shown in Equation 1, used the fitness of the RNN used to update the weights to determine how much these new (locally found) weight values effect those of the colony. The second variant of Φ was set to a predetermined constant instead

of being calculated or adjusted by fitness. This process essentially allows for a running average (either with a fixed update or dynamic update based on fitness) of the best weights found for each connection in the superstructure. When a new RNN is generated, it uses the current weight values in whatever edges that were extracted from the superstructure on the master process. This allows for the colony to share information about the best weights found for each connectioin, adapting them in a manner similar to a running average as new best candidate RNNs and weights are found.

3.2 Memory Cell Selection

For any particular node in the super-structure, ANTS also has the ability to utilize the pheromones present to select which memory cell type a particular node will be in the generated network. A node could chosen to be either an LSTM [21], a GRU [22], an MGU [23], a UGRNN [24], or a Δ-RNN cell [25]. We refer the reader to these works for the formulations of these memory cells. Pheromones are deposited and updated for each of these memory cell possibilities as described below.

3.3 Altering Graph Traversal with Ant Species

As mentioned above, we explored various strategies for guiding ant traversal over the connectivity superstructure. Inspired by role specialization in real colonies, we implemented ant agents that explored the connectivity graph in specific ways. First, we started with a generic ant agent, called the *standard ant*, which was allowed to traverse through the massively connected colony superstructure in an unbiased manner. This, in essence, recovers the standard simple ant agent in classic ACO, which has complete freedom to explore any piece of a given graph structure. However, it became quickly apparent that this type of ant would get "stuck" in the network, generating a significantly high number of recurrent connections before finally reaching an output node (explained in Figure 1). This meant that the RNN candidates extracted for local fine-tuning were rather dense, and in turn, compute-heavy (featuring many extraneous parameters as is characteristic of over-parameterized models).

To prevent this problem, our first tactic was to alter the pheromone deposit function by adding extra pheromones to forward paths upon initialization as well as after every pheromone update. If the total number of pheromones on the forward edges out of a node was less than 75% of the total number of pheromones on the recurrent edges out of the node, the pheromones on each forward node were multiplied by the ratio of the total sum of outgoing recurrent edge pheromones over the total sum of the outgoing forward edge pheromones. This biasing method yielded better proportions of forward and backward paths.

Even with this forward path bias added to the pheromone deposit function, when using standard ants, we found that ANTS still tended to favor the generation of fairly dense networks. Altering the number of ant agents used to explore the structure as a means to control density of RNN candidates proved to help

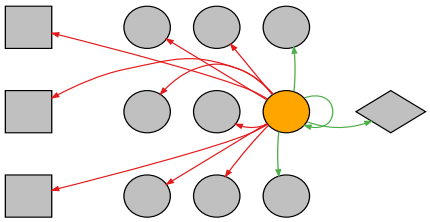

Fig. 1: Potential paths an ant can take from a given node (in orange) with the massively-connected superstructure. The number of recurrent paths (red) far outnumber the forward paths (green). This problem is exacerbated as the possible recurrent time scale increases, which results in multiple backward recurrent connections for each red connection, each going back a different number of time steps in the past.

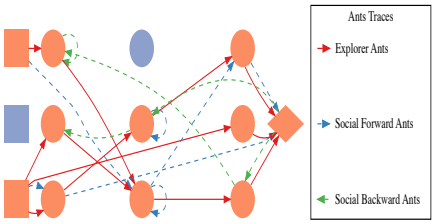

Fig. 2: In multi-role traversal, explorer ants (red) first select the forward paths in the network, creating a basic structure for the RNN. The social ant agents then select from the nodes chosen by the explorer ants with forward recurrent ants (blue) creating additional forward recurrent connections and backward recurrent ants (green), moving backwards from the output toward the input, creating backward recurrent connections between the same nodes.

somewhat but was rather unwieldy and entailed far too much external human intervention. Instead, we developed an ant agent role specialization scheme that we found worked far better as an automatic control mechanism to control the network size and synaptic density.

The first agent role, the *explorer ant*, can only choose from forward connections in the connectivity superstructure. The connections selected by this specialized agent are utilized to generate the base neural structure upon which recurrent connections are then be added to. After the explorer ants have selected the possible nodes and forward connections, two additional specializations of what we call *social ants* are then used: *i) forward recurrent ants* and, *ii) backward recurrent ants*. Social ants are first restricted to only visiting nodes that have already been selected by the explorer ants. In the case of the forward recurrent ants, when a path is chosen, the ant specifically creates a recurrent connection that moves forward in the network along the same path, along with a selected time skip (determined by pheromones). Backward recurrent ants, on the other hand, move backwards through the network and, for each path between nodes they take, a backward recurrent connection is added, along with a selected time skip (also determined by pheromones). Figure 2 provides an example of possible pathways that these specialized agents can take in a colony superstructure.

In addition to the development of specialized ant agents as described above, we explored two modes for general ant movement; *i)* ants were allowed to pick edges that could jump over layers in the colony (*i.e.*, the superstructure is massively connected, with a plethora of skip connections), or *ii)* ants were only allowed to select edges between consecutive layers (*i.e.*, the superstructure is

fully connected, with no skip connections). This was tested to see the impact that layer skipping would have on the sparsity and performance of generated RNNs. Jumping and non-jumping modes were tested for both the standard ants (with and without forward-path bias) and the specialized ant agent roles.

3.4 Updating Pheromone Values

Different strategies for pheromone placement were also examined. We define τ as the pheromone value, α as the pheromone decay parameter, W as the weights of the evaluated (candidate) RNN, and η as the candidate model's fitness. Specifically, we describe four different functional schemes used to model pheromone deposits.

The first strategy we implemented for ANTS is standard for classical ACO setups. This deposit scheme rewards well performing RNNs with a fixed (constant) pheromone deposit while penalized ill-performing RNN models by evaporating the pheromone trace by a constant evaporation value, C. Specifically, this approach is defined as:

$$\tau_{new} = \tau_{old} \pm C \tag{2}$$

The second strategy we implemented was one that used the fitness (value) as a parameter to guide pheromone deposit. This has been shown to improve ACO performance in prior studies [15]. This scheme is defined as follows:

$$\tau_{new} = (1 - \alpha) \cdot \tau_{old} + \alpha \frac{1}{\eta} \tag{3}$$

The third strategy was to use the values of the neural synaptic weights themselves to control/guide the deposit of pheromones. Specifically, we inserted a penalty on the weights, specifically an L1 penalty (assuming a Laplacian prior of the synaptic weight values), in order to encourage regularization that favored sparser connectivity structure. This form of weight decay is sometimes applied to ANNs when controlling for over-parameterization and sparse weight matrices (with many near hard-zero values) are highly desirable. L1 regularization was applied to the pheromone deposition calculation in the following manner:

$$\tau_{new} = (1 - \alpha) \cdot \tau_{old} + \alpha \Big\{ \frac{1}{\eta + \frac{\gamma}{n} \|W\|} \Big\} \tag{4}$$

The fourth and final strategy we employed was to insert an L2 penalty to regularize the RNN candidate weights. This assumes a Gaussian prior over the synaptic weight values and is sometimes referred to in ANN literature as "weight decay". We incorporate L2 regularization into pheromone deposition according to the following formula:

$$\tau_{new} = (1 - \alpha) \cdot \tau_{old} + \alpha \Big\{ \frac{1}{\eta + \frac{\gamma}{2n} \|W\|_2} \Big\} \tag{5}$$

We developed these L1 and L2 functional variations of pheromone deposit schemes in the hopes that they would ultimately encourage/reward the uncovery of sparse, compact RNN predictive models.

3.5 Pheromone Evaporation

Lastly, pheromone trace values (deposited on the superstructures synaptic edge pathways) were allowed to evaporate or "decay" after each generation of an RNN in order to reduce the amount of pheromones on synaptic edges that have not been recently beneficial and to encourage exploration [15, 14, 26]. Pheromone values are updated (or decayed) according to the following equation:

$$\tau_{updated} = (1 - \beta) \cdot \tau_{current} + \beta \cdot \tau_{original} \tag{6}$$

where $\tau_{updated}$ is the pheromone value after the update, $\tau_{current}$ is the current pheromone value, $\tau_{original}$ is the original baseline pheromone value, and β is the pheromone evaporation rate. This function evaporates the pheromone back towards the original baseline value.

4 Results

ANTS was compared to both NEAT and EXAMM, as well as traditional layered RNN architectures. All ANTS and EXAMM experiments generated 2000 total RNNs, training each for 10 epochs. NEAT, on the other hand, was allowed to generate 420,000 RNNs. If we assume that a forward pass (forward propagation) and a backward pass (backprop calculation) are approximately the same computationally, this generously gave NEAT approximately 10 times the amount of compute time (as 2000 RNNs trained for 10 epochs would equivocate to 20,000 forward and 20,000 backward passes). The RNNs with non-evolvable (fixed) architectures were allowed to train for 70 epochs. Every experiment was repeated 10 times to compute means and standard deviations in order to ensure a proper statistical comparison.

ANTS used a colony superstructure with 12 input nodes, 3 hidden layers, each with 12 hidden nodes, and a single output node. Recurrent synapses could span 1, 2 or 3 steps in time. The resulting connectivity superstructure consisted of 49 nodes, 924 edges, and 3626 recurrent edges. While this may seem modest compared to modern convolutional architectures, which may consist of millions of connections, it is important to note that the RNNs generated from this superstructure are unrolled over 7200 time steps (according to the time series length of the training and testing data samples) when trained locally via backpropagation through time (BPTT). This means algorithms such as ANTS must handle (fully-unrolled) networks of up to 3,528,000 nodes, 6,652,800 edges, and 26,107,200 recurrent edges with errors from the final output (predictor) potentially back-propagated over up to 28,000 synaptic connections.

The dataset utilized in this study is an open access time series dataset taken from a coal fired powerplant. The data was introduced in previous neuro-evolution studies for time series data prediction [11, 27]. It consists of 12 possible parameters, recorded for 10 days with each parameter recorded at each minute. These 12 parameters were used to predict the flame intensity parameter (the response variable, in regression parlance). Results were generated by training

RNNs on 5 days worth of data taken from one of the coal burners from this data set. Fitness values (mean absolute error) were calculated on the other 5 days, which was data that was treated as a test set.

1,600 experiments were conducted in order to include all combinations of the ANTS options/variations (described below). Each experiment was repeated 10 times to obtain robust results. These ANTS experiments generated, trained, and evaluated 32 million RNNs. Experiments were scheduled on a 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 experiment utilized 15 nodes. Overall, the experiments took approximately 30 days to complete.

4.1 Backpropagation Hyperparameters

All ANNs were trained with backprop and stochastic gradient descent (SGD) using the same hyperparameters. SGD was run with a learning rate $\eta = 0.001$ and used Nesterov momentum ($mu = 0.9$) to smooth out the local gradient descent. No dropout regularization was used since it has been shown in other work to reduce performance when training RNNs for time series prediction [18]. To prevent exploding gradients, gradients were re-scaled using gradient clipping (as prescribed by Pascanu *et al.* [28]) when the norm of the gradient was above a threshold of 1.0. To improve performance in the case of vanishing gradients, gradient boosting (the opposite of clipping) was used when the norm of the gradient was below a threshold of 0.05. The forget gate bias of the LSTM cells had 1.0 added to it as this has been shown to yield significant improvements in training time by Jozefowicz *et al.* [29]. Weights for RNN in all other cases were initialized as described in the section describing our communal weight sharing 3.1 scheme for ANTS and or by EXAMM's Lamarckian weight inheritance [3].

4.2 ANTS Options and Hyper-parameters

The influence/effect of individual ANTS hyper-parameters was carefully investigated in this study. A pheromone decay rate of $\alpha = 0.05$ and a pheromone evaporation rate of $\beta = 0.1$ were chosen as they were shown to be effective in preliminary tests and is within the recommended standard range [15]. The other ANTS parameters we considered were:

1. Number of ants : {20, 40, 80, 160}.
2. Regularization update parameter: {0.25, 0.65, 0.90}.
3. Initializing RNN using communally shared weights with constant Φ values of ({0.3, 0.6, 0.9}), using Φ as calculated by a function of fitness, and basic randomized weight initialization.

The application of the examined heuristics that appear in the figures and tables that follow are labeled as follows:

1. Function $\Phi : \Phi()$

2. Constant Φ: $\Phi_{value\ of\ \Phi}$
3. L1 Pheromone regularization: $L1_{value\ of\ \gamma}$ (Equation 4)
4. L2 Pheromone regularization: $L2_{value\ of\ \gamma}$ (Equation 5)
5. Standard Ant Species: Without Bias (Std) and With Bias ($StdBias$)
6. Multi Species Ants:
   - Explorer Ants: Exp
   - Explorer Ants and Forward Social Ants: $ExpFwd$
   - Explorer Ants and Backward Social Ants: $ExpBwd$
   - Explorer Ants, Forward and Backward Social Ants: $ExpFwdBwd$
7. Layer Jumping: AJ
8. No Layer Jumping: OJ

4.3 Performance of Individual Heuristics

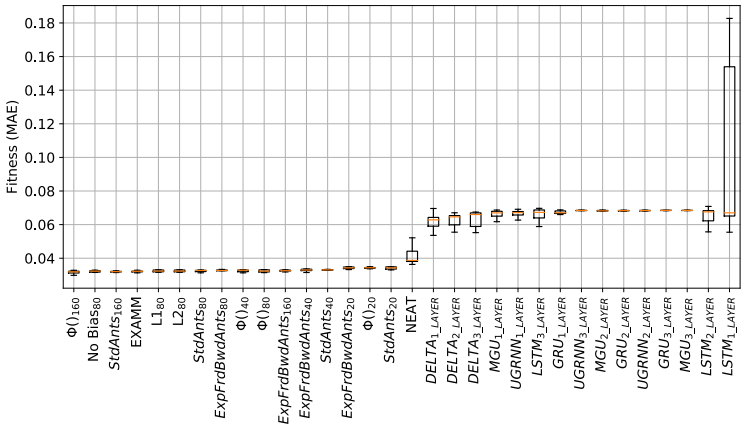

Fig. 3: Performance of NEAT, EXAMM, & individually applied ANTS heuristics against fixed memory cell RNNs.

Figure 3 presents the performance of ANTS when each each heuristic is applied separately. Furthermore, it presents for comparison the performance of the state-of-the-art EXAMM, NEAT, and traditional fixed standard RNNs. While ANTS in this case (augmented only by individual heuristics) did not outperform EXAMM except for some outliers, both EXAMM and ANTS showed dramatically better performance than NEAT, even though NEAT was given a significant amount of extra compute time. ANTS, EXAMM and NEAT also significantly outperformed traditional RNNs. Some of the gain over NEAT is most likely due to the use of backpropagation by EXAMM and ANTS since NEAT uses fairly simple and non-gradient based recombination operations to adjust weights.

	Top 10			Top 25			Top 100			Top 250		
	Mean	Median	Best	Mean	Median	Best	Mean	Median	Best	Mean	Median	Best
$\Phi()$	3(0)	4(0)	3(0)	9(0)	7(0)	9(0)	26(0)	23(0)	31(8)	58(0)	54(0)	49(8)
ConstΦ	7(0)	6(0)	7(0)	14(0)	14(0)	12(0)	60(0)	63(0)	54(8)	147(0)	149(0)	155(16)
NoΦ	0(0)	0(0)	0(0)	2(0)	4(0)	4(0)	14(0)	14(0)	15(0)	45(0)	47(0)	46(0)
L1	2(0)	4(0)	0(0)	9(0)	8(0)	3(3)	42(0)	34(0)	30(4)	96(0)	96(0)	91(4)
L2	5(0)	5(0)	6(0)	13(0)	12(0)	16(1)	40(0)	45(0)	38(3)	100(0)	98(0)	95(12)
StdAnts	0	0	0	1	0	0	3	0	0	20	19	0
StdBiasAnts	0	0	0	0	0	0	3	1	0	23	16	0
ExpAnts	0	0	10	0	0	25	1	0	100	10	6	250
ExpFrdAnts	6	7	0	14	15	0	45	49	0	98	103	0
ExpBkwAnts	0	0	0	0	0	0	0	0	0	0	0	0
ExpFrdBkwAnts	4	3	0	10	10	0	48	50	0	99	106	0
No Jump	0	0	5	0	0	13	0	0	52	0	0	128
Layer Jump	10	10	5	25	25	12	100	100	48	250	250	122
20 Ants	0	0	2	0	0	6	0	0	24	0	0	65
40 Ants	2	0	3	5	1	7	14	15	23	50	57	63
80 Ants	4	3	2	8	11	6	44	45	26	82	80	60
160 Ants	4	7	3	12	13	6	42	40	27	118	113	62

Table 1: Heuristic Ranking Statistics

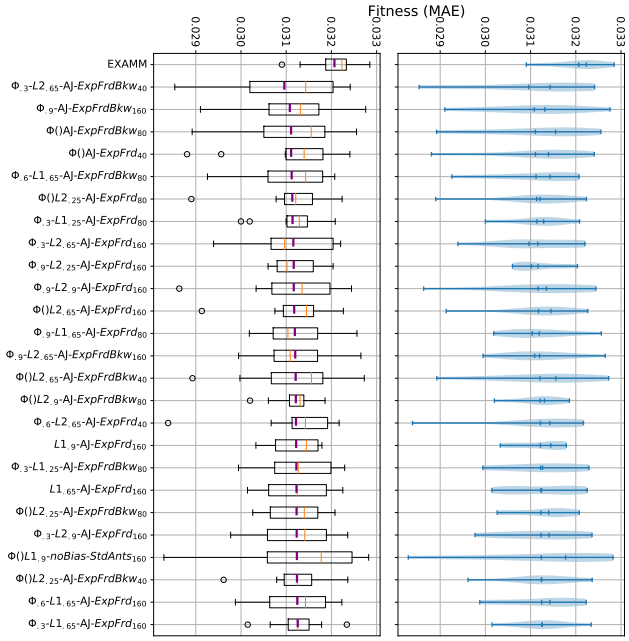

Fig. 4: Performance of EXAMM and the top 25 ANTS experiments

4.4 Performance of Combined Heuristics

The combined application of multiple different heuristics, as illustrated in Figure 4, yielded ANTS results that outperformed *all baselines*, including the fixed RNNs, NEAT, *as well as* EXAMM. Table 1 provides statistics ranking each of the heuristics based on how many times the experiments that utilized them appeared in the top 10, 25, 100, and 250 best results as determined by the mean, median, and the best performance of the RNN generated in the experiment's 10 repeats. Values in parentheses are the number of times an experiment that only

utilized that single heuristic appeared in that top ranking. The utilization of multiple heuristics dominated the top results, with individually-applied heuristics not appearing in the top 10, and only 4 times in the top 25 (only as best results).

Communal weight sharing proved to be very important, yielding strong performance, with all of the top 10 utilizing either functional or constant Φ parameters. Furthermore, it also occurred 2 (mean), 4 (median) and 4 (best) times in the top 10, and 14 (mean), 14 (median), and 15 (best) times in the top 25. Additionally, all of the best performing RNNs used layer-jumping ants, which tend favor more sparse connectivity patterns. Most of the best results used pheromone weight-regularization, with L2 regularization appearing at a nearly 50% rate in the top 10, 25 and 100 results. The regularization factor was also high, at 65% or 90%, for most of the 25 best experiments that used it.

All of the top 250 best results utilized the multiple ant species heuristic, which strongly supports the use of specialized ants. The number of ants varied between 20 and 160 for all the top 25 results in the mean and median case, with a larger number of ants tending to perform better. However the case of 20 ants did occasionally appear in the best cases, even sometimes in the top 10 and, furthermore, these networks tended to be rather sparse but very well performing. This may suggest that the experiments that utilized more ants had an easier time finding the most important structures, but also potentially had extraneous connections which were not needed. In contrast, the experiments with less ants had less of a chance of finding these important structures due to lower (overall) connectivity. This suggests that further optimizations could be designed to better guide ANTS towards the discovery of more efficient networks.

Perhaps one the most interesting items to observe is the performance distribution when multiple ant agent roles was used in ANTS. The entirety of the best found RNNs, up to the top 250 were from explorer ants only, so these generated RNNs only had recurrent connectivity in terms of whatever the various memory cells offered. However, for the mean and median performance of the experiments, nearly all the top 25, 100, and 250 consisted of explorer and forward recurrent roles or explorer, forward, and backward recurrent ant specializations – with only a very few of the only explorer ant only configurations showing up in the top 100 and 250. First, this suggests that backward recurrent connections (which are most commonly utilized in RNNs) were less effective than forward recurrent connections. Second, it also appears that adding these recurrent connections tended to make the RNNs perform significantly better on the average and median cases, while the RNNs which were generated with only explorer ants had the ability to occasionally find RNNs that generalized quite well. These results certainly suggest further study in order to better understand the effect of combining recurrent connections and memory cells. In addition, perhaps alternative strategies can be developed that retain the stability of adding recurrent connections while still efficiently finding well-generalizing RNNs.

5 Discussion

To the best of our knowledge, this work represents the first application of ant colony optimization (ACO) to the problem of neuro-evolution/neural architecture search for recurrent neural networks with varying recurrent time spans and more complex connectivity patterns, with the introduction of the novel Ant-based Neural Topology Search (ANTS) algorithm. ANTS generates candidate RNNs from a massively-connected superstructure (the colony/swarm), taking advantage of ACO for structural optimization and concepts from neuro-evolutionary/genetic approaches for maintaining populations of RNN candidates that are trained locally and asynchronously (making ANTS a memetic procedure as well). A hallmark of ANTS is its computational formalization of role specialization as done by real ant colonies – ant agents are prevented from getting stuck "wandering" around the superstructure through the use of different ant roles, which are constrained to only explore different components of the underlying complex graph space. We also utilize a noval communal intelligence strategy for sharing and updating the best found weights within the colony.

Our experimental results show that using ants with different roles generated RNNs that were not only sparse but performant – these candidates almost entirely outperformed the more standard ant traversal strategies even when standard ants were biased to select forward paths. Furthermore, communal weight sharing greatly improved the accuracy of the generated RNNs[‡]. Additionally, allowing ants to jump (or skip) layers proved to not only boost performance but also to increase sparsity. Lastly, to our knowledge the introduction of L1 and L2 regularization into the ACO pheromone deposition process is quite novel if albeit a bit unconventional. Our results show by playing with the form of the pheromone adjustment function, we can increase the likelihood that sparser RNNs are found that also outperform schemes that do not incorporate regularization/constraints. The strategies we formalize in this work are generic and could be applied to any other ACO algorithm's pheromone update process.

The proposed ANTS metaheuristic not only provides advances and new concepts for the field of ant colony optimization research to further explore but also shows strong promise for its use as an alternative neuro-evolution algorithm for automated RNN architecture search. It significantly outperforms the well-known NEAT algorithm (even when NEAT is given an order of magnitude more computation), and, more importantly, ANTS outperforms the state-of-the-art EXAMM neuro-evolution algorithm on the studied time series problem.

The work also opens up multiple avenues for future study and presents interesting questions. In particular, why were explorer ants able to find the best performing networks while performing quite poorly in the mean and median cases? Why did explorer ants combined with social recurrent ants perform extremely well in the mean and median cases but not in the best cases? Answering experimental questions such as these could lead to insights as to how recurrent

[‡]Corraborating prior studies that have also shown the benefits of similar initialization schemes [20, 3].

connections that skip multiple steps of time interact with recurrent memory cells, potentially leading to the design of more expressive RNN structures that better capture longer-term dependencies in sequential data. Finally, future work should entail investigation of ANTS on other time series datasets as well as sequence modeling (and classification) problems more commonly explored in mainstream statistical learning research, such as language modeling [30, 25].

Acknowledgements

This material is in part supported by the U.S. Department of Energy, Office of Science, Office of Advanced Combustion Systems under Award Number #FE0031547. We also thank Microbeam Technologies, Inc. for their help in collecting and preparing the coal-fired power plant dataset.

References

1. Yang, X.S.: Nature-inspired metaheuristic algorithms. Luniver press (2010)
2. Stanley, K.O., Miikkulainen, R.: Evolving neural networks through augmenting topologies. Evolutionary computation 10(2), 99–127 (2002)
3. 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
4. Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., Salakhutdinov, R.: Dropout: a simple way to prevent neural networks from overfitting. The Journal of Machine Learning Research 15(1), 1929–1958 (2014)
5. 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)
6. Yao, X., Liu, Y.: A new evolutionary system for evolving artificial neural networks. IEEE transactions on neural networks 8(3), 694–713 (1997)
7. Kassahun, Y., Sommer, G.: Efficient reinforcement learning through evolutionary acquisition of neural topologies. In: ESANN. pp. 259–266. Citeseer (2005)
8. Xie, L., Yuille, A.: Genetic cnn. In: Proceedings of the IEEE International Conference on Computer Vision. pp. 1379–1388 (2017)
9. Miikkulainen, R, L.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)
10. Desell, T.: Large scale evolution of convolutional neural networks using volunteer computing. In: Proceedings of the Genetic and Evolutionary Computation Conference Companion. pp. 127–128. ACM (2017)
11. ElSaid, A., Benson, S., Patwardhan, S., Stadem, D., Desell, T.: Evolving recurrent neural networks for time series data prediction of coal plant parameters. In: International Conference on the Applications of Evolutionary Computation (Part of EvoStar). pp. 488–503. Springer (2019)
12. Holland, J.H., et al.: Adaptation in natural and artificial systems: an introductory analysis with applications to biology, control, and artificial intelligence. MIT press (1992)

13. Dorigo, M.: Optimization, learning and natural algorithms. PhD Thesis, Politecnico di Milano (1992)
14. Mavrovouniotis, M., Yang, S.: Evolving neural networks using ant colony optimization with pheromone trail limits. In: 2013 13th UK Workshop on Computational Intelligence (UKCI). pp. 16–23. IEEE (2013)
15. Sivagaminathan, R.K., Ramakrishnan, S.: A hybrid approach for feature subset selection using neural networks and ant colony optimization. Expert systems with applications 33(1), 49–60 (2007)
16. Desell, T., Clachar, S., Higgins, J., Wild, B.: Evolving deep recurrent neural networks using ant colony optimization. In: European Conference on Evolutionary Computation in Combinatorial Optimization. pp. 86–98. Springer (2015)
17. Elman, J.L.: Finding structure in time. Cognitive science 14(2), 179–211 (1990)
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. O'Donnell, S., Bulova, S., Barrett, M., von Beeren, C.: Brain investment under colony-level selection: soldier specialization in eciton army ants (formicidae: Dorylinae). BMC Zoology 3(1), 3 (2018)
20. Desell, T.: Accelerating the evolution of convolutional neural networks with node-level mutations and epigenetic weight initialization. In: Proceedings of the Genetic and Evolutionary Computation Conference Companion. pp. 157–158. ACM (2018)
21. Hochreiter, S., Schmidhuber, J.: Long short-term memory. Neural computation 9(8), 1735–1780 (1997)
22. Chung, J., Gulcehre, C., Cho, K., Bengio, Y.: Empirical evaluation of gated recurrent neural networks on sequence modeling. arXiv preprint arXiv:1412.3555 (2014)
23. 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)
24. Collins, J., Sohl-Dickstein, J., Sussillo, D.: Capacity and trainability in recurrent neural networks. arXiv preprint arXiv:1611.09913 (2016)
25. Ororbia II, A.G., Mikolov, T., Reitter, D.: Learning simpler language models with the differential state framework. Neural computation 29(12), 3327–3352 (2017)
26. Liu, Y.P., Wu, M.G., Qian, J.X.: Evolving neural networks using the hybrid of ant colony optimization and bp algorithms. In: International Symposium on Neural Networks. pp. 714–722. Springer (2006)
27. Ororbia, A., Elsaid, A.A., Desell, T.: Investigating recurrent neural network memory structures using neuro-evolution (2019)
28. Pascanu, R., Mikolov, T., Bengio, Y.: On the difficulty of training recurrent neural networks. In: International conference on machine learning. pp. 1310–1318 (2013)
29. Jozefowicz, R., Zaremba, W., Sutskever, I.: An empirical exploration of recurrent network architectures. In: International Conference on Machine Learning. pp. 2342–2350 (2015)
30. Mikolov, T., Karafiát, M., Burget, L., Černockỳ, J., Khudanpur, S.: Recurrent neural network based language model. In: Eleventh annual conference of the international speech communication association (2010)