2025 34th IEEE International Conference on Robot and
Human Interactive Communication (RO-MAN)
August 25-29, 2025. Eindhoven, The Netherlands

2025 34th IEEE International Conference on Robot and Human Interactive Communication (RO-MAN) | 979-8-3315-8771-0/25/$31.00 ©2025 IEEE | DOI: 10.1109/RO-MAN63969.2025.11217597

Intelligent Sampling for Predicting the Performance of Hub-based Swarms

Puneet Jain[1], Chaitanya Dwivedi[2] and Michael A Goodrich[1]

***Abstract*— This paper presents an inductive learning algorithm to predict the performance of hub-based swarms solving the best-of-N problem. Since a major constraint in learning swarm behavior is the high computational cost of obtaining sample data, it is desirable to ensure the right samples are used to train the models. The paper's main contribution is formulating and comparing various sampling techniques to improve performance prediction using manageable amounts of training data. We compare *random* sampling with *in-distribution* sampling and *out-of-distribution* sampling, and then apply the lessons learned to modify *random* sampling to improve sampling. Results show that *in-distribution* sampling has the best F1 across different sampling techniques for classifying *slow* versus *fast* convergence. Model performance indicates that an informed combination of *in-distribution* and *out-of-distribution* sampling produces the highest classification accuracy of the swarm's *time-to-converge*.**

I. INTRODUCTION

Agent-based Models (ABMs) can be used to solve problems in a distributed way, such as Chemical, Biological, Radiological, Nuclear, or Explosive incident response [1], perform optimization [2], and even manage cooperating robot teams [3] and multiple UAVs [4]. Designing these agent-based models to perform well is challenging, since it is hard to predict emergent behavior [5]. This paper uses a framework inspired by [6], [7] to predict the performance of a colony solving the best-of-N problem [8]. Previous work [7] shows that graph neural networks (GNNs) can be used to inductively learn the behavior of ABMs solving the best-of-N problem, for varying swarm sizes and environments. This paper improves performance predictions by modifying the proposed graph formulation, learning model, and sampling techniques from [7]. The goal is to provide a solution to two unresolved challenges: expensive ABM sampling and exploding state space of the swarm.

The paper's contributions are: (a) An analysis of the possible benefits of sampling behavior trajectories from ABM simulations. (b) A comparison of the proposed sampling methods for collecting training data for more efficient learning, i.e., better predictions with less data. (c) A new feature vector to enable learning over larger swarms without exploding the feature vector size. This paper focuses on one swarm/environment setting to improve the classification accuracy for *time-to-converge* (*slow* versus *fast*) for a hub-based swarm solving the best-of-N problem. Note: the terms colony and swarm are used interchangeably in this paper.

*The work was supported by US Office of Naval Research grant N00014-21-1-2190. The work does not represent sponsor opinions.

[1]Department of Computer Science, Brigham Young University, Utah, USA. puneetjain2895@gmail.com

[2]Amazon AGI, California, USA

II. RELATED WORK

Natural Collective Behaviors. The study of collective behavior in nature has been a significant source of inspiration for developing agent-based models (ABMs). Research has documented various forms of collective intelligence, from bird flocking patterns [9] to the complex decision-making processes in honeybee colonies during nest selection [10]. Similarly, ant colonies have been studied for their efficient food-gathering strategies [11], while schools of fish demonstrate remarkable collective movement patterns [12]. Animal swarms exhibit emergent intelligence, self-organization, robustness, and adaptability. ABMs allow collective behaviors to be simulated, providing a powerful tool for exploring emergent phenomena in controlled virtual environments. These explorations help bridge the gap between observational studies and theoretical understanding of complex biological systems [13]. Some examples of ABMs include [14] which used differential equations to design ABMs and [15] which leveraged finite state machines with additional memory.

Best-of-N Problem. The best-of-N problem involves a group of agents that collectively identify the highest quality option among N alternatives in a decentralized manner [16]. Each option typically has an associated quality that agents can assess but with some degree of noise or uncertainty [17]. Collective site selection in swarm robotics [18], distributed sensing and decision-making in sensor networks [17], and consensus formation in social networks [19] can all be formulated as the best-of-N problem.

Graph Neural Networks for ABMs. The integration of ABMs with graph-based representations has emerged as a sophisticated approach in modeling complex systems. Several pioneering frameworks have been developed to transform traditional finite state machine based ABMs into more expressive graph representations [20], [21]. This transformation enables one to leverage the inherent structural relationships between agents while preserving the state-transition dynamics that characterize agent behavior. In multi-agent systems, Graph Neural Networks (GNNs) have demonstrated remarkable versatility and effectiveness. Particularly noteworthy is their application in traffic engineering scenarios [22], where they capture the complex interactions between multiple vehicles, traffic signals, and infrastructure elements. Similarly, GNNs are used in path prediction tasks [23], where they can effectively model and forecast the movement patterns of multiple agents while accounting for their mutual influences and

environmental constraints. The field has also seen significant advancement in sub-graph learning approaches [24], [25]. Analyzing sub-graphs can capture local patterns and relationships that might be obscured when considering the entire graph structure, which is especially useful when processing the complete graph might be computationally prohibitive.

Sampling Approaches for Agent-Based Models Previous work on sampling approaches for ABMs has explored various techniques to address computational challenges, improve efficiency, and enhance the analysis of these complex systems. In [26], authors applied Latin Hypercube Sampling (LHS) to an agent-based model of immune response to mycobacterium tuberculosis infection. They demonstrated that LHS allowed for a more comprehensive exploration of the parameter space compared to simple *random* sampling. [27] proposed an adaptive sampling approach for ABMs that iteratively refines the sampling based on the model's output. The method showed improved efficiency in exploring complex parameter spaces compared to static sampling methods. [28] applied Approximate Bayesian Computation (ABC) to estimate parameters of agent-based economic models. The paper showed how ABC can be used to perform Bayesian inference in complex ABMs where traditional likelihood-based methods are not feasible.

III. Agent-Based Model and Collective State Graph

This section briefly summarizes the ABM used in this paper to solve the best-of-N problem and draws on [20]. The prior work demonstrated that this ABM can solve the best-of-N problem. We present a more efficient representation of the *collective state*, which depicts the state of the swarm and the environment at any time in a way that scales to larger swarms and environments without the computational complexity of previous representations. We then discuss how the *collective state graph* (CSG) is formed from the swarm's transitions. Understanding the CSG is necessary for understanding tradeoffs in various ABM sampling techniques.

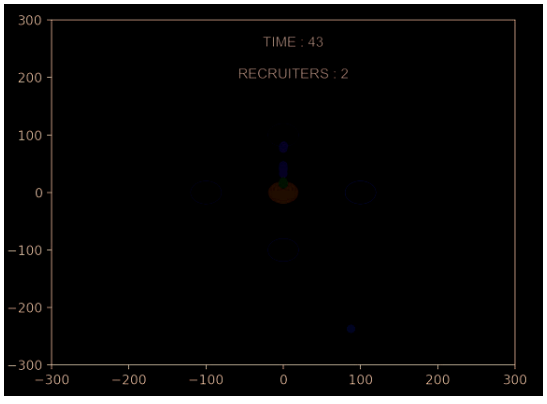


Fig. 1: ABM. The red agent is exploring. Yellow agents are traveling between site and hub. The agent at the site is assessing. The agents at the hub are observing or recruiting.

A. Best-of-N

Figure 1 illustrates the spatial best-of-N problem used in this paper. The scenario involves four sites (open ovals), ten agents (filled dots), and a central hub (filled oval). The agents explore the environment to locate potential sites. When an agent discovers a site, it returns to the hub to inform the others. If an agent fails to find a site, it comes back to the hub to observe the actions of other agents. These agents move back and forth between the hub and the sites of interest to evaluate the sites and recruit more agents. Recruiters can sense when a quorum, or a sufficient number of agents, has gathered at the hub. Once this quorum is achieved, the collective decides that the site is the solution to the problem.

B. Agent-Based Model

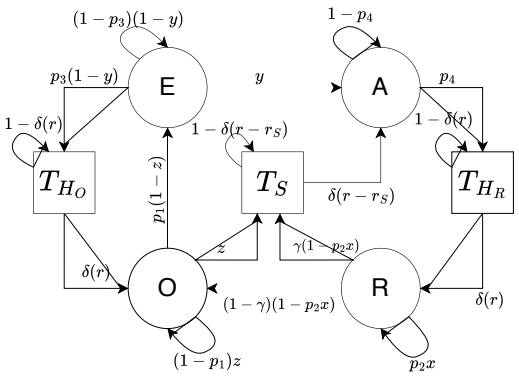


Fig. 2: State machine for our agent-based model.

The ABM shown in Figure 2 is adapted from [6], [7]. The ABM adheres to the Markov condition, meaning that each agent's next state is solely dependent on its current state. Each agent runs its version of the state machine shown in Figure 2, consisting of the states which are *site-agnostic* (Observe (O), Explore (E), Travel to Hub to Observe (T_{H_O})) as well as states which are *site-oriented* (Assess (A), Recruit (R), Travel to Hub to Recruit (T_{H_R}), and Travel to Site (T_S)). The parameters used for the transitions are defined in Table I.

TABLE I: ABM parameters. $Q(s_i)$ = quality of a site i, r = distance between agent and hub, r_S = distance between agent and site S, and $|R|$ = number of agents in Recruit state.

Parameter	Values
x	$2/(2+e^{-7Q(s_i)})$
y	$q\delta(r-r_S)$
z	$\delta(p_r)/\|R\|$
p_r	binomial($\|R\|$,0.1)
p_1	binomial(1,0.01)
p_2	binomial(1,0.99)
p_3	binomial(1,0.02)
p_4	binomial(1,0.1)
γ	$Q(s_i)^{0.5}$
τ	0.5

State transitions are based on the agent's current state and its position relative to sites, the hub, and other agents. The edges of the state machine illustrate the transition

State ↓ | Info →	agents @ s_0	agents @ s_1	agents @ s_2	agents @ s_3	agents elsewhere	agents @ H
R	K(s_0)	K(s_1)	K(s_2)	K(s_3)	0	K(Hub)
A	K(s_0)	K(s_1)	K(s_2)	K(s_3)	K(Other)	0
T_{H_R}	K(s_0)	K(s_1)	K(s_2)	K(s_3)	K(Other)	0
T_S	K(s_0)	K(s_1)	K(s_2)	K(s_3)	K(Other)	0
O	0	0	0	0	0	K(Hub)
E	0	0	0	0	K(Other)	0
T_{H_O}	0	0	0	0	K(Other)	0

Site Info	X(s_0)	Y(s_0)	Q(s_0)	X(s_1)	Y(s_1)	Q(s_1)	X(s_2)	Y(s_2)	Q(s_2)	X(s_3)	Y(s_3)	Q(s_3)

TABLE II: New feature vector v (length 54) to accommodate for any number of agents, but limiting to 4 best sites. s_0 is the site with the highest number of agents, s_1 the second highest, and so on. X and Y are the coordinates, Q is the quality, and K is the number of agents oriented to that site. The feature vector is a 1D concatenation of the rows, where the site order changes based on which sites are discovered. This also encodes if the agent is at the hub or somewhere else. Since we do not know if the agent is at the site or not, we put it in the "other" basket.

probabilities. Transitions from travel states (T_{H_O}, T_{H_R}, T_S) are dictated by δ functions, which imply that agents only move out of these states when they reach their respective destinations, either the hub (for T_{H_O} and T_{H_R}) or a site (for T_S). Each agent makes a decision about changing state at each simulation time step. Quorum is achieved when more agents than the threshold (τ) are recruiting for the same site at the same time.

C. Collective State Graph

This section defines the *collective state* of the swarm and the *collective state graph*.

1) Collective State: In previous work [7], agent-based state vectors were used to be able to generalize to any environment. This scales very poorly to larger swarms. This paper uses a site-based *collective state* vector, denoted by v, defined in Table II. Row labels correspond to the states shown in Figure 2. The *collective state* vector scales better to more complex environments because it considers only the top four sites in the environment at any time (top sites based on number of agents oriented towards that site). Restricting the vector to the top four sites is sufficient for most problems because previous work has shown a high correlation between the difference in quality of the top two sites and both the time to converge and the probability of success [29]. Furthermore, the collective state vector scales to larger swarms because it uses the number of agents that favor sites in various states, denoted by $K(s)$, rather than individual agent preferences.

2) Collective State Graph: The *collective state graph* (CSG) represents how the swarm behavior evolves through time, from one collective state to another. The CSG is a graph, denoted by G_{CSG}, whose vertex set V, edge set E, and edge weighting $w : E \mapsto \Re$ are defined as follows:

- The vertex set V consists of all possible *collective state* vectors v. In other words, the nodes of the CSG represent possible swarm states.
- The set of edges E is the set of all possible transitions between collective states. The structure of the individual state machines shown in Fig. 2 shows the transitions are directional. If the swarm can transition between state v_i and v_j, then a directed edge exists (v_i, v_j).
- The edge weight, denoted $w(v_i, v_j)$, is the probability that the transition occurs from vertex v_i to vertex v_j.

The *time-to-converge* for a given *collective state* is defined as the average time the swarm takes to reach a quorum threshold from that state. It is not possible to actually instantiate G_{CSG} in code because it requires either (a) prohibitive numbers of simulations or (b) the mapping of the individual state machines into a first-order Markov chain with a prohibitive number of states. Consequently, the classifier needs to estimate time-to-converge from samples of the CSG.

IV. Learning the Swarm Behavior

This section presents the classifier architecture and the processing framework of the data fed into the learning model. Figure 3 shows the architecture used, modified from [7], to generate a 2D classification output for a given sub-graph. The goal of the classifier is to predict whether a given *collective state* will result in a *slow* or a *fast time-to-converge*.

A. Learning Architecture

We use a Graph Attention Network (GAT) based classifier. Previous work [7], [24], [30] has shown that GAT-based networks can be used to inductively learn to predict ABM performance classes from sub-graphs of the CSG. GAT networks, using GATConv layers, use an attention mechanism that can result in better performance [30]. Figure 3 shows the network architecture used in this paper, with three GATConv layers. The first two GATConv layers are followed by activation functions (*elu*). The output of the third GATConv layer is added with a skip connection through a linear layer. This mixed output is then passed through a linear layer to generate the embeddings, and another linear layer to generate the classification of time to converge to a site.

B. Model Parameters

The learning rate is set to 0.0001 with a decay of 0.0005. The decay acts as a regularizer, mitigating the risk of model overfitting [31]. The model is run for five epochs, after which a plateau in loss is observed. Using lessons from prior work, the model has a hidden layer dimension of $h = 128$, uses the *AdamW* optimizer, and computes classification loss using the cross entropy metric. We set the GAT heads to 4, since the average vertex degree is be 3.757 in the training set.

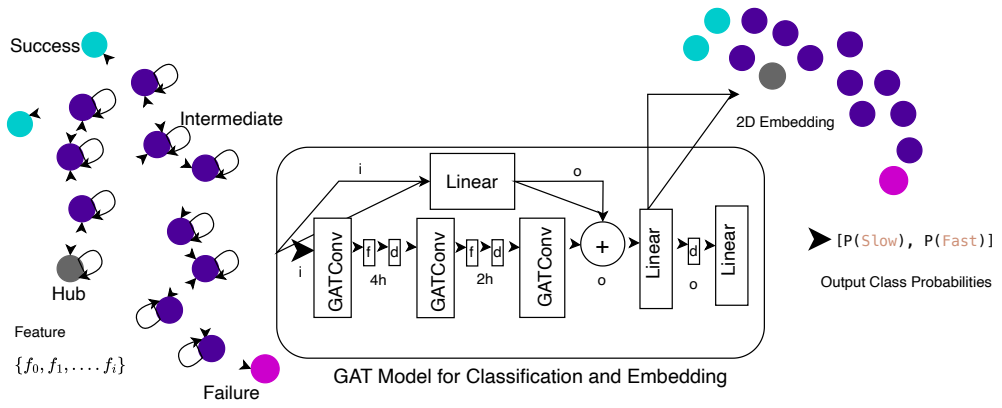


Fig. 3: Learning architecture to predict convergence time. The graph on the left shows a sample CSG with vertices labeled by whether the swarm chooses the correct site (Success) or not (Failure). The filled circles on the right represent the latent vector information identified form the learning architecture. The colors in the embedding correspond to the vertex labels.

C. Creating the Sampled CSG

The purpose of this subsection is to describe the process for creating sampled CSGs, denoted by $\hat{G}$, since we cannot work with the complete G_{CSG}. The section identifies the assumptions and processes followed to overcome the limited ability to reasonably estimate the CSG. The sampled CSG is defined as the vertex set $\hat{V}$, edge set $\hat{E}$, and edge weighting $\hat{w}$, giving, $\hat{G} = \{\hat{V}, \hat{E}, \hat{w}\}$. The hat indicates an estimate of the corresponding element of G_{CSG}. We now describe how each estimate is obtained.

Vertex Set: The vertex set $\hat{V}$ is obtained as follows: (a) Generate a starting node according to one of the sampling techniques described in Section V-A. (b) For each starting node, create multiple trajectories by simulating the ABM until convergence or up to 10000 steps. 30 simulations are used for training and 60 are used for testing. (c) For each simulation from a starting node, record samples of length $\min(128, \textit{time-to-converge})$. A maximum trajectory length of 128 was subjectively chosen as it provides adequate nodes to sample batch sub-graphs as defined in Section IV-D. In cases when starting nodes are close to quorum, shorter trajectory lengths may occur, resulting in a smaller sampled CSG since most trajectories will not reach the maximum length of 128 steps. Since each vertex visited in this process represents a feasible *collective state*, $\hat{V} = \{v : v \in \{\text{trajectory}\}\} \subseteq \text{V}$.

Edge Set: The edge set $\hat{E}$ is determined by the transitions observed in the sampled trajectories including self-transitions. Let $T = \{(v_i, v_j) : v_i \text{ transitions to } v_j\}$. We use undirected edges because removing direction allows the different layers of the learning architecture to better aggregate information from sub-graph structures. Thus, the sampled edge set is the set of *undirected* edges $\hat{E} = \{\{v_i, v_j\} : (v_i, v_j) \in T \text{ or } (v_j, v_i) \in T\}$. Since the edges in E and $\hat{E}$ are directed and undirected, respectively, $\hat{E} \not\subseteq E$.

Edge Weighting: Recall that the edge weighting function w represents the transition probability between edges. The estimate of the edge weighting function for the undirected edge $\{v_i, v_j\} \in \hat{E}$ is obtained by counting the number of *directed transitions* from vertex v_i to vertex v_j or vice versa. $C(v_i, v_j)$ denotes this count. The transition probability is estimated by normalizing by the total number of transitions from vertex v_i including self-loops, $\hat{w}(v_i, v_j) = \frac{C(v_i, v_j)}{\sum_{v_k \in N(v_i)} C(v_i, v_j)}$ where $N(v_i)$ is the set of neighbors of vertex v_i.

D. Sampling Sub-graphs for Batch Processing

Sub-graphs from $\hat{G}$ are obtained using the following steps. If a node v has fewer neighbors than the samples requested, *neighbor sampling* returns the complete neighborhood of v, denoted by $N_{\text{samp}}(v)$. Note that *neighbor sampling* is sampling without replacement.

- Randomly select $v \in \hat{V}$ with uniform probability.
- Randomly sample four neighbors of v, denoted by $N_{\text{samp}}(v) = \{v_0, v_1, v_2, v_3\}$ with uniform probability using *neighbor sampling*.
- Sample four neighbors for each $v_k \in N_{\text{samp}}(v)$ using *neighbor sampling*.
- The vertex set of the sub-graph S_{batch} is the vertex v, its neighbors $N_{\text{samp}}(v)$, and their sampled neighbors. The edge set of the sub-graph S_{batch} is the set of all edges in $\hat{E}$ that come from vertices in the sub-graph, and their corresponding weights from $\hat{w}$.

The next section both describes how some nodes in $\hat{V}$ are assigned labels and discusses how each sub-graph S_{batch} that includes at least one labeled node can be used for training.

V. Experiment Design

This section describes the independent variables, which are the different sampling conditions for choosing the initial states from which $\hat{G}$ is created. It then discusses the node-labeling process, followed by the hypotheses. Finally, the choice of dependent variables is discussed and the train/test split for the experiments is described.

A. Independent Variables

Generating samples is expensive: (a) Running an ABM trial is time consuming, even when agents move much faster than real-time. (b) Storing all ABM trajectories and training on them requires more memory than is practicable. The independent variable is the sampling technique for obtaining the the initial state in the sampled CSG. Multiple trajectories are generated from each initial state to form a sampled CSG. Simulations that start close to where a quorum is reached, like when multiple agents are initialized to favor the same site, tend to converge fast and generate correspondingly short trajectories. By contrast, simulations that begin far from a quorum state, like when many agents are favoring very different sites and few agents are observing, tend to converge slow and generate correspondingly long trajectories.

1) Trajectory Sampling: The *trajectory sampling condition* generates trajectories in two phases. In the first phase, 100 simulations were run beginning from the *origin*, which is defined as all agents are at the hub in the Observe state. Simulations are run until convergence, which requires agents to explore the environment, discover relevant sites, recruit other agents, and reach quorum threshold. Each simulation generates a trajectory consisting of a sequence of collective states. In the second phase, initial collective states are selected by choosing 10 points from each of the 100 trajectories, yielding 1000 initial collective states.

The trajectory sampling condition favors *in-distribution samples*. "In-distribution" means the kinds of collective states that are likely to be seen in typical ABM trials as the swarm explores the world. Descriptively speaking, the individual agent-states (e.g, Observe, Explore, etc.) for each initial collective state have a wide distribution with a notable concentration in the Observe state due to the bias toward initial collective states close to the origin. Additionally, there are relatively few agents in Assess and Travel-to-Hub agent-states because these states tend to be far from the origin.

2) Random Sampling: The *random sampling condition* uniformly chooses agent-states from the set of all possible agent-states. As with the trajectory sampling condition, 1000 random starting states are generated. Sites are uniformly chosen for agents in site-oriented states. Positions are uniformly chosen between hub and site for agents in travel states.

The random sampling condition distributes agents evenly across possible agent states. This condition sample the space of possible initial collective states more uniformly than the trajectory sampling condition without bias toward either *in-distribution* or *out-of-distribution* collective states. The random sampling condition includes more *out-of-distribution* points than the trajectory sampling condition.

3) Informed Random Sampling: The *informed random sampling* deliberately chooses more *out-of-distribution* initial collective states than the random sampling condition. This can be useful because the ABM parameters are tuned in such a way that the colony is likely to choose the best site, which means that typical simulations do not typically visit collective states that lead to bad collective decisions. 1000 initial collective states are obtained using random sampling and trajectories are run from these initial states. The starting collective states are then labeled as "slow" or "fast" depending on how many transitions are required before the quorum threshold is reached. Five new initial collective states are generated for each slow state by randomly chaning agent-states with a probability of 0.1, which roughly translates to 1 out of 10 agents changing its agent-state. This makes the set of initial nodes more than 1000.

Naturally, the informed random condition generates agent-states that are distributed similarly to the random condition, but with a higher percentage of initial collective states far quorum decision threshold because such states occur more often when convergence is slow.

4) Near Extremes Sampling: The *near extreme sampling condition* favors initial collective states that are very unlikely to be found in the other conditions. The near extremes sampling condition emphasizes two types of rare states: (a) when nearly all agents are in a single site-agnostic agent-state, and (b) when nearly a quorum of agents are in a single site-oriented agent-state. This sampling approach includes collective states when there is almost a quorum favoring a bad site. Each type of rare collective state has the same number of samples in the set of initial collective states

The near extremes sampling condition generates initial collective states that are out-of-distribution, predominantly in single site-agnostic collective states or collective states near a quorum threshold.

B. Semi-supervised Labeling for Batches

The learning architecture is used to classify collective states as *fast* or *slow*, where a time threshold of 400 transitions was subjectively chosen because it divided training classes roughly evenly.

Multiple batches of sub-graphs S_{batch} are formed for each sampling condition. It is not practicable to assign class labels for all nodes in S_{batch}, so labels are assigned only for nodes sampled as a starting node during initial sampling. These labels are determined by running multiple simulations from the starting node until convergence: 30 for training set and 60 for the test set. The mean *time-to-converge* of the simulations for each of these starting nodes is used to classify the node as *fast* or *slow*. Simulations are limited to trajectories no greater than 10000 steps to keep computation time small enough.

Each training batch consists of 32 sub-graphs with at least one labeled node. This approach greatly reduces computations and improves quality of labels compared to [7] where a single trajectory is used as a batch, labeling all nodes encountered according to their time-to-convergence averaged across multiple simulations where they occur.

C. Motivation and Hypotheses

A classifier is trained on training data from a particular sampling condition and then tested by sequentially classifying initial collective states for each sampling condition. The first two hypotheses address conditions under which each learned model will work best. The motivation for the first hypothesis is that a classifier trained on *in-distribution*

data will work best when tested on *in-distribution* data. The motivation for the second hypothesis is that a classifier trained on the widest range of *in-distribution* and *out-of-distribution* data will have the best performance when tested on data from each of the sampling conditions.

Hypothesis 1: A classifier trained on data from the *trajectory sampling condition* will be superior to other classifiers when tested on data from the *trajectory sampling condition*.

Hypothesis 2: A classifier trained on data from the *informed random condition* will outperform classifiers trained from data from the other sampling conditions sampling when using data from all test sets.

These hypotheses are evaluated in Section VI-A, which will show that the classifier trained on the trajectory-sampling condition performs best, supporting the first hypothesis but refuting the second. We formulate two additional hypotheses based on these results.

Hypothesis 3: Combining training samples from the *trajectory* sampling condition and *near extremes* sampling condition will improve classification accuracy compared to classifiers trained using only samples from individual sampling conditions.

Hypothesis 4: Adding additional starting collective states obtained from *informed random* sampling condition to a blend of *trajectory* and *near extremes* training data will not improve performance.

D. Dependent Variables

The weighted F1 score weighted by the number of true instances per class to account for class imbalance, is used to compare the performance. This F1 score is the harmonic mean of precision and recall.

E. World and Swarm Conditions

Experiments were run for colonies with 10 agents. Initial positions of agents for simulations are described in Algorithm 1. The quorum threshold for all simulations is set to $\tau = 0.5$ during the Recruit state, meaning that for a colony of 10 agents, 6 agents must be recruited for the same site simultaneously for the simulation to conclude.

Four sites are used in the experiments, with site qualities set to $Q(s) = (0.913, 0.196, 0.388, 0.693)$. Subjective evaluations of the best-of-N model indicated that the relatively high and similar values of the first and fourth sites make the decision problem reasonably challenging for the colony. The distances to the sites is subjectively chosen as $D(s) = 150$, with all sites being equidistant from the hub.

Along with different techniques to sample the space for starting conditions, we store 128 time steps of data from the starting node for each simulation. We found that the average degree of the graphs is around 4. We also tested varying neighborhoods from 64 to 1024, and found 128 time steps over 30 simulations give a big enough neighborhood to learn the graph structure and time to converge.

For this paper, we do not attempt to make all the training and testing data sizes the same, since even when the initial number of labeled nodes is the same, our technique of forming batches of sub-graphs in Section V-B generates a different number of training and testing samples for each technique, due to variation in sampled CSGs.

Algorithm 1 Agent Initialization Based on State

Input: agent-state s.
Output: agent-position $[x,y]$, agent-direction θ
for each agent **do**
  $\theta \sim \mathscr{U}[-\pi, \pi]$
  switch s
    case Observe (O) or Recruit (R):
      $[x,y] \leftarrow [0,0]$
    case Assess (A):
      $[x,y] \leftarrow [X(s_i), Y(s_i)]$
    case Explore (E):
      $x \sim \mathscr{U}[-1000, 1000], y \sim \mathscr{U}[-1000, 1000]$
    case Travel to Hub to Observe (T_{H_O}):
      $x \sim \mathscr{U}[-1000, 1000], y \sim \mathscr{U}[-1000, 1000]$
      $\theta \leftarrow \arctan(\frac{y}{x})$
    case Travel to Hub to Recruit (T_{H_R}):
      dist $\sim \mathscr{U}[0,1]$
      $[x,y] \leftarrow dist \times [X(s_i), Y(s_i)]$
      $\theta \leftarrow \arctan(\frac{y}{x})$
    case Travel to Site (T_S):
      dist $\sim \mathscr{U}[0,1]$
      $[x,y] \leftarrow dist \times [X(s_i), Y(s_i)]$
      $\theta \leftarrow \arctan\left(\frac{Y(s_i)-y}{X(s_i)-x}\right)$
    end switch
end for

Comments: The direction of the agent is set uniformly within $[-\pi, \pi]$ unless updated in one of the cases. For O or R, the position is set to the hub, and to the site for A. The position is set randomly in the environment for E and T_{H_O}. The direction is set towards the hub for T_{H_O}. For T_{H_R}, the position and direction are on line from the hub to site, and from the site to hub to site for T_S.

VI. Results and Discussion

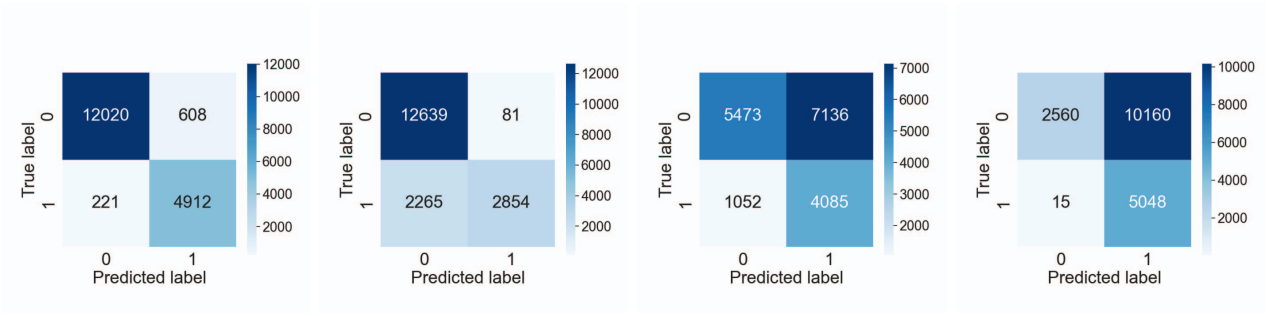
This section presents the classification results from two experiments for the two-class classification problem of *slow* and *fast*: (a) Experiments with training on individual sampling conditions. (b) Experiments with data blended from multiple sampling conditions.

A. Training on Individual Sampling Condition

This section evaluates the first two hypotheses. Simulation and model parameters are given in Section V-E and IV-B, respectively. After training one instance of the GAT model on data collected from each of the sampling techniques defined in Section V-A, validation/testing is done using fewer sampled starting points but using 60 sample trajectories instead of 30.

Classification Results for Individual Sets: F1 scores are the primary performance measure. F1 scores for the mean *time-to-converge* classification problem are given in Table III. Confusion matrices for the most interesting results are shown in Figure 4, where Class 0 is *fast* and class 1 is *slow*.

Discussion: Results are now discussed.



(a) *Trajectory* Train (b) *Random* Train (c) *Informed Random* Train (d) *Near Extremes* Train

Fig. 4: Performance of models trained on independent training sets on the *Trajectory* test set.

Training Data	Testing Data: *Trajectory*	*Random*	*Informed Random*	*Near Extreme*
Trajectory	0.9531	0.9977	0.9964	0.9865
Random	0.8589	0.9981	0.9976	0.9488
Informed Random	0.5510	1.0	0.9977	0.9749
Near Extreme	0.3829	0.9987	0.9988	0.9970

TABLE III: F1 scores for individual sampling techniques

1) Trajectory Condition Training: The model trained on the *trajectory* training set performs well on all test sets. Interestingly, the model's worst performance is on the *trajectory* test set, though the model trained on the *trajectory* training set has the highest F1 score for *trajectory* test set. These data support hypothesis 1.

2) Random Condition Training: The model trained on the *random* training set performs well except on the *trajectory* test set, though it outperforms models trained on the other two training sets on the *trajectory* test set. The model trained on the *random* training data performs especially poorly for *slow* instances, as seen in the lower left in Figure 4b.

3) Informed Random Condition Training: The model trained on the *random* training data performs well on the *random* test data and even better on the *informed random* test data. High performance on the *informed random* test data is unsurprising since the data in the *random* sampling condition is a subset of the *informed random* condition. The model trained on *informed random* training data does not perform well on the *trajectory* test data, refuting hypothesis 2. The upper right portion of Figure 4c shows especially poor performance for the *fast* class because *informed random* sampling biases the training data to the *slow* class.

4) Near Extremes Condition Training: The model trained on the *near extremes* training data performs poorly on the *trajectory* test set. It performs well on *random* and *informed random* test sets, although not as well as other sampling conditions. The upper right portion of Figure 4d shows that most misclassification error comes from the *fast* condition.

B. Blending Training Sets Across Sampling Conditions

Hypothesis 3 and 4 were evaluated by training models on two different blended datasets. The *Combined-2* dataset combines 45000 simulations from *trajectory* with 45000 from *near extremes*. The *Combined-3* dataset adds additional training data from 10000 simulations from *informed random*. Table IV presents results.

Training Data	Testing Data: *Trajectory*	*Random*	*Informed Random*	*Near Extremes*
Combined-2	0.9531	0.9984	0.9976	0.9938
Combined-3	0.9538	0.9994	0.9995	0.9970

TABLE IV: F1 scores for training on blended sets

1) Combined-2: Trajectory and Near Extremes: Since the *combined-2* training data include *trajectory* training data, it is unsurprising that the model trained using the *combined-2* training data performs well on the *trajectory* test set. Interestingly, the good performance occurs even though the training data had only half the training samples from the *trajectory* training set. Including *near extremes* samples did not negatively impact performance on the *trajectory* test data, while improving our performance on the *near extremes*, *random*, and *informed random* test sets. The F1 scores show that *combined-2* has similar performance to the validation scores of *informed random* sampling and better than validation scores for *random* sampling. These data support hypothesis 3.

2) Combined-3: Trajectory, Near Extremes, and Informed Random: The model trained on the *combined-3* training data qualitatively performs slightly better on the *trajectory* and *near extremes* test sets than the model trained on the *combined-2* training data. The model also qualitatively provides a slight performance increase for the *random* and *informed random* test sets. This suggests that using training data from *informed random* sampling condition along with the *combined-2* sampling condition adds more information to the model training. This refutes hypothesis 4 and instead suggests that a good data mix should contain all 3 sets - *trajectory*, *random* (or *informed random*), and *near extremes*.

C. Discussion

Using *in-distribution* training samples yields accurate predictions for the majority of cases that are going to occur when a hub-based swarm solves the best-of-N problem.

However, results show that using only *in-distribution* training data performs poorly on edge cases which occur when the swarm reaches an adversarial or rare configuration. *Out-of-distribution* samples are therefore needed to train the model. Including a carefully chosen mix of trajectory, random, and extreme (all agents in a single state) swarm configurations improves performance prediction under all testing conditions with the same amount of data as single sampling strategies.

VII. Summary and Future Work

This paper demonstrates that the time to converge for best-of-N models can be predicted using efficient sampling techniques for ABM simulations. By converting ABM states into a scalable feature vector called the *collective state*, and forming a *collective state graph* (CSG) from state transitions, the study shows that sampled CSGs can be used to generate batches of undirected sub-graphs. These sub-graphs are then used to train Graph Attention Network models to classify *slow* versus *fast* convergence times. Among the four proposed sampling conditions, *in-distribution* sampling (trajectory) performed best overall, and combining different sampling methods further improved performance.

A key limitation is that measurements were only taken for one set of site qualities, site distances, and agent population sizes. This restricts the generalizability of the findings. Additionally, the study only focused on the classification task of time-to-converge, without exploring other performance metrics such as success probability.

Future work should generalize the experiments to more diverse swarm and environment configurations. Exploring different data mixes to determine the best performance on convergence prediction tasks is also recommended. A two-step method that first detects if a node is *in-distribution* or out-of-distribution, followed by specialized models for each, could optimize data needs and performance prediction.

Use of AI

The content of this paper is original to the authors (also included in thesis). Microsoft Copilot was used to shorten some sections with prompt "Rewrite subsection *x* so that Fig *z* can be removed. Use brief qualitative descriptions of the figure." The AI-generated revisions were then edited.

References

[1] C. M. Humphrey and J. A. Adams, "Robotic tasks for chemical, biological, radiological, nuclear and explosive incident response," *Advanced robotics*, vol. 23, no. 9, pp. 1217–1232, 2009.

[2] M. Dorigo, A. Colorni, and V. Maniezzo, "Distributed optimization by ant colonies," 1991.

[3] J. Wang and M. Lewis, "Human control for cooperating robot teams," in *2007 2nd ACM/IEEE International Conference on Human-Robot Interaction (HRI)*, pp. 9–16, IEEE, 2007.

[4] M. L. Cummings, S. Bruni, S. Mercier, and P. Mitchell, "Automation architecture for single operator, multiple UAV command and control," tech. rep., Massachusetts Inst Of Tech Cambridge, 2007.

[5] G. Valentini, H. Hamann, and M. Dorigo, "Global-to-local design for self-organized task allocation in swarms," *Intelligent Computing*, 2022.

[6] P. Jain, C. Dwivedi, V. Bhatt, N. Smith, and M. A. Goodrich, "Performance prediction of hub-based swarms," *arXiv:2408.04822 [cs.MA]*, 2024.

[7] P. Jain, C. Dwivedi, V. Bhatt, N. Smith, and M. A. Goodrich, "Performance prediction of hub-based swarms," *Philosophical Transactions A*, 2024.

[8] G. Valentini, E. Ferrante, and M. Dorigo, "The best-of-N problem in robot swarms: Formalization, state of the art, and novel perspectives," *Frontiers in Robotics and AI*, vol. 4, Mar. 2017.

[9] C. W. Reynolds, "Flocks, herds and schools: A distributed behavioral model," in *Proceedings of the 14th Annual Conference on Computer Graphics and Interactive Techniques*, pp. 25–34, 1987.

[10] T. D. Seeley and S. C. Buhrman, "Nest-site selection in honey bees: How well do swarms implement the "best-of-N" decision rule?," *Behavioral Ecology & Sociobiology*, vol. 49, no. 5, 2001.

[11] M. Dorigo and L. M. Gambardella, "Ant colonies for the travelling salesman problem," *BioSystems*, vol. 43, no. 2, pp. 73–81, 1997.

[12] B. L. Partridge, "The structure and function of fish schools," *Scientific american*, vol. 246, no. 6, pp. 114–123, 1982.

[13] N. Horsevad, H. L. Kwa, and R. Bouffanais, "Beyond bio-inspired robotics: How multi-robot systems can support research on collective animal behavior," *Frontiers in Robotics and AI*, vol. 9, Jun 2022.

[14] D. J. Sumpter, *Collective animal behavior*. Princeton University Press, 2010.

[15] M. Dorigo, M. Birattari, C. Blum, A. L. Christensen, A. Reina, and V. Trianni, "Ant colony optimization and swarm intelligence," *Lecture Notes in Computer Science*, vol. LNCS-11172, 2018. Proceedings of 11th ANTS International Workshop.

[16] G. Valentini *et al.*, "Collective decision-making with minimal interactions in robot swarms," *Nature Scientific Reports*, vol. 13, 2023.

[17] A. Reina, G. Valentini, C. Fernández-Oto, M. Dorigo, and V. Trianni, "A design pattern for decentralised decision making," *PloS One*, vol. 10, no. 10, p. e0140950, 2015.

[18] G. Valentini, H. Hamann, M. Dorigo, *et al.*, "Self-organized collective decision making: the weighted voter model.," in *AAMAS*, pp. 45–52, 2014.

[19] A. Baronchelli, "The emergence of consensus: A primer," *Royal Society Open Science*, vol. 5, no. 2, p. 172189, 2018.

[20] P. Jain and M. A. Goodrich, "Designing and predicting the performance of agent-based models for solving best-of-n," in *2023 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*, pp. 1076–1083, 2023.

[21] M. Mesbahi and M. Egerstedt, *Graph Theoretic Methods in Multiagent Networks*. Princeton University Press, 2010.

[22] G. Bernárdez, J. Suárez-Varela, A. López, X. Shi, S. Xiao, X. Cheng, P. Barlet-Ros, and A. Cabellos-Aparicio, "Magnneto: A graph neural network-based multi-agent system for traffic engineering," *IEEE Transactions on Cognitive Communications and Networking*, vol. 9, no. 2, pp. 494–506, 2023.

[23] X. Mo, Y. Xing, and C. Lv, "Heterogeneous edge-enhanced graph attention network for multi-agent trajectory prediction," *arXiv preprint arXiv:2106.07161*, 2021.

[24] W. Hamilton, Z. Ying, and J. Leskovec, "Inductive representation learning on large graphs," *Advances in neural information processing systems*, vol. 30, 2017.

[25] D. Nguyen, W. Luo, T. D. Nguyen, S. Venkatesh, and D. Phung, "Learning graph representation via frequent subgraphs," in *Proceedings of the 2018 SIAM International Conference on Data Mining*, pp. 306–314, SIAM, 2018.

[26] S. Marino, I. B. Hogue, C. J. Ray, and D. E. Kirschner, "A methodology for performing global uncertainty and sensitivity analysis in systems biology," *Journal of Theoretical Biology*, vol. 254, no. 1, pp. 178–196, 2008.

[27] J.-S. Lee, T. Filatova, A. Ligmann-Zielinska, B. Hassani-Mahmooei, F. Stonedahl, I. Lorscheid, and D. C. Parker, "The complexities of agent-based modeling output analysis," *Journal of Artificial Societies and Social Simulation*, vol. 18, no. 4, p. 4, 2015.

[28] J. Grazzini, M. G. Richiardi, and M. Tsionas, "Bayesian estimation of agent-based models," *Journal of Economic Dynamics and Control*, vol. 77, pp. 26–47, 2017.

[29] P. Jain and M. A. Goodrich, "Processes for a colony solving the best-of-N problem using a bipartite graph representation," in *Distributed Autonomous Robotics Systems, DARS*, 2021.

[30] P. Velickovic, G. Cucurull, A. Casanova, A. Romero, P. Lio, Y. Bengio, *et al.*, "Graph attention networks," *stat*, vol. 1050, no. 20, pp. 10–48550, 2017.

[31] K. You, M. Long, J. Wang, and M. I. Jordan, "How does learning rate decay help modern neural networks?," *arXiv:1908.01878*, 2019.