IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received June 3, 2020, accepted June 21, 2020, date of publication June 29, 2020, date of current version July 13, 2020.

Digital Object Identifier 10.1109/ACCESS.2020.3005359

BioMASS, a Spatial Model for Situated Multiagent Systems That Optimizes Neighborhood Search

CANDELARIA ELIZABETH SANSORES-PÉREZ[1], (Member, IEEE), AND JOEL ANTONIO TREJO-SÁNCHEZ[2]

[1]Complex Systems Simulation Laboratory, Universidad del Caribe, Cancún 77528, Mexico
[2]CONACyT—Centro de Investigación en Matemáticas, Mérida 97302, Mexico

Corresponding author: Candelaria Elizabeth Sansores-Pérez (csansores@ucaribe.edu.mx)

This work was supported by the Mexican Ministry of Public Education under the program "Programa Fortalecimiento de la Calidad Educativaa (PFCE)."

ABSTRACT We present BioMASS, a new model to implement a spatially explicit environment that supports constant-time sensory (neighborhood search) and locomotion functions for situated *multiagent systems* (MAS). In contrast, the spatial models currently provided by agent-based modeling and computer simulation (ABMS) platforms have computational costs that grow quadratically with perception range and linearly with the number of agents. To conserve computation, existing ABMS models of complex systems are oversimplified, by limiting the environment size, perception ranges, or the number of agents. BioMASS achieves constant time search and locomotion for the majority of function calls by implementing a linked list, nearest-neighbor data structure. This model makes the functions largely independent of the environment size, perception range, and the number of agents. We conduct a theoretical and experimental study of BioMASS compared to other spatial models. Experiments performed using a prey-predator swarm model show that BioMASS significantly outperforms the continuous and hybrid models in terms of execution time, allowing for higher-resolution modeling and simulation of complex systems.

INDEX TERMS Complexity analysis, multiagent system, simulation.

I. INTRODUCTION

Agent-based modeling and computer simulation (ABMS) is an approach to modeling the dynamics of complex systems. It is widely used to study these systems in a variety of scientific disciplines, such as economics [1]–[3], ecology [4]–[7], sociology [8]–[11] and biology [12]–[14]. Complex systems often exhibit self-organizing behavior and emergent properties as a result of the interactions among individual components. In ABMS, the behavior of individual components is modeled using abstractions such as autonomy, communication with other agents, the ability to perceive and respond to changes in the surroundings, problem-solving, conflict resolution, and the potential to make independent choices. By tracking the properties and decisions of a large number of individual agents, a computer simulation can reproduce the behavior of the complex, multi-agent system. In other words, the computer simulation encapsulates the behavior of a complex system in real-world scenarios through its algorithms, mathematical expressions, and equations. Experiments can then be performed by running the simulation with different populations of agents and observing directly the effect of these heterogeneous, autonomous entities interacting in a non-linear fashion in a shared environment.

Over the years, numerous agent-based modeling and simulation frameworks have been developed [15]–[22] to carry out such experiments in different domains. These frameworks address simulation aspects like discrete time-event scheduling, charting, support for geographic information systems (GIS), and 2D or 3D spatial modeling [23]–[26]. Although these platforms are frequently used they have no specific programming syntax for agent design and construction; these are general-purpose tools able to address multiple application domains.

In this context, we see the need for a spatially explicit ABMS framework that can represent an environment suitable for agents that support complex sensory and locomotion functions. For example, one use case is to model agents with a limited perception range that need to efficiently explore the environment beyond their immediate surroundings. These

The associate editor coordinating the review of this manuscript and approving it for publication was Donatella Darsena.

functions are of primary importance in a simulation that relies on accurately modeling local interactions among a group of different (possibly continuously moving) entities.

The classical method adopted to solve these types of problems involves a discrete virtual space (usually a two- or three-dimensional rectangular lattice) and a search algorithm whose computational cost per individual increases with the perception range and with the number of discrete cells in the search area. To reduce this cost, the number of cells must be kept low. This is done either by reducing the perception range or by increasing cell size (sacrificing detail in the environment representation). Another approach is to use a continuous space where the computational cost per agent increases linearly with the number of agents. However, models with a large number of agents (1000s or more) perform poorly under this scheme. To avoid very costly processing in this case, a collection of agents can be modeled as a group and represented as a special kind of agent. But this again sacrifices detail in the model representation, especially regarding the degree of heterogeneity among the agents.

Oversimplifying the spatial representation or the number of individuals may work for certain systems. However, in [4], Sansores *et al.* described several scenarios that require large spatial dimensions, fine spatial granularity and a large number of agents at the same time. Moreover, the perception range of the agents may change over time, so it is desirable to preserve the heterogeneity that forms the foundation of ABMS models. The combination of these requirements creates an additional level of complexity in the spatial model that is not addressed by any current ABMS framework. Our main contribution focus on the BioMASS spatial model.

A. CONTRIBUTION

In this paper, we address the stated problems of classical spatial models and provide a solution that can combine a large space with fine granularity while accommodating a large number of agents. The model also supports agents with widely varying and even dynamic neighborhood lookup ranges, without significantly affecting performance. The philosophy behind the design of the model is to make the time complexity of the agent functions independent of the size of the space and the number of agents, as much as possible.

The remainder of this article is as follows. Section II describes the problem that motivated this work. Section III presents a formal description of the ABMS multi-agent system. Section IV describes and analyzes the classical schemes used to explicitly represent the environment in an ABMS model. Section V describes the main contribution of this work. Section VI presents a benchmark simulation experiment to assess the performance of frameworks that implement the different spatial models and discusses the results. Finally, Section VII presents some concluding remarks and suggestions for future work.

II. PROBLEM DESCRIPTION

ABMS simulations are designed and implemented as multi-agent systems, to solve tasks in a distributed manner. Most ABMS models for the study of complex systems can be classified as either a behavioral simulation or an agent-based system simulation. These categories reflect different requirements in terms of spatial and time resolution, so they are used in different experimental scenarios.

In behavioral simulations, individual-level rules give rise to complex collective patterns. This early ABMS paradigm requires not many individuals: 10s to 100s at most. This paradigm requires fine spatial and temporal granularity to observe the individual behavioral interactions in the simulated environment. Typically, this type of simulation does not require large space and time dimensions, as demonstrated by classical swarm and predator–prey models [27]–[29].

Agent-based system simulations were conceived as a complementary tool to state variable models. In these models, the physiological processes and local interactions of individuals give rise to emergent properties in the population. This paradigm demands a large number of individuals, 1000s or more if possible, and implies a large-scale environment to accommodate those individuals. Further, since the goal is to generate emergent population dynamics, the simulation is run over a long time scale, usually several years or decades. For example, this permits the simulation to infer how a population changes over multiple generations. In contrast with behavior-based simulations, the temporal and spatial granularity are coarser. Trophic network models are an example of classical simulations in this category [30]–[32].

There are cases that require combining both paradigms, such as [4]. It is sometimes necessary to incorporate the behavioral factor as an additional differentiating aspect in agent-based systems simulations, especially when the behavior has a direct impact on the macro variables under study. However, such combined models make agent-based simulations computationally expensive. It becomes necessary to manage a large number of interacting agents, while keeping track of each agent's variability: its local interactions, complete life cycle, and adopted behavior in response to its changing internal state and the environment. An important property of these simulation models is that they are spatially explicit; therefore, perception and sensory functions are a fundamental form of interaction. The combination of large spaces with fine granularity also adds another level of computational complexity: the agents need to sense a large fraction of the environment. This requirement translates into neighborhood search functions, which are very costly. Further, the variability among agents implies that they can have different perception ranges, making it difficult to optimize this function in traditional spatial models.

The above discussion highlights the pressing need for a spatially explicit ABMS framework that can efficiently handle neighborhood exploration and locomotion functions.

III. ABMS MULTIAGENT SYSTEM FORMALIZATION

ABMS simulations are founded on multiagent system design principles. This MAS formalization is general and applicable to a wide range of systems. However, it is specifically tailored to models requiring an explicit representation of the spatial environment. Therefore, we emphasize the agents' awareness of their location in the environment and allow the agents to have heterogeneous and dynamic perception skills. Also, we stress a physical description of the environment.

The following multiagent system is inspired by [33] and is defined as a synchronous system consisting of:

- A set of agents $A = \{a_1, a_2, \ldots, a_n\}$ where a_i is any given agent for $1 \leq i \leq n$
  - Each agent a_i behaves according to a finite set of disjoint actions $Ac_i = \{ac, ac', \ldots\}$
- An environment is modeled by $E = \{S, e_0, \tau\}$ where the set of agents A is situated and where the agents live.

The environment is defined as a triple: the set of all possible states $S = \{e, e', \ldots\}$ the environment during the MAS execution, the initial state that describes the environment at the beginning of the simulation, and a transformation function τ, that represents the effect of agents' actions on the environment. The future state of the environment is difficult to predict from the initial conditions, because the state transitions are determined by agents' actions.

The decision-making mechanism [34], [35] of each agent a_i is based on a three-phase, cyclic process: (1) perception, (2) deliberation, and (3) action. In the perception, agents sense the environment where they are situated. In the deliberation, an internal mechanism selects the action (based on the agent's internal state and the perceived environment's state) from a limited repertoire of actions or behaviors. As the purpose of this section is not to explain the modeling details of existing agent architectures, here we will just assume that the MAS implementation chooses the most appropriate deliberation mechanism for the problem: reactive [36], cognitive [37], or hybrid [38], according to the problem it addresses.

In a situated MAS, the agents are sensitive to spatial relationships in the sense that these relationships can impose constraints and limit their available actions. Thus, concepts like location, perception scope and neighborhood are of primary importance. Each agent $a_i \in A$ is represented by a vector of variables $a_i.\vec{v}$ which are attributes of an agent a_i. The local state $a_i.st_k$ of agent a_i, represents all the values of each variable in $\vec{v}$ at time k. Each agent consists of:

- A physical location $a_i.loc_k$, of a_i in E at step k.
- A perception scope $a_i.perception_k$ (integer or real) defined as the distance r that agent a_i can sense to look for other agents at time k. Agent a_i perceives a_j at time k if the distance to agent a_j from agent a_i ($dist(a_i, a_j)$) is less than or equal to r. Note that the perception scope $a_i.perception_k$ can be a dynamic variable.
- A neighborhood $N(a_i)^r_k$. Set of agents perceived by agent a_i at step k; *i.e.*, $N(a_i)^r_k = \{a_j | dist(a_i, a_j) \leq r\}$. The distance $dist(a_i, a_j)$ is defined according to the implementation of the environment in the MAS model (discrete, continuous, or hybrid).

The execution of the MAS is divided into k rounds or steps. During the k-th round the system state passes from $MAS.st_{k-1}$ to $MAS.st_k$, then all agents perform their deliberation functions to select actions. The global state of the MAS is the union of the states of every agent in A and the environment state S_k at step k; *i.e.*, $MAS.st_k = \{a_1.st_k \cup a_2.st_k \cup \ldots \cup a_i.st_k \cup S_k\}$. The system starts execution at state $MAS.st_0$. Finally, an action in Ac_i is modeled as a transition function $f(a_i) : (a_i.st_k, MAS.st_k) \rightarrow a_i.st_{k+1}$ which determines the new configuration of agent a_i at step $k+1$. The transition function $f(a_i)$ receives as input the state a_i and the configuration of the multiagent system at step k.

In the next section, we present the most common computational models used by MAS to represent spatially explicit environments, and discuss how they are implemented.

IV. CLASSICAL SPATIAL MODELS FOR SITUATED MAS

Traditionally, the internal model of the environment in a situated MAS is either discrete or continuous. This choice determines how the agents can be physically organized and the forms of the sensory and locomotion functions used by the agents. We will also discuss the hybrid model introduced by [23]. This model uses partial *discretization* of the underlying continuous model in order to improve the execution performance of some functions such as neighborhood search.

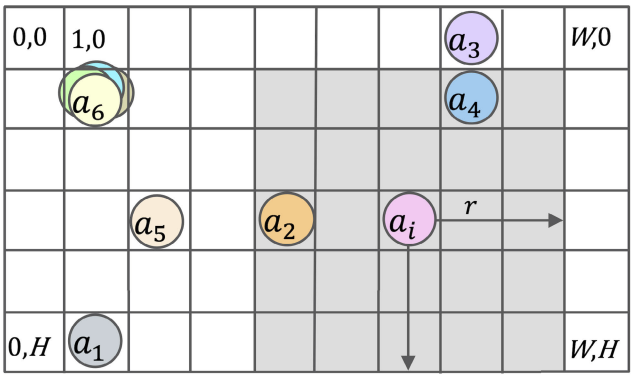

FIGURE 1. A discrete spatial model E of $(W \times H)$ cells containing a set of agents $A = \{a_1, a_2, a_3, a_4, a_5, a_6, \ldots, a_i\}$. The perception scope of a_i at step k is the shaded subgrid. The neighborhood lookup function of a_i iterates over all cells in the shaded area and adds all agents occupying those cells to its neighborhood set $N(a_i)^r_k = \{a_2, a_4\}$.

A. DISCRETE MODEL

The discrete model is the most widely supported by simulation tools. The simplicity of its implementation makes it very intuitive to use for most domain experts. Basically, it depicts the environment where the agents live as a grid of $W \times H$ cells (Fig. 1). Each agent occupies at most one cell and can move from cell to cell. In some uncommon variants agents can move to cells that are not adjacent. One cell can contain more than one agent, a feature known as *multi-occupancy*. Cell (1, 1) in Fig. 1 is occupied by multiple agents. The basic implementation of multi-occupancy attaches a list of agents

Algorithm 1 Algorithm for Neighborhood Lookup of Agent a_i at Distance $r = a_i.perception_k$ in a Discrete Spatial Model

Input: The agent a_i and grid $E_{W\times H}$.
Output: The set of neighbors at distance r at step k, $N(a_i)^r$.

```
r = a_i.perception_k
N(a_i)^r_k ← ∅
for k ← a_i.x − r to a_i.x + r do
    for l ← a_i.y − r to a_i.y + r do
        N(a_i)^r ← {N(a_i)^r ∪ a_j} for all a_j located in E_(k,l)
    end for
end for
return N(a_i)^r
```

to each cell. Therefore, the computational complexity of a multi-occupancy model would increase in proportion to the basic list of operations required to keep track. We now present a brief description of the functions *insert*, *remove*, *move*, and *neighborhood lookup* that must be performed by each agent in a discrete spatial model.

- *Insert*. This action occurs when agent a_i first enters the environment E at step k. The agent contains its own location $a_i.loc_k$, consisting of a column $col \leq W$ and a row $row \leq H$. The complexity of this action is constant $O(1)$, even in the case of multi-occupancy.
- *Remove*. To remove agent a_i from the environment E at step k, the MAS obtains its location $a_i.loc_k$ and eliminates the agent from its cell. The complexity of this action is $O(1)$. In the case of multi-occupancy, this function takes $O(n)$ in the worst case, to update the list of agents which contains at most n elements.
- *Move*. Let $a_i.loc_k = (col, row)$ be the location of agent a_i at step k, and let $a_i.loc_{k+1}(col', row')$ be the new location of a_i. The system updates the value as $a_i.loc_{k+1} = (col', row')$ and stores agent a_i in the list of the cell at the new position. This function can be seen as a combination of the insert and remove functions. The complexity of this action is typically $O(1)$. The worst case is for the *move* function to take $O(n)$ when there is multi-occupancy in the starting position.
- *Neighborhood lookup*. Obtaining the neighborhood in the discrete model is computationally expensive. To avoid very costly processing, the search space tends to be reduced to a few surrounding cells. Let $N(a_i)^r_k$ be the neighborhood of agent a_i within distance r at step k, where $r = a_i.perception_k$. To obtain $N(a_i)^r_k$, the search algorithm traverses all cells whose edges intersect the perception scope of agent a_i, as shown in Fig. 1. All agents a_j in the searched cells are added to the neighborhood $N(a_i)^r_k$. In pseudo-code 1 we present the algorithm to perform the neighborhood lookup.

  Now, we show in Lemma 1 that the time complexity of the neighborhood lookup is $O(W^2)$. Without loss of generality, we assume that $W > H$. If multi-occupancy is allowed, the cells can contain a list of $O(n)$ agents, and all agents can be contained in $N(a_i)^r_k$.

Lemma 1: A neighborhood lookup for agent a_i takes $O(W^2)$ time units in the discrete model.

Proof: Note that $a_i.perception_k$ can be as large as W. An agent a_i traverses the entire sub-grid within its perception scope ($O(W^2)$ cells in the worst case). Therefore, a_i performs $O(W^2)$ operations to obtain $N(a_i)^r_k$. □

Hence, when both, the number of agents, and their perception is large, the simulation becomes very slow.

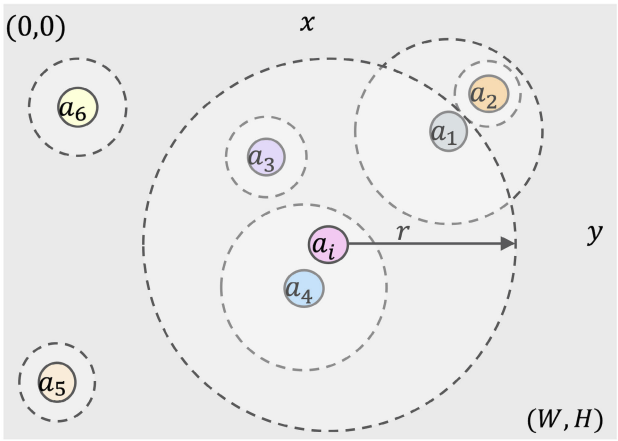

FIGURE 2. A continuous spatial model with Cartesian coordinates. The set of agents $A = \{a_1, a_2, a_3, a_4, a_5, a_6, \ldots, a_i\}$ is situated in the space and their different perception scopes are depicted by dotted lines. The neighborhood set of agent a_i is $N(a_i)^r_k = \{a_1, a_3, a_4\}$. Note that the neighborhood relation is not symmetric: agents a_1 and a_3 do not have agent a_i in their neighborhood.

B. CONTINUOUS MODEL

This model is a plane with Cartesian coordinates. Let $\mathcal{C}$ denote this plane with dimensions $W \times H$, as depicted in Fig. 2. The agent locations (x, y) are implemented as a hash table indexed by the agent identifiers. That is, the position $a_i.loc_k$ of agent a_i at step k is stored in the hash table $\langle id, (x, y)\rangle$, not with the agent. Note that the number of locations stored in the hash table is the same as the number of agents, and hence independent of the space size (unlike the discrete model, where the number of cells is the size of the space).

The continuous space also uses an auxiliary hash table of the form $\langle coord, list\rangle$, where $coord = (x, y)$ is the key and *list* is a list of agents sharing the same position. This hash table is only used in case of multi-occupancy to make related queries more efficient. (For example, the query of how many agents are located at coordinates (x, y) is inexpensive to perform if the auxiliary hash table exists, but requires $O(n)$ time of the main hash table otherwise. Regarding time complexity, basic functions like *insert*, *move* and *remove* involve updating both hash tables. The *neighborhood lookup* function uses only the main hash table $\langle id, coord\rangle$ for both the single occupancy and multi-occupancy cases. Now, we explain how the continuous model performs these operations for each agent a_i, and analyze their time complexity.

- *Insert.* Let a_i be a new agent in the environment at step k, and let (x, y) be its position. The continuous model inserts an item $\langle a_i, (x, y)\rangle$ into the main hash table. The time complexity of this function is $O(1)$. In the case of multi-occupancy, an entry is also inserted in the auxiliary hash table, as a one-element list $[a_i]$ with key $coord = (x, y)$. If the key already exists, then a_i is inserted at the end of the list with that key. In both situations the time complexity is $O(1)$.
- *Remove*. To remove an agent a_i from $coord = (x, y)$ at step k, the entry $\langle a_i, coord\rangle$ pair is deleted from the main hash table. The time complexity of this operation is $O(1)$. With multi-occupancy, an extra operation is required to delete agent a_i from the list in the auxiliary hash table with key $= coord$. If a_i is the only agent in the list, then the entry $\langle coord, list\rangle$ is eliminated. The time complexity for this operation remains $O(1)$. Otherwise, the list is traversed to find and remove agent a_i. The time complexity for this case is $O(n)$.
- *Move.* Let a_i be an agent situated at (x, y) during step k, and let (x', y') be its new coordinates. First, this function computes the index of the main hash table where the pair $\langle a_i, coord\rangle$ will be modified, then it replaces the value $coord$ to the new value (x', y'). The time complexity of this operation is $O(1)$. For the case of multi-occupancy, two extra operations have to be realized in the auxiliary hash table: agent a_i has to be inserted in its new coordinates (x', y') and removed from the old coordinates (x, y). Both operations were described in the *insert* and *remove* functions. Therefore, the complexity for the *move* function is at most $O(n)$.
- *Neighborhood lookup.* Let $N(a_i)_k^r$ be the neighborhood of agent a_i within a Euclidean distance r at step k, where $r = a_i.perception_k$. Fig. 2 illustrates agent a_i situated in the continuous space. Its perception scope r is shown by the dotted line. The pseudo-code 2 shows the algorithm to iterate over n elements in the main hash table $\langle id, coord\rangle$ and identify agents a_j whose Euclidean distance from a_i is less than or equal to r. The time complexity of this function is always $O(n)$.

C. HYBRID MODEL

We named this type of model *hybrid* since it is a Cartesian plane, like the continuous model, but also uses a kind of *discretization*. It is designed to make the neighborhood lookup more efficient. Let Ω be the space in the hybrid model. Assume that there exists a continuous space with dimension $W \times H$. The hybrid model Ω discretizes the continuous space by defining a grid of $l \times m$ non-overlapping subspaces called "buckets" $\mathcal{B} = \{B_{0,0}, B_{0,1} \ldots, B_{l,m}\}$, as shown in Fig. 3. Without loss of generality we assume that each bucket is a square. The *discretization value* Δ defines the granularity of the space. Without loss of generality, we assume that Δ is a divisor of W and H. Thus, $l = \frac{W}{\Delta}$ and $m = \frac{H}{\Delta}$.

Algorithm 2 Algorithm for Neighborhood Lookup of Agent a_i at Distance $r = a_i.perception_k$ in a Continuous Space

Input: The agent a_i, the space $\mathcal{C}$.
Output: The set of neighbors at distance r at step k, $N(a_i)_k^r$.

```
1:  N(a_i)^r ← ∅
2:  for all (a_j, coord) ∈ C do
3:      h ← hash(a_j)
4:      coord ← C.get(h)
5:      a_j.loc = coord
6:      if dist(a_i, a_j) ≤ r then
7:          N(a_i)^r ← {N(a_i)^r ∪ {a_j}}
8:      end if
9:  end for
10: return N(a_i)^r
```

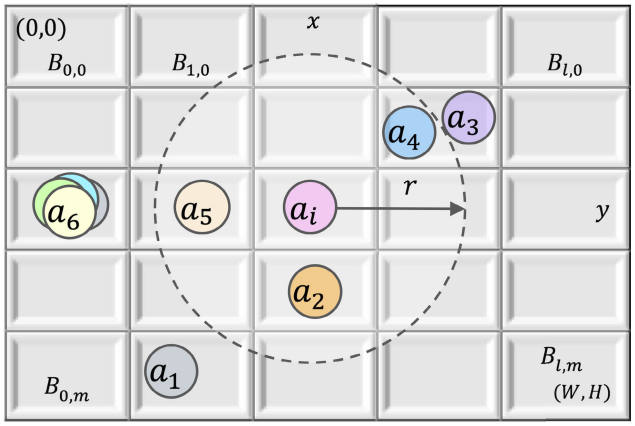

FIGURE 3. A hybrid spatial model with a continuous space divided into buckets. The set of agents A is situated in the space with real Cartesian coordinates. The neighborhood set of agent a_i is $N(a_i)_k^r = \{a_2, a_4, a_5\}$. Note that agent $a_3 \notin N(a_i)_k^r$: although it happens to be in a bucket that is perceived by a_i, it is not within the Euclidean distance r.

The implementation of the hybrid model uses a hash table to represent Ω. The agents contained in bucket $B_{l,m}$ are designated by a list in the hash table $(bucket, list)$ where $bucket$ is an integer coordinate pair (w, h) and $list$ is a list of agent identifiers. As in the previous models, we will analyze the basic functions performed by agent a_i with Cartesian coordinates $a_i.loc$. Notice that the variables in the description are normalized with the *discretization* value Δ, in this way, the operations are realized in terms of buckets and not in terms of the real coordinates of the agents.

- *Insert.* Let a_i be a new agent to be inserted in the environment at step k, and (x, y) be the Cartesian coordinates of its location. An agent a_i is inserted into a bucket $B_{w,h}$ if its location is contained within the area of $B_{w,h}$. The integer coordinates of the bucket containing (x, y) are $w = \lfloor x/\Delta \rfloor$ and $h = \lfloor y/\Delta \rfloor$. The pair (w, h) is the index of the hash table $(bucket, list)$, and the action is completed by either creating a new, one-element list with a_i at this index or by inserting a_i at the end of the existing

list. In this way, all agents whose coordinates fall into the same $B_{w,h}$ are grouped together. The time complexity of this function is $O(1)$.

- *Remove*. Let a_i be the agent to be removed from the environment at step k, and (x, y) its location. The integer coordinates of its bucket are $w = \lfloor a_i.x/\Delta \rfloor$ and $h = \lfloor a_i.y/\Delta \rfloor$. The pair (w, h) is the index of the list in the hash table $(bucket, list)$ where the agent can be found. The *list* is searched to remove agent a_i. The time complexity of the remove function is $O(n)$.
- *Move*. Let a_i be an agent situated in the hybrid model at coordinates (x, y) and let (x', y') be its new location. Also, let $B_{w,h}$ be the current bucket of agent a_i. If the new location is situated in the same bucket, $(x', y') \in B_{w,h}$, then it is not necessary to perform any operation on the table $(bucket, list)$. The change in the real coordinates is handled by replacing the values of $a_i.x$ and $a_i.y$. Thus, the time complexity of the *move* function when it does not involve a change of bucket is $O(1)$. If $(x', y') \notin B_{w,h}$, it is necessary to perform the *insert* and *remove* functions on the hash table $(bucket, list)$ to reflect the change of buckets in the discrete space. The real coordinate value of a_i also has to be updated. Therefore, the complexity for the *move* function involving a change of buckets is $O(n)$.
- *Neighborhood lookup*. This function is similar to that of the discrete model, in that its time complexity depends strongly on the *discretization* size. Let $N(a_i)_k^r$ be the neighborhood of agent a_i within a Euclidean distance r at step k, where $r = a_i.perception_k$. Also, let $B_{w,h}$ be the current bucket of a_i. The algorithm to obtain $N(a_i)_k^r$ first finds all buckets whose boundaries intersect the perception scope of agent a_i, as shown in Fig. 3. Let $B(a_i)_k^r = \{B_{0,0}, B_{0,1} \ldots, B_{col,row}\}$ be the subset of buckets perceptible to agent a_i at step k. For each bucket in $B(a_i)_k^r$, the system retrieves the corresponding list from the hash table $\langle bucket, list \rangle$ and traverses the list to identify the set of agents a_j such that $dist(a_i, a_j) \leq r$, in Pseudo-code 3 we present this algorithm. If the bucket size is much larger than the typical size of a neighborhood lookup, then a bucket will include large numbers of agents that are not in the neighborhood of a_i. On the other hand, if the bucket size is much smaller than the typical size of a neighborhood lookup, then many buckets have to be inspected to cover the perception scope of a_i. This can be highly inefficient, with a time complexity similar to that of the discrete model. As we show in the next lemma, the time complexity of this function is $O(n \times W^2)$, considering that there are W^2 buckets and that each bucket can contain a list of $O(n)$ agents.

Lemma 2: A neighborhood lookup for agent a_i takes $O(n \times W^2)$ time units in the hybrid spatial model.

Proof: Without loss of generality, assume that $W > H$. Notice that the perception scope of agent a_i can be as large as W. Each agent must traverse all buckets within its perception scope ($O(W^2)$ in the worst case). Therefore, a_i performs $O(W^2)$ operations to obtain the subset $B(a_i)_k^r$. Since each bucket has a list of $O(n)$ agents, the neighborhood lookup function for a_i can take $O(n \times W^2)$ time units. □

Algorithm 3 Algorithm for Neighborhood Lookup of Agent a_i at Distance $r = a_i.perception_k$ in the Hybrid Space

Input: The agent a_i, the space Ω and the *discretization* value Δ.

Output: The set of neighbors at distance r at step k, $N(a_i)_k^r$.

```
r ← a_i.perception_k
rdisc ← r/Δ
w ← a_i.x/Δ
h ← a_i.y/Δ
N(a_i)^r ← ∅
for col ← floor(w − rdisc) to floor(w + rdisc) do
  for row ← floor(h − rdisc) to floor(h + rdisc) do
    h ← hash(col, row)
    L ← Ω.get(h)
    for k ← 1 to length(L) do
      a_j ← L(k)
      if dist(a_i, a_j) ≤ r then
        N(a_i)^r ← {N(a_i)^r ∪ {a_j}}
      end if
    end for
  end for
end for
return N(a_i)^r
```

If the simulation model is very dense, with many agents in each bucket, this model [23] recommends a *discretization* equal to the maximum perception scope of an agent a_i. On the other hand, if the model is very sparse, it recommends a *discretization* value equal to twice the maximum perception scope. However, we have noticed an important issue in the agent-based modeling approach that is not addressed by existing simulation tools. In the study of complex systems, the model can have individual agents with widely varying perception scopes, and hence different needs for optimizing the neighborhood lookup. The previous recommendation is only helpful for the case of homogeneous individuals with the same perception range. Further, there is still the question of how to address dynamic perception ranges, since all the previous approaches assume that agents do not change their perception range during the simulation. In the next section we propose a new spatial model that addresses these issues.

V. THE PROPOSED BioMASS SPATIAL MODEL

Let $\mathcal{L}$ be the space in the BioMASS model with dimensions $W \times H$. $\mathcal{L}$ can be portrayed as a Cartesian plane with real coordinates. The implementation uses a quadruply linked list. The nodes forming the list are agents with four pointers: two in the horizontal dimension and two in the vertical dimension.

The pointers *prevx* and *nextx* link to the agents with the next smallest and next largest x coordinate, respectively. A node points to itself if there is no other agent with a smaller/larger x coordinate. Initially, $\mathcal{L}$ is configured with two "sentinel" agents a_0 and a_1, which represent the corners of the 2D space with coordinates $(0, 0)$ and (W, H).

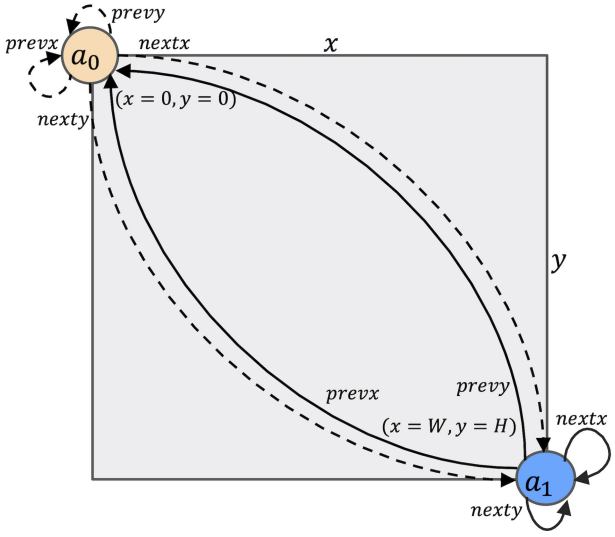

FIGURE 4. The quadruply linked list in $\mathcal{L}$ representing the BioMASS spatial model with a_0 and a_1 as sentinel nodes delimiting the simulation space. The depicted links between nodes are valid only when the space is empty.

The initial setup is illustrated in Fig. 4. The sentinel node a_0 has 2 pointers to itself, *prevx* and *prevy*. The pointers, *nextx* and *nexty*, point to a_1. Similarly, for a_1 the two 'previous' pointers point to a_0, while its 'next' pointers point to itself. Thus, $\forall a_i \in \mathcal{L}$, $a_i.prevx$, $a_i.nextx$, $a_i.prevy$, and $a_i.nexty$ are pointers to its four closest agents in the horizontal and vertical dimensions. Therefore, $\forall a_i \in \mathcal{L}$, we have $a_i.x <= a_i.nextx.x$, $a_i.y <= a_i.nexty.y$, $a_i.x >= a_i.prevx.x$, and $a_i.y >= a_i.prevy.y$. This data structure guarantees that the agents situated in the environment are always linked to the closest agents in both dimensions. Now, we will analyze the basic functions performed by an agent a_i in $\mathcal{L}$.

- *Insert.* Let a_i be a new agent to be inserted at step k, and (x, y) the Cartesian coordinates of its location $a_i.loc$. First, a_i must be situated in the horizontal dimension, by finding the two closest agents in x. To do so, the system transverses $\mathcal{L}$ by starting from a_0 (a_1) and following the chain of *nextx* (*prevx*) pointers. This process stops at the first occurrence of a node with $x \geq a_i.x$ ($x \leq a_i.x$). That node becomes the next(prev) node with respect to a_i, and all pointers in the horizontal dimension are re-linked to insert a_i in the list. The same operations apply to insert agent a_i in the vertical dimension. Fig. 5 and Fig. 6 illustrate the insert function. The time complexity of this function is $O(n)$ in the worst case.
- *Remove.* Let a_i be the agent to be removed from $\mathcal{L}$ at step k. Independently of its location, agent a_i abandons its position in the linked list by updating all the references to itself from the previous and next agents in both dimensions. This function is time complexity $O(1)$.
- *Move.* Let a_i be an agent situated in the space $\mathcal{L}$ at coordinates (x, y) during step $k - 1$, and let (x', y') be its new Cartesian coordinates during step k. First, the *prevx* or *nextx* linked list is traversed (depending on where the x' coordinate can be reached through) starting from a_i, up to the first node whose x coordinate is less than x'. That node becomes the reference node after which agent a_i will be inserted in the horizontal dimension after removing a_i from its current position on

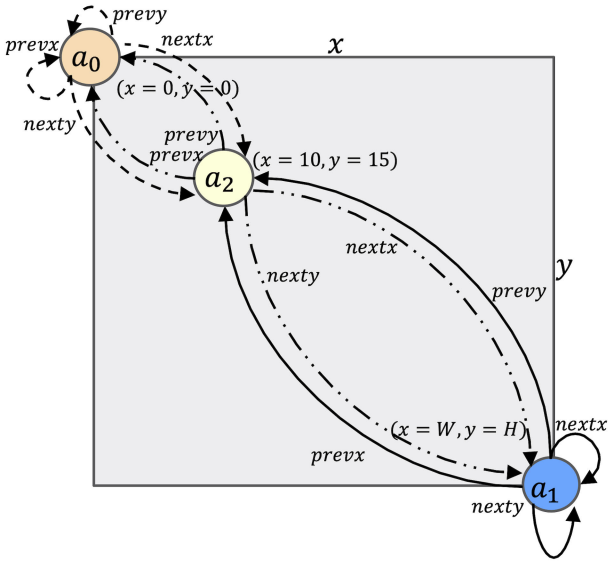

FIGURE 5. The space $\mathcal{L}$ with only one agent, a_2, pointing to sentinels a_0 and a_1 in both dimensions as the previous and next agent nodes respectively.

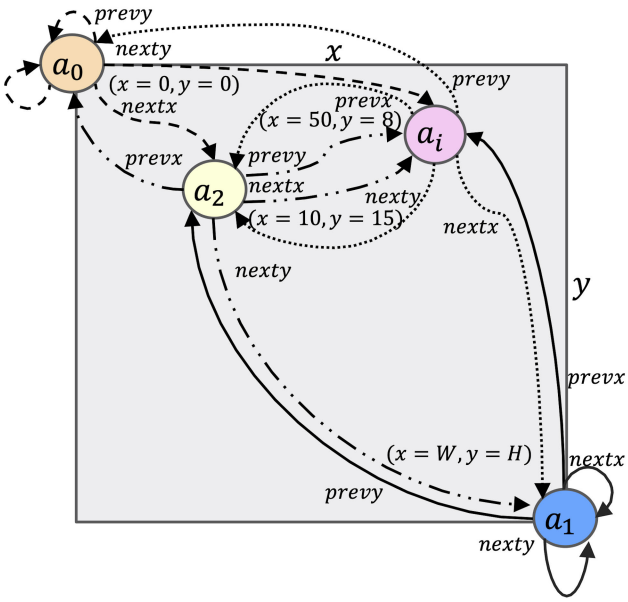

FIGURE 6. The space $\mathcal{L}$ after inserting a_i at coordinates (50, 8). Note that, the links of a_i do not necessarily point to the same agent in both dimensions.

the list. This steps apply for the vertical dimension too. Note that in realistic simulations of complex systems (see Section II), the movement of agents is smooth: changes in position at each time step are small (traverse a constant number of agents on average) so that local interactions of the agents can be modeled at high resolution. Thus, the *move* function takes $O(1)$ in most circumstances. However, we do not rule out the worst case, where the *move* function takes $O(n)$ because it needs to traverse a large portion of the list. We will see this occurring frequently in any situation where the agents are densely occupying a small portion of the space and move quickly relative to their local density. If this happens frequently, it is a sign that the simulation is not set up correctly, because its resolution is not high enough to model the short-range interactions correctly. Another extreme case might happen if agents could teleport from one end of the space to the other in a given dimension. This is not the case for most ABMS of complex systems.

- *Neighborhood lookup.* Let $N(a_i)_k^r$ be the neighborhood of agent a_i within a Euclidean distance r at step k, where $r = a_i.perception_k$. The algorithm to obtain $N(a_i)_k^r$ is as follows. First, it traverses the list in the x dimension in both directions starting from a_i, looking for nodes whose x value is within r from $a_i.x$. For each node that complies with this condition, its two-dimensional Euclidean distance is calculated to verify that the node is indeed within the perception range of a_i. If so, it is included in the set $N(a_i)_k^r$. This process is illustrated in Fig. 7. Observe that agents $\{a_2, a_3, a_4\}$ are all within a distance r from a_i in the x dimension, but not all are within the perception scope. Therefore, the neighborhood set is $\{a_3, a_4\}$. The pseudo-code 4 shows that this calculation only requires a few operations. The time complexity of this function is usually $O(1)$, due to the fact that agent nodes are ordered by proximity in the list. An agent's neighbors in space are also neighbors in the list, so the search time is roughly constant. The worst case for this function is $O(n)$, when $N(a_i)_k^r = \{A \setminus \{a_i\}\}$; that is, when all agents in the space are neighbors of a_i. This case is very rare and not realistic for ABMS simulations. For this to happen, the agent would have to either perceive the entire space or all the agents would have to be concentrated in a small space comparable to the perception range of a_i. Neither situation is common in a properly configured complex system model. However, in such an uncommon case, the *neighborhood lookup* function would not be necessary, because it would be more efficient for agents to have global access to the list of all agents in the simulation.

Table 1 summarizes the time complexities of the different spatial models. BioMASS has the best theoretical performance with respect to the time complexity of the *neighborhood lookup* function, but the time complexity of its *insert* function is high compared to the other models. However, most simulation experiments are not limited by the insertion time

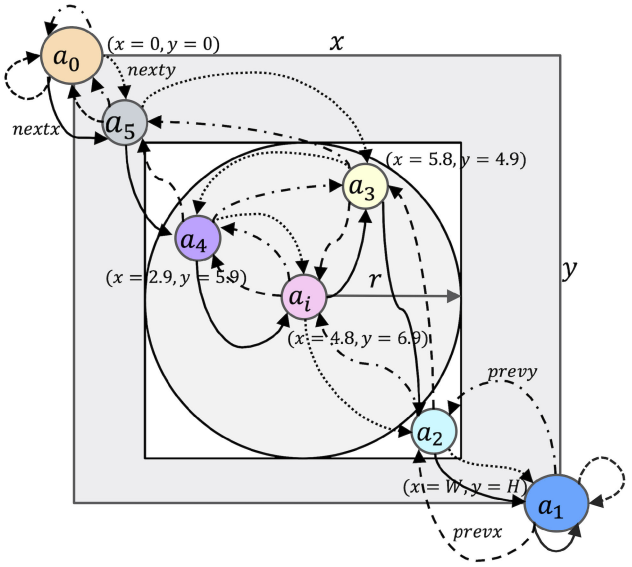

FIGURE 7. BioMASS spatial model depicting the neighborhood $N(a_i)_k^r = \{a_3, a_4\}$. The area perceived by agent a_i at step k is a disk of radius $r = a_i.perception_k$. Note that although agent a_2 is identified as a candidate neighbor while traversing the linked list in both dimensions, its Euclidean distance from a_i is greater than r.

TABLE 1. Time complexity of agent functions provided by the spatial models and widely supported by agent-based simulation toolkits.

Function	BioMASS Space	Discrete Space	Continuous Space	Hybrid Space
Insert	$O(n)$	$O(1)$	$O(1)$	$O(1)$
Remove	$O(1)$	$O(1)$	$O(1)$	$O(n)$
Move	$O(1)$	$O(1)$	$O(1)$	$O(n)$
Neighborhood lookup	$O(1)$	$O(W^2)$	$O(n)$	$O(n \times W^2)$

of agents, since this is done just once to set up the initial scenario. Rather, to model the dynamics of complex systems with local interactions, the sensory function is of primary importance since it is used most intensively. In the case of the BioMASS space, the time complexities of the *move* and *neighborhood lookup* functions are both $O(1)$.

As we explained before, the worst case implies that the density of agents is high compared to the perception scope. Such cases are rare in the simulation of complex systems. When they arise, it often means that the resolution of the simulation is unsuitable for the problem. Furthermore, if they do happen frequently, the functions can be implemented differently. For example, if agents are able to perceive the whole space then it is simpler to provide them with a list of all individuals in the space. In contrast, in the continuous space the *neighborhood lookup* function is always $O(n)$ for all agents, independently of the perception range. In the hybrid and discrete spaces, time complexity in $O(n)^2$ are also rare (when agents perceive most of the environment), the *neighborhood lookup* function is still a quadratic function of the perception, which make them less efficient than the worst cases of the other spaces. Hence, theoretically the BioMASS *neighborhood lookup* function should be the most efficient.

Algorithm 4 Algorithm for Neighborhood Lookup of Agent a_i at Distance $r = a_i.perception_k$ in the BioMASS Space

Input: The agent a_i, the space $\mathcal{L}$.
Output: The set of neighbors at distance r at step k, $N(a_i)_k^r$.

```
r = a_i.perception_k
N(a_i)^r ← ∅
node ← a_i.nextx
while node.x − a_i.x ≤ r and node ≠ a_1 do
    if dist(a_i, node) ≤ r then
        N(a_i)^r ← {N(a_i)^r ∪ {node}}
    end if
    node ← node.nextx
end while
node ← a_i.prevx
while a_i.x − node.x ≤ r and node ≠ a_0 do
    if dist(a_i, node) ≤ r then
        N(a_i)^r ← {N(a_i)^r ∪ {node}}
    end if
    node ← node.prevx
end while
return N(a_i)^r
```

VI. EXPERIMENTATION

In this section, we compare the four spatial models through a case study: the classical swarm model proposed by [39]. The case study simulates a large flock of prey birds that attempt to avoid a small number of predator birds flying in the same environment. Individual prey and predator birds are modeled as autonomous agents living in a 2D toroidal (opposite boundaries of the space are joined) continuous space. The behavior of the agents is modeled as follows:

- *Prey.* This agent tries to stay close to prey within its perception scope and avoid predators within a fear radius. It finds neighboring prey agents and builds an ordered list of the closest neighbors, which will be updated in every step of the simulation. Therefore, a prey agent needs to call the *neighborhood lookup* function each step. The prey agent then calculates avoidance or attraction vectors to each one. With respect to other prey, the agent flies away (flies closer) if the neighbor is too close (far away). Hence, this vector modifies the agent's own velocity. Next, the prey avoids predators. It loops through all predators and gets the distance vector to each one. If a predator is within the prey fear radius, it adds an avoidance vector that also modifies the agent's velocity. Finally, the prey moves to the location determined by its new velocity vector.
- *Predator*. In every step of the simulation, a predator agent chooses a random prey and attempts to eat it. The perception scope of the predator is considered large enough to perceive all prey in the environment. The prey information is available during the whole simulation as a list. If the predator already has a target from the previous step, it first checks whether the magnitude of the distance vector to the target is within its kill radius. If so, the current target is considered dead and the predator randomly selects a new prey. Then it calculates the distance vector to the new prey and modifies it to take into account acceleration over time. The result is the attack vector. Finally, it adds the attack vector to its own velocity vector to chase the prey. If its velocity exceeds the maximum speed, it is reduced to the maximum speed threshold. Finally, the predator moves in space to the location determined by its new velocity vector.

Since the BioMASS spatial model implements a continuous space with Cartesian coordinates, we decided to compare it only to similar models. That is, we did not test the swarm scenario with the discrete spatial model, which performs poorly in large spaces with neighborhood lookup ranges beyond the agent's surrounding cells. We selected two simulation platforms for comparison, Repast [25] and MASON [23]. Our selection criteria were as follows: (1) the platform supports one of the two models with Cartesian coordinates, continuous or hybrid; (2) it is a general-purpose simulation tool; and (3) it is an ongoing project still releasing new versions. Also, the Java programming language was an important requirement to standardize the experiments between simulation platforms. Finally, while selecting the platforms we also took into consideration less critical characteristics such as robustness, running speed, and open-source support.

A. BENCHMARK

We performed a set of experiments comparing the BioMASS model to the continuous and the hybrid models. We set up the following conditions. There exist 6000 prey and 4 predators. The width of the space is 1200 and the height is 800. Notice that the fact that the BioMASS spatial model is not grid-based allows us to adopt larger spatial dimensions without sacrificing performance. The perception scopes of the prey can be homogeneous or heterogeneous. The perception scope of the predators is constant since they perceive all prey. The two configurations are as follows:

- *Homogeneous perception scope*. The objective of this configuration is to evaluate the performance of the models assuming that all preys have an identical perception scope of $preyPerception = 200$.
- *Heterogeneous perception range*. The objective of this configuration is to test the performance of the models when agents have a range of perception scopes. There exist complex system models where the perception scope varies across agents and over time. We adopt a simple way of testing performance under these conditions. We choose a high value and a low value for the perception scope, and assigned each value to one half of the prey population. Specifically, the initial setup contains 3000 prey with $preyPerception = 20$ and 3000 prey with $preyPerception = 200$. This variability is sufficient

to test the efficiency of the neighborhood search methods within a large space and with a large number of agents. In models with dynamic perception scopes, there will be additional instructions in a simulation step to update these values, but these instructions are not dependent on the current scope range or the size of the space. Therefore, this configuration is also a reasonable benchmark for dynamic perception models.

We used Repast for the continuous model (we refer to this model as Repast). With respect to choosing the discretization parameter of the hybrid model, we are not able to define the space as dense or sparse in advance ([23]), and in fact we do not expect a single value of the discretization to be suitable for every agent at all time steps. Therefore, we performed two sets of experiments for each configuration using MASON. In one experiment we define a *discretization* value equal to the maximum perception scope an agent is likely have ($d = 200$ for the homogeneous experiment), reflecting the assumption that the spatial model is usually dense. In another experiment, we define a *discretization* value equal to twice the maximum perception scope ($d = 400$ for the homogeneous experiments), reflecting the assumption that the spatial model is usually sparse. We refer to these models as Mason200 and Mason400 respectively. Additionally, we perform hybrid model experiments for the heterogeneous configuration with *discretization* values of $d = 20$ (Mason20) and $d = 40$ (Mason40), corresponding to the lower value of the perception scope. Finally, we refer to our model as BioMASS. All simulations were executed on an iMac with a 3.2 GHz Quad-Core Intel Core i5 processor. The iMac had 16GB of memory and was running MacOS Catalina.

The hybrid models are at a disadvantage compared to the continuous and BioMASS models, because the *discretization* parameter is constant and must be chosen in advance. In fact, taking into account the suggestions of the authors of MASON, we conclude that the *discretization* value has no meaning in a complex model meeting one of the following conditions: (1) the individuals have many different neighborhood lookup ranges; or (2) the agents' movement in space forms clusters dynamically, so the space is dense in some zones and sparse in others. The heterogeneous Predator-Prey swarm model used in this experiment meets both conditions. For instance, in Fig. 8 the space is covered by small patches with a dense distribution of agents, and also numerous small areas without agents. In Fig. 9, however, there are large areas without agents that would be ideal for a hybrid model with a large *discretization* value, but also some dense areas where a small *discretization* value would be better. Both cases were taken from the same configuration and same run of the experiment, at different time steps.

We performed 500 simulation runs to test the performance of the BioMASS, continuous, and hybrid models. For the homogeneous configuration, we performed 50 simulations for each one, BioMASS, Mason200, Mason400, and Repast. For the heterogeneous configuration we performed 50 simulations for each one, BioMASS, Mason20, Mason40, Mason200, Mason400, and Repast.

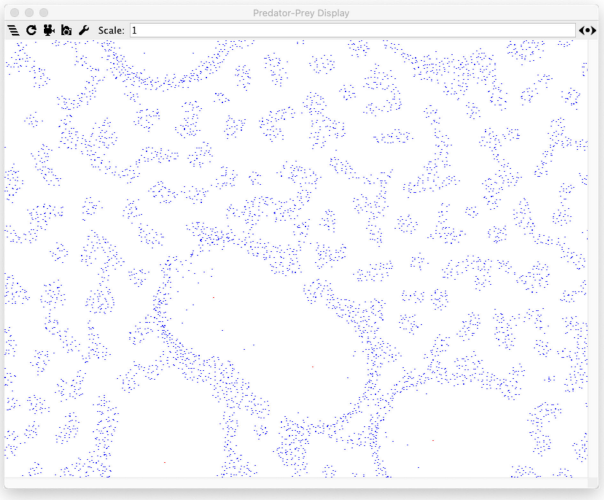

FIGURE 8. The Predator-Prey swarm model graphic display. The figure illustrates the first steps of a simulation run. Here, the space is covered by a patchwork of densely and sparsely areas. This distribution changes at each simulation step.

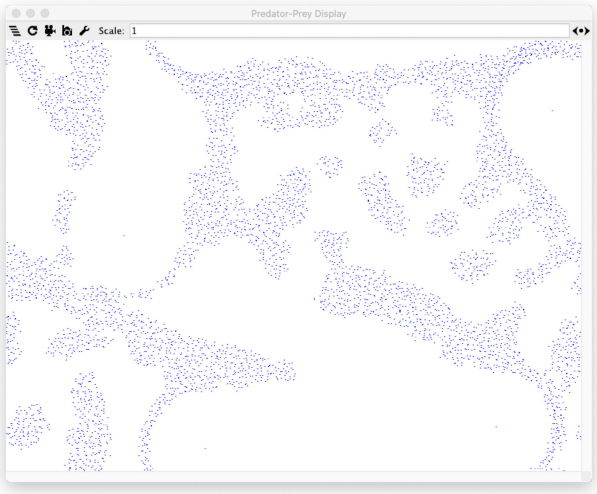

FIGURE 9. The Predator-Prey swarm model graphic display. The figure illustrates an advanced state of the simulation run. Compared to Fig. 8, there are fewer dense and sparse patches but the patches are much larger.

B. RESULTS AND ANALYSIS

Each simulation run consists of 1000 time steps. For each simulation run we gather the average, minimum, and maximum duration of a step in terms of execution time.

1) HOMOGENEOUS PERCEPTION

Figures 10, 11, and 12 present the results of the simulations. Fig. 10 depicts the maximum time step duration of each run. Fig. 11 depicts the minimum time step duration of each run. Fig. 12 depicts the average time step duration of each run.

Observation 1. The average time steps in the BioMASS simulation are shorter than both the hybrid models and the continuous model.

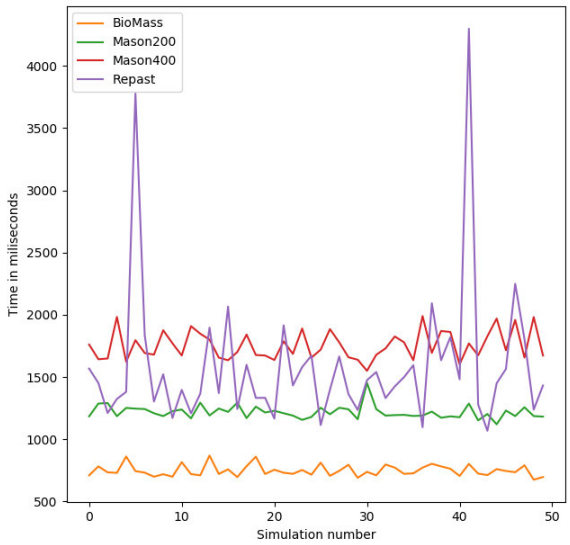

FIGURE 10. Maximum time step duration of each run for the simulations with homogeneous perception scope.

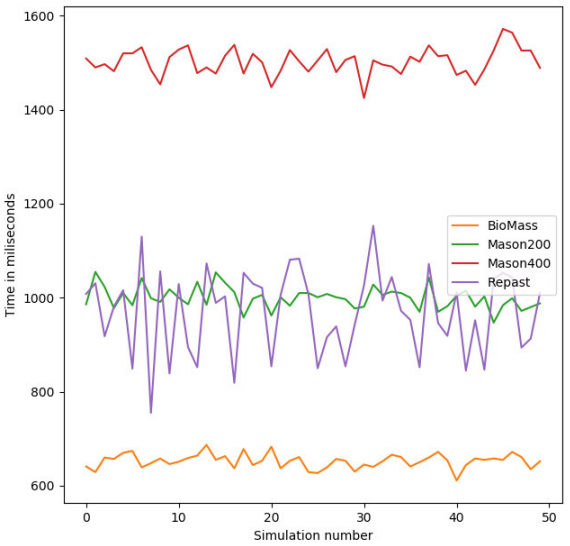

FIGURE 11. Minimum time step duration of each run for the simulations with homogeneous perception scope.

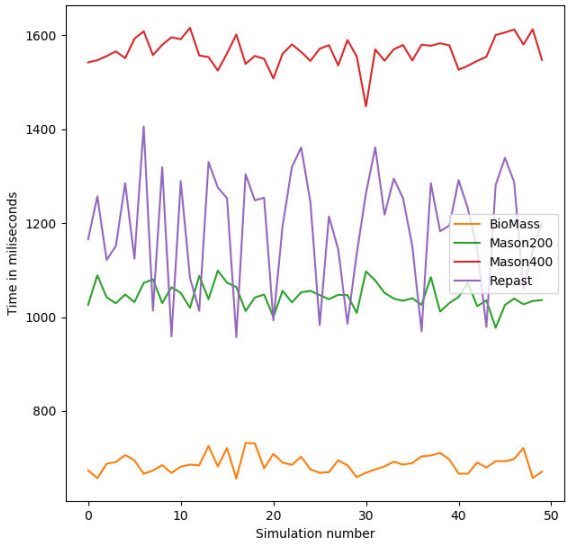

FIGURE 12. Average time step duration of each run for the simulations with homogeneous perception scope.

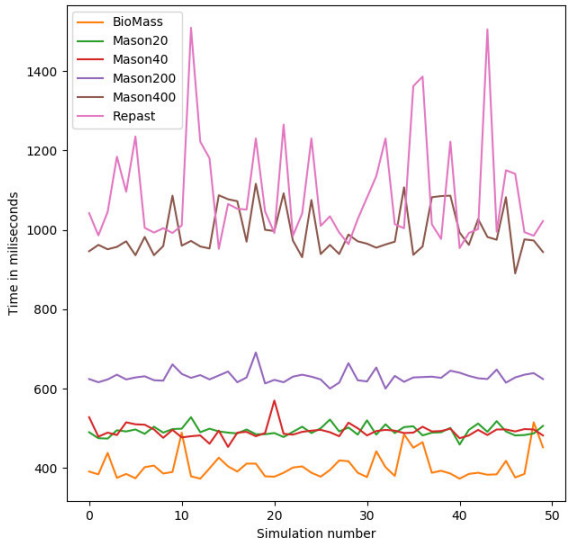

FIGURE 13. Maximum time step duration of each run for the simulations with heterogeneous perception scope.

Observation 2. The minimum time of steps in the continuous model outperforms the hybrid models in most cases.

Observation 3. In terms of time step duration, the hybrid model with the smaller discretization (Mason200) outperforms the continuous model (Repast) and hybrid model with larger discretization (Mason400).

Observation 4. In most cases, the time step duration in the continuous model is smaller than the time step duration of the hybrid model with the larger discretization (Mason400).

Observation 5. In a few simulation runs, the maximum time step in the continuous model is much larger than the maximum time step of the BioMASS and hybrid models.

2) HETEROGENEOUS PERCEPTION

Figures 13, 14, and 15 present the results of the six simulations with heterogeneous perception scopes. As in the previous subsection, Fig. 13 depicts the maximum time step duration for each of the 50 simulation runs, while Fig. 14 depicts the minimum and Fig. 15 depicts the averages.

Observation 1. The average time step durations are lowest in the BioMaSS model.

Observation 2. The hybrid models with the smallest discretization values (Mason20 and Mason40) have similar performance. They outperform the hybrid models with large discretization values (Mason200, and Mason400).

Observation 3. The time step durations of hybrid model Mason200 are smaller than those of the hybrid model Mason400 and the continuous model (Repast).

Observation 4. The time step durations of the hybrid model with the largest discretization value Mason400 are similar to those of the continuous model (Repast).

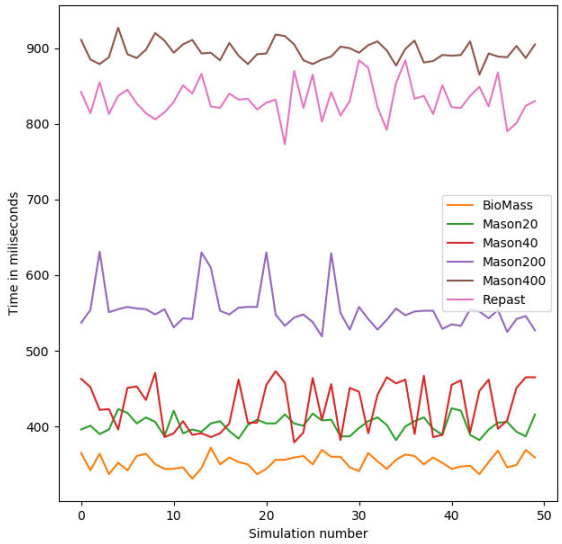

FIGURE 14. Minimum time step duration of each run for the simulations with heterogeneous perception scope.

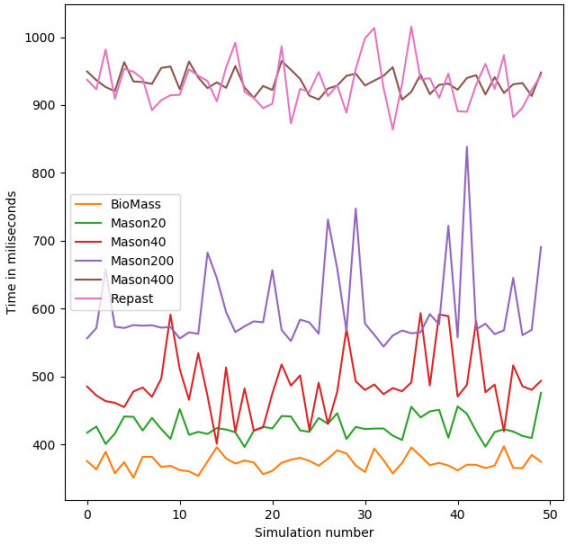

FIGURE 15. Average time step duration of each run for the simulations with heterogeneous perception scope.

VII. CONCLUSION

We presented BioMASS, a multi-agent spatial model to simulate spatially explicit environments in a MAS. In terms of simulation time, our model significantly outperformed traditional continuous and discrete models, as well as hybrid models with various discretization values. The main reason for this accomplishment is that the spatial model is designed to perform the neighborhood search function with $O(1)$ time complexity in almost all cases. This allows a large number of agents to use the search function intensively without significantly affecting the simulation time. The BioMASS model is very advantageous for agent-based models of complex systems, which require a large population of agents interacting both directly and through the environment in order simulate realistic emergent behaviors. We analyzed the time complexity of the different spatial models both theoretically and experimentally through simulation. The simulation experiments tested various perception scopes, to test the execution time of the spatial model for extreme values. The results of these experiments are depicted in Figs. 10 to 15, and clearly demonstrate that BioMASS outperforms the other models. In future work, we propose extending the BioMASS multi-agent space model to three dimensions.

REFERENCES

[1] L. Tesfatsion, "Agent-based computational economics: Growing economies from the bottom up," *Artif. Life*, vol. 8, no. 1, pp. 55–82, Jan. 2002.
[2] J. D. Farmer and D. Foley, "The economy needs agent-based modelling," *Nature*, vol. 460, no. 7256, pp. 685–686, Aug. 2009.
[3] L. Tesfatsion, "Agents come to bits: Towards a constructive comprehensive taxonomy of economic entities," *J. Econ. Behav. Org.*, vol. 63, no. 2, pp. 333–346, Jun. 2007.
[4] C. E. Sansores, F. Reyes-Ramírez, L. E. Calderon-Aguilera, and H. F. Gómez, "A novel modeling approach for the 'end-to-end' analysis of marine ecosystems," *Ecol. Informat.*, vol. 32, pp. 39–52, Mar. 2016.
[5] F. Bousquet and C. Le Page, "Multi-agent simulations and ecosystem management: A review," *Ecol. Model.*, vol. 176, nos. 3–4, pp. 313–332, Sep. 2004.
[6] V. Grimm and S. F. Railsback, "Agent-based models in ecology: Patterns and alternative theories of adaptive behaviour," in *Agent-Based Computational Modelling: Applications in Demography, Social, Economic and Environmental Sciences*, F. C. Billari, T. Fent, A. Prskawetz, and J. Scheffran, Eds. Heidelberg, Germany: Physica-Verlag, 2006, pp. 139–152.
[7] C. M. Buchmann, F. M. Schurr, R. Nathan, and F. Jeltsch, "Habitat loss and fragmentation affecting mammal and bird communities—The role of interspecific competition and individual space use," *Ecol. Informat.*, vol. 14, pp. 90–98, Mar. 2013.
[8] S. Zheng and H. Liu, "Improved multi-agent deep deterministic policy gradient for path planning-based crowd simulation," *IEEE Access*, vol. 7, pp. 147755–147770, 2019.
[9] R. Axelrod, "Advancing the art of simulation in the social sciences," in *Simulating Social Phenomena*, R. Conte, R. Hegselmann, and P. Terna, Eds. Berlin, Germany: Springer, 1997, pp. 21–40.
[10] R. K. Sawyer, *Social Emergence: Societies as Complex Systems*. Cambridge, U.K.: Cambridge Univ. Press, 2005.
[11] R. K. Sawyer, "Artificial societies: Multiagent systems and the micro-macro link in sociological theory," *Sociol. Methods Res.*, vol. 31, no. 3, pp. 325–363, Feb. 2003.
[12] S. Camazine, N. R. Franks, J. Sneyd, E. Bonabeau, J.-L. Deneubourg, and G. Theraula, *Self-Organization in Biological Systems*. Princeton, NJ, USA: Princeton Univ. Press, 2001.
[13] M. Pogson, M. Holcombe, R. Smallwood, and E. Qwarnstrom, "Introducing spatial information into predictive NF-κB modelling—An agent-based approach," *PLoS ONE*, vol. 3, no. 6, pp. 1–6, 2008.
[14] M. Soheilypour and M. R. K. Mofrad, "Agent-based modeling in molecular systems biology," *BioEssays*, vol. 40, no. 7, 2018, Art. no. 1800020.
[15] N. R. Jennings, "On agent-based software engineering," *Artif. Intell.*, vol. 117, no. 2, pp. 277–296, 2000.
[16] F. Bergenti, E. Iotti, S. Monica, and A. Poggi, "Agent-oriented model-driven development for JADE with the JADEL programming language," *Comput. Lang., Syst. Struct.*, vol. 50, pp. 142–158, Dec. 2017.
[17] M. Wooldridge, N. R. Jennings, and D. Kinny, "The Gaia methodology for agent-oriented analysis and design," *Auton. Agents Multi-Agent Syst.*, vol. 3, no. 3, pp. 285–312, 2000.
[18] G. Kardas, "Model-driven development of multiagent systems: A survey and evaluation," *Knowl. Eng. Rev.*, vol. 28, no. 4, pp. 479–503, Dec. 2013.
[19] S. Rodriguez, N. Gaud, and S. Galland, "SARL: A general-purpose agent-oriented programming language," in *Proc. IEEE/WIC/ACM Int. Joint Conf. Web Intell. (WI), Intell. Agent Technol. (IAT)*, Warsaw, Poland, Aug. 2014, pp. 103–110.
[20] C. Bădică, Z. Budimac, H.-D. Burkhard, and M. Ivanovic, "Software agents: Languages, tools, platforms," *Comput. Sci. Inf. Syst.*, vol. 8, no. 2, pp. 255–298, 2011.

[21] J. J. Gómez-Sanz, C. R. Fernández, and J. Arroyo, "Model driven development and simulations with the INGENIAS agent framework," *Simul. Model. Pract. Theory*, vol. 18, no. 10, pp. 1468–1482, Nov. 2010.
[22] A. S. Rao and M. P. Georgeff, "BDI agents: From theory to practice," in *Proc. 1st Int. Conf. Multi-Agent Syst.*, San Francisco, CA, USA, 1995, pp. 312–319.
[23] S. Luke, C. Cioffi-Revilla, L. Panait, K. Sullivan, and G. Balan, "MASON: A multiagent simulation environment," *Simulation*, vol. 81, no. 7, pp. 517–527, Jul. 2005.
[24] A. Grignard, P. Taillandier, B. Gaudou, D. A. Vo, N. Q. Huynh, and A. Drogoul, "GAMA 1.6: Advancing the art of complex agent-based modeling and simulation," in *PRIMA 2013: Principles and Practice of Multi-Agent Systems*, G. Boella, E. Elkind, B. T. R. Savarimuthu, F. Dignum, and M. K. Purvis, Eds. Berlin, Germany: Springer, 2013, pp. 117–131.
[25] M. J. North, N. T. Collier, J. Ozik, E. R. Tatara, C. M. Macal, M. Bragen, and P. Sydelko, "Complex adaptive systems modeling with repast simphony," *Complex Adapt. Syst. Model.*, vol. 1, no. 1, pp. 1–26, Dec. 2013.
[26] F. Klügl, "SeSam: Visual programming and participatory simulation for agent-based models," in *Multi-Agent Systems: Simulation and Applications*, A. M. Uhrmache and D. Weyns, Eds. Boca Raton, FL, USA: CRC Press, 2009, pp. 477–508.
[27] J. H. Anderson, J. A. Downs, R. Loraamm, and S. Reader, "Agent-based simulation of muscovy duck movements using observed habitat transition and distance frequencies," *Comput., Environ. Urban Syst.*, vol. 61, pp. 49–55, Jan. 2017.
[28] A. L. Cronin, "An agent-based model of nest-site selection in a mass-recruiting ant," *J. Theor. Biol.*, vol. 455, pp. 54–63, Oct. 2018.
[29] L. Pérez and S. Dragićević, "ForestSimMPB: A swarming intelligence and agent-based modeling approach for mountain pine beetle outbreaks," *Ecol. Informat.*, vol. 6, no. 1, pp. 62–72, Jan. 2011.
[30] D. McDermot and K. A. Rose, "An individual-based model of lake fish communities: Application to piscivore stocking in lake Mendota," *Ecol. Model.*, vol. 125, no. 1, pp. 67–102, Jan. 2000.
[31] L. Parrott and R. Kok, "A generic, individual-based approach to modelling higher trophic levels in simulation of terrestrial ecosystems," *Ecol. Model.*, vol. 154, nos. 1–2, pp. 151–178, Aug. 2002.
[32] E. H. van Nes, E. H. R. R. Lammens, and M. Scheffer, "PISCATOR, an individual-based model to analyze the dynamics of lake fish communities," *Ecol. Model.*, vol. 152, no. 2, pp. 261–278, 2002.
[33] M. Wooldridge, *An Introduction to MultiAgent Systems*. Hoboken, NJ, USA: Wiley, 2009.
[34] A. Drogoul, F. Michel, and J. Ferber, "Multi-agent systems and simulation: A survey from the agent community's perspective," in *Multi-Agent Systems*, vol. 5, A. M. Uhrmacher and D. Weyns, Eds., 1st ed. Boca Raton, FL, USA: CRC Press, 2009, pp. 3–59.
[35] M. R. Genesereth and N. J. Nilsson, *Logical Foundations of Artificial Intelligence*. San Francisco, CA, USA: Morgan Kaufmann, 1987.
[36] T. Tyrrell, "The use of hierarchies for action selection," *Adapt. Behav.*, vol. 1, no. 4, pp. 387–420, Mar. 1993.
[37] A. S. Rao and M. P. Georgeff, "An abstract architecture for rational agents," in *Proc. 3rd Int. Conf. Princ. Knowl. Represent. Reasoning*, Cambridge, MA, USA, 1992, pp. 439–449.
[38] J. P. Müller and M. Pischel, "The agent architecture InteRRaP: Concept and application," German Res. Center Artif. Intell., Kaiserslautern, Germany, Tech. Rep. RR 93-26, 1993.
[39] C. W. Reynolds, "Flocks, herds and schools: A distributed behavioral model," in *Proc. 14th Annu. Conf. Comput. Graph. Interact. Techn. (SIGGRAPH)*, New York, NY, USA, 1987, pp. 25–34.

CANDELARIA ELIZABETH SANSORES-PÉREZ (Member, IEEE) received the B.S. degree in computer systems engineering from the Instituto Tecnológico de Mérida, Mérida, Mexico, in 1994, and the M.S. and Ph.D. degrees in computer science from the Universidad Complutense de Madrid, Madrid, Spain, in 2008.

Since 2008, she has been a Full Professor with the Department of Basic Sciences and Engineering, Universidad del Caribe. Her current research interests include simulation of complex systems, multiagent systems, and algorithms.

JOEL ANTONIO TREJO-SÁNCHEZ was born in Mérida, Yucatán, Mexico, in 1981. He received the B.S. degree in computer science from the Autonomous University of Yucatán, the M.S. degree in electrical engineering from CINVESTAV, and the Ph.D. degree in computer sciences from CICESE, in 2014.

He is currently a Research Fellow with the CONACyT—Centro de Investigación en Matemáticas. His research interests include distributed and parallel computing and the design of algorithms.