Proposal for a Flexible Benchmark for Agent Based Models

Elizabeth Koning
Department of Computer Science
University of Illinois, Urbana Champaign
Urbana, IL, USA
ekoning2@illinois.edu

William Gropp
Department of Computer Science
University of Illinois, Urbana Champaign
Urbana, IL, USA
gropp@illinois.edu

***Abstract*—As hardware capabilities and scientific understanding of social and biological systems improves, researchers are able to model complex system with increasing detail. This change increases the demand on Agent Based Modeling (ABM) software, making the performance of the software more critical. For researchers to choose the best system to implement their models, they must understand not only the capabilities of the platforms, which is available in the documentation, but also the relative performance to other software options. Currently, there is a dearth of information on the performance of ABM platforms and no standardized way to compare the performance of different platforms.**

To address this gap, the proposed NothingModel benchmark is a flexible benchmark to reflect a wide variety of ABMs and allow the developers of models and ABM platforms to compare platforms under a range of conditions. Set up as a series of building blocks, it allows the users to vary the scale, memory use, communication, computation, and heterogeneity of a sample model, implemented with multiple tools. Along with the written description of the model, we include a reference implementation in NetLogo to demonstrate the specifics and provide comparison to other tools.

***Index Terms*—Benchmarking, Agent Based Models, ABM, Multi-agent systems, MAS, Performance**

I. Introduction

As hardware capabilities and scientific understanding of social and biological systems improves, researchers are able to model complex system with increasing detail. Agent Based Models (ABMs) are a popular way of representing these complex systems, representing the actors, whether human, plant, or animal, as agents interacting and moving around an environment. As more complex systems are represented using ABMs, the performance of the software becomes more critical.

There are many platforms to support the development of ABMs, including NetLogo [8], Repast [5], and Repast HPC [3].

For researchers to choose the best system to implement their models, they must understand not only the capabilities of the platforms, which is available in the documentation, but also the relative performance to other software options. Currently, there is a dearth of information on the performance of ABM platforms and no standardized way to compare the performance of different platforms.

A flexible benchmark for ABMs will help users of modeling software choose the best tool for their task. While one piece of software may be optimized for a large number of agents, another may have an advantage if the model has a large variety of agent behavior.

The NothingModel, proposed here, begins with a model of nothing – the agents do not move, communicate, or perform any actions. From this baseline, the model changes into a more realistic and complex system, varying in five areas: scale, computation, memory, communication, and heterogeneity. This list comes from the lists of factors impacting performance in [1, 4].

This flexible benchmark is designed to help communicate between ABM software developers and modelers. For the developers of platforms supporting ABMs, it provides a way to demonstrate their software's strengths, whether that be in models focused on models balanced across the five factors, or models with heavy communication demands. As for modelers, it provides a way to create a test model with a similar structure as their own model, and run it with a variety of software platforms.

In this paper, we provide a description of the benchmark, so that it may be implemented with ABM software packages. We include an implementation in NetLogo as a reference: https://github.com/kodingkoning/NothingModel.

II. Background

A variety of benchmarks have been proposed for ABMs, including in [1, 2, 4, 6, 7]. Each of these benchmarks aims for measuring different elements of a model, and so has different types of agents and different parameters. Table I shows the factors included in each benchmark discussed in this section. The list of features has been combined from [1, 4].

A. StupidModel

StupidModel, proposed in [6] and the earliest of these benchmarks, simplifies a model to a very simple ABM and then adds complexity to make a total of 16 versions. The most basic version uses a collection of "bug" agents that move in random directions at every timestep. Each of the next 15 versions increase in complexity. For example, the second version adds an action where the agent grows by a constant amount. Most (10 out of the 15) versions change the actions or initialization of the agents from the first version, and the

979-8-3503-6460-6/24/$31.00 ©2024 IEEE
DOI 10.1109/IPDPSW63119.2024.00149

TABLE I
FEATURES INCLUDED IN BENCHMARKS

Feature	Benchmark				
	Particle Benchmark[1]	***Circles Benchmark[2]***	***Person Benchmark[4]***	***StupidModel [6]***	***Person Benchmark [7]***
Computation	No	No	Yes	Variation in computation of summary statistics	Compute intensity included but not varied
Scale	Yes for population size, No for environment size.	Yes	Yes	Yes for population size, No for environment size.	Yes for population size, No for environment size.
Memory	No	No	No	No	No
Communication	No	Yes for neighborhood size, No for message size, No for message frequency	Yes for neighborhood size, Yes for message size, No for message frequency	No	No
Heterogeneity	Multiple types of agents, but same set of agent types for all runs	No	No	Second type of agent only in version 16, but some new actions for original agent in other versions	No

other 5 focus on displaying information to the user, such as graphing the number of agents.

B. Person Benchmark

In [7], the authors present a model of a "Person" agent with actions *walk*, *interact*, and *compute*. The size of the environment is always a 2000×2000 grid. They also use a fixed interaction radius of 3, to determine if the randomly moving agents will interact with one another by exchanging small messages. For a compute workload, they use a Discrete Fourier Transformation (DFT).

In [4], the authors base their benchmarking approach on [7]. They address the limitations of [7]'s assessment, as it is fixed in amount of communication, amount of computation, and the workload distribution. To provide a more general benchmark, they parameterize the size of the compute task, the maximum interaction radius, and the size of the messages that are sent. These changes expose more performance factors, but it still only includes a single agent type. Without variation in the actions taken by the agents, the benchmark does not represent many models.

C. Circles Benchmark

The benchmark presented in [2] focuses on the relationship between neighboring agents. A fixed radius near neighbors (FRNNs) search is often the element of a model that limits the performance. Many models require communication between an agent and their closest neighbor or their neighborhood, which is often done using a partitioned grid. The Circles model uses a circular environment and requires each agent to identify its neighborhood at each time step. The strength of this model is its focus on what the authors see as the most limiting aspect of a model. However, it is limited to that one factor, lacking variation in issues such as communication and computation workload.

D. Particle Benchmark

The benchmark proposed by [1] simulates particle agents, where each particles is moving around the environment space and reacting with other particles. They present it as a good proxy for social and ecological simulations as well. In this benchmark, there are three types of agent: Particle A, Particle B, and Particle C. Particle A and B start in the environment, moving around randomly. When one A and one B are in near enough proximity, they react and become a single Particle C. This benchmark does include variation in the types of agents, but every run includes the same three agents types.

III. PROPOSAL

The proposed benchmark uses a similar approach to [6]'s StupidModel, which starts with a minimal model, and adds layers of complexity to each version. The proposed model starts with agents who do not move, change, or interact, making a "NothingModel". From this core model, agents and attributes can be added to the model, like building blocks. This allows the benchmark to span from the simplest possible model to complex, computationally demanding model. It is not feasible to run every possible combination, so a subset of the possible models must be strategically chosen.

The variations proposed here are to give a range of possibilities for the ways realistic models are most likely to vary. Other patterns may be useful to users of the model, and such changes are encouraged.

A. NothingModel

In the simplest model, there will be one agent per grid location, and they will have no actions. They may have functions to be created and destroyed, but their "action" step will be empty. For NetLogo, this may be the patches. The agents in any implementation do not need to be able to move around the environment, they merely need to be present as entities.

The appropriate number of timesteps may change between tests, but a preliminary test with no timesteps may be useful to account for the time spent in starting the simulation, rather than in the body of the tasks.

B. Scale

Scale is the simplest factor to vary. Most models use an input parameters for population and environment size. With the NothingModel using one agent per grid cell, an initial scale is set. The population or environment size can be changed together to keep the density constant, or varying one or the other can change the density.

Depending on the actions of the agents, the number may change throughout execution. Both population and environment are likely to impact the performance, and, especially with the introduction of parallelism, the ratio will be influential as well. Two ways the model could use distributed parallelism are by assigning agents to processes, or by assigning areas of the environment to the processes. Changing the size of one or the other may change the performance in the case of parallelism more than a sequential run.

C. Computation

Computation includes both performing calculations and accessing the agent's own attributes. The computational step will be done through:

- Random number generation
- Pairwise matrix multiplication
- Computing factors

All three of the calculations are chosen because they should be straightforward to implement in any programming language and are similar to computations that may occur in actual models.

For the implementation of the computation, the goal of RNG and the multiplication is to mimic behaviors typical of ABMs. Because of this, each should be done in the typical way for the platform. Pairwise matrix multiplication requires a local list of numbers, increasing memory usage temporarily, but RNG can be done with minimal memory use. The factorization task is included to be a loop of computation, not to be efficiently implemented. Therefore, the computation should be implemented in the most naive and tedious way, dividing the iteration number by every number less than itself, down to 2.

D. Memory

While different platforms may include different data types, the core types we are working with are numbers, strings, agents, and lists, as those are the types available in NetLogo [8]. Though NetLogo allows mixing of types in a list, we will limit lists to a single type of item, to include more strictly typed languages.

The main way of varying memory use is adding a list to an agent. This can be done with a list of any of the other data types. It can start at a single item in the list, and increase to as large of a list is able to be executed on the hardware for the test.

E. Communication

Communication can vary in many ways; it can vary by size of message, frequency of message, and pattern of message partners. Some models will have messages shared only with neighbors, but others may require all-to-all patterns. These may have very different performance results. Using the approach from NetLogo [8], we will allow the agents to communicate by viewing other agents' properties.

Key patterns to consider are:

- All-to-all
- Summary statistics (gather)
- Pairs (remaining fixed throughout execution)
- Pairwise (changing throughout execution)
- Neighborhoods

While the implementation of the communication patterns may depend on the design of the platform, the same communication should be able to occur across platforms. In an all-to-all patterns, all agents (either globally or of a type) will access some summary statistic, such as an average property across the population, and may act based on that information. With the gather of a summary statistic, similar information can be calculated and then stored for the user to access after the simulation. In the pair patterns, agents access properties of a single other agent, and either access the information about the same agent repeatedly, or different agents at each step. Finally, with the neighborhoods, agents access information about the nearby agents, where the distance may vary between agent types.

The size and frequency of the communication may also change. Size may change by whether the agent views only the location, or the location and a list of properties. Frequency may change, as the communication may not need to occur every timestep. Message size can start at the minimum data size, while frequency can start at every timestep and then be tested at lower frequencies.

In the NetLogo communication structure, the agents access data from other agents, which requires all communicated data to also be a property of an agent. The implication of this structure is that communication requires memory to vary as well.

F. Heterogeneity

Heterogeneity is about how similarly the agents behave. Some platforms may show performance improvements if the agents all behave similarly or if each agent performs the same actions every timestep. The level of heterogeneity is the degree to which the other performance factors vary in a single simulation. Because of this requirement of variety, it quickly becomes an issue of an unreasonable number of permutations, and so a set of variations need to be chosen. Adding different types of agents can be done by taking the other factors (computation, memory, communication) and combining them in a different way for a new agent type.

One aspect of heterogeneity is the distribution of the agents across the environment. Models may perform differently if the

agents cluster in one area, such as when they are attracted to a resource. While this behavior does not fit into scale, communication, computation, or memory, it may separate otherwise similar implementations, especially those with parallelism. Alongside this, the amount of movement each agent is given at every timestep may change the performance, especially if different threads or processes are managing the agents.

Another form of heterogeneity is the frequency of actions. Frequency is included in communication, but can be applied as well to computation or changes to agent properties. Like the distribution across the environment, this may be implemented differently, and the system may optimize for more or less frequent events.

G. Example Permutations

A proposed set of combinations to show the impacts of the performance factors would be:

- NothingModel, varying scale (keeping agent density constant)
- Varying only computation (increasing all types), with no memory or communication used
- Varying only memory, with no communication or computation used
- Varying only communication (increasing all aspects), with no computation or memory used
- Agents cluster in one area, moving randomly, with the stride changed between executions
- Unique agent types from one to the number of agents in the simulation (randomly generated properties and behaviors)
- Frequency of actions, varied from 0% probability to 100% probability of occurring, for both communication and computation
- "Low", "Medium", and "High" levels of communication, memory use, and computation per agent, as well as density of agents

Notice, most of these do not have a maximum threshold for the factor that is varied. Instead, when running the tests, the maximum acceptable runtime should be selected, and the varied aspect should be increased until the model cannot complete in the given time. "Low", "Medium", and "High" are also not specifically defined. These may depend on the hardware used and can be tuned based on varying each factor alone.

The reference implementation in NetLogo uses BehaviorSpace experiments to demonstrate each permutation, while sharing code between experiments when possible. The base NetLogo code is less than 250 lines. Depending on the scale of the tests, they can take seconds, minutes, or hours to run, but the provided examples are on the order of seconds.

IV. Conclusions and Future Work

We have presented NothingModel, a flexible benchmark for ABMs. It is able to represent models from a baseline of no actions to complex interactions between agents. Easy use of the NothingModel requires a robust set of implementations for popular and specialized ABM platforms, which is currently planned work building on the model design.

The number of possible combinations building on the NothingModel is infinite, but the number is narrowed down to a set of permutations to span some of the most common problem types. The ideal set of combinations may not be the ones laid out here. However, the flexible building block approach to the benchmark allows for users to formulate the benchmark to the elements that are most relevant for their use case.

The next step for the NothingModel is implementation using popular ABM tools, including [3, 5, 8], using the example permutations to recommend best use cases for each tool.

References

[1] Eidah Alzahrani, Paul Richmond, and Anthony J. H. Simons. "A Formula-Driven Scalable Benchmark Model for ABM, Applied to FLAME GPU". en. In: *Euro-Par 2017: Parallel Processing Workshops*. Ed. by Dora B. Heras et al. Lecture Notes in Computer Science. Cham: Springer International Publishing, 2018, pp. 703–714. ISBN: 978-3-319-75178-8. DOI: 10.1007/978-3-319-75178-8_56.

[2] Robert Chisholm, Paul Richmond, and Steve Maddock. "A Standardised Benchmark for Assessing the Performance of Fixed Radius Near Neighbours". In: *Euro-Par 2016: Parallel Processing Workshops*. Ed. by Frédéric Desprez et al. Cham: Springer International Publishing, 2017, pp. 311–321. ISBN: 978-3-319-58943-5.

[3] Nicholson Collier and Michael North. "Parallel agent-based simulation with Repast for High Performance Computing". en. In: *SIMULATION* 89.10 (Oct. 2013), pp. 1215–1235. ISSN: 0037-5497, 1741-3133. DOI: 10.1177/0037549712462620. URL: http://journals.sagepub.com/doi/10.1177/0037549712462620.

[4] Andreu Moreno et al. "Designing a benchmark for the performance evaluation of agent-based simulation applications on HPC". en. In: *The Journal of Supercomputing* 75.3 (Mar. 2019), pp. 1524–1550. ISSN: 1573-0484. DOI: 10.1007/s11227-018-2688-8. URL: https://doi.org/10.1007/s11227-018-2688-8.

[5] Michael J North et al. "Complex Adaptive Systems Modeling with Repast Simphony". In: *Complex Adaptive Systems Modeling* 1 (Mar. 2013). DOI: 10.1186/2194-3206-1-3. URL: https://doi.org/10.1186/2194-3206-1-3.

[6] Stephen F. Railsback, Steven L. Lytinen, and Stephen K. Jackson. "Agent-based Simulation Platforms: Review and Development Recommendations". en. In: *SIMULATION* (Sept. 2006). DOI: 10.1177/0037549706073695.

[7] Alban Rousset et al. "A survey on parallel and distributed multi-agent systems for high performance computing simulations". In: *Computer Science Review* 22 (Nov. 2016), pp. 27–46. ISSN: 1574-0137. DOI: 10.1016/j.cosrev.2016.08.001. URL: https://www.sciencedirect.com/science/article/pii/S1574013715300435.

[8] U. Wilensky. *NetLogo*. Evanston, IL, 1999. URL: http://ccl.northwestern.edu/netlogo/.