AGH UNIVERSITY OF SCIENCE AND TECHNOLOGY

FIELD OF SCIENCE: ENGINEERING AND TECHNOLOGY

SCIENTIFIC DISCIPLINE: INFORMATION AND COMMUNICATION TECHNOLOGY

DOCTORAL THESIS

SCALABLE SIMULATION OF SOCIAL PHENOMENA BASED ON SIGNAL PROPAGATION MODELING

Author: Mateusz Paciorek, MSc

Supervisor: Wojciech Turek, PhD, associate professor

Completed in:

AGH University of Science and Technology

Faculty of Computer Science, Electronics and Telecommunications

Institute of Computer Science

Krakow, 2022

I would like to express my sincere gratitude to my supervisor, Prof. Wojciech Turek. Your expertise helped me understand the problems I faced, while your patience was invaluable in the process of formulating the research goals and during my work on this thesis.

I would also like to thank Prof. Renata Słota and Prof. Aleksander Byrski for their guidance in my prior education. The knowledge I gained under your tutelage was irreplaceable in my research work.

I could not have finished this thesis without my fiancée, Anna Wodzisz. Your constant love and support sparked in me the strength necessary to complete my work, and your company kept me sane throughout this time.

Finally, I would like to thank my brother, Wojciech Paciorek, for proofreading this thesis.

Contents

1 Introduction

This chapter provides the context necessary to understand the concepts presented in this document. A general summary of the research areas relevant to the topic of this work is presented, along with the motivation behind this research. The thesis of the research is also stated, reinforced by the critical goals and main contributions.

1.1. Motivation

Computer simulations constitute an important field of study that is rapidly developing and increasing in the range of applicability. A concept central to simulation is a model, which usually serves as a way of expressing some structure, properties, and rules regarding real-world objects that are intended to be reflected in the simulation. The growing interest in the use of simulations in research sparked the development of new modeling methods, the creation of models for new problems, more general studies on efficiency, ease of use, universal tools, and the standardization of simulations [54].

One type of model, which is focused on the utilization of individual and independent entities (*agents*), is especially interesting in the context of studies on social phenomena. This class of models is often referred to as *agent-based models* (ABM). The approach to simulation that makes use of ABM is commonly referred to as *agent-based modeling and simulation* (ABMS), extending the model with decision-making algorithms that aim to reproduce the behaviors observed in the modeled scenario, usually giving agents autonomy and the ability to interact with each other. The ability to represent individuals in the real-world scenario as self-contained beings provided by this approach leads to its popularity in such research areas as traffic management, architecture (especially in the context of safety and evacuation facilities in case of emergency), urban design, artificial life, biology, and many more.

A large number of social phenomena fit naturally into the capabilities of *spatial simulations*, in which the model represents some physical space and, in the case of ABMS, the individuals occupying this space. In such models, it is common to represent the environment as metric spaces, of which 2D or 3D Euclidean space is often the most obvious choice, depending on the dimensionality requirements. However, the continuous Cartesian coordinate system is often not necessary — as the time is usually measured in discrete increments, the precise position of an object in space is superfluous, while the discretized set of possible positions is a much simpler solution, sufficient for many cases. This research focuses on this class of simulations: discrete spatial ABMS.

As the usefulness of the simulations increases, so does the scope and complexity of the created models. A number of factors commonly used to improve the simulation results influence the required computing resources:

– increasing the scope of the model, e.g. from one building to a city block or an entire city, or increasing the number of agents in the model,

– increasing the granularity of the model, e.g. subdividing the structure representing the simulation space into smaller parts,

– increasing the complexity of the model, e.g. representing the environment or agents using more parameters, or increasing the frequency or complexity of interactions between agents,

– increasing the time scope of the simulation, usually by increasing the number of iteration steps.

All the mentioned factors need an accompanying increase in resources: memory (either volatile or persistent) or computing power.

Further analyzing the resource requirements, many applications rely heavily on the speed of obtaining results. The examples include such critical areas as decision systems operating in the live environment — medical emergencies, evacuations — but also commercial simulations, where the time factor carries a significant monetary value. An additional constraint on the aspect of the simulation speed is the process of model development, usually requiring repeated trials for troubleshooting or fine-tuning of the parameters.

On the other hand, setting aside the critical limitations regarding the time available for simulation execution, the need of extensive amounts of memory poses another significant challenge. The growing size of the data that need to be processed quickly outpaces the reasonable amount of volatile memory that can be assigned to a single computing machine.

Analysis of all factors leads to the conclusion that the only sensible approach to the growing needs of simulations is the parallelization of the computations. In 1965, Gordon Moore [46] observed a yearly doubling in the number of components that can fit into the same circuit space, effectively allowing for a similar increase in computing capabilities. Later, in 1975, he revised the observation to doubling every two years, which has since been referred to as ”Moore’s law” and reported to be roughly accurate. However, around the year 2010, the Moore’s law has been — and currently is being — widely proclaimed as either already outdated or nearing the point when it no longer holds, suggesting significant limitations on the predicted capabilities of any single computing machine. Fortunately, such limitations do not transfer from ” vertical scaling” (i.e. increasing the computing capabilities by improving the components of a single machine) to ”horizontal scaling” (i.e. increasing the computing capabilities by adding more machines). The technologies most frequently used to achieve scalability can be roughly divided into two classes: the ones using GPUs (Graphics Processing Units), and the ones relying on multiple CPUs (Central Processing Units), usually paired with separate sets of resources and operating systems. The former provide thousands of cores, but their architecture could be loosely compared to the SIMD class according to Flynn’s taxonomy [18], which greatly limits their applicability to the broader class of models. The latter encompasses a variety of different technologies that implement a similar core concept: cloud environments, HPC (High Performance Computing) environments, edge computing, and similar solutions. The versatility of such environments, as well as their popularity and usually relatively low cost of utilization, make them a perfect solution to the problems discussed. In this research, HPC will represent this class of resources: a supercomputer comprised of multiple computing nodes, each with the capabilities of a single multicore computing machine, connected by a network.

In most cases, the parallelization of simulations is not a trivial task, as usually the problem is not ”embarrassingly parallel”, i.e. they are not structured in a way that allows a simple allocation of parts of the problem to multiple processes. The most common obstacle is the existence of some form of large data structure accessed by multiple agents, either explicit, e.g. in the form of globally shared information, or implicit, e.g. the access and modification of the environment or interaction between agents, which can result in the state of the whole simulation to become such a data structure. For the proper execution of such a simulation, it is necessary to manage the access to this structure to avoid inconsistency, which introduces a natural bottleneck in the form of a single point of synchronization for all involved processes. Additionally, the details of any given model can further constrain the distribution of the computations — while specific applications and models were successfully parallelized with good results [47, 55, 66], no solution for an efficient and non-intrusive distribution exists in general for ABMS.

This research aims to address this challenge — efficient, transparent, and universal method of parallelization for discrete spatial ABMS.

1.2. Main thesis

The main thesis of this research is the following:

It is possible to create a highly scalable method of distribution for discrete spatial agent-based models and simulations that will not impact the correctness of the simulation, ensuring that the results obtained will not differ from its non-distributed version. Using the concept of signal propagation can allow representing a variety of models, with a focus on social phenomena. The method can be defined at the level of abstraction enabling its use with more complex environments, without the limitations of two-dimensional grid-based representation.

1.3. Goals of this research

The main goal of the presented research is the creation of a simulation method for discrete spatial ABMS with the following qualities:

– ensuring the distribution does not impact the distributed model and the changes to its state, i.e. preserving the *correctness*, which will be discussed further in this thesis,

– high scalability, with the capabilities of effective utilization of thousands of computational cores available in HPC infrastructures,

– applicability to a wide range of scenarios, with a focus on social phenomena, e.g. urban environments, artificial life etc.

A secondary goal of this research is linked to the widening of the applicability of the method. As the majority of the simulation methods are limited to the grid-based, two-dimensional model environment representation, this research aims to increase the flexibility of the proposed method to allow higher-dimensional models to be implemented, as well as models that are represented using more abstract structures, e.g. graphs.

1.4. Contributions presented in this thesis

The most important results yielded by the research presented in this thesis are the following:

Definition of distributed ABMS correctness. The objective of this research was the development of a simulation method that, among other characteristics, ensures that the model is ” properly executed”, which initially lacks the proper definition. To fully address this issue, it was mandatory to explore this area and formulate the definition necessary for future verification. The definition was created on the basis of previous preliminary research and experience with distributed simulations. A method of correctness verification was also proposed.

Generalization of environment model structure in the core method. The core simulation method that was selected as the foundation for improvements suffered from several limitations. The first step towards the development of the solution presented in this research was to remove some of these limitations, which were centered

around the supported structures of the modeled environment. As the original method was created with grid-based environments in mind, the related data structures and algorithms were inflexible, which in turn severely limited the range of models that can be expressed using this method. The reliance on grid-based environment representation was removed, at the same time providing the tools necessary to maintain full compatibility with those simpler environments.

Scalable, correct, and widely applicable simulation method. The main contribution of this work is the simulation method for agent-based models. The method ensures high scalability, preserves the model constraints (i.e. is *correct*) and provides support for a wide range of simulation cases. These qualities fully satisfy the goals of this research and are thoroughly verified.

1.5. Structure of this thesis

This work is structured as follows:

Chapter 2 outlines the scientific background of the presented research and shows the existing approaches to the problems addressed by this work, as well as their shortcomings and limitations.

Chapter 3 introduces and defines the concept of the correctness of the simulation method. The preliminary research that led to the formulation of this definition is also presented.

Chapter 4 presents several improvements to the core methods used as a foundation for the proposed solution. Improvements are mostly related to the generalization of the environment model structure of the method and the dimensionality of the models that can be adapted to this method.

Chapter 5 proposes a solution to the problems stated at the beginning of this work that also addresses all the goals set for this research.

Chapter 6 proposes a scheme to test the correctness of the method and evaluates the solution using this scheme.

Chapter 7 shows the results of the scalability tests of this method.

Chapter 8 presents the models based on real-world scenarios applying this method. Additional extensions of the method related to further widening of the range of models that can benefit from the proposed method are also presented.

Chapter 9 summarizes the research and presents the conclusions.

Appendix A supplements several of the chapters with a description of test models that were used in some stages of the research.

2 Agent-based discrete spatial simulations

This chapter introduces key concepts and definitions related to agent-based simulations. Existing methods and solutions are presented that serve as either a basis for the research presented in this work, or to highlight the limitations of commonly used techniques. The goals presented in the previous chapter are also restated in terms of agent-based context.

2.1. Definition of an agent

The key concept that serves as the foundation for the technologies discussed in this study is *agent* [48, 61, 72]. Over the course of its use, the concept was defined in various manners, leading to many discussions, but the following traits are usually agreed on:

- autonomy - ability to act on its own behalf, make decisions to achieve its own goals,
- interactivity - ability to interact with other agents,
- perception - ability to observe the environment and react accordingly,
- persistence - being continuously present in the environment, as opposed to being executed on demand.

However, the seemingly established, ”middle ground” definition of agent does not fully reflect the roles that can be assigned to such agent. In modeling, the term *agent* is often used to refer to the representation of the individual from the real world (e.g. a person) in the model. In this case, the role of this agent in the model is to serve as a unit that aggregates some attributes assigned to this individual. If simulation of such model is considered, it is usually most intuitive to extend this unit with a decision-making logic that will reproduce some behavior of the real-world counterpart of the agent. Thus, the same concept that was referred to as an agent in the model becomes the state of the agent. Finally, further broadening the scope, when the simulation is distributed, some processes are created that utilize computing resources to perform the operations necessary for the simulation to progress. These processes possess some state (e.g. some part of the model), can communicate with each other, cooperating with a common goal (updating the state of the model according to the predefined rules), have complete autonomy in doing their part of the work, and can react to some changes in the communication medium.

These three examples of agents clearly show the need for further clarification. In this research, the second type of agent, a modeled individual with decision-making logic, is the central concept. Therefore, an *agent* for the purpose of the research presented in this thesis should be understood as follows:

An agent is a program or a part of a program, operating continuously within a specified environment, with capabilities of observing the environment, interacting with other agents present in the environment, and making autonomous decisions.

2.2. Introduction to agent-based simulations

Although useful in the process of architectural design and artificial intelligence, probably the most influential application of the agent concept is in computer simulation. The methodology emerging from this approach is called *agent-based modeling and simulation* (ABMS) [13, 54]. Since its inception, ABMS has attracted increasing interest. Its use was deemed highly beneficial in a wide variety of research topics, such as modeling changes in economic behavior [28] or in research on phenomena that occur in two-party elections [35]. Its capabilities were deemed revolutionary for contemporary science [2]. Advantages of ABMS find their best use in micro-scale models in which groups of autonomous entities are represented at the level of each individual being assigned an agent as a representation. The resulting models still prove to be excellent tools for studying real-world social scenarios such as road traffic [66], pedestrian dynamics [56], urban design [51], building evacuation [49], etc. An important common characteristic shared by all of those applications is the necessity of representing non-uniform individuals: having different attributes or parameters, in different states, or following different decision-making algorithms. The common alternative type of simulations, based on differential equations, while immensely useful in the representation of continuous physical phenomena, usually does not provide mechanisms expressive enough for this type of application. As a consequence, ABMS became irreplaceable in some research areas.

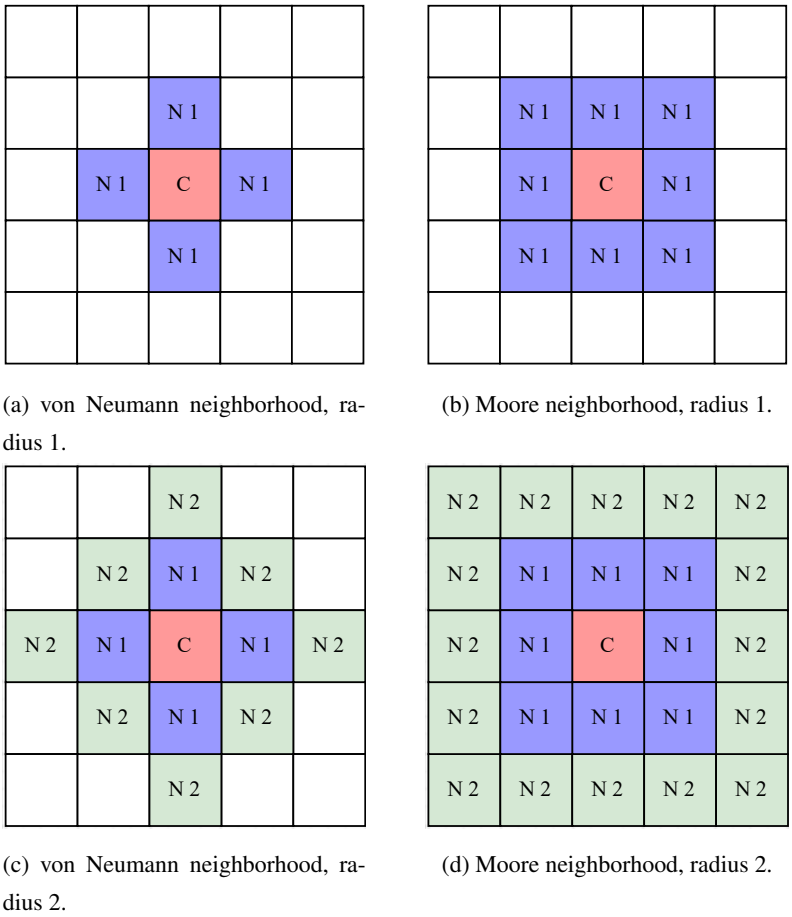
The use of ABMS does come at a cost — being able to assign an agent to each simulated entity requires a significant amount of computational resources. The discussion of the factors contributing to the increase in computational costs for simulations in general included in Section 1.1 holds for ABMS in particular as well. The conclusion is valid as well — the increasing size and complexity of ABMS requires an investigation into the possibilities of parallelization.

2.3. Parallelization of discrete spatial ABMS

The ultimate goal of the distribution of the spatial ABMS is to obtain super-scalability. A general rule worth following in the case of any distributed program that aims to efficiently utilize large numbers of cores is to limit the amount of data that needs to be transferred between computing units. As this communication is not present in the non-distributed version of the program, it is easy to observe that the larger it becomes, the larger portion of the time has to be spent on this part of the program, instead of the actual workload. However, as has been observed in [16], minimizing the size of a message is not enough to ensure good scalability of the system. The concept of *scale invariance* is discussed, described as the limitation of the number of other processes with which a given process has to communicate. To achieve the super-scalability, this number must be constant, independent of the total number of the processes in the system. As a result, all communication in the system can be described as *local*, i.e. occurring only between processes that are somehow related to each other, often by the adjacency or neighborhood in some underlying structure of the problem solved by the system.

A well-researched intuitive approach to this problem is the division of the environment [10], in which the workload of any given process depends on the state of the fragment of the simulation model assigned to this process. In this approach, the processes need to perform synchronization with neighboring processes, i.e. processes handling the adjacent fragments of the model state. The complexity and volume of this synchronization has a critical impact on the efficiency of the parallelization — it can range from a constant small portion of data, which can yield very good scalability, to data proportional to some function of the number of processes or the size of fragments assigned to the processes, which in extreme cases can negate or outweigh the benefits of paralellization, making it inefficient or even harmful for the speed of calculations.

The common representation of an environment in discrete spatial ABMS is a two-dimensional grid of cells. Each cell can be in any of the given states declared for the created model, e.g. occupied, empty, etc. The inter-



(a) von Neumann neighborhood, radius 1.

(b) Moore neighborhood, radius 1.

(c) von Neumann neighborhood, radius 2.

(d) Moore neighborhood, radius 2.

Figure 2.1: Neighborhood definition variants commonly used in ABMS.

action between cells follows the rules of neighborhood, usually von Neumann neighborhood ("4-neighborhood") or Moore neighborhood ("8-neighborhood") with a radius of 1. Figure 2.1 shows the examples of both types of neighborhood: von Neumann (Figure 2.1a, Figure 2.1c) and Moore (Figure 2.1b, Figure 2.1d) for a radius of 1 and 2, with the cell (marked with C) and its neighbors within range of 1 ($N1$) and within range of 2 ($N2$). This type of simulation environment relying on a grid is often incorrectly referred to as *cellular automaton* (CA). While the main discrepancy from the original definition of CA lies in the strict rules governing the creation of subsequent states of the simulation (generations), the representation of the state is very similar to the one described above.

If the considered models are limited to such type of environment, the problem of division is significantly less challenging, as it is enough to subdivide the grid into rectangles of identical shapes and sizes, or as close to identical as the geometry of the initial grid allows. Therefore, the main challenge is the development of a mechanism responsible for updating the state of the simulation, including any necessary synchronization, which will ensure good scalability of the model. However, it is worth noting that an additional goal of this research is to increase the applicability of the created method, including the extension to more complex environment representations.

The crucial challenge related to distribution, which exists in the general class of ABMS, is the lack of restrictions regarding the interaction between agents and the observation of the environment. It is a common practice to define a parameter describing the radius of "sight" for the agent — all interactions are performed using only the data from the cells within the distance limited by this radius. The first important problem arises from the influence of this radius on both the expressiveness and the scalability of the model:

– The greater the radius, the more agents are aware of their environment and can make decisions based on a larger picture of the whole state.

– The greater the radius, the greater part of the state necessary for the decision can be owned by some other process, requiring communication to perform observations and interactions.

As a result, the intuitive approach leads to a direct trade-off between the limits of the efficiency of parallelization and the complexity of the model.

2.3.1. Types of agent-based simulations with regard to parallelization

When considering the parallelization of an ABMS state update, one has to carefully analyze how the model handles the updates in sequential execution. A publication [52] that is part of the research presented in this thesis contains the analysis that has been reiterated and expanded in this section. The type of ABMS in the context of parallelization can be partitioned into two categories, depending on the influence of the processing of any cell on the state of simulation:

– the processing results in changes limited to the processed cell — further in the text referred to as *classic-CA*,

– the processing results in changes in another cell (or cells), regardless of whether the processed cell is changed or not — further in the text referred to as *CA-inspired*.

It is worth noting that whether the processing of any given cell involves scanning its neighborhood does not influence the above categorization. The discriminating factor is the possibility of influencing cells other than the one that is being processed. The problem of accessing adjacent cells that are not owned or governed by a given process is typically [21] solved by using a cache of the last known state of the required remotely handled cells. Each process, in addition to storing the state of the cells assigned to it, keeps the read-only state of a one-cell-wide margin synchronized with the respective owners of those cells at given intervals, e.g. after each iteration. This margin is often referred to as *buffer zones* [10] or *ghost region* (the cells in these regions are referred to as *buffer cells* or *ghost cells*, respectively).

The classic-CA category is significantly less challenging of the two. Considering the model that does utilize ghost cells, the decisions regarding any given cell can be entirely inferred from the data in those cells synchronized after the previous iteration — if the cell lies on the border of the region assigned to the process — and the previous state of the cells accessible to the process, including the state of the cell in question. The update can be immediately written as the next state of the cell, since no other cell can change the outcome for that particular cell. As such, this problem belongs to the *embarrassingly parallel* [45] class of problems. A very good example of such a problem is Conway's Game of Life [19]. Regardless of the grid partitioning, as long as the state of the local and remote neighbor cells are known to each cell, all decisions and updates are entirely local for any given cell in any given process. After each iteration, the updated state of the border cells is sent to update read-only ghost cells, and the simulation can continue to express the same behaviors as the sequential version. A more thorough analysis of this specific scenario is included in [63], along with further investigation of the influence of delayed synchronization and data loss in communication. Additionally, this category is not limited to deterministic simulations, as it seems to not be affected by the stochastic processes used in the model, as shown in the analysis of the simulated spatio-temporal evolution of urban land use [73].

Unfortunately, only a very limited subset of ABMS applications can be modeled in this way. A majority of models require some influence of a cell on another cell, usually changing the states of several cells in unison, e.g. an agent moving between two cells. As a result, any operation trying to apply changes across the border between grid parts must be consulted with the state of the second cell on the other side, as well as any other cells that might try to change the same cell. An example of possible conflict is shown in Figure 2.2. The grid is divided into two parts assigned to two processes $P1$ and $P2$. The agents are represented by circles and intend to perform moves according to the arrows. The conflict between agents $A1$ and $A2$ can be resolved locally, but the involvement of

$A3$ requires communication in order to convey the intention of $A3$ to $P1$ and another instance of communication to return the outcome to $P2$.

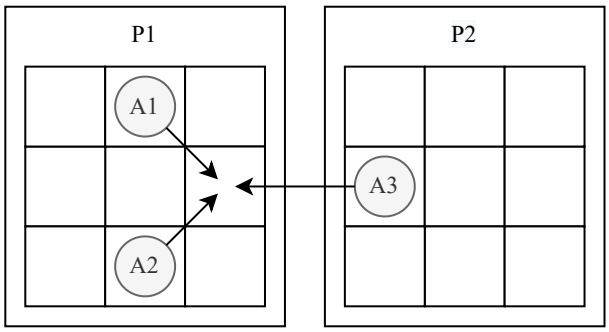


Figure 2.2: Example of conflicting agent decisions.

It is possible to avoid this conflict if the model is deterministic and the data necessary to predict the validity of actions is available locally — this conclusion is also presented and elaborated in [66]. Unfortunately, this cannot be applied to stochastic models, as the decisions in the remote process cannot be predicted. Another approach to circumvent this problem is the application of a model with a built-in mechanism that handles or prevents collisions. A very good example of a model that meets these criteria is the simulated life of microscopic benthic creatures — *foraminifera* — which was modeled in a way that prevents the occurrence of conflicts [10]. The model allowed multiple agents to coexist in a single cell and no action of one agent could influence the behavior of another. The distribution of this simulation required an additional synchronization step of aggregating remote updates to determine the final state of a cell, but did not involve any collisions and allowed for relatively easy parallelization.

Both discussed cases rely heavily on specific properties of the modeled scenarios, and therefore their applicability is not guaranteed for an arbitrarily selected model. A large subset of more complex simulation scenarios do not display any of those facilitation factors. As such, a class of CA-inspired, nondeterministic models allowing for the creation of mutually exclusive updates remains a significant challenge for efficient parallelization.

Usually, the approach to solving this problem relies on a careful analysis of a given problem and development of a solution dedicated to this specific problem. An example is the thorough analysis of the polymer dissolution phenomena in [5], where the authors suggest several parallelization schemes with different drawbacks and limitations. It should be noted that this problem does not actually utilize agents and, as such, does not belong to the ABMS domain, but the nature of the discussed challenges is very similar to the ones encountered in ABMS.

Development of a universal solution that could provide guidelines for model creation ensuring good scalability is a far more difficult task. However, the success would limit the work required for parallelization of the newly created ABMS.

2.3.2. Existing approaches

Generalization of the distribution and synchronization scheme for ABMS has already been attempted in the past. The most thorough analysis is presented in [8], where three different schemes have been discussed in detail. The following is a description and additional analysis of each collision-free migration strategy applied in the grid with the Moore neighborhood.

Location-ordered migration. This method relies on the fact that any given cell and its contents can influence only the neighboring cells. The grid is subdivided into a lattice of 3×3 cell regions and the cells are labeled according to their locations within the region. An example of such labeling is shown in Figure 2.3. As a result, all

cells with the same label are separated by the distance of at least two cells. Another way of stating the same, which bears a lot of importance for this method, is that no cell neighbors two cells with the same label — Figure 2.3 shows the influence range for each location labeled as 4. A single step of the simulation is then divided into nine decision-synchronization substeps that involve only the cells with the same label. In the resulting scheme, no cell can be influenced by two different sources in the same substep. This strategy seems to be the most intuitive approach to collision avoidance and as such, although not referred to as *location-ordered migration*, has been used in the past in works such as [34], where it facilitated the simulation of pedestrian dynamics on GPUs.

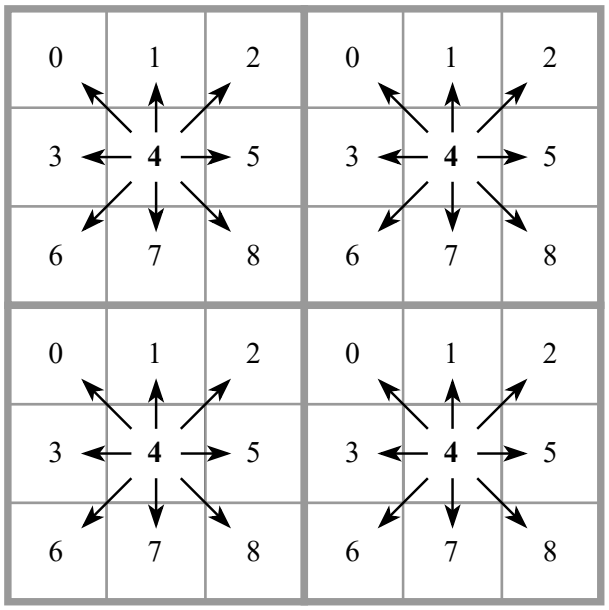
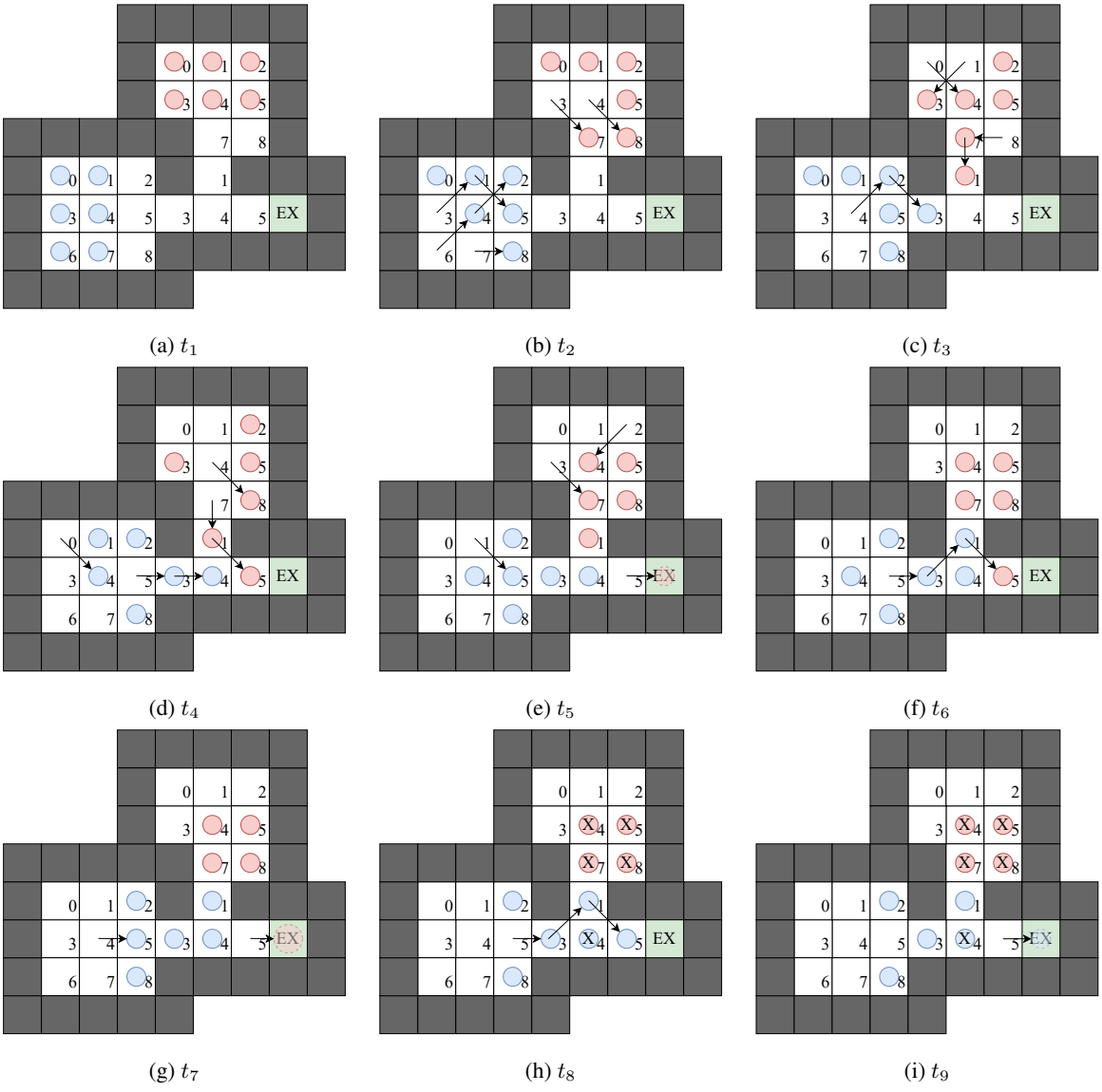


Figure 2.3: Grid subdivision in location-ordered migration.

The authors themselves state that this strategy generates a lot of substeps with all related costs: environment scanning, decision loop, remote communication, etc. An additional, unmentioned drawback of this approach is the lack of support in case of several agents being allowed to occupy the same cell. In that case, a single cell could generate conflict, although it could be resolved locally within the same process that generated the conflicting decisions, which greatly reduces the complexity of this issue.

In the context of the goal of this work, it is necessary to point to yet another, much more subtle drawback of this strategy. The labeling of the grid is based purely on the geometry of the grid, without taking the characteristics of the environment into account. Therefore, ordering the locations introduces an implicit prioritization that does not correspond to the model design. The demonstration of the consequence of this behavior can be seen in Figure 2.4 (the same problem was shown in a similar figure originally published in [52]). In this example, there are two rooms filled with agents representing people, as well as an exit in the corridor that joins the rooms marked EX. Agents are allowed to traverse the white cells in order to reach the exit, while the grey cells are impassable and represent walls. The location-ordered labels are shown in the corners of the empty cells. In each of the time steps represented in the subfigures, the full cycle of substeps is performed with arrows showing the movement performed by the agents. The time steps t_1–t_6 (Figure 2.4a–Figure 2.4f) show the agents initiating their travel towards the exit, but the steps t_7–t_9 (Figure 2.4g–Figure 2.4i) yield a crucial observation: due to the cell labeling within the corridor, the agents marked with X signs will not be able to perform any movement until the bottom left room (initially populated by blue agents) is completely empty. Steps t_8 and t_9 will repeat as long as there are more blue agents, except for the one stuck in the crossing. It is easy to show that if the room was extended toward the left or bottom, all its inhabitants would still be prioritized over the ones marked by X, while extending the top room (initially



(a) t_1 (b) t_2 (c) t_3

(d) t_4 (e) t_5 (f) t_6

(g) t_7 (h) t_8 (i) t_9

Figure 2.4: Unexpected phenomenon emerging from location-ordered migration.

populated by red agents) would only result in an increase in the number of agents waiting until the corridor is completely empty. The most important conclusion from this observation is that this method introduced a behavior that was not explicitly dictated by the model.

Direction-ordered migration. This strategy deals with the problem of conflicts using the same principal idea as the previous one, i.e. separation of decisions into batches that are incapable of creating collisions. In this case, the substeps do not focus on any specific grid subdivision, but on the direction of movement instead. An example of such an ordering could be the agents performing moves in clockwise order, as shown in Figure 2.5 — first to the top, then top-right, then right, and continuing until the last substep moves all agents with the intention of moving to the top-left. As a result, no two agents can cause the update of the same cell, as each cell is in a given directional relationship with only one cell (e.g. to the right of only one cell).

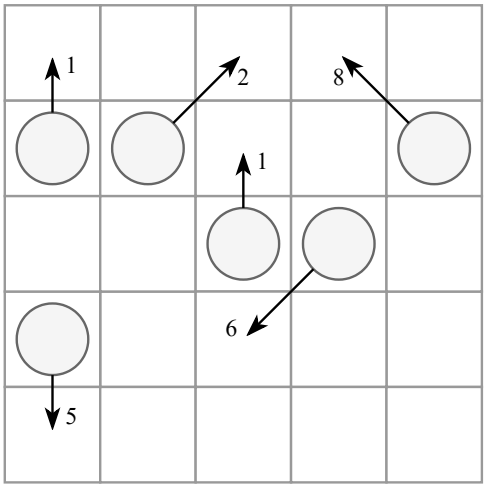
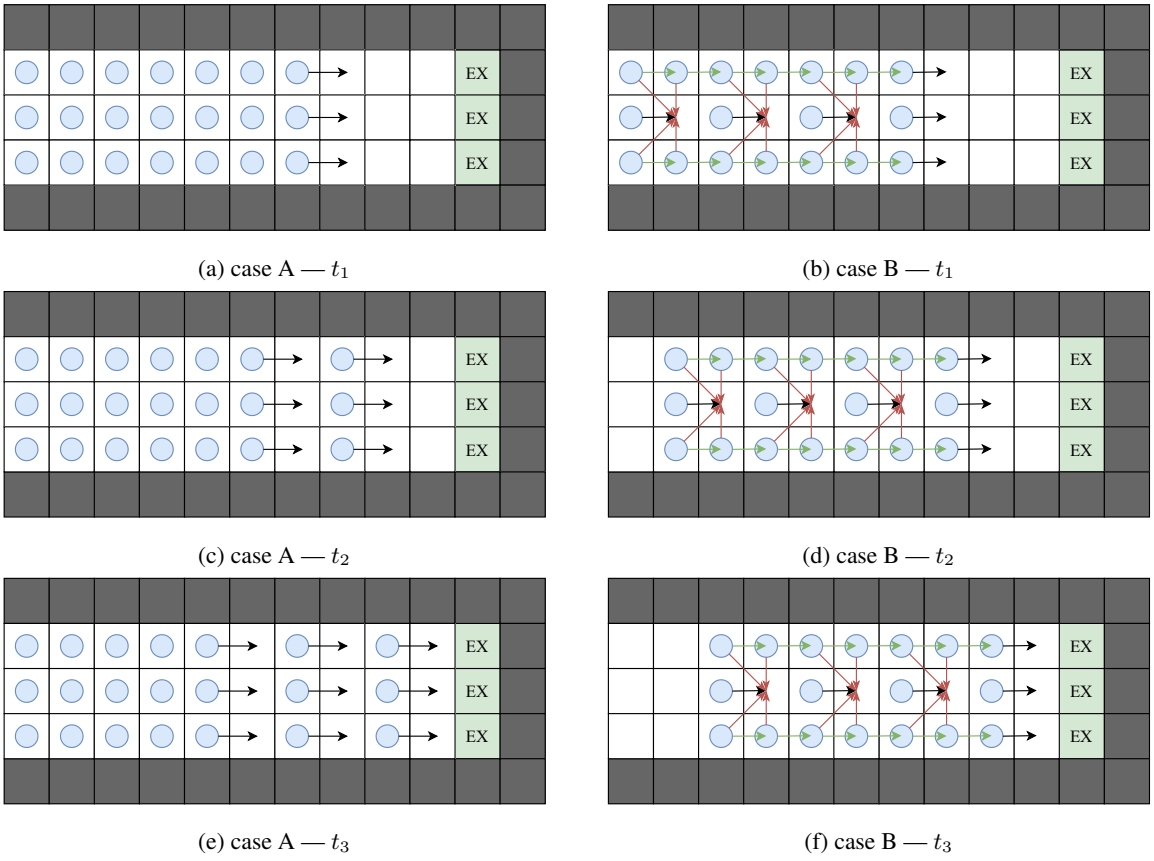


Figure 2.5: Direction ordering in direction-ordered migration.

The above commentary applies to this method as well: repeated cycles of decision making and synchronization, lack of support for multiple agents in a single cell, and implicit prioritization that is not built into the model. In this case, it is much easier to provide an example of the significant influence of this prioritization on the simulation. Whenever two incoming streams of agents would compete for the space in a single output stream with lesser throughput than the combined throughput of the incoming ones (e.g. two corridors merging into one with the same width), one of the streams would have to wait until the other one is fully exhausted.

Trial-and-error migration. The third of the collision-free strategies does not attempt to create non-colliding substeps. In this case, all agents can make decisions simultaneously, but in case of a conflict, only one move will be performed. Each rejected action is compensated with an additional opportunity for the decision given to the agents that did not win the conflict. As the initially chosen direction becomes unavailable to the processed agent in this iteration, this process can only repeat up to 8 times, once for each direction, assuming the decisions of the agent were rejected in each of the previous steps.

Like all the previously mentioned strategies, this one also generates additional subcycles of simulation. In this case, the possibility of handling multiple agents in the same cell is built into the method, but the hidden influence of this strategy on the phenomena occurring in the grid can now result from two different sources. The first, trivial influence is the need to explicitly prioritize agent decisions for conflict resolution and selection of the movement to be applied. The second, less obvious, is the introduction of ”unfairness” in the handling of agents. The agent that settles for the less desirable action as its initial decision (or is unable to perform any action) will not have



(a) case A — t_1 (b) case B — t_1

(c) case A — t_2 (d) case B — t_2

(e) case A — t_3 (f) case B — t_3

Figure 2.6: Unexpected phenomenon emerging from trial-end-error migration.

the opportunity to perform an additional decision, in which it could achieve much better results. A good example of this phenomenon is shown in Figure 2.6 (the same problem was shown in a similar figure originally published in [52]). This example shows two groups of agents in a passageway that ends with exit cells marked EX. In the left example (case A) the crowd is densely packed, while in the right example (case B) there are several empty cells in the crowd. Following the time steps in case A it is visible that in t_1 (Figure 2.6a) only the front column can perform any movement, leaving space for the second column to move in t_2 (Figure 2.6c), third column in t_3 (Figure 2.6e) and so on, until the whole crowd is moving in a pattern of alternating occupied and empty columns. At the same time, case B presents a completely different behavior. In the first time step t_1 (Figure 2.6b) the front column is allowed to move, as well as the agents that have empty spaces directly in front of them, but all other agents attempted to move into the empty spots as well and were rejected (red arrows). Then, after the first trial was finished, all agents attempted to move forward, this time successfully (green arrows). The resulting state at the beginning of t_2 (Figure 2.6d) contains an identical formation of agents as in the previous step, but moved forward by one cell. The same will repeat in further steps, effectively leading to almost doubling of the throughput of the corridor.

2.3.3. Problem definition

The mentioned strategies clearly possess several undesired qualities, other than the additional cycles of decision making and synchronization, that must be taken into consideration when designing the synchronization scheme:

– All methods are designed as migration strategies for a grid-based environment with a Moore or von Neumann neighborhood. The consequence is that their main focus lies in transporting agents between adjacent cells,

while preserving one-agent-per-cell restriction. A method designed with these limitations in mind will not be easily adaptable for models that stray from those restrictions, such as a different definition of neighborhood, different types of actions, multiple agents occupying the same cell, different modes of interaction between different types of agents, etc.

– Each method introduced a change into the model, which was not originally intended for the represented real-world scenario. Consequences may be drastic and clearly visible in the resulting simulation, forcing the creator to take those changes into consideration and adjust the model. However, they can also be subtle or very complex in nature, leading to changes that are not easily observable but fundamentally change the results, making them unfit for creating solutions to be applied in the real world. In such a case, it is much more challenging for the model creator to thoroughly measure the modeled scenario and ensure that all variables and results match the collected data. In case of not being able to measure the reference values associated with the scenario, e.g. due to the lack of ability to reproduce it in the real world, the created model can be simply incorrect without a meaningful way to troubleshoot or even notice the discrepancies.

These conclusions serve as guidelines for the research presented in this thesis. In the context of this analysis, the desired qualities of the parallelization method presented in Section 1.3 can be rephrased as follows:

– The method must introduce no hidden changes in the model. If a mechanism would require additional decisions not already present in the model, it must be explicitly required to be included by its creator.

– The method must allow good scalability, capable of efficiently utilizing HPC resources.

– The method must not limit its applicability to a narrow class of ABMS, instead providing as much flexibility and extensibility as possible.

To assess whether a method possesses those properties, each of them must be expressed in a way that allows a measurable goal to be established. The lack of discrepancies introduced into the model is a challenging quality to express mathematically, therefore Chapter 6 will focus on this challenge. Scalability will be assessed using widely used metrics such as speedup and efficiency in Chapter 7. Finally, as the applicability is an unquantifiable quality, Chapter 8 will present various models that have been successfully implemented using this method, as well as ongoing research regarding the compatibility with other models and more broad principles often used in modeling.

2.4. Signal propagation method

To address the problem of accessing remote information in process of decision-making, a different approach to agent sensing has been proposed: *signal propagation* [62, 75]. Inspiration from nature processes such as smell diffusing in the air serve as a basis for this method. The responsibility for providing long-range information is transferred from an agent to the environment, allowing the agent to only observe the state of its own cell for the purpose of scanning the environment for other agents or other features. As this method is crucial for the parallelization of ABMS, its full explanation is included below.

The loosely worded formulation of this approach is the following: each cell of the grid receives an additional subdivision layer serving as a *signal medium*. This subdivision consists of a 3×3 grid, in which each cell contains information regarding the strength of the signal in a single direction. The direction represented by a given cell corresponds to its geometrical location within the cell, i.e. the top-left subcell shows the strength of the signal incoming from the cell to the top-left of the current cell.

The more strict original definition requires the listing of the requirements regarding the type of models to which it is applicable, as well as the formalization of the description of the state of such models:

– The environment of the model is represented by a rectangular grid consisting of $m * n$ grid cells of equal size, denoted gc_{ij}.

– Each cell is connected to its 8 neighbors, as in the Moore neighborhood.

– Each cell can either be a *free cell* or an *obstacle*.

– Free cell can be occupied by a *task* T or an *agent* A.

– Obstacle cannot be occupied by an agent or a task.

– Tasks appear in the environment dynamically and are assigned a priority, which can change over time.

– Agents can move from the cell they occupy to an adjacent (connected) cell. Their purpose is to reach a task, which also represents execution of the task, and remove it from the grid.

The set of grid cells GC can be then defined as in Equation 2.1 — a cell is denoted by its grid coordinates:

$$GC = \left\{ gc_{i,j} : \begin{matrix} 1 \leq i \leq m, \\ 1 \leq j \leq n \end{matrix} \right\} \tag{2.1}$$

Furthermore, the set of subcells GSC can be defined within context of each grid cell as in Equation 2.2, where each cell $gc_{i,j}$ is assigned a subset of subcells $gsc_{i,j}^{x,y}$. The central subcell $gsc_{i,j}^{0,0}$, which would represent the signal in the direction of the current cell, is not used, as it serves no purpose. In the following equations this restriction still holds, but it will be omitted for brevity.

$$GSC = \left\{ gsc_{i,j}^{x,y} : \begin{matrix} 1 \leq i \leq m, \\ 1 \leq j \leq n, \\ x, y \in \{-1, 0, 1\}, \\ \neg(x = 0 \wedge y = 0) \end{matrix} \right\} \tag{2.2}$$

The correspondence between the subcoordinates x and y and the direction of the signal can be further explained as follows: the presence of a task in cell $gc_{i,j}$ will generate a directional signal with a value dependent on its priority in the subcells defined in Equation 2.3:

$$\forall_{x \in \{-1,0,1\}} \forall_{y \in \{-1,0,1\}} gsc_{i-x,j-y}^{x,y} \tag{2.3}$$

The visual representation of the same principle is shown in Figure 2.7. Cells are outlined in solid borders and numbered according to the coordinates outside axes, subcells are outlined in dashed borders and numbered according to the subcoordinates inside axes. As an example, the signal has been propagated to the subcell $gsc_{i+1,j}^{-1,0}$, which satisfies Equation 2.3. For this case, the chosen orientation of the coordinate system denotes the cell coordinates with row first and column second, with origin in the top-left corner of the grid, as the negative values are usually not used.

Each subcell is associated with (i.e. contains) a signal value. The following function in Equation 2.4 will be used to denote the signal value in any given grid subcell:

$$\begin{gathered} GSV : GSC \to \mathbb{R} \\ GSV(gsc_{i,j}^{x,y}) = gsv_{i,j}^{x,y}, \begin{matrix} 1 \leq i \leq m, \\ 1 \leq j \leq n, \\ x, y \in \{-1, 0, 1\}, \\ \neg(x = 0 \wedge y = 0) \end{matrix} \end{gathered} \tag{2.4}$$

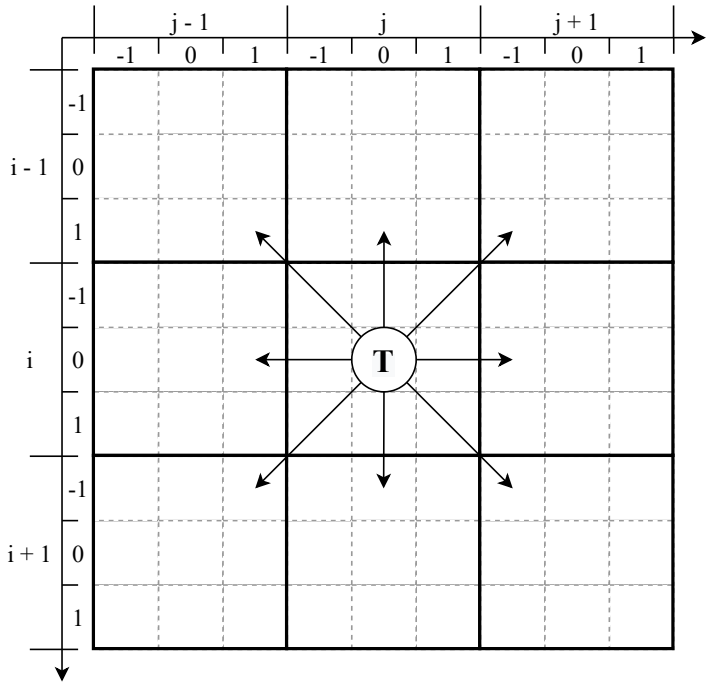


Figure 2.7: Direction of initial signal propagation.

The value of the signal should be interpreted as the incentive of an agent that would occupy the given cell to move in the direction of the considered signal. It is the composition of the priority of the task and the distance to the task, with the ability to aggregate information regarding multiple tasks in the same direction. On the other hand, the agents themselves emit a signal opposite to the signal of the task, although not necessarily of the same absolute value. This feature allows the agent to discourage other agents from crowding the same task and, in the presence of other tasks, allows the agents to prefer the tasks with no other agent in their vicinity. Additionally, information about completed tasks and other agents does not persist indefinitely in the environment, which would lead to agents being drawn to now empty locations.

To achieve these properties, the initial signal is subject to the following transformation over time:

– The signal spreads in the direction determined by its original direction, with the value reduced by some factor due to the increasing distance to the source. This behavior allows the signal to reach further areas of the grid while retaining the information about distance.

– The signal is reduced by a predefined factor due to its dissipation. This behavior allows the signal to dynamically adapt to changes in the environment, leading to the gradual disappearance of the signal associated with completed tasks and agents that are no longer present at this location.

Moreover, it is necessary for the signal to be able to reach all cells within reach, so that any agent has access to information regarding tasks that require its attention.

To represent the weakening of the signal with distance, time, and the ability to reach all unobscured parts of the grid, the following *signal propagation function* $GSPF$ for the grid environment is defined:

$$
\begin{gathered}
GSPF : \mathbb{R} \to \mathbb{R} \\
GSPF(gsv_{i,j}^{x,y}) = \\
\begin{cases}
SAF \cdot \left(gsv_{i,j}^{x,y} + SSF \cdot \left(gsv_{i+x,j+y}^{x,y} + gsv_{i+x,j+y}^{x+y,x+y} + gsv_{i+x,j+y}^{x-y,y-x}\right)\right) & \text{if } x = 0 \vee y = 0 \\
SAF \cdot \left(gsv_{i,j}^{x,y} + SSF \cdot gsv_{i+x,j+y}^{x,y}\right) & \text{otherwise}
\end{cases}
\end{gathered}
\tag{2.5}
$$

where SSF is *signal suppression factor*, i.e. the factor that reduces the value of the signal with distance, and SAF is *signal attenuation factor*, i.e. the factor that reduces the value of the signal with time. The visual representation of the signal propagation, omitting values and factors, is shown in figure Figure 2.8. The solid arrows show the initial propagation of signal from the task, the dashed arrows show one step of propagation, and the dotted lines show another step of propagation. This scheme ensures that the signal reaches each cell in the grid.

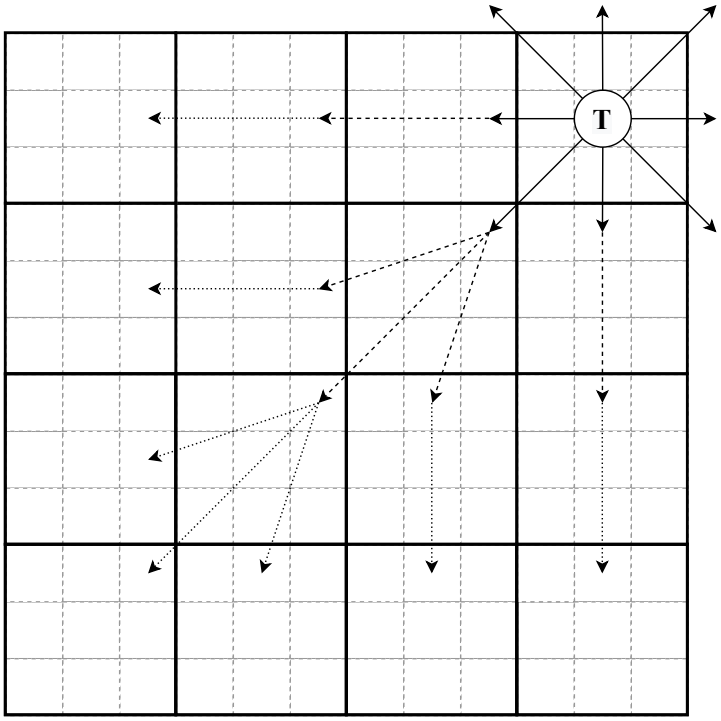


Figure 2.8: Scheme of signal propagation in grid environment.

The step of signal propagation can be repeated multiple times for each step of the simulation, or repeated only once per some number of simulation steps, taking the additional parameter defining the speed of the signal into consideration. Additionally, the authors of this approach proposed the mechanism of signal diffraction, allowing the signal to reach all cells even in the presence of obstacles on the grid.

The method of signal propagation effectively alleviates the problem arising from the need of environment scan across multiple processes. As all information can be provided by the signal present in the cell occupied by an agent, the decision can be taken without additional reading of the state of other cells. As the state of direct neighbors is usually critical for the agent, e.g. to prevent entering an occupied cell, the agents can be allowed to read the state of those cells. As a result, all models that use the presented signal propagation method can limit agents to accessing only the state of adjacent cells. The problem of reading the state of adjacent cells that are governed by another process can be easily addressed with the use of ghost cells.

2.5. Xinuk simulation method

This research is not focused on creating the complete simulation method from the ground up. The basis for the development of an effective parallelization method is the existing set of principles and mechanisms implemented as a ready-to-use ABMS platform named Xinuk described in [10]. Other platforms such as FLAME [11] or Repast [12] provide a similar level of complexity, i.e. a complete set of tools required for the creator of a simulation. The Xinuk platform differs from the other mentioned platforms in several ways, which the authors elaborate

on in [10]. The crucial difference lies in the use of the signal propagation method in the Xinuk platform, which is important for the purpose of accommodating models that require long-range environment scanning.

The selected approach was originally designed for models representing the environment as a 2D grid with Moore neighborhood, with agents able to make decisions that influence themselves, other agents, and the environment. As mentioned before, the distribution is aided by the signal propagation method explained in Section 2.4. As a consequence, it is possible to use a ghost cells concept for the purpose of inter-process synchronization, in this case named *buffer zone*.

The method performs very well in the context of scalability. The authors show very good speedup for up to thousands of computing cores, which means efficient applicability in HPC environments. The distribution is based on the subdivision of the grid into equally sized rectangle regions (the difference of one row or column is possible if the width or height of the grid is not divisible by the number of horizontal or vertical partitions) assigned to *worker processes*, also called *workers*. The full cycle of a single iteration for a single worker consists of the following steps:

- Iterate over all cells in the region, visiting cells row by row. If a decision would influence the remote cell, save the changes in the buffer zone for later synchronization.
- Iterate over all cells in the region, performing a step of signal propagation. If the signal should propagate into the remote cell, save the signal change in the buffer zone as well. This step can be repeated multiple times per iteration or once every several iterations, depending on the configured speed of the signal.
- Send the contents of the buffer zone to the owners of the cells mapped by the buffer cells.
- Receive the contents of the buffer zones from the neighboring workers.
- For each received buffer cell, combine the local contents and signal of the cell with the incoming contents and signal.

The last step leaves room for the possibility of conflict which must be resolved within the cell itself, which can pose a significant difficulty. In Chapter 5 a different set of steps is presented, while in Chapter 6 an approach to measure and assess the correctness of the distribution is proposed, as well as a comparison of both methods in terms of this newly defined metric. Additionally, Chapter 4 presents a modification of this base method that allows to remove the reliance on the 2D environment representation.

As was presented in this chapter, some of the more basic problems of parallelization and distribution of discrete spatial ABMS have been successfully solved:

- ghost cells as a means of remote cell state observation,
- division of simulation environments into parts assigned to worker processes as a baseline for the distribution,
- signal propagation method as a way to limit the range of environment scanning.

Those solutions provide the tools necessary for the distribution to occur. However, they are not enough to provide great scalability and ensure the correctness of the distributed simulation on their own. The commonly used representation for spatial models is not expressive enough for more complex environments, while the solutions mentioned above are designed with such a simple representation in mind. At the same time, there are many approaches to the application of conflicting decisions resulting from distribution, but none of them was created with correctness as one of the main objectives. As a result, each of the existing methods presented in this chapter introduces some bias into the model update process. Therefore, the challenge of creating a distributed state update method for ABMS remains unsolved.

3 Correctness in distributed discrete spatial ABMS

This chapter presents the steps undertaken to develop a proper definition of simulation method correctness. Some of the contents presented in this chapter were originally shown in publications [50] and [52] that are part of the research presented in this thesis. They serve as preliminary research that paved the way towards the better understanding of the concept of ”method correctness”.

3.1. Preliminary research towards distributed ABMS correctness definition

The first steps toward eliminating any influence the distribution method might have on the model were presented in a publication [50] mentioned above. The initial approach did not use the method described in Chapter 5 and was largely based on corrections and adaptations applied to specific models. The changes applied to the general method were minor, focusing instead on a better adaptation of existing models. The models used as examples were the same as those presented in Appendix A.

All models were influenced by randomness in decision-making processes. Lacking the ability to perfectly reproduce the same random results with varying degrees of distribution (i.e. with different numbers of computing processes or workers), the approach used in this attempt relied on the following statement: for any model, if the changes in metrics collected in different degrees of distribution are significantly smaller than the changes in metrics collected with different parameter configuration, the influence of distribution can be considered negligible.

In order to apply this statement to test cases, a set of configuration variants was created for each model. The selection of relevant parameters is presented in Table 3.1, Table 3.2, and Table 3.3, while the full set of parameters was originally published in [50]. The variant originally described as *default* will be referred to as *variant 0* in this text. The abbreviations used in the header rows are expanded and explained in Appendix A for each model. The variants 0 and 2 of the fire emergency evacuation model presented in Table 3.1 are identical, but in the generation of the initial environment the fire agents were placed five times more frequently, which greatly accelerated the spread of the fire.

Table 3.1: Fire emergency evacuation configuration variants.

Variant	HMS	FSF	HSV	FSV	ESV
0	1.0	1.0	-0.1	-0.001	1.0
1	1.0	1.0	-0.1	-0.5	1.0
2	1.0	1.0	-0.1	-0.001	1.0
3	1.0	1.0	-0.01	-0.001	1.0

Table 3.2: Foraminifera habitat configuration variants.

Variant	FSE	FRC	FRT	FLC	ARR	AEC	FSV
0	0.3	0.5	0.8	0.2	0.07	0.1	-1.0
1	0.3	0.5	0.8	0.2	0.07	0.05	-1.0
2	0.3	0.5	0.8	0.2	0.07	0.1	-5.0
3	0.3	0.5	0.8	0.1	0.07	0.1	-1.0

Table 3.3: Rabbits and lettuce configuration variants.

Variant	RSE	RRC	RRT	RLC	LRF	LEC
0	0.5	0.3	1.0	0.1	2.0	0.6
1	0.5	0.3	1.0	0.1	2.0	0.45
2	0.5	0.3	1.0	0.1	5.0	0.6
3	0.5	0.3	1.0	0.5	2.0	0.6

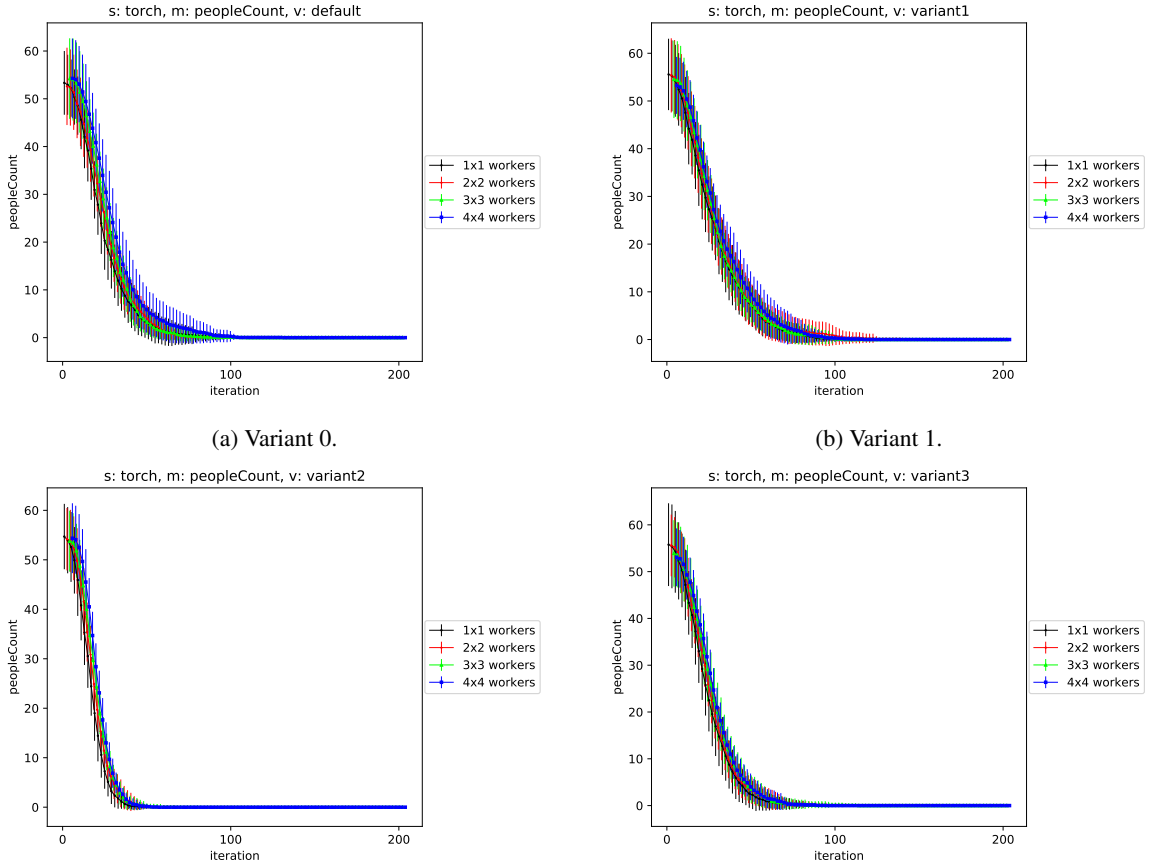
Each configuration variant was executed on a single worker, four workers, nine workers, and finally sixteen workers. Each experiment was repeated multiple times to collect a statistical sample. Two types of plots will be presented:

timelines — which present the change of the value of a metric over iterations. Each line represents all repetitions of a single configuration for a single degree of distribution, marking the mean of all repetitions with bars of standard deviation. Timelines are grouped by configuration, comparing different degrees of distribution in one plot.

aggregates — which present the average value across all iterations. Each aggregate takes the form of a boxplot, i.e. the shape that marks the median (horizontal bar), the first and third quartiles ($Q1$ and $Q3$, bottom and top sides of a rectangle containing median), minimal and maximal value ("whiskers" extending from the rectangle), and finally outliers that do not fit into the range $[Q1 - 1.5 \cdot IQR, Q3 + 1.5 \cdot IQR]$, where $IQR = Q3 - Q1$ (circles). The aggregates are grouped into fours that correspond to the same configuration, and the fours are grouped once again showing all variants on one plot.

In the case of the fire emergency evacuation model, the conclusion was that despite the fact that the model had to be extended with the ability of a human agent to form a crowd to cope with collisions, the goal statement suggested that the model was not affected by distribution. Figure 3.1, which was also included in the discussed study, shows the timeline of the metric tracking the number of people present in the environment. In this case, the shape of the plot differs between the variants, but the mean values in each plot are close and follow the same curve. As an example, the value of the metric reaches 0 around 100th iteration in Figure 3.1a and Figure 3.1b, but it only takes 50 iterations to remove all human agents from the environment in Figure 3.1c and 75 in Figure 3.1d. Similar conclusion arose from Figure 3.2, which shows the aggregates of the numbers of deaths (Figure 3.2a) and escapes (Figure 3.2b) of people. In all cases, the boxes for the same configuration are at very similar levels, but *variant 2* is significantly different from the other ones in both metrics. Therefore, the conclusion was that the model was not significantly altered by the method used for parallelization.

The test model representing a foraminifera habitat yielded similar results. As shown in Figure 3.3, *variant 0* (Figure 3.3a) and *variant 1* (Figure 3.3b) resulted in very similar timelines, where the population of foraminifera was reduced to zero, resulting in extinction. At the same time, *variant 2* (Figure 3.3c) led to the emergence of a stable population of foraminifera, and finally the *variant 3* (Figure 3.3d) shows oscillations, which hints that the scenario might be satisfying the Lotka-Volterra equations. However, the aggregates shown in Figure 3.4 seem to



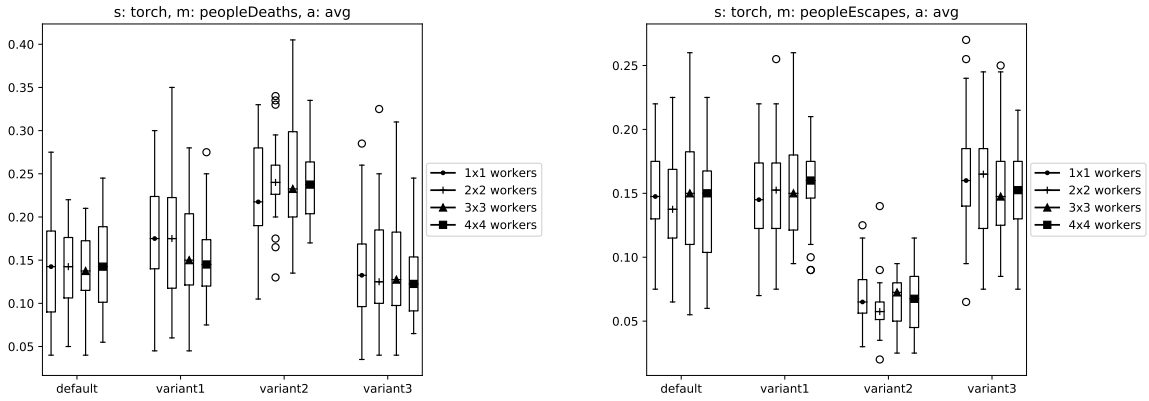
(a) Variant 0.

(b) Variant 1.

(c) Variant 2.

(d) Variant 3.

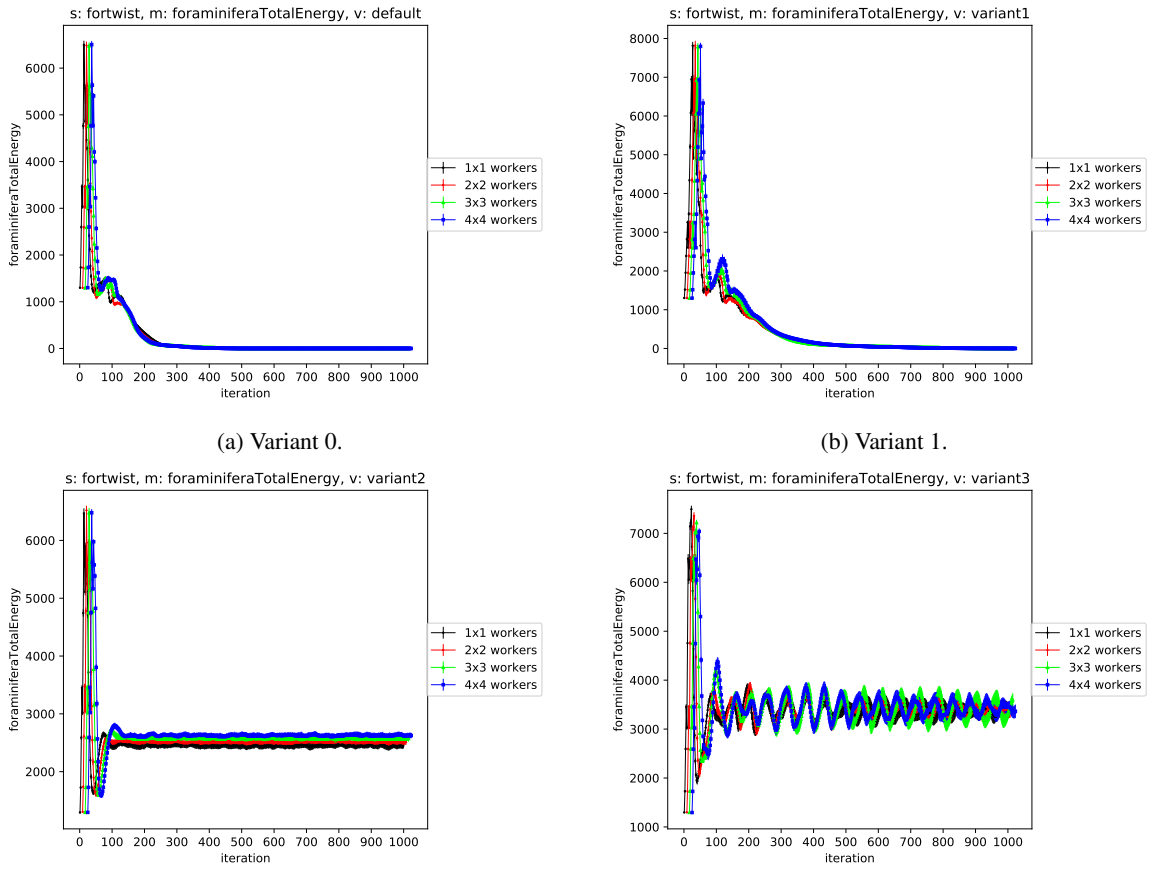
Figure 3.1: Fire emergency evacuation — timelines of number of people. Source: [50]



(a) Aggregate of number of people deaths.

(b) Aggregate of number of people escapes.

Figure 3.2: Fire emergency evacuation — aggregates of sample metrics. Source: [50]



(a) Variant 0.

(b) Variant 1.

(c) Variant 2.

(d) Variant 3.

Figure 3.3: Foraminifera habitat — timelines of total energy of foraminifera. Source: [50]

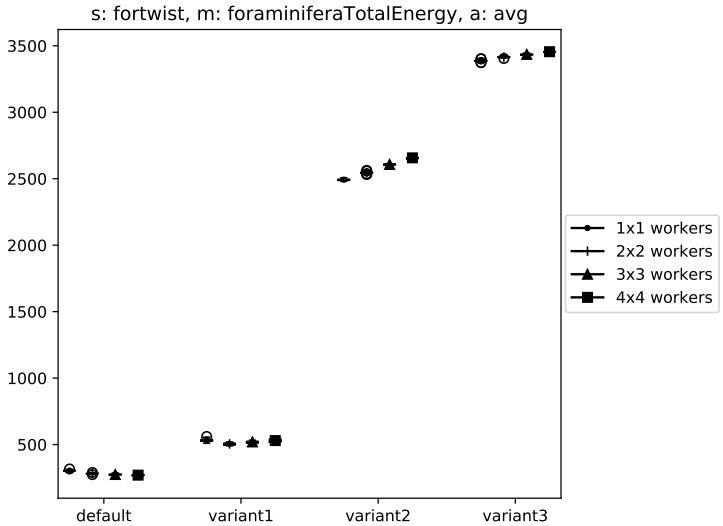
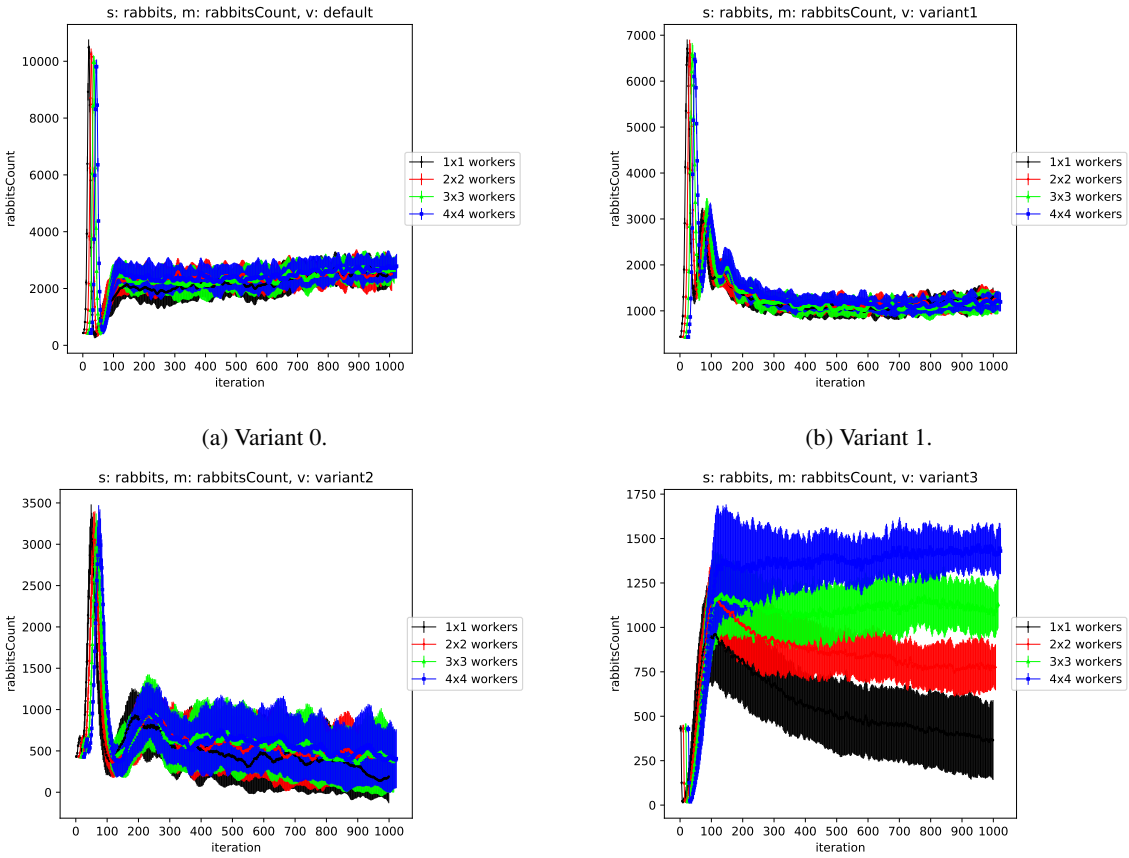


Figure 3.4: Foraminifera habitat — aggregate of total energy of foraminifera. Source: [50]

express little to no variance in the results, yet the differences between the distribution degrees are visible. However, the differences between variants are decidedly outweighing the differences within variants, which satisfies the assumptions made at the beginning of [50].



(a) Variant 0.

(b) Variant 1.

(c) Variant 2.

(d) Variant 3.

Figure 3.5: Rabbits and lettuce — timelines of rabbit count. Source: [50]

Nonetheless, the final example presents itself as a significant problem. The timelines of rabbit agent count in Figure 3.5, especially in *variant 3* (Figure 3.5d), show significant discrepancies that are enough to suggest that essentially each degree of distribution results in a completely different scenario. The only obvious difference between sequential execution and different degrees of distribution is the conflict resolution policy, which allows two rabbits to combine into one rabbit with energy equal to the sum of the energies of colliding rabbits. Finding the precise cause of the observed differences is not an easy task, but there are several possibilities: the new rabbit created by this combination process can have enough energy to reproduce, possibly multiple times in succession, or the rabbit can travel a much greater distance than two separate rabbits, because only one deduction of life activity cost from its energy occurs instead of two. These changes can cause a cascade of further changes to the state of the model that would be impossible in this configuration, essentially altering the model and its constraints.

The conclusions of the cited work can be summarized as follows: it is possible to adapt the model to the original method, but the responsibility for keeping the model true to the scenario lies on the model creator, and it is very easy to accidentally introduce significant changes into the logic. This conclusion was correct in terms of the original problem and the original verification method. However, the method of assessing the influence — or lack thereof — of the method on the model was subjective and largely based on the analysis of the timelines and aggregates of the metrics. It remains uncertain whether the slight differences observed in the foraminifera habitat model were significant enough to lead to observable changes in a longer runtime of the simulation, e.g.

after 100000 ($1e5$) iterations instead of 1000 ($1e3$). On the other hand, it is unclear whether such a runtime makes sense in the context of the modeled scenario and the precision of the model itself. Therefore, the definition of a specific number of iterations that need to be tested in the way presented in [50] is not a correct way towards a better verification that would be model-independent.

3.2. Definition of distributed simulation method correctness

The presented research and partially successful experiments served as a foundation for the definition of the distributed simulation method correctness:

A distributed simulation method is correct if the results obtained from simulation of any model implemented with the use of this method are not changed by the presence or a degree of the distribution.

Satisfying this definition requires proving the lack of change in the results for *all* models that can be implemented using this method. Therefore, the characteristics of the distributed simulation method must ensure that the model structure and its state update algorithm are not affected by the distribution, regardless of the specific model. However, in some cases, proving this for the method in general is not possible. At the same time, preliminary research presented in this chapter demonstrated that the tested method seemed to be correct for fire emergency evacuation. Such observations are important because they show that the method works well with a given model, but it is not accurate to claim the correctness of the method in general. To address this type of situation, the definition of correctness limited to a single model was formulated:

A distributed simulation method is correct for a given model if the results obtained from simulation of this model implemented with the use of this method are not changed by the presence or a degree of the distribution.

Several aspects of this second definition are crucial and need to be emphasized:

– The method cannot be verified in isolation. It is necessary to provide a model, as the correctness does not necessarily extend to other models.

– The results of the execution of a model are defined as the data that can be obtained from that execution, e.g. in the form of collecting some metrics throughout the simulation run. The inner workings of the model or its updates are not taken into account. The consequence of this is that the addition of some new metrics to the same model requires a new verification.

– Multiple different degrees of distribution must be analyzed. The correctness is not measured directly because it depends on the presence or absence of a difference.

– Results from non-deterministic models cannot be compared directly across runs, as they can differ even in identical configurations unless the source of randomness is fixed. Instead, the statistical distribution of the results needs to be compared, as the artificial removal of randomness can hide some phenomena observable in the model.

4 Core method environment model generalization

In this chapter, the problems existing in the Xinuk core method are highlighted. Modifications to this method and to the signal propagation method are presented that will significantly increase its flexibility and applicability. The changes will be focused on removing the reliance on the 2D grid-based environment representation present in both mentioned methods.

4.1. Increasing flexibility of the model structure in core method

The core method used as a starting point for this research, which was described in Section 2.5, is limited to a regular grid with the Moore neighborhood. This is caused by the grid representation, relying on the two-dimensional array implementation with index-based adjacency.

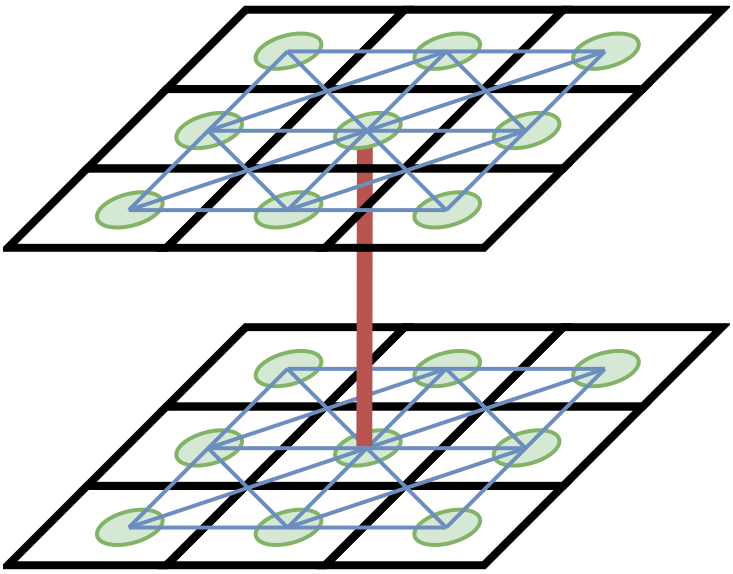


Figure 4.1: Example of connected floors forming non-planar graph.

The research presented in [49] has shown that a multitude of models cannot be presented using a simple 2D grid due to elements derived from real-world 3D environments. At the same time, the nature of such simulations usually does not call for the utilization of a 3D grid, which would cause a vast increase in memory and time requirements. A very good example is a building consisting of multiple floors interconnected via staircases. Mapping to a 2D grid, e.g. by placing each floor in a separate area of a grid, is impossible due to the connections between the floors. Another, more formal phrasing of the same problem is the following: the graph representing the grid cells as vertices and the neighborhood relationship between cells as edges may be non-planar, which precludes it from being represented on a 2D grid. A very simplified example of the configuration of real-world environment,

impossible to be simply mapped to a 2D grid, is shown in Figure 4.1. Green circles depict the vertices representing the cells, while blue lines represent edges representing the full Moore neighborhood of each separate floor. The additional wide red line represents a connection between the centers of floors, e.g. staircase, ladder, etc. It is impossible to rearrange the cells into a single 2D plane so that all edges are represented by a neighborhood between two cells.

4.2. Transition from grid environment representation to graph

The unnecessary limitation of being able to model only 2D-grid-based environments is relatively simple to remove. The solution emerges from the brief discussion presented above — the problematic arrangement is easily expressed using a graph. Therefore, the method was modified to forgo the array-like grid representation with a forced neighborhood derived from adjacency in favor of a graph representation with a fully configurable neighborhood.

The idea of representing the environment as a set of indivisible fragments remains unchanged. To this end, the set of cells C is defined to contain all possible places any agent can reach or interact with for a given model. As opposed to grid cells GC, they are not assigned any data that would imply their location within the environment. On the other hand, the set GC can be used as a specific instance of C for a given model — this modification is intended to expand the possibilities without removing the ability to use grid-based models. In the following examples, the set of grid cells GC will be referred to as C^{Moore} and the specific cells $gc_{i,j}$ simply as $c_{i,j}$.

Navigation between cells is provided by the edges connecting the cells. Each edge is labeled to convey information about the direction from one cell to another. The consequence is that each bidirectional connection must be assigned a different label from the perspective of each cell, which in practice requires two separate unidirectional edges in such scenarios. The side effect of this requirement is the possibility of creating one-way connections between cells, facilitating the creation of models benefiting from such features, e.g. traffic simulations, escalators in shopping centers or airports, etc. To achieve this, each model, in addition to the set of cells, must define a set of possible directions D. As one of the main focuses of this modification was ensuring that the new representation does not lose any of the expressiveness present in the old one, the grid-based environment can be easily represented using the modified approach. The following set of 2D grid-based directions D^{Moore} can be used to recreate adjacency-based connections:

$$D^{Moore} = \{N, NE, E, SE, S, SW, W, NW\} \tag{4.1}$$

Each of the directions can be derived from the index-based Moore neighborhood used in the subcells concept described in Chapter 2. Such mapping $I2D_{Moore}$ is shown in Equation 4.2:

$$I2D^{Moore} : \{(x, y) \mid x, y \in \{-1, 0, -1\}, \neg(x = y = 0)\} \to D^{Moore}$$

$$I2D^{Moore}(i) = \begin{cases} N & \text{if } i = (-1, 0) \\ NE & \text{if } i = (-1, 1) \\ E & \text{if } i = (0, 1) \\ SE & \text{if } i = (1, 1) \\ S & \text{if } i = (1, 0) \\ SW & \text{if } i = (1, -1) \\ W & \text{if } i = (0, -1) \\ NW & \text{if } i = (-1, -1) \end{cases} \tag{4.2}$$

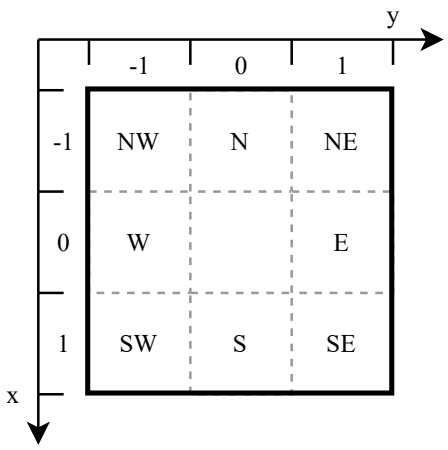


Figure 4.2: Mapping from Moore neighborhood subcells to directions.

To facilitate the analysis of this mapping, the same has been shown in Figure 4.2, in which the subcells are labeled with the direction they correspond to. Using this mapping process, it is possible to replicate the same environment configuration that could be modeled using the old approach.

An additional useful function to be defined is the actual retrieval of the neighbor of a given cell in a given direction:

$$NGH : C \times D \to C \tag{4.3}$$

The definition of a mapping from the Moore neighborhood NGH^{Moore} using the set C^{Moore} of grid cells is presented in Equation 4.4, again aided visually by Figure 4.3.

$$NGH^{Moore} : C^{Moore} \times D^{Moore} \to C^{Moore}$$
$$NGH^{Moore}(c_{i,j}, d) = \begin{cases} c_{i-1,j} & \text{if } d = N \\ c_{i-1,j+1} & \text{if } d = NE \\ c_{i,j+1} & \text{if } d = E \\ c_{i+1,j+1} & \text{if } d = SE \\ c_{i+1,j} & \text{if } d = S \\ c_{i+1,j-1} & \text{if } d = SW \\ c_{i,j-1} & \text{if } d = W \\ c_{i-1,j-1} & \text{if } d = NW \end{cases} \tag{4.4}$$

However, the graph representation is not limited by the same constraints as the grid. The example from Figure 4.1 can be easily modeled by using a different set of directions, or extending the D^{Moore} set — by adding the directions representing *up* and *down*, it is possible to connect the cells exactly as denoted by the red line. This improvement would make the method capable of modeling other, much more complex environments that would not be able to efficiently utilize a fully 3D environment, such as multistory buildings, insect nests, habitats of burrowing animals, etc.

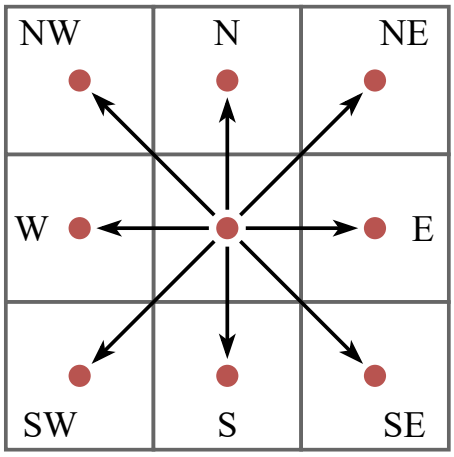


Figure 4.3: Grid overlaid with a part of graph representing the same environment.

4.3. Adaptation of signal propagation method to graph

From the newly created graph representation of environment emerged the challenge of maintaining the applicability of signal propagation. The original formulation of this technique was firmly grounded in the grid geometry, therefore, it required redesigning in the context of graph.

As in the case of the grid version of signal propagation, a signal value is associated with a connection between cells. For the graph representation, the propagation of this value can be defined for each cell-direction pair, instead of using the subcells concept. The new signal value SV can be defined as in Equation 4.5:

$$SV : (C \times D) \to \mathbb{R} \tag{4.5}$$

The signal propagation relies heavily on the relations between directions, originally expressed in terms of displacement vectors and represented by subcells. The simplified rule of propagation states that the signal pointing in some direction must influence the signal pointing in the *same* direction, but in the cell located in the *opposite* direction to the signal. As an example, if a subcell pointing to the *right* (or east) of the given cell contained some signal value, it should be propagated to the *right* subcell of the *left* (or western) neighbor of the cell. Therefore, it is necessary to define a relation OPP between directions which marks them as opposite to each other:

$$OPP : D \to D \tag{4.6}$$

Such a relation can be easily defined for the Moore neighborhood as OPP^{Moore} in Equation 4.7. This relation is symmetrical in the case of Moore neighborhood, but the definition in the form of a function allows for the

introduction of an asymmetrical configuration, and therefore this notation was selected.

$$OPP^{Moore} : D^{Moore} \to D^{Moore}$$

$$OPP^{Moore}(d) = \begin{cases} S & \text{if } d = N \\ SW & \text{if } d = NE \\ W & \text{if } d = E \\ NW & \text{if } d = SE \\ N & \text{if } d = S \\ NE & \text{if } d = SW \\ E & \text{if } d = W \\ SE & \text{if } d = NW \end{cases} \tag{4.7}$$

Trivially, the following will hold for the graph created from a grid using the neighborhood definition in Equation 4.3:

$$NGH^{Moore}(c_1, d) = c_2 \iff NGH^{Moore}(c_2, OPP^{Moore}(d)) = c_1 \tag{4.8}$$

The specific signal propagation function defined for the 2D grid Moore neighborhood used a special rule regarding diagonal signal propagation. Instead of spreading the signal only in a diagonal direction, the signal pointing in a diagonal direction is also influencing the cells in the direction *adjacent* to the one opposite to the signal. Therefore, an additional relation ADJ between directions is necessary to mark the mentioned adjacency of the directions:

$$ADJ : D \to D^* \tag{4.9}$$

This relation is more complex, as a single direction can be adjacent to multiple directions, e.g. 2 in Moore neighborhood, as shown in Equation 4.10.

$$ADJ^{Moore} : D^{Moore} \to D^{Moore*}$$

$$ADJ^{Moore}(d) = \begin{cases} \{NW, NE\} & \text{if } d = N \\ \{N, E\} & \text{if } d = NE \\ \{NE, SE\} & \text{if } d = E \\ \{E, S\} & \text{if } d = SE \\ \{SE, SW\} & \text{if } d = S \\ \{S, W\} & \text{if } d = SW \\ \{SW, NW\} & \text{if } d = W \\ \{W, N\} & \text{if } d = NW \end{cases} \tag{4.10}$$

Once again, to allow models to use an asymmetrical relation, the form of the function is chosen as a more flexible option. The Figure 4.4 shows an example of both relations defined for the set of directions D^{Moore} derived from grid-based representation of the neighborhood: OPP^{Moore} (opposite) and ADJ^{Moore} (adjacent).

Finally, the signal propagation function SPF can be reformulated in terms of the concepts introduced. For each cell-direction pair the updated value of the signal can be calculated:

$$SPF : (C \times D) \to \mathbb{R} \tag{4.11}$$

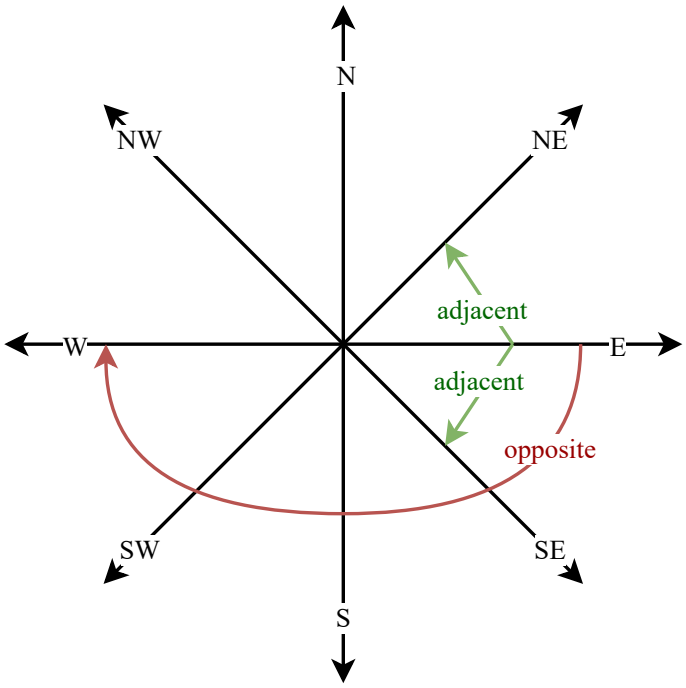


Figure 4.4: Example of relations between directions derived from grid-based environments: directions opposite and adjacent to E (east) direction.

Using all grid- or Moore-specific relations and definitions mentioned above, the signal propagation for the Moore neighborhood can be rewritten in terms of graph representation as follows in Equation 4.12:

$$SPF^{Moore} : (C^{Moore} \times D^{Moore}) \to \mathbb{R}$$
$$SPF^{Moore}(c, d) =$$
$$\begin{cases} SAF \cdot \left(SV(c,d) + SSF \cdot \sum_{d' \in (ADJ^{Moore}(d) \cup \{d\})} SV(NGH^{Moore}(c,d), d')\right) & \text{if } d \in \{N, E, S, W\} \\ SAF \cdot \left(SV(c,d) + SSF \cdot SV(NGH^{Moore}(c,d), d)\right) & \text{otherwise} \end{cases} \tag{4.12}$$

The parameters SSF and SAF remain the same as in the original signal propagation definition. The resulting function will yield results identical to the one defined for the grid.

The new graph representation of the environment is far more flexible and especially useful for scenarios in 3D space (majority of real-world scenarios) that do not utilize the full 3D space. Specific applications might require the definition of custom directions, which in turn might require careful planning regarding the definition of ADJ and OPP functions involving those new directions. The function proposed in Equation 4.11 can be easily adapted to new direction sets and even completely replaced by a different scheme. The major benefit of the presented improvement, aside from the mentioned flexibility, is the detachment from the index-based, error-prone calculations to more intuitive, direction-based ones.

5 Abstract super-scalable correct ABMS method

Chapter 2 thoroughly discusses the problems related to state synchronization in parallelized simulation and the challenge of designing a proper scheme for agent migration. Furthermore, the observations made in that chapter lead to the conclusion that the method should not be limited to handling only models with single agent per cell restriction and movement being the only available action, as it significantly reduces the scope of scenarios it can be applied to. As shown further in Section 2.5, the approach used as the core for this research is also constrained, allowing only the resolution of conflicts as a *post factum* correction of a cell state. It could be argued that this limitation is much more restrictive than those analyzed in Chapter 2. At the same time, one of the main goals of this research is to propose a method that imposes as few restrictions on the models as possible while providing good scalability. Therefore, in this chapter, a proposition of a new decision and state synchronization scheme is presented. The basis for this method is an approach presented in Section 2.5, with the generalized environment model presented in Chapter 4. However, the new method is not a direct extension of the original one — the core principles are preserved, but the proposed method ensures correctness of the distribution and provides an abstraction over many components: elements constituting the environment, agents and their states, decision-making algorithms. The same scheme was presented in a publication [52] that is part of the research presented in this thesis.

The proposed method is not limited to the grid environment — on the contrary, it fully utilizes the generalization presented in Chapter 4. The scope of the actions available to the agent is not limited to movement, and, finally, no intrinsic restriction exists in the method that precludes placing multiple agents in a single cell, as long as the model allows it. This method aims to achieve the goals outlined in Chapter 1 and is analyzed with respect to all those qualities in further chapters:

– correctness, analyzed in Chapter 6,

– scalability, analyzed in Chapter 7,

– applicability, analyzed in Chapter 8.

5.1. Model structure for the proposed method

Before the method itself can be described, it is necessary to define the model structure it operates on. The notation introduced in the following section will be later used to describe a simple model to better visualize the role of each model component.

5.1.1. Model definition

The model used by the proposed method can be defined as the following tuple of sets:

$$M = (C, A, E, S, U, P) \tag{5.1}$$

where the constituent sets are the following:

C — a set of available cells,

A — a set of possible actor-related states,

E — a set of possible environment-related states,

S — a set of allowed cell states,

U — a set of possible cell state updates,

P — a set of allowed plans.

The role of each set is discussed in detail below.

Set of available cells C. Since the new method preserves the discrete nature of the base method, it is necessary to provide a set of all discrete units of the environment, referred to as cells. The representation and definition of this set C is directly related to the specific implementation. Common examples of specific representations include a graph, in which cells are represented by vertices and the connections between them by edges, or a 2D grid, for which a set of cells GC was defined in Equation 2.1 (in Chapter 2). The presented method is independent of the chosen representation.

Set of possible actor-related states A. For simplicity of representation, the agent-related states for any given cell are provided as a separate set A. There are no additional restrictions imposed on this set — if a model allows multiple agents to occupy a single cell, then a group of agents with their internal states can be a single element of A.

Set of possible environment-related states E. The second important component of the model state representation, the environment-related states for any given cell are represented as elements of set E. Once again, the elements are wholly dependent on the model — they can be simple markers denoting the type of a cell, or more complex descriptions of the internal structure of the part of environment.

Set of allowed cell states S. The set S combines the components describing agent- and environment-related states into a complete cell state. It is defined as in Equation 5.2:

$$S \subset A \times E \tag{5.2}$$

It is important to note that S does not necessarily contain all possible combinations of state components, as the features of the environment might impose some restrictions on agents in the cell.

Set of possible cell state updates U. The elementary component of a decision the agents are allowed to make in the presented method is an *update*. The set of allowed updates U is defined as in Equation 5.3:

$$U : \{S \to S\} \tag{5.3}$$

Each element of the set U can be either a total function or a partial function. As a consequence, it is possible for a given update function to not be applicable to a given cell state, especially in the case of conflicting updates. This possibility is an intended behavior and is used as a means to detect such conflicts.

Set of allowed plans P. Finally, the complete and independent decision made by an agent is represented as a *plan*. A set of all plans allowed by the model can be defined as the subset of the cross product shown in Equation 5.4:

$$P \subset C \times C \times U \times (U \cup \{\varnothing\}) \times (U \cup \{\varnothing\}) \tag{5.4}$$

Similarly to the set of possible states, the set P also does not necessarily contain all possible combinations of components. The specific constraints emerge from the logic of the model. For better clarity, each element p in this set can be represented as the following tuple:

$$p = (source, target, action, consequence, alternative), \quad \begin{aligned} p &\in P, \\ source, target &\in C, \\ action &\in U, \\ consequence, alternative &\in U \cup \{\varnothing\} \end{aligned} \tag{5.5}$$

A plan is assigned a *source* cell and a *target* cell, although it is not necessary for them to be different cells. Furthermore, neither the model nor the algorithm described further in this chapter imposes any limitations on the definition of a neighborhood in the environment. However, it is usually useful to limit the range of performed actions, usually to the immediate neighborhood of the source cell. In that case, it is the responsibility of the set P to exclude plans with non-adjacent source and target cells.

The intention of the agent expressed by the created plan is represented by three components, of which one is mandatory and the other two are optional:

– *Action* — the update intended for the target cell. Can be accepted or rejected during the iteration.

– *Consequence* — the (optional) update intended for the source cell in case of the action being accepted.

– *Alternative* — the (optional) update intended for the source cell in case of the action being rejected.

The exact use of those components will be explained in detail in Section 5.2, as they serve as the focal point of the entire algorithm.

5.1.2. Exemplary model

To aid in explaining this structure, it is useful to present an example — a simple model of a 3×6 grid environment with the Moore neighborhood in which an agent can migrate to an empty adjacent cell. Additionally, each agent counts its actions in the form of two integers: one for each successful migration and another for each occurrence of staying in the cell. The described model structure is defined as follows, using the notation introduced in the previous section:

$$M' = (C', A', E', S', U', P') \tag{5.6}$$

The set of cells C' is a set of grid cells denoted by their coordinates:

$$C' = \{c_{x,y} \mid x \in \{0, 1, 2\}, y \in \{0, 1, 2, 3, 4, 5\}\} \tag{5.7}$$

The model allows only a single agent in each cell, therefore, the agent-related states A' include either a state of a single agent or the absence of the agent:

$$A' = \{(m, i) \mid m \in \mathbb{R}, i \in \mathbb{R}\} \cup \{\varnothing\} \tag{5.8}$$

The state of an agent consists of two numbers: the count m of successful migrations and the count i for iterations in which it stayed idle. The symbol $\varnothing$ represents the " null" element (the absence of agent), as opposed to the other popular meaning of this symbol being an empty set.

The environment is represented with the use of markers — each cell is either free or contains an obstacle:

$$E' = \{e_{free}, e_{obstacle}\} \tag{5.9}$$

The interaction of the environment and the agents is the following: the free cell can be occupied by any agent, while the cell with an obstacle cannot be occupied. This is reflected by the set of allowed states S':

$$S' = \{(a, e_{free}) \mid a \in A'\} \cup \{(\varnothing, e_{obstacle})\} \tag{5.10}$$

The definition of available updates in this model requires an additional distinction between update types for readability. The model supports three types of updates, which are subsets of U': entering the cell (U'_{enter}), leaving the cell (U_{leave}), and staying in the cell (U'_{stay}). Therefore, the set of updates U' for this example is the following:

$$U' = U'_{enter} \cup U'_{leave} \cup U'_{stay} \tag{5.11}$$

Additionally, each update, aside from operating on the cell state, is linked to the context of a specific agent. Therefore, to satisfy Equation 5.3, each update in each of the mentioned update subsets can be created with the use of the respective "update generator" assigned to its subset: UG'_{enter}, UG'_{leave}, or UG'_{stay}. To create any single update that can be used in this method, a generator function must be applied to the agent a that causes this update (usually as an action):

$$\begin{gathered} UG'_{enter} : A' \to U'_{enter} \\ UG'_{enter}(a = (m, i)) = (\varnothing, \varnothing) \to ((m+1, i), \varnothing) \end{gathered} \tag{5.12}$$

$$\begin{gathered} UG'_{leave} : A' \to U'_{leave} \\ UG'_{leave}(a) = (a, \varnothing) \to (\varnothing, \varnothing) \end{gathered} \tag{5.13}$$

$$\begin{gathered} UG'_{stay} : A' \to U'_{stay} \\ UG'_{stay}(a = (m, i)) = (a, \varnothing) \to ((m, i+1), \varnothing) \end{gathered} \tag{5.14}$$

As shown in the definitions of the generators, all update instances require a specific state of the cell they are applied to: U'_{enter} elements can only be applied to the cell with no agent in it, while U'_{leave} and U'_{stay} elements can only be applied to the cell containing the linked agent a. The applicability mentioned for the Equation 5.3 is clearly visible here — an attempt to apply any update from U'_{enter} to a non-empty cell will fail, signifying the occurrence of a conflicting update.

In the model described above, an agent is able to express two types of intentions represented by a plan: moving into one of the adjacent empty cells, or staying idly in the original cell:

$$P' = P'_{migration} \cup P'_{idle} \tag{5.15}$$

Both types of plans can be generated similarly to updates, with the use of the two "plan generators" $PG'_{migration}$ and PG'_{idle}, but apart from the agent creating the plans, the source cell is necessary for both types, and the target cell is necessary for the migration plan. Migration requires the cells to be adjacent. To achieve this

goal, the model uses NGH^{Moore} neighborhood with D^{Moore} directions, limited to the size of the environment determined by the set C':

$$\begin{gathered} PG'_{migration} : C \times C \times A' \to P'_{migration} \\ PG'_{migration}(c_{source}, c_{target}, a) = (c_{source}, c_{target}, UG'_{enter}(a), UG'_{leave}(a), UG'_{stay}(a)) \\ \text{where } \exists_{d \in D^{Moore}} NGH^{Moore}(c_{source}, d) = c_{target} \end{gathered} \tag{5.16}$$

$$\begin{gathered} PG'_{idle} : C \times A' \to P'_{idle} \\ PG'_{idle}(c, a) = (c, c, UG'_{stay}(a), \varnothing, \varnothing) \end{gathered} \tag{5.17}$$

Considering the possibility of an agent $a = (m_a, i_a)$ intending to migrate from cell $c_{x1,y1}$ to cell $c_{x2,y2}$, the plan p_m representing this intention would be described as follows:

$$\begin{aligned} p_m &= PG'_{migration}(c_{x1,y1}, c_{x2,y2}, a) = \\ &= (c_{x1,y1}, c_{x2,y2}, UG'_{enter}(a), UG'_{leave}(a), UG'_{stay}(a)) = \\ &= (c_{x1,y1}, c_{x2,y2}, (\varnothing, \varnothing) \to ((m_a + 1, i_a), \varnothing), (a, \varnothing) \to (\varnothing, \varnothing), (a, \varnothing) \to ((m_a, i_a + 1), \varnothing)) \end{aligned} \tag{5.18}$$

Similarly, the second possibility of the same agent $a = (m_a, i_a)$ intending to remain in the cell $c_{x1,y1}$ can be expressed as a plan p_i:

$$\begin{aligned} p_i &= PG'_{idle}(c_{x1,y1}, a) = \\ &= (c_{x1,y1}, c_{x1,y1}, UG'_{stay}(a), \varnothing, \varnothing) = \\ &= (c_{x1,y1}, c_{x1,y1}, (a, \varnothing) \to ((m_a, i_a + 1), \varnothing), \varnothing, \varnothing) \end{aligned} \tag{5.19}$$

As can be inferred from the descriptive names of UG functions and the structure of a plan described in the previous section, the plans represent the following detailed intentions of agents:

– If the *action* of plan p_m is accepted, the agent a will "enter" the *target* cell $c_{x2,y2}$, and as a *consequence* it will "leave" the *source* cell $c_{x1,y1}$. If the *action* is rejected, the agent a will not "enter" the *target* cell $c_{x2,y2}$, and as an *alternative* it will "stay" in the *source* cell $c_{x1,y1}$.

– Similarly, if the *action* of plan p_i is accepted, the agent a will "stay" in the *target* cell $c_{x1,y1}$, and no *consequence* is provided. If the *action* is rejected, nothing will happen, as no *alternative* is given. However, due to the model constraints, it is not possible for the *action* to be rejected, as no other legally created *plan* can create a conflict by being successfully applied to the cell $c_{x1,y1}$: no other agent can try to "stay" in this cell, as it is occupied only by agent a, and no agent can "enter" this cell, as it is not empty (i.e. all plans from U'_{enter} are defined for empty cells only).

The process of accepting and rejecting plans, the application of updates, and the further exploration of this exemplary model will be presented in Section 5.2.2, after the algorithm using this model structure is explained.

5.2. Proposed model state update algorithm

Based on the definition of the model introduced in Section 5.1, this section presents the update and synchronization algorithm used by the proposed method. Examples of this algorithm will be shown later in this section, based on the exemplary model described in the previous section.

5.2.1. Subdivision of simulation iteration step

The algorithm maintains the previous principle of simulation execution, where the entire simulation is run in small time-steps representing some short duration of time in the simulated environment. Each time-step, usually called *iteration*, can be simplified to a single portion of the model state update, followed by the synchronization of any data that migrated between workers.

This algorithm significantly changes the steps (*phases*) that constitute the iteration, proposing the following order of operations:

- Planning phase.
- Actions phase.
- Consequences and signal propagation phase.
- Ghost cells update phase.

Each phase is followed by a synchronization. The phases are described in detail below.

Planning phase. This step consists of traversing the environment, collecting all plans from all cells. This includes the decisions of the agents or changes caused by the environment, depending on the model. Contrary to the original approach, no changes are made to the state of the simulation at this point.

The phase is ended by sending each plan to the worker that is the owner of the region containing the target cell of that plan. In most cases, the majority of plans will target another cell within the region owned by the same worker, which does not require additional communication. However, for plans operating across the border between workers, this step will be necessary to ensure that the plan is available to the worker that has the authority over the cell that is intended to be changed.

Actions phase. In this phase, all plans targeting any given cell are available to the worker that owns this cell. Plans are rearranged in random order (”shuffled”) and processed in this new order. Each plan is then validated against the current state of the target cell. If the model allows the action to be applied to the cell in this state (i.e. the update function representing the action of the plan is defined for the current state of the cell), it is accepted and immediately applied, replacing the state of the cell with the result. Otherwise, the action is rejected. For each applied action, the consequence of its plan is selected for the application, while for each rejected action the alternative of its plan is selected.

The phase is ended by sending each plan to the worker that is the owner of the region containing the source cell of the plan. This operation is similar to the ending of the previous phase.

Consequences and signal propagation phase. In this phase, all plans are back in their source cells, either applied to the target cells or rejected. All consequences (for accepted actions) or alternatives (for rejected actions) are applied, replacing the state of the cell with the result. It is a crucial observation that the important task for the model creator is designing a separation that does not allow the application of actions to invalidate the future consequences or alternatives, as they are not validated at this point.

Additionally, as the state of cell contents is not subject to any future changes in this iteration, it is possible to perform a ”cleanup” of the cells — any change that is limited to a single cell and can be performed only after all agents have resolved their decisions. This additional cleanup might not be necessary for majority of the models, as the changes that would fit into this step can usually be part of a plan. It is also a reasonable stage for inspecting the changes that occurred in the model state.

Finally, after all changes to the contents of cells are finished, the signal propagation can be performed.

The phase is ended by sending any signal updates that should influence the remote cells to the workers that are the owners of the regions containing those cells.

Ghost cells update phase. In this phase, the received signal updates are applied to the cells they are intended for. In most cases, this amounts to adding all values that share the same cell and direction pair, as they represent the change in the same signal.

As no other changes are necessary, the phase ends with sending the final state of border cells to the workers that contain cells connected to those border cells, in order to update the read-only ghost cells in those workers.

At this point, the state of the simulation is ready for the beginning of the next iteration.

The presented algorithm in conjunction with the division of the environment introduces two potential differences between non-distributed execution and execution with some degree of distribution:

– accessing remote data,

– synchronization after each phase.

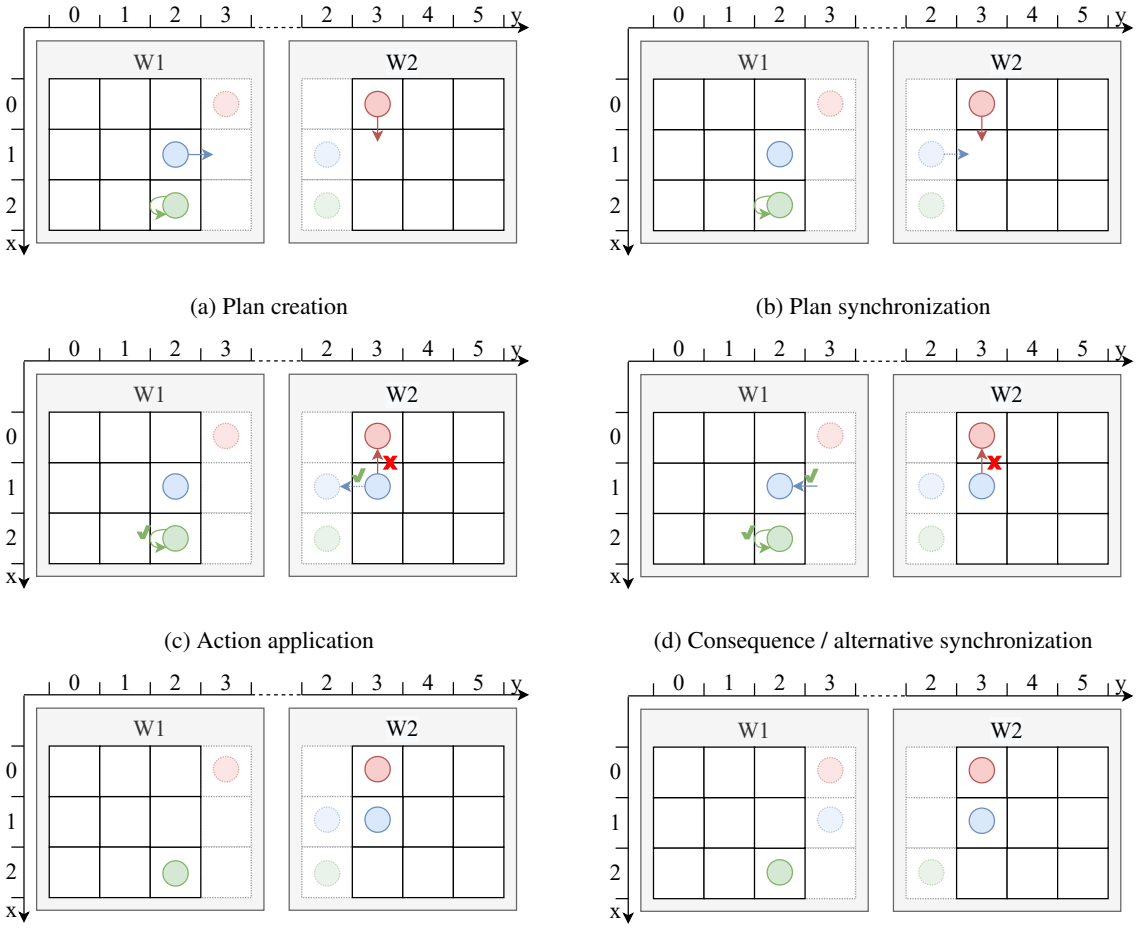
The first issue is directly addressed by the signal propagation method and the use of ghost cells. Regardless of the presence of distribution, all agents should rely on the signal in their decision-making algorithms. In cases where direct observation is necessary, e.g. checking the state of the adjacent cells, all such remote cells should be available in the ghost region of the worker. As decisions are made based on the state after the previous iteration, the update at the end of each iteration in the ghost cells update phase ensures the state is up-to-date before the agent is allowed to make any decisions. Therefore, the environment observed by an agent is identical regardless of the degree distribution.

The second issue should be analyzed separately, as they differ between phases. In case of exchanging consequences/alternatives, they are required to be always applicable and independent, therefore, the order of their application does not matter for the resulting state of a given cell. As with exchanging signal updates and new states of ghost cells, it is only necessary for all of them to arrive at the proper worker before the next phase, which is ensured by the synchronization. Finally, the exchange of plans poses some challenges, as they can be mutually exclusive, and the order of their arrival can dictate which of the conflicting ones will be selected to be applied. The rearrangement of the plans in a random order after all are available in the proper worker ensures that there will be no difference in their treatment, regardless of the distribution degree or its absence.

The conclusion emerging from this analysis is that this method does not allow for any differences in the execution of the same model with the same parameters with different degrees of distribution, except the stochastic ones. Therefore, this method satisfies the first, general definition of correctness proposed in Chapter 6. However, in the following chapter it will also be tested against the second, specific definition of correctness, with a test model.

5.2.2. Application of the algorithm to the exemplary model

Further exploring the exemplary model presented in Section 5.1.2, an example of the state of environment in a simulation using this model can be visually represented as in Figure 5.1. The simulation is executed with two workers $W1$ and $W2$, each responsible for a 3×3 portion of the original 3×6 grid environment. The semi-transparent cells outlined in the dots are the ghost cells, representing the last known state of the cells across the border, owned by the other worker. Attention should be paid to the auxiliary axes, as the y axis is not continuous - numbers 2 and 3 are repeated for both workers, because ghost cells replicate column 3 for $W1$ and column 2 for $W2$.



(a) Plan creation

(b) Plan synchronization

(c) Action application

(d) Consequence / alternative synchronization

(e) Consequence / alternative application

(f) Ghost cells synchronization

Figure 5.1: Decision and synchronization phases in presented approach.

The empty white squares represent empty cells, i.e. cells with state of $(\varnothing, \varnothing)$. The squares containing circles represent the cells occupied by agents. For better visual clarity, the agents will be referred to as a_R, a_B, and a_G (for the red, blue, and green circles, respectively). Therefore, in Figure 5.1a, cells $c_{0,3}$, $c_{1,2}$, and $c_{2,2}$ have states of $(a_R, \varnothing)$, $(a_B, \varnothing)$, and $(a_G, \varnothing)$, respectively. The internal states of agents, i.e. the migration and idle counters, are omitted in the visualization.

In addition to the presentation of the initial state of the simulation, Figure 5.1a also shows the plans created in the planning phase. The agents a_R and a_B decided to migrate to the same cell, while a_G decided to remain idle. The resulting plans are the following:

- $p_R = PG'_{migration}(c_{0,3}, c_{1,3}, a_R)$, represented by the red arrow,
- $p_B = PG'_{migration}(c_{1,2}, c_{1,3}, a_B)$, represented by the blue arrow,
- $p_G = PG'_{idle}(c_{1,2}, c_{1,3}, a_B)$, represented by the green arrow.

One of the plans — p_B — targets a cell that lies in a region owned by the worker other than the one that owns the source cell. Therefore, it must be sent from W_1 to W_2 at the end of this phase, as shown in Figure 5.1b.

After the communication, the second phase commences, in which the plans are randomly ordered and processed. In this example, p_B has been validated and applied before p_R. Therefore, as shown in Figure 5.1c, a_B entered the cell $c_{1,3}$ (and increased its migration counter) through the action of p_B, making the action of p_R inapplicable in the process, as it is not defined for non-empty cells. At the same time, p_G was validated and applied —

increasing the idle counter of a_G — without any interaction with other plans, as it targeted a different cell. This resolution leads to the selection of the consequences of p_B and p_G, and the alternative of p_R as the updates to be performed in the next step.

As the plan p_B has a source cell that is located in a region owned by W_1, it has to be sent back to that worker at the end of this phase, as shown in Figure 5.1d.

In the third phase, the selected updates are applied:

– alternative of p_R increases the idle counter of a_R,

– consequence of p_B removes a_B from the original cell $c_{1,2}$,

– consequence of p_G is empty, so no additional update is performed.

The state after those changes is shown in Figure 5.1e. The signal propagation is omitted in this example, as it would make the images much less readable, while the signal itself is a component used in the decision-making process, which is not presented here. As a result, the signal synchronization step is also omitted.

The final step, shown in Figure 5.1f, is sending the updates to the ghost regions, removing outdated and no longer correct information about the old location of a_B from the ghost cells of $W2$ and adding information about the new position of a_B in the ghost cells of $W1$. The states of a_R and a_G are updated in the ghost cells as well, as both agents increase their idle counters during this iteration.

This concludes the iteration of the exemplary model.

5.3. Distribution and efficiency of the proposed method

The method described in this chapter was created with scalability in mind, which is why the synchronization and lack of access to parts of the environment are accounted for in the algorithm presented in the previous section. The method assumes that any plan and/or update can operate across the boundary between distributed parts of the environment, with no limitations. However, the method itself does not guarantee good scalability. It is necessary to understand the influence of the approach to the distribution on the efficiency of the resulting simulation system. The presented method supports the good practices outlined below.

5.3.1. Efficient inter-worker communication architecture

It is important to pay attention to the efficiency of communication and resource utilization. First, it is important to ensure the limited, constant number of connections between workers. Otherwise, increasing communication overhead might outweigh the benefits from increasing the computing power. Therefore, it is advised to limit the communication range of a worker, allowing only to exchange messages with adjacent workers (i.e. workers that contain at least one cell adjacent to some cells of a given worker). As was observed in [16], it is necessary to ensure *scale invariance* to maintain scalability in large parallel systems. Scale invariance is defined as having a constant number of processes with which a given process has to communicate, regardless of the total number of processes in the whole system. This is exactly the quality that is being pursued in the limitation presented here. Second, the method supports any shape of the regions assigned to the workers and the complexity of the connections between the cells. However, the total size of the boundaries between the regions is a good approximation of the amount of communication that occurs in each synchronization step. Additionally, due to frequent synchronization, any imbalance in the amount of workload among workers will result in some workers finishing their allotment of work earlier and idly waiting for their neighbors. The practice shows that all of those aspects can be taken care of by selecting an environment partitioning method that prioritizes equal, regularly shaped, connected regions. An example

of such partitioning for a 2D grid is a simple division into identical rectangles, which yields regions of equal size, with a maximum of eight neighboring workers, and minimized straight border lines. These characteristics should, in turn, provide a balanced workload, a constant number of communications regardless of the number of workers, and a total volume of transferred data proportional to the square root of the size of the grid. Other, more complex methods of environment partitioning might be desirable, depending on the model, but the mentioned limitations should be taken into consideration.

Another aspect which is not mentioned in the algorithm is the internal logic of the model, especially the details regarding the decision making. The proposed method does not specify any constraints for the data available to the decision-making entity (e.g. an agent), but the ghost cells approach is mentioned. The consequence of employing this mechanism is that the only information regarding the state of the simulation is the state of the local cells and the cells mirrored by ghost cells. Naturally, the size of the ghost region is equal to the amount of data that will have to be synchronized to keep the information valid. In an extreme case, if each worker keeps the whole environment as its ghost region, it is necessary to obtain the updated state of those cells from each worker, violating the good practices mentioned before and severely impacting the scalability of the system. Further analysis suggests that the size of this region should be kept proportional to the size of the border, as it ensures that the benefits from the efficient partitioning of the environment are not overshadowed by the ghost cells synchronization. An example of a reasonable choice of the size of the ghost region for the 2D environment partitioned into rectangles is to keep a one-cell-wide outline of ghost cells around the local region, which is also used in the exemplary model shown in Section 5.2.2 (see Figure 5.1).

The limitation of the size of the ghost region might be problematic for models that require scanning some area around the agent to obtain the data necessary to make a decision. As an example, if an agent should be able to scan an area within a distance of 5 cells, the said agent occupying a cell at the border of a region with one-cell-wide ghost region might be able to see only one cell across the border, making it unable to perceive the presence of other agents in that direction. The solution to this problem is hinted at in the third step of the algorithm — the use of the signal propagation method [62, 75]. The ability to directly observe the state of any cell should be substituted with the ability of information about interesting phenomena to spread in the environment, making the decision process dependent only on the current cell and adjacent cells, if needed. This approach is in line with all the other good practices presented here and should be considered when designing a model.

5.3.2. Optimization of synchronization between phases

Each of the phases described in Section 5.2 is finalized by synchronization with other workers. The algorithm explicitly calls for sending each of the plans, updates, signal updates, or cell states to a target worker. Taking advantage of the limited number of workers, it is advisable to batch the data in each of these synchronization instances. It is entirely possible for each worker to send a single message containing the entirety of the necessary data for a given phase. This allows for improved further optimization depending on the chosen communication method, e.g. compression of the payload.

Another interesting observation is that the lack of centralized tracking of simulation progress makes the workers aware solely of the state of their neighbors. Therefore, it is only necessary to ensure that the local synchronization was successfully completed before proceeding to the next iteration. In case of simulations with fluctuations in the computing cost of an iteration for a given worker (e.g. the distribution of agents changing over time with successive migrations), it is possible for some less loaded workers to safely execute further iterations that might offset future increases in load. At the same time, despite the desynchronization in iteration number, there is no risk of losing consistency of the simulation state, as the worker is allowed to proceed only after receiving up-to-date information regarding its ghost region. This desynchronization is limited by the distance between workers, as shown in this

example with a single "strip" of workers connected with:

- $W1$ is processing step N of the simulation,
- $W2$ is finished with step N of the simulation, has already sent the messages after this step to $W1$ and $W3$, received the message from $W3$ and is waiting for the response from $W1$,
- $W3$ is finished with step $N + 1$ of the simulation, has already sent the messages after this step to $W2$ and $W4$, received the message from $W4$ and is waiting for the response from $W2$,
- $W4$ is finished with step $N + 2$ of the simulation, has already sent the messages after this step to $W3$ and $W5$, received the message from $W5$ and is waiting for the response from $W3$,
- the pattern repeats for all subsequent workers.

The same observation also suggests that a momentary slowdown of a single worker will not halt computations on all resources. Instead, the effects will "ripple" across the workers, with the possibility of further workers not being impacted at all by this occurrence.

5.3.3. Parallelization of the update application

Each phase of the algorithm described in Section 5.2 consists of some operations performed on the subset of cells assigned to a given worker. In the planning phase, for each cell, the states of some other cells are usually inspected, but they are not allowed to be changed at this point. In all of the remaining phases, the changes in any given cell are independent of each other and do not depend on the states of neighboring cells.

These observations lead to the conclusion that the unit of work related to a single cell in each of the phases is fully independent of other units of work. As a result, they can be executed in any order, or even in parallel. This raises the opportunity for the efficient use of multi-core workers executing each phase of the algorithm in parallel, with synchronization occurring only at the end of each phase, when the communication to other workers is necessary.

5.3.4. Enforcing model constraints in probabilistic method

All plans are randomly rearranged prior to validation and application, in order to eliminate any bias that might stem from processing them in the order of their creation, which would likely prioritize locally created plans. As a result, the designer of the model loses some control over the behaviors that can be observed within the simulation.

If it is unacceptable for the validity of the model, the presented method is able to preserve all its qualities while offering the designer to take the complete responsibility for the ordering of the plans. The "shuffling" can be safely replaced by sorting the plans, provided that the creator of the model can define a strict total order for the set of plans P. Alternatively, non-strict total order might be acceptable with an additional random reordering for each sequence of equal plans. In both cases, the bias is eliminated by enforcing ordering rules that are independent on the order of creation or receiving of any given plan (unless explicitly accounted for in the total order provided by the designer of the model).

If random ordering does not pose any problem with regard to the model constraints, it is still crucial to fully understand the possible impact of any order of updates on the model state. During the work towards creation of the algorithm presented in the previous section, an interesting (and undesired) phenomenon occurred within one of the considered models. The model was a simplification of the predator-prey scenario, similar to one of the test models presented in Appendix A. The model allowed for two agent types: *wolfs* (acting as predators) and *rabbits* (acting as prey). The rabbits tried to avoid the wolves, while the wolves pursued the rabbits. In an occurrence of

the meeting of two agents of different types, the wolf consumed the rabbit, increasing its energy. The initial, naïve adaptation of this model to the presented method, the plans could be informally described as follows:

– Wolf intends to enter the neighboring cell that must not contain another wolf; in case of success, the wolf will be removed from the original cell; in case of failure, the wolf will stay in the original cell.

– Rabbit intends to enter the neighboring cell that must not contain another rabbit; in case of success, the rabbit will be removed from the original cell; in case of failure, the rabbit will stay in the original cell.

Therefore, the only collisions were those caused by two agents of the same type entering the same cell. Additionally, applications of updates that introduced an agent into a cell were resolved with an immediate check for a wolf meeting a rabbit, which prompted the consumption scenario.

Due to this manner of update resolution, an unexpected behavior was observed. If a wolf was adjacent to a rabbit, it was very likely that it would try to move into the rabbit's cell, while the rabbit would try to move into another empty cell. The resulting updates were the following:

– The rabbit placed itself in an empty cell, due to the action of its plan.

– In the same phase, the wolf placed itself in the rabbit's original cell, still containing the rabbit, and consumed it.

– In the next stage, due to the consequences of the plans, the wolf is removed from the original cell, as is the rabbit (although the second update fails due to the lack of the rabbit in the original cell).

If the problem with removing the rabbit from its original cell is ignored, the simulation continues its progress. Simple tracking of the locations of agents has shown that the rabbit was able to repeatedly run from the wolf. However, a more thorough inspection has shown that the energy of the wolf is being increased, as if it was successfully catching the rabbit. As a result, the bunny was forced into the superposition of being both eaten and alive at the same time, which prompted the labeling of this phenomenon as "Schrödinger's rabbit" (see [60] for context).

The solution of the problem arising from this behavior is to postpone any interaction between agents to the "cleanup" step, mentioned in the third phase of the algorithm. Using this principle, all consequences and alternatives are allowed to be applied without interruption, and all necessary interactions can be resolved afterwards, once the state of the cell does not expect any other updates until the next iteration.

5.4. Summary of the features of proposed method

The new method solves the most significant problems with the methods discussed in Chapter 2. The decision whether the two plans are mutually exclusive or not lies completely in the hands of the model creator, as they must provide all building blocks of the simulation: the creation of plans for a given cell, the assessment whether a given plan is applicable to a given cell state, and the application of a given change to the state of a given cell.

The abstraction introduced by the plan provides enough flexibility to allow actions other than agent migration. The same principles can be used to create a copy (e.g. offspring) of an actor in another cell without vacating the original cell, to allow an agent to change the state of the environment in another cell without entering it, etc.

Random rearrangement of plans targeting the same cell serves as a guarantee that no decision is implicitly prioritized. Therefore, the phenomena observed in the *location-* and *direction-ordered* strategies analyzed in Chapter 2 will not occur, and no agent will have the opportunity to make a new decision if the old one fails, as in the *trial-and-error* strategy. On the other hand, if the model would benefit from such retries — which can resemble a sort of "hivemind" or "collaboration" behavior, in which agents seem to be aware of the decisions of others and

act accordingly, if their priority is lower — it should be included in the model explicitly. Additionally, if the model is not suitable for random order of plans, the rearrangement step can be easily replaced by sorting, in which the priority will be explicitly visible in the model.

At the same time, the method still maintains the locality of each step, i.e. after synchronization, all operations can be performed by each worker independently. If the guidelines outlined above are followed, the volume of data sent is limited to sizes similar to the ones in the original approach described in Section 2.5. Therefore, the preliminary analysis suggests that the good scalability of the base method is preserved.

6 Verification scheme and results

In this chapter, which builds upon the conclusions from Chapter 3, a further research is presented, which has led to the establishment of the verification process for the second, model-specific correctness of the distribution method for ABMS. Additionally, the results of the verification of the test cases are presented and discussed.

6.1. Method of verification

Taking important notes from the observations presented in Chapter 3, the new approach to verification of the distribution method for a given model was created. To avoid the necessity of an expert in the field related to the model, performing an analysis of the changes to the model and assessment whether they are acceptable, the method should instead ensure that the differences can be described quantitatively. The proposition of such a verification method was presented in a publication [52] that is part of the research presented in this thesis.

The first step is already required from the creators of the model — careful analysis of the modeled scenario and selecting crucial metrics that describe the most important observed behaviors. This step is especially important, as only those metrics can be later analyzed in terms of correctness. As the examples described in Chapter 2 show, the influence can be subtle and not clearly noticeable, but if the proper behaviors are monitored and collected in the form of metrics, they should become visible in further steps of verification.

As a next step, a process similar to the one described in Chapter 3 is performed: a simulation is executed multiple times, collecting metrics from many repetitions with different degrees of distribution, maintaining the same parameter configuration. In this case, it is not necessary to create different configuration variants for comparison purposes, however, it is advisable to do so to test multiple different behaviors that can occur in the simulation, depending on the parameters.

Finally, the obtained results are statistically analyzed. The importance of this step lies in the fact that a large number of models utilize randomness in some parts of their decision-making algorithms. Therefore, a simple comparison of metrics from two executions is bound to show some differences, and statistical analysis is needed instead. For the purposes of this research, the following approach was used:

- The metric to be analyzed is selected.
- The recorded values for that metric are grouped: for each degree of distribution, an array of samples is created, where a single sample is a set of values obtained for a given iteration in all repetitions. As an example, if the experiment was conducted for 4 degrees of distribution, with iteration number of 1000, with 30 repetitions, the grouping would result in 4 sets of arrays, each containing 1000 sets of 30 values.
- For each iteration, the sets of values obtained for each degree of distribution were compared using the Kruskal-Wallis test [39]. The null hypothesis states that the compared samples come from the same distribution and should display no statistically significant difference. The key result of this test is *p-value*, and the commonly accepted interpretation of this value is that the p-value below 0.05 suggests rejection of the null

hypothesis (samples are significantly different), while the value above 0.05 suggests accepting of the null hypothesis (samples are not significantly different).

– The p-values for all iterations are averaged and all values less than 0.05 are counted.

– The results are presented in the form of a table summarizing both the average p-value and the percentage of p-values below 0.05.

It is worth noting that the presence of some p-values below 0.05 is not unexpected, especially for smaller numbers of repetitions, even if the collected metrics are from the same distribution. The number of such values will be increased if the model utilizes the stochastic processes more extensively. Therefore, the main hint that differences have been introduced due to the distribution comes from low average p-values and a significant share of values below 0.05, not their presence alone.

As a further guideline for the interpretation of the results, it is useful to consider the following example: applying the same analysis to random samples from the same normal distribution with a mean of 0 and standard deviation of 1 in place of actual metrics, the results tend to be the following: average p-value of 0.5, 5% of p-values below 0.05, results ranging from 0 to 1 — i.e. p-values are uniformly random, as there is no preference to any of the samples dominating the other. At the same time, sampling a different distribution tends to return much lower p-values, with the share of values below 0.05 easily exceeding 10%–15% and an average of 0.3 or less. The magnification of this difference corresponds to the scale of differences between the distributions, where significantly different distributions can even cause the majority ($> 95\%$) of the p-values to be lower than the selected threshold of 0.05.

6.2. Verification of the proposed simulation method

In Section 5.2 an analysis of the characteristics of the proposed method was presented, demonstrating that the method satisfies the definition of correctness of the method in general. However, the verification method presented in the previous section has been applied to the method and a test model to showcase the verification process. The chosen test case was the model that presented the most difficulty in adaptation to the core method presented in Section 2.5, i.e. rabbits and lettuce scenario described in Section A.1 (see Appendix A). The remaining two models were omitted due to relatively easy adaptation to the previous method. The results were originally presented in a publication [52] that is part of the research presented in this thesis.

Due to the way the new method was designed, the conflict resolution violating the rules set by the model can be removed. All decisions can be expressed as the following plans:

– Rabbit reproduces — target: an adjacent cell without a rabbit agent; action: place a new rabbit agent; consequence: reduce the energy of the rabbit agent according to the reproduction costs; alternative: reduce the energy of the rabbit agent according to the living costs.

– Rabbit dies — target: same as source; action: remove the rabbit agent; consequence: none, alternative: none.

– Rabbit moves — target: an adjacent cell without a rabbit agent; action: place a copy of the moving rabbit agent with energy reduced according to the living costs; consequence: remove the rabbit agent; alternative: reduce the energy of the rabbit agent according to the living costs.

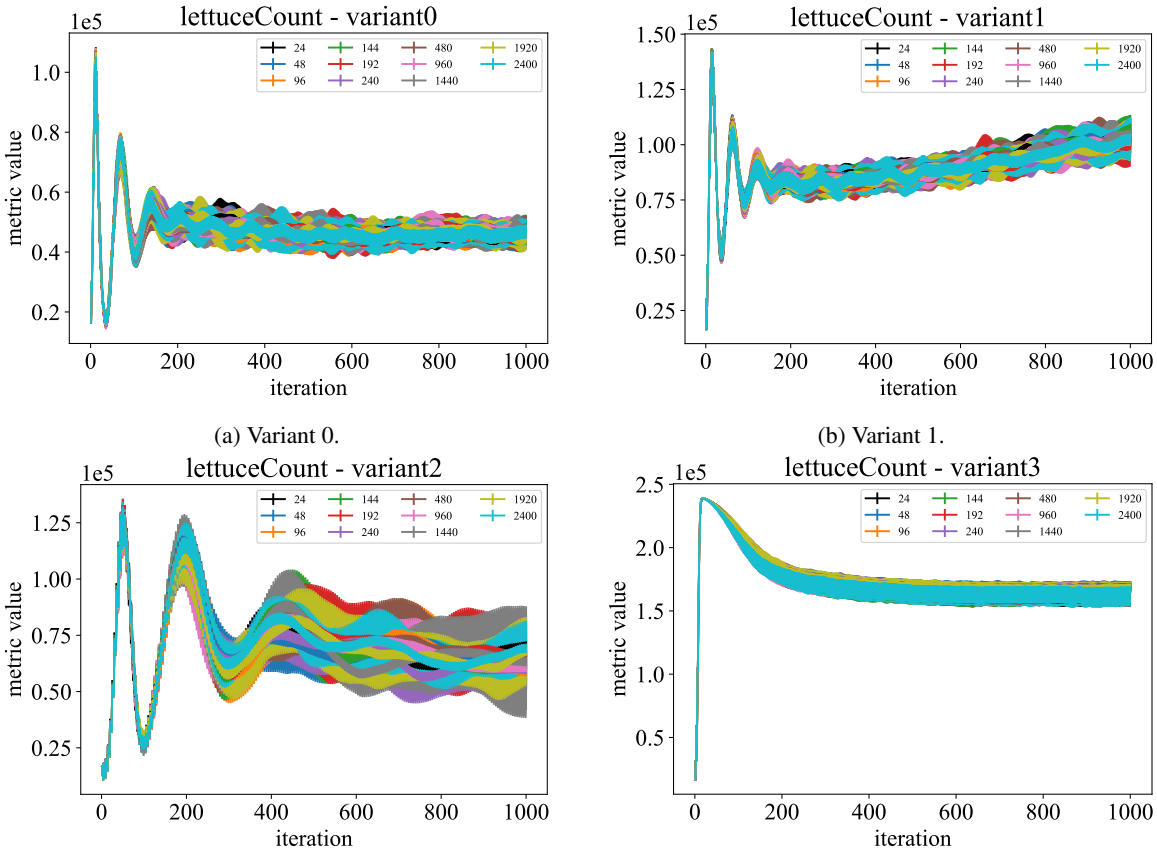
– Rabbit idle — target: same as source, action: reduce the energy of the rabbit agent according to the living costs; consequence: none; alternative: none.

– Lettuce reproduces — target: adjacent cell without a lettuce agent; action: place a new lettuce agent; consequence: none; alternative: none.

Forgoing conflict resolution eliminates the necessity of handling two agents of the same type entering the same cell. The new scheme disallows two lettuce agents to reproduce into the same cell (one will do nothing instead) and two rabbit agents either moving or reproducing into the same cell (one will be idle instead). Lettuce and rabbit can enter the same cell, but in the finalization step, the cell will be updated by the rabbit agent consuming the lettuce agent.

6.2.1. Non-deterministic simulation approach

The rabbits and lettuce model has been tested using the proposed method of verification. The same four configuration variants listed in Table 3.3 (in Appendix A) have been used in this experiment. Each configuration was executed 10 times with an iteration limit of 1000. The experiment was performed on the Prometheus supercomputer (detailed information in Chapter 7, where this type of data is relevant), an environment offering nodes with 24 computational cores. The simulation was distributed using 1, 2, 4, 6, 8, 10, 20, 40, 60, 80, and 100 nodes (i.e. 24, 48, 96, 144, 192, 240, 480, 960, 1440, 1920, and 2400 cores), each degree of distribution producing a separate set of samples for each iteration.



(a) Variant 0.

(b) Variant 1.

(c) Variant 2.

(d) Variant 3.

Figure 6.1: Rabbits and lettuce — timelines of lettuce count. Source: [52]

The results were visualized using timelines, as in the first approach to verification. The results are presented in Figure 6.1. Unfortunately, the multitude of series on the same plot significantly reduces the visibility of this type of

visualization, but no obvious discrepancies were observed. The most suspicious results were obtained from variant 2 (Figure 6.1c), where many oscillating series overlap each other in a wide strip of standard deviation values.

Table 6.1: Summary of average p-values and percentage of p-values below threshold for rabbits experiment

	variant 0		variant 1		variant 2		variant 3	
metric	$\bar{p}$	$p < 0.05$	$\bar{p}$	$p < 0.05$	$\bar{p}$	$p < 0.05$	$\bar{p}$	$p < 0.05$
rabbit count	0.47	4.2%	0.42	3.9%	0.48	1.8%	0.44	1.4%
rabbit reproductions count	0.47	2.9%	0.44	4.2%	0.48	2.7%	0.45	1.1%
rabbit total energy	0.47	3.9%	0.42	4.1%	0.48	2.5%	0.44	1.4%
rabbit deaths	0.48	3.0%	0.44	4.7%	0.48	2.7%	0.45	2.7%
rabbit total lifespan	0.48	3.3%	0.44	3.9%	0.48	3.0%	0.46	3.2%
lettuce count	0.55	2.4%	0.44	5.7%	0.52	0.1%	0.47	1.8%
consumed lettuce count	0.47	2.8%	0.42	4.5%	0.48	3.0%	0.44	1.6%
lettuce total lifespan	0.50	3.2%	0.45	7.3%	0.47	5.2%	0.53	1.5%

Applying the proposed method of verification to the collected metrics yielded the results presented in Table 6.1. All average p-values are close to the 0.5 observed for samples of the same distribution, ranging from 0.42 to 0.55. The count of low p-values that could hint at the significant difference between samples is well below 5% of all the returned values for most tests, except three instances of values 5.2%, 5.7% and 7.3%. None of those values is paired with an exceptionally low average p-value, and none is exceptionally high itself. The conclusion of the analysis is that no metric in any of the executions is statistically changed by the distribution.

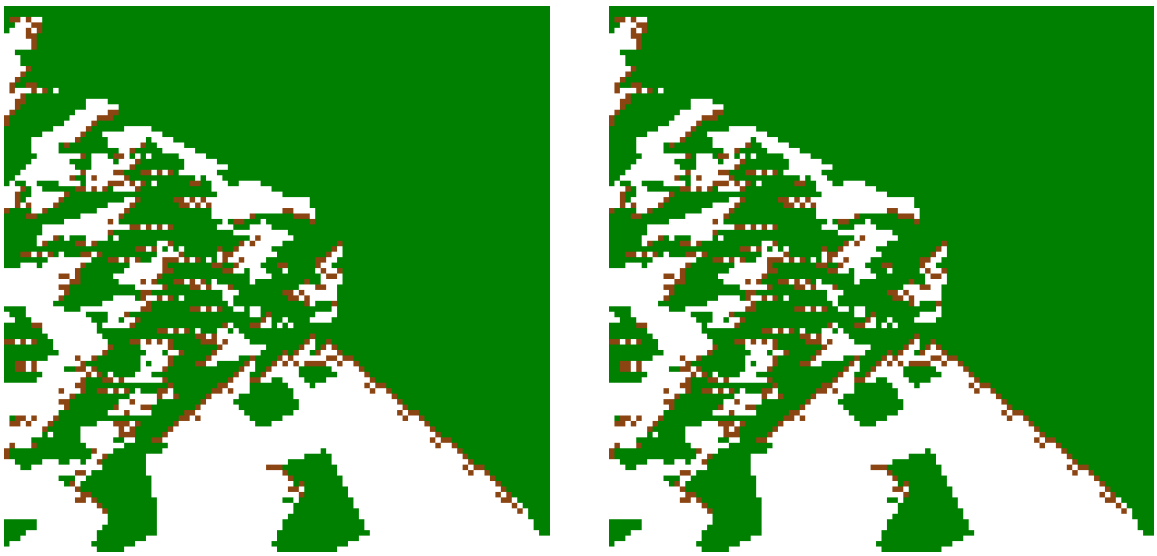
6.2.2. Deterministic simulation approach

As an additional, more fine-grained method of verification, the experiment was repeated removing any non-determinism from the model or distribution method. In each instance of randomly choosing a neighbor cell, a total order was defined for directions. The random rearrangement of plans was exchanged for sorting. It should be noted that this process fundamentally changes the modeled scenario and invalidates all possible conclusions that may come from the analysis of the metrics. Therefore, this approach is not suitable for the analysis of any model, unless the model is deterministic itself, in which case different approaches to the distribution can be used (please refer to Chapter 2). The purpose of this alteration of the model is the further analysis of the method itself. As the method does not affect the results of the random choices, the method remains unchanged, but due to the lack of randomness, the results should now be identical. Therefore, if the method has any influence on the state and progress of the simulation, it will be visible in this experiment. The experiment was otherwise very similar to the previous one, but the executions of the same configuration were not repeated, as collecting a statistical sample is not necessary.

The results show no difference between the same variant executed with different degrees of distribution. The visualization of the state of the simulation was identical as well and a sample is provided in Figure 6.2, where both images represent the state in the same iteration. Figure 6.2a shows the state in an execution on a single core, while Figure 6.2b shows the state in the same iteration on 100 cores, with a grid divided into 10×10 equal squares. The full visualization is available online[1].

The conclusion from this experiment reinforces the one from the previous one — distribution of the simulation of the test model using the method proposed in Chapter 5 does not influence the observed behaviors or simulation results.

[1] `https://youtu.be/9W-zmyQo-K8`



(a) Execution on 1 core. (b) Execution on 100 cores.

Figure 6.2: Deterministic rabbits and lettuce — visualisation of the same iteration.

7 Scalability of the new method

This chapter shows the scalability of the proposed method. The results were originally presented in [52], which is a part of the research shown in this thesis. The scalability is assessed with the use of popular and widely accepted methods of scalability measurement: speedup measured in strong and weak scalability experiments, and Karp-Flatt metric.

7.1. Scalability testing environment

Scalability tests were performed on the Prometheus supercomputer located in the AGH Cyfronet computing center in Krakow, Poland. According to the TOP500 list compiled for November 2021[1], it is the 440th fastest supercomputer. Prometheus is a peta-scale (2.35 PFlops) cluster utilizing HP Apollo 8000 nodes with two Xeon E5-2680v3 12-core 2.5GHz CPUs each. The nodes are connected via InfiniBand FDR network, forming the HPC environment with 55,728 cores in total.

The selected test scenario was the rabbits and lettuce model with the default (variant 0) set of parameters.

7.2. Strong scalability

One of the most popular methods of measuring the scalability of a parallel program is the analysis of the change in execution time with the addition of computational cores, for the constant size of a problem, commonly referred to as *strong scalability*. This approach is the most straightforward, due to the direct applicability of the results to the actual usage of the parallel program. It is worth noting that in this research the calculated speedup does not compare the best sequential program with the parallelized one, but instead compares the executions of the same parallel program on different numbers of cores. The consequence is that the results are likely to show better speedup than would be obtained comparing the parallel program to the sequential one, because designing the sequential version could allow for better optimization for single-core execution and would remove any overhead introduced by using the tools required for communication between processes.

The strong scalability experiment yields measurements that allow calculation of the *speedup* obtained by increasing the number of cores, defined as follows in Equation 7.1:

$$S_N = \frac{T_1}{T_N} \tag{7.1}$$

where S_N is the speedup achieved for N cores, T_1 is the time of execution on 1 core and T_N is the time of execution on N cores. The ideal scenario in which the computations are split evenly between all N cores

[1] https://www.top500.org/system/178534/#ranking

would lead to the execution time being always one Nth of the total time, leading to the ideal linear speedup, as in Equation 7.2:

$$T_N^{ideal} = \frac{T_1}{N} \implies S_N^{ideal} = \frac{T_1}{T_N^{ideal}} = \frac{T_1}{\frac{T_1}{N}} = N \tag{7.2}$$

In practice, a fraction of the problem that does not benefit from parallelization always exists, preventing the ideal scenario from occurring. This observation is described by Amdahl's law [1], which formulates the speedup in terms of the fraction p of the problem that benefits from parallelization, as in Equation 7.3:

$$S_N = \frac{1}{(1-p) + \frac{p}{N}} \tag{7.3}$$

The denominator of this formulation can be divided into two parts: $1 - p$, which represents the fraction of the problem that cannot be parallelized (sequential part), and $\frac{p}{N}$, which represents the parallelized part. The ideal speedup can be derived from this equation by assuming $p = 1$, i.e. no sequential part of the problem exists.

By rearranging this equation, it can be shown that there exists a limit to the speedup (assuming a non-zero sequential part of the problem), which cannot be exceeded by adding more cores, as in Equation 7.4:

$$\lim_{N \to \text{inf}} S_N = \frac{1}{(1-p)} \tag{7.4}$$

In this case, the explanation is that regardless of the speedup achieved in the parallelizable part of the problem, the sequential part will always take the same amount of time, limiting the overall speedup.

The experiment was carried out to measure the performance of the method with the constant size of a problem, which in this case was set as $2.4e7$ cells. This size was selected to obtain execution times for core counts large enough to offset possible noise in the measurements. Each experiment was repeated 5 times to further reduce the influence of noise on the final results. The memory requirements of the problem proved to be problematic — when the size was enough to observe non-negligible execution times for larger core counts, the memory requirements of the problem part for each core for lesser core counts exceeded the limits set in environment. Due to this limitation, the experiment was executed on 10, 20, 40, 60, 80, 100, 150, and 200 nodes, i.e. 240–4800 cores. As a consequence, the speedup calculation had to be adjusted to use the available data in a way that fits the original idea. The concept of ideal speedup was used to calculate the reference execution time, leading to the following calculation of speedup for this experiment:

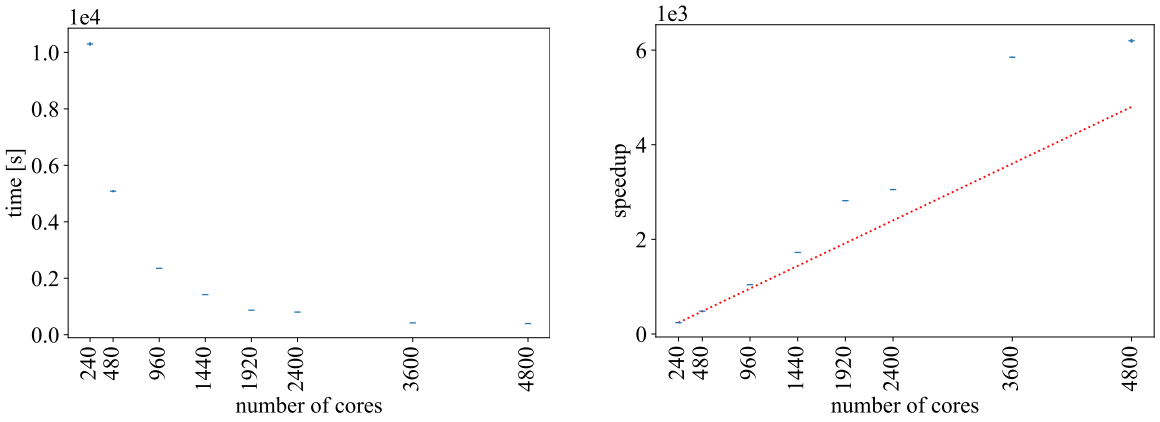
$$S_N = \frac{240 \cdot T_{240}}{T_N} \tag{7.5}$$

Figure 7.1a shows the measured execution times. Each horizontal bar shows the mean time, while the vertical bar shows the standard deviation calculated from all repetitions. The data points roughly follow the hyperbolic shape expected in this type of experiment.

At the same time, Figure 7.1b shows the calculated speedup. The red dotted line marks the ideal speedup. The speedup calculated from the measured execution times suggests a significant superlinear scaling of the solution. The detailed explanations for the presence of superlinear scaling can vary vastly from case to case, but generalized causes can be limited to the following [58]:

1. The actual number of computation cycles was lower in parallelized execution - most prevalent cause for searching algorithms, in which one process can find the piece of data quickly and prevent other processes from performing operations that would otherwise be performed in sequential execution.

2. Sharing the cache memory - due to sharing cache between computing units (cores, processors), one cache miss can lead to replacement that will benefit all other units sharing the same cache, i.e. the replacement will be performed only once.



(a) Execution time.

(b) Speedup.

Figure 7.1: Execution times and calculated speedup for strong scalability. Source: [52]

3. Availability of larger combined cache memory - due to introduction of more computing units (cores, processors) the total size of cache is increased, allowing for less frequent cache replacement.

The tested case does not fit into the first category of programs, as the simulation will be required to execute all steps and all iterations in each case. The reduction in the number of cycles might stem from randomness that could reduce the number of agents that need to perform the planning and action loop, but this possibility is already mitigated by multiple executions of each run. No memory is shared by the workers, which eliminates the second suggested cause. Therefore, the most probable scenario is the variation of the third cause. Additional measurements performed outside the experiment show that the memory available on each of the nodes is not sufficient to contain the data processed in the case of lesser core counts. More precisely, as the system was implemented using the Java Virtual Machine (JVM), multiple additional invocations of Garbage Collector (GC) were recorded for lesser core counts than for larger ones. In this context, the superlinear scaling appears to not be a performance boost obtained due to the characteristics of the proposed solution, but rather an anomaly resulting from the poor memory performance for such problem sizes assigned to each core. In summary, drawing conclusions about the scalability of the solution from this experiment alone might be misleading and requires additional investigation.

7.3. Weak scalability

A very useful observation can be taken from the previous experiment: the proposed solution and its implementation are much better suited for larger problems distributed over a large number of cores, or smaller problems executed on a few cores. This quality of a problem is one of the principles behind a different experiment type — *weak scalability*. This type of scalability measurement is very adequate for simulations in general, as it requires a problem that can have its dataset indefinitely increased. Usually, the increased resource requirements of simulations emerge from increasing the size of the simulated environment or increasing the resolution of the simulation, which perfectly fits the weak scalability experiments. This is especially useful in the context of simulations increasing in size and complexity to the point of exceeding the capabilities of a single machine, which is the focus of this research.

The difference between this type of experiment and strong scalability is that the problem size is increased to match the assigned resources. In other words, the size of the problem part assigned to each core is constant. The

speedup in the case of this type of experiment is calculated as in Equation 7.6:

$$S_N = \frac{N \cdot T_{1,1}}{T_{N,N}} \tag{7.6}$$

where S_N is the speedup achieved for N cores, $T_{1,1}$ is the time of execution on 1 core with a unit size of a problem (chosen for the instance of an experiment) and $T_{N,N}$ is the time of execution on N cores for a problem N times larger than a unit size. Once again, the ideal scenario can be defined: the computations are split evenly between all N cores, and increasing the problem size increases the execution time proportionally. The ideal speedup would be linear, as in Equation 7.7:

$$T_{N,N}^{ideal} = N \cdot \frac{T_{1,1}}{N} \implies S_N^{ideal} = \frac{N \cdot T_{1,1}}{T_{N,N}^{ideal}} = \frac{N \cdot T_{1,1}}{N \cdot \frac{T_{1,1}}{N}} = N \tag{7.7}$$

It is worth noting that in the ideal case the execution time is constant, as the benefits from the parallelization are exactly opposed by increasing size of the problem.

The speedup observed in the weak scalability experiment can be described using Gustafson's law [23]. The calculations begin with the idea of workload W:

$$W = (1 - p) \cdot W + p \cdot W \tag{7.8}$$

where p is a fraction of a problem that can be parallelized. The theoretical workload can be then defined in the context of parallelization, as only the non-sequential part will be subject to the speedup:

$$W_N = (1 - p) \cdot W + N \cdot p \cdot W \tag{7.9}$$

And finally, the theoretical speedup, assuming the same time of execution for the sequential and parallel execution, is given by the Gustafson's law as in Equation 7.10:

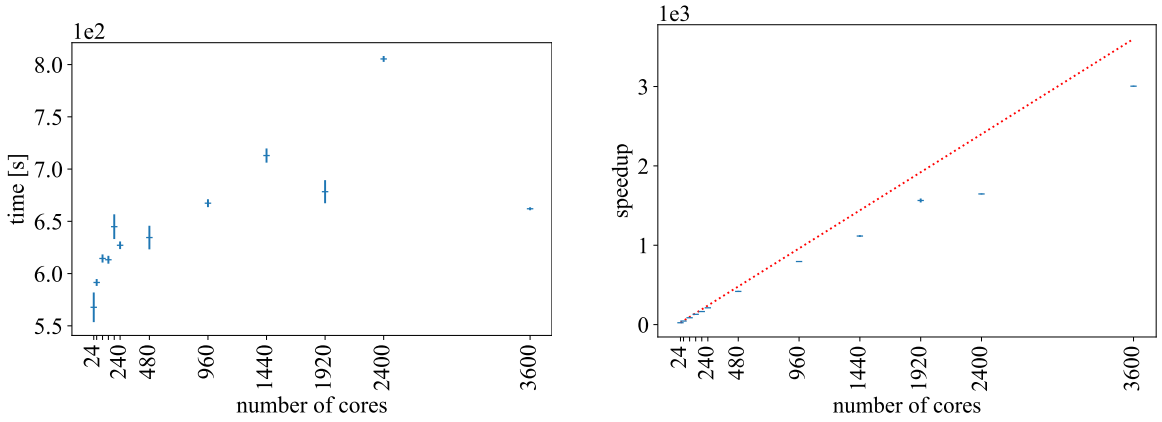
$$S_N = \frac{T \cdot W_n}{T \cdot W} = 1 - p + N \cdot p \tag{7.10}$$

This formula does not define any limits to the speedup that can be achieved. Under the assumption that the sequential part remains the same and the parallelized part of the problem can be increased in size indefinitely, it is always possible to further reduce the relative influence of the sequential part of the problem on the overall speedup. Additionally, the resulting speedup is linear, although the slope is not necessarily 1 — more precisely, the slope of the speedup line is equal to the fraction of the problem that benefits from parallelization. Therefore, Gustafson's law, like Amdahl's law, does allow the existence of an ideal speedup $S_N = N$ in case of nonexistent sequential part of the problem (i.e. $p = 1$).

The experiment performed to measure the execution times for weak scalability was very similar to the previous one. The main exception was the problem size, which was set to always result in a constant size per core — $1e4$ cells. As previously, each experiment was repeated 5 times to eliminate any random disturbances. The experiment was executed on 1, 2, 4, 6, 8, 10, 20, 40, 60, 80, 100, and 150 nodes, i.e. 24–3600 cores.

Figure 7.2a shows the measured execution times with vertical markers for the standard deviation. While the theoretical ideal time of execution is a constant value for all numbers of cores, the results show expected deviations and an increase in runtime. The relative difference between the times measured for the smallest and highest number of cores shows an increase of 16%. The significant increase in execution time for the 2400 cores shows the largest relative difference — 41%.

In Figure 7.2b it can be observed that the speedup conforms to Gustafson's law — the calculated values appear to be roughly linear, with a slope coefficient of approximately 0.8, with no signs of any significant deviation from this trend.



(a) Execution time.

(b) Speedup.

Figure 7.2: Execution times and calculated speedup for weak scalability. Source: [52]

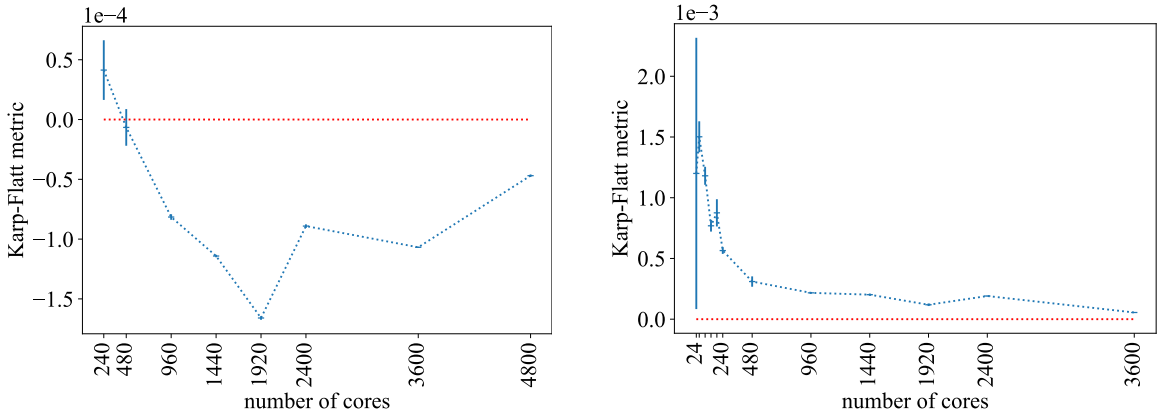
The conclusion from this experiment is that the method does not possess any quality that would increase the share of the sequential part of the problem. A significant increase in both problem size and available resources (by a factor of 150) did not lead to a significant increase in runtime, suggesting that the further increase in resources would allow handling even larger problems efficiently.

7.4. Karp-Flatt metric

Another useful approach to measuring scalability is *Karp-Flatt metric* [33] (also named *serial fraction*), which provides an insight into the the extent to which a particular program is parallelized. In its simplest form, it can be expressed as the following formula:

$$sf = \frac{\frac{1}{S} - \frac{1}{N}}{1 - \frac{1}{N}} \tag{7.11}$$

where sf denotes *serial fraction* (a fraction of the program that did not benefit from parallelization), S is the obtained speedup, and N is the number of cores. The metric proves to be a very useful tool to assess how much the program actually benefits from the distribution.



(a) Karp-Flatt metric for strong scalability experiment.

(b) Karp-Flatt metric for weak scalability experiment.

Figure 7.3: Karp-Flatt metric values

When using the speedup values calculated in Section 7.2 to calculate the Karp-Flatt metric, the obtained results are as shown in Figure 7.3a, with a red dotted line marking the ideal serial fraction of 0 (i.e. the whole program is fully parallelized). Unfortunately, the values suggesting superlinear scalability cause the serial fraction to drop below zero. This outcome does not yield any useful information about the parallelization of the program.

Although originally the Karp-Flatt metric has been only declared to be consistent with Amdahl's law, the formula is expressed in terms of speedup and number of cores. Therefore, it does allow for the use of speedup obtained in weak experiments. When the serial fraction values are calculated using the values obtained in Section 7.3, the observed results are as shown in Figure 7.3b. In this case, the values remain above the ideal line, but are very small. The values for the low core counts are mostly in the range 0.0005–0.0015, which can be expressed as 0.05%-0.15%. As the number of cores increases, the serial fraction values decrease, ending in the range of 0.00005–0.0002 (0.005%–0.02%). This leads to the conclusion that the part of the algorithm unable to benefit from parallelization is very small and unlikely to cause significant loss of efficiency when the simulation is distributed using even larger numbers of cores.

8 Method applicability

This chapter discusses the applicability of the proposed method. First, examples of models successfully adapted to this method are presented. The chapter then summarizes the research on the further expansion of the method to allow for a wider variety of scenarios.

8.1. Successful application cases

In this section, examples of successful model adaptations that take advantage of the presented method are provided. The models were implemented at different stages of the development of the presented method, therefore some discrepancies may occur. In cases where the differences were significant — with some of them providing crucial observations for the method under development — they are mentioned and discussed.

8.1.1. Office tower evacuation

The research presented in this section was originally presented in a publication [49] that is part of the work presented in this thesis.

The model, developed in conjunction with the presented method, addressed the problem of a real-world fire evacuation scenario for a real building, presented in [24]. Two key problems needed to be considered to construct an accurate representation of the modeled phenomena:

– The behaviors of modeled people should be able to reproduce the behaviors observed in real-world scenarios.

– The modeled building did not require to be modeled in 3D, as only the floors and doorways are taken into consideration, but was also not directly convertible to 2D.

In addition to that, the model should be efficient and scalable, able to benefit from the assignment of increasing computing resources to its execution.

The first problem — realistic behavior of people — has been resolved by taking inspiration from the SDM (*Social Distances Model* [69]), which proposes an approach in which social beings are aware of their surroundings, trying to keep some distance from other beings. SDM is in turn based on *proxemics* [26], which describes different types of distances between two people and how those distances are subconsciously respected in social interactions. The inspiration taken from these concepts is similar to [56] in its basic assumptions:

– Each person can occupy a single cell in a discrete environment.

– Each person pursues some goal, usually represented as a specific point in environment the person intends to get to.

– Each person tries to avoid getting too close to other people.

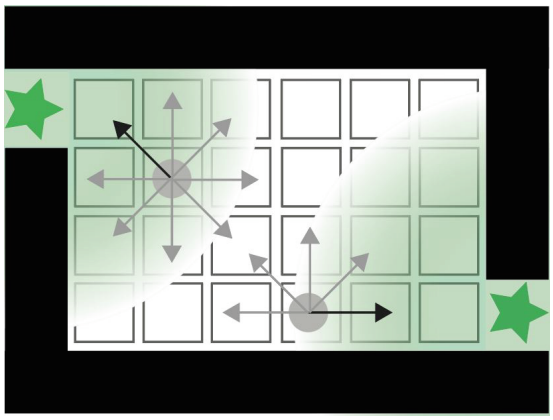


Figure 8.1: Visualization of the dynamic influence of the signal on agent decisions. Source: [49]

The last two assumptions act as opposing forces that influence the current behavior of the modeled being. In order to ensure good scalability and the ability to divide the environment between several computing nodes (workers), the guidelines presented in Section 5.3 were applied. In particular, awareness of the surroundings has been implemented using the signal propagation method, using the negative signal emitted by each human being as a repelling force for the decision-making logic of each being. The same mechanism was used to guide people towards the exit of the building, which was the common destination, with the use of the positive signal emitted by each exit door. This decision proved especially useful, as the specific exit was not predefined for any person, instead relying on the propagation of signal to dynamically attract an agent towards the most favorable exit. The signal emitted by exits was calculated prior to the simulation, so the agents do not have to wait for it to propagate in real time. This approach can be compared to a *floor field* [68] of *potential field* [70] concepts.

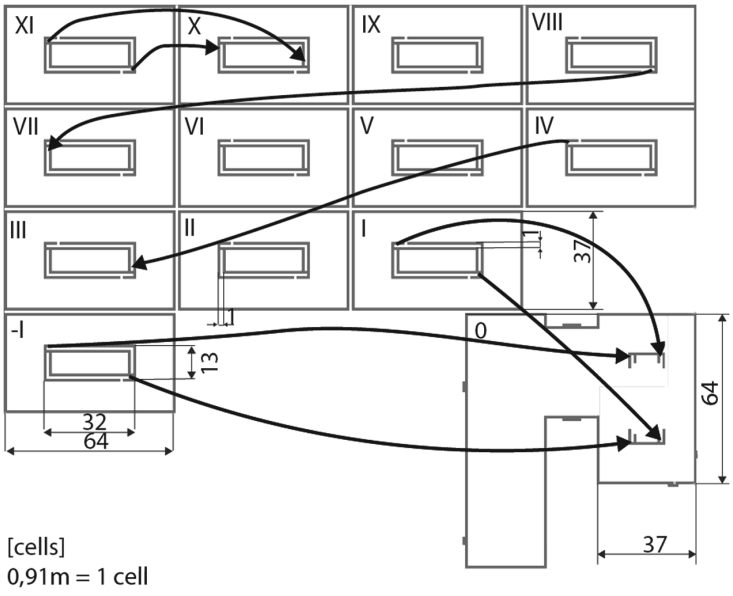


Figure 8.2: Evacuated building plan. Source: [49]

The resulting rules influencing the behavior of modeled beings are visualized in Figure 8.1. Two agents, marked with gray circles, are trying to keep their distance from each other. At the same time, they try to reach one of the exits marked with green stars. Without scanning the nearby area, each agent is able to analyze the signal available in its cell and determine the best direction, marked with a black arrow. Therefore, regardless of the details of the building plan and the existence of borders between partitions of the environment, the same logic will allow an

agent to reach its destination while maintaining its distance to other agents.

The second problem — representation of the building plan — was solved with the use of environment model generalization presented in Chapter 4. The modeled building consisted of several floors (the more detailed plan will be presented further), connected by the staircases. The problem was very similar to the one presented in Figure 4.1 (see Chapter 4). Therefore, for the purposes of this model, an additional type of connection between cells was added, representing going ”up” and ”down” in the staircase. Although some of the details in the original publication differ from the ones presented in this thesis, they solve the same problem using the same idea.

The modeled building consisted of 14 floors connected by two staircases. Floors I–XI and floor -I were almost identical to each other, while floor 0 was unique in its shape and contained all exits from the building. The plan is presented in Figure 8.2. The up/down connections are marked for some of the floors, but most of them are omitted to avoid excessive markings that would render the plan unreadable. In general, the connections between floors XI and X repeat for all other floors. The floor XII was present in the building plan, but was omitted in the evacuation drill, and therefore it was not included in the model.

The evacuation process was divided into two phases:

– evacuation after alarm on floors V, VI, and VII, as in [24],

– evacuation after alarm in the entire building.

To simulate the reluctance and preoccupation of the people, they were added to the building over time, joining the ongoing evacuation process. This process attempted to match the pre-evacuation times presented in [24].

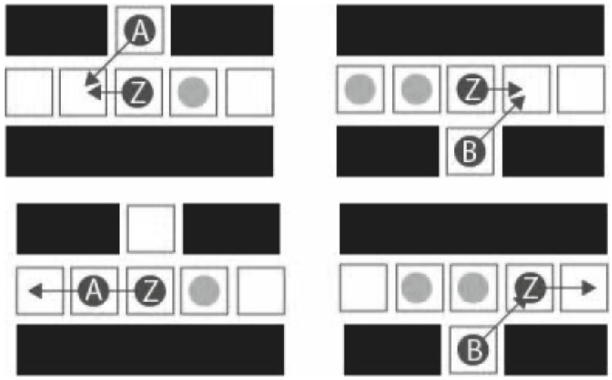
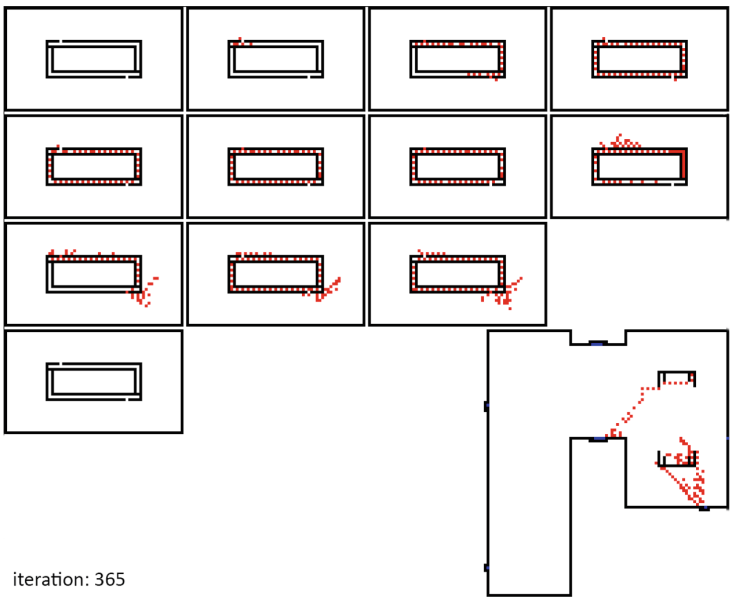


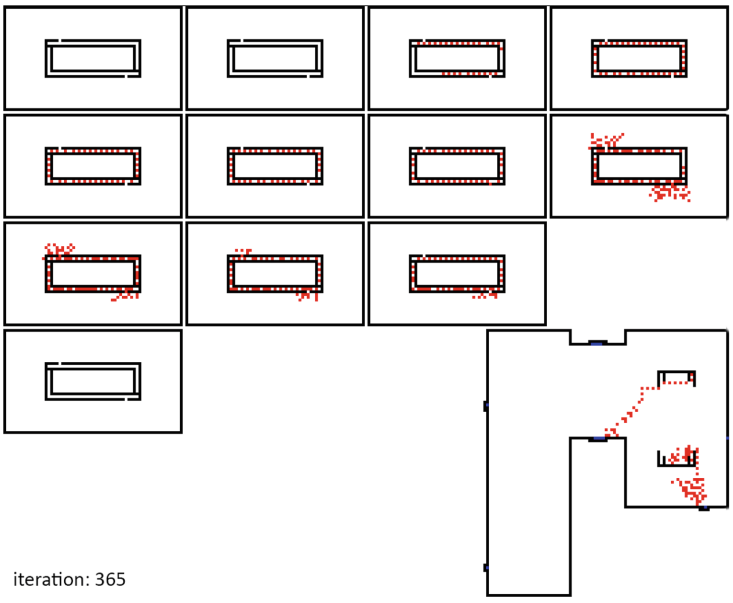
Figure 8.3: Prioritization resulting from the sequential order of update application. Source: [49]

Two important observations during the development of this model were made, one regarding the logic of decision making, and the other regarding the order of update/plan application.

The first one arose from the animated visualization of the evacuation process (the link to the recording is available in [49]). Once the agents reached the bottleneck in the form of the staircase entrance, the attracting force was strong enough to allow them to form a dense crowd. Some agents were observed to move back and forth, ”bouncing” back from the crowd, and then approach the exit again. The further analysis helped determine the cause: the logic responsible for the decision making checked adjacent cells to find all possible (i.e. empty) destination cells, and then selected the cell associated with the most attracting (or least repulsing) signal. Therefore, if an agent was blocked by another agent occupying the cell that was optimal for the deciding agent, the second-best direction was the one that allowed the agent to back away from the crowd. This created a strange behavior, resembling an agent forgetting its goal in favor of maintaining social distance in one iteration, and immediately returning to the pursuit of its goal in the next iteration. The solution to this problem was the alteration of the decision-making algorithm: if the cell associated with the most attracting signal is unavailable, the agent remains in its original cell without moving.



(a) Sequential order of updates.



(b) Random order of updates.

Figure 8.4: Comparison of plan/update processing order. Source: [49]

The second observation was another instance of hidden bias that can significantly influence the simulation progress, similar to the ones discussed in Chapter 2. As the method presented in Chapter 5 was still being developed, the model initially used the more naïve order of decision processing and application — the agents performed their movements in an order determined by the order in which the environment data structure was traversed by the system. In the used implementation the iteration of the underlying grid was performed row by row, beginning with the top one, and from left to right in each row. This resulted in visible prioritization of the decisions made by agents above and/or to the left of the agents that could possibly collide with them (see Figure 8.3). Figure 8.4 shows the visualization of an arbitrarily chosen iteration using this order of updates and the same iteration using random order, which is in line with the method presented in Chapter 5. When the order is predefined (Figure 8.4a), the agents entering through the top entrances of the staircase on each floor tended to have priority over agents already present in the staircase. At the same time, agents entering through the bottom entrances had to wait for the staircase to empty in most cases. This resulted in asymmetry in some floors: most of the crowds waiting at the top entrance already entered the staircase, while the crowds at the bottom entrances were still waiting. Symmetrically, the top staircase contained more agents than the bottom one, because the agents descending from the upper floors had to wait until crowds entered the staircase on the lower floors before moving past the entrance. It is safe to assume that this behavior would be reversed if the grid was rotated by $180\,\mathrm{deg}$ or if the order of traversal was reversed. The visualization of the simulation with the random order of updates (Figure 8.4b) does not show any strong bias toward any side of the building. On the other hand, if such behavior is desired, e.g. if the momentum of people descending the staircase should prevent anyone from entering the staircase, it should be explicitly expressed within the model or its logic, without reliance on implementation details affecting the outcome.

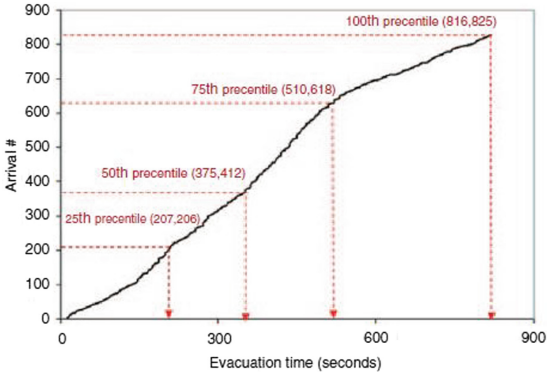
To analyze the influence of those observed phenomena on the simulation results, the four combinations of the simulation were executed:

– *Moving Sequential* — agents always move, the order is determined by traversal.

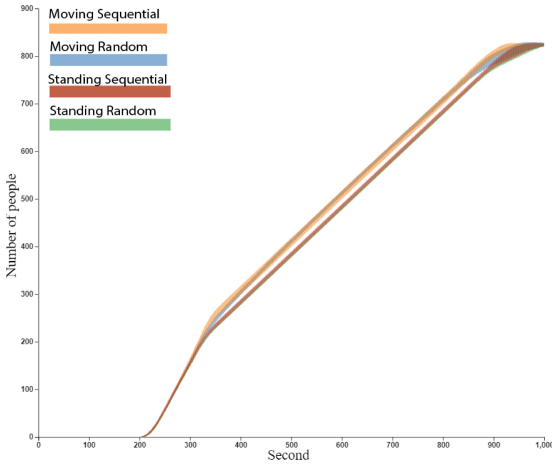
– *Moving Random* — agents always move, the order is determined randomly.

– *Standing Sequential* — agents do not move if the optimal cell is occupied, the order is determined by traversal.

– *Standing Random* — agents do not move if the optimal cell is occupied, the order is determined randomly.



(a) Empirically measured evacuation curve. Source: [24]



(b) Simulated evacuation curve. Source: [49]

Figure 8.5: Comparison of empirically measured and simulated evacuation curves.

The results of this experiment were measured as the cumulative number of people that left the building over time. The evacuation curve created from this data is shown in Figure 8.5, compared to the original data collected during the drill.

The obtained results form a very regular curve, in which, after the initial delay, the agents started to exit the building. The initial slope is much steeper than the rest of the curve, due to people already present on the lower floors. After that, the flow was limited by the throughput of the staircases. The last portion of the curve is even less steep, as the few remaining agents had to leave the mostly empty staircases and get to the exits.

The difference between different variants is barely visible in Figure 8.5b, with *Moving* behavior slightly more effective in the initial phase. The difference between *Sequential* and *Random* variants is almost nonexistent, which — perhaps counter-intuitively — highlights the dangers of introducing an unexpected bias into the simulation. The asymmetry was not visible in the final results, as the chosen metric was not sensitive to such phenomena. At the same time, the actual behavior of agents observable in the visualization was clearly not intended.

When comparing Figure 8.5b with Figure 8.5a, it is clearly visible that the original curve was much less regular and smooth. This difference is expected, as the model did not account for the specific behavior of the people, which is usually far from uniform in such situations. Similarly, the exact location of each person and the exact time they started to follow the evacuation instructions were not provided. Finally, the description of the original results seems to mismatch the data, as it is stated that no person left the building before the general alarm, i.e. within the first 200s of evacuation. On the other hand, the time elapsed until the last person exited the building is very close: ca. 820s in the original data and ca. 900s in simulation.

The conclusion was that this experiment was very promising for the evaluated method. Despite assuming some uniform ”average” behavior for all agents, the results obtained from the simulation were very similar to the ones measured in the real-world evacuation drill. With a more advanced model, taking more intricacies of human behavior into consideration, it seems very likely that this simulation could yield valuable information regarding the fire safety of a modeled building. At the same time, the scalability of the method ensures that results can be obtained quickly, efficiently utilizing the assigned resources.

8.1.2. Pedestrian traffic in urban area

The research presented in this section was originally presented in a publication [51] that is part of the work presented in this thesis.

Following the outbreak of the Covid-19 pandemic, a large amount of research in a variety of scientific fields gathered increasing popularity, due to the potential usefulness in fighting the disease or its effects. One such area was the interaction between urban structures and the spread of the virus. It is an especially important topic nowadays, when paradigms change rapidly and various directions of further evolution are proposed [15, 36, 38, 57, 59], often contradicting each other in some aspects.

The main motivation of this research lies in the high urbanization of Poland (more than 60% in 2019), mostly focused in small- and medium-sized cities [37], combined with the fact that densely populated areas such as cities are much more susceptible to natural disasters, including epidemics [64]. Therefore, it is especially important to assess the risks related to specific urban and architectural structures with regards to the spread of the pandemic, in order to gain insight necessary to amend the situation. As the *urban tissue* is subject to constant growth and change, it is crucial to learn as much as possible from existing areas to help in the development of safer and more resilient cities and districts. The observation of this process in Poland in recent history [25, 31, 41] shows that the resulting structures are defective in terms of functional layout and urban form, with the irregularities emerging from the socio-economic changes being a recurring topic in discussions.

The elementary tools used to study the phenomena occurring in urban areas are direct observation (either in person or utilizing recordings, e.g. from security surveillance) and surveying the people present in those areas. Those methods are time-consuming and usually require the participation of a significant number of people, especially if the research is supposed to yield results that are applicable to a wider population. This is one of the reasons why the use of simulations is gaining increasing interest in this field. The use of agent-based microscopic models provides the ability to track the behavior of individual entities and introduce some variation in the modeled beings.

The selected approach was based on the same principles as in [49], i.e. using the modified version of proxemics rules [70] within a discrete environment. Therefore, the solution presented in Chapter 5 was utilized. The main objective was to propose a highly scalable and efficient simulation of pedestrian behavior that will allow the user to take the changes in behavior due to Covid-19 [29, 32] into consideration. Moreover, the model should be accurate and able to represent behaviors that match the empirically collected data.

The proposed model uses a discrete grid of cells, each assigned a specific area type. The main function of the type is to provide the associated *walkability factor*, which serves as a measure of the eagerness of the pedestrian to enter this area. Values range from 0 (impassable) to 1 (dedicated to pedestrian traffic). This factor is essential in the calculation of preferred paths, which will be described further in the text.

The most important part of the environment is the presence of pedestrian destinations. Each destination, usually representing a single building or place, consists of the following elements:

– A set of coordinates corresponding to the entrances to the destination. A single entry is often sufficient for places such as bus stops or small shops, but residential buildings often have multiple entrances.

– A set of destination types. This type determines the purpose of the destination in broad terms, e.g. services, residential place, etc.

– Population of the destination. A number of people that permanently occupy this destination, with non-zero values usually reserved for residential buildings.

Both the *walkability factor* and destinations are used in the path creation process. This process precedes the simulation and yields the preferred paths that pedestrians will use to reach a given destination. A single destination is chosen, and its entrances are assigned an entity that constantly produces a signal. The spreading of the signal is modified to allow two important phenomena to occur:

– The original propagation method presented in Section 2.4 is changed to allow the signal to ”bend” in all directions, which in turn allows it to reach the same cell via different paths.

– In addition to the normal suppression and attenuation factors, the signal is also reduced by *walkability factor*.

These changes ensure that the signal has a chance to take the pedestrian preferences (represented by the *walkability factor*) into consideration. Examples of such occurrences might include a path through a greenery area having a stronger signal than the much longer pavement around the area, or a path across a street in an unmarked, dangerous place being preferred over a safe, but very distant crossing.

The signal is allowed to cover the whole grid and is converted to a path using a simple rule: for each cell, the optimal direction to choose in the next movement step is the direction associated with the strongest signal present in that cell. The resulting paths take all necessary factors into consideration: *walkability factor*, distance to the nearest entrance, and the number of entrances.

The final component of the model is the agent representing the modeled pedestrian and its behavior. The lifecycle of each agent consists of three stages that also determine the necessary attributes assigned to it: creation, destination selection, and navigation.

For the creation of an agent, it is necessary to know the distribution of the habits of people regarding exiting the buildings. Combining this data with the population of the building, it is possible to determine whether a person should leave a given building through a given entrance in a given time. As this data is usually sensitive to the time of the day, the model can be fine-tuned to any number of periods within a day with different distribution data.

Once the agent is created, it is assigned a destination. As in the previous stage, it is necessary to provide the model with data regarding the distribution of the destination types. Similarly to agent creation, this distribution can change over the course of the day. Based on the distribution of target types chosen by pedestrians for this time of the day, a destination type is selected, followed by a selection of the specific destination of this type. The specific destination is selected randomly, but the model can be easily adjusted to take some factors into consideration, e.g. distance or some more specific subtype of the destination. During the research, an additional special type of destination has been created: *wandering*, which causes the agent to periodically change destinations, emulating a stroll without any specific destination.

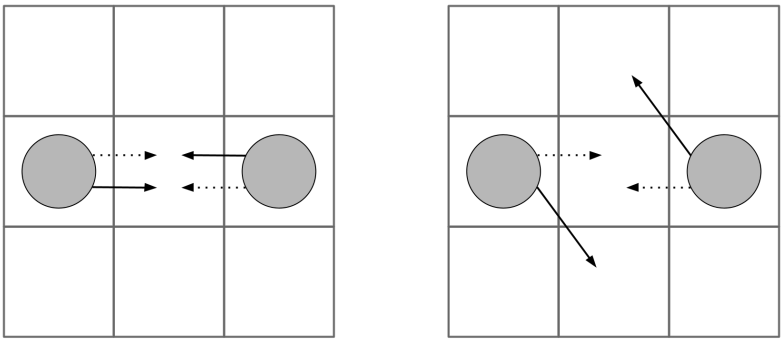
When the agent already has a specific destination, it begins to follow the precalculated path towards the destination. In the most basic form of navigation, the decision regarding the direction in each iteration of the simulation is simplified to the selection of the known optimal direction. However, as the main focus of the research was the study of behavior under epidemic conditions, this behavior was extended to allow the person to perceive and react to other agents. Therefore, each agent creates a marker at its location that carries several pieces of information:

– which agent is the source of the marker,

– what is the distance and direction from the origin point of the marker,

– what was the time of the creation of the marker.

In the following iterations (and usually at a rate faster than the rate of agent movement), the marker spreads its copies to the adjacent cells, with proper adjustment to the information about the distance and direction to the origin. Only copies with the lowest distance are preserved as it is possible for them to reach the same point from different directions and with different distance data. If the difference between the current time and the creation time of the marker reaches a predefined limit, the marker is removed. The idea behind the markers is similar to the signal and its propagation, but in this case the signal is discrete, does not combine with other instances of the signal, and carries some information different from the strength of the signal. It is also not allowed to spread indefinitely, and instead it is limited in its lifetime. The signal serves as a medium of communication between agents — if an agent reaches a cell with a marker created by another agent, it knows that the source of this marker is nearby.



(a) Social avoidance disabled. (b) Social avoidance enabled.

Figure 8.6: Behavior of pedestrians in proximity to other pedestrians. Dotted arrows represent precalculated optimal directions, solid arrows represent the chosen movement direction. Source: [51]

The use of markers allows agents to apply social avoidance: if an agent knows that there is another agent in a given direction, it can adjust its path to avoid contact. If that agent is in the direction different from its intended optimal direction of movement, it can continue without any adjustments. Otherwise, it will try to select an adjacent direction, preferring the clockwise neighbor. The behavior emerging from this modification is shown in Figure 8.6, where Figure 8.6a shows the simple navigation without avoidance, while Figure 8.6b shows the avoidance affecting the decisions.

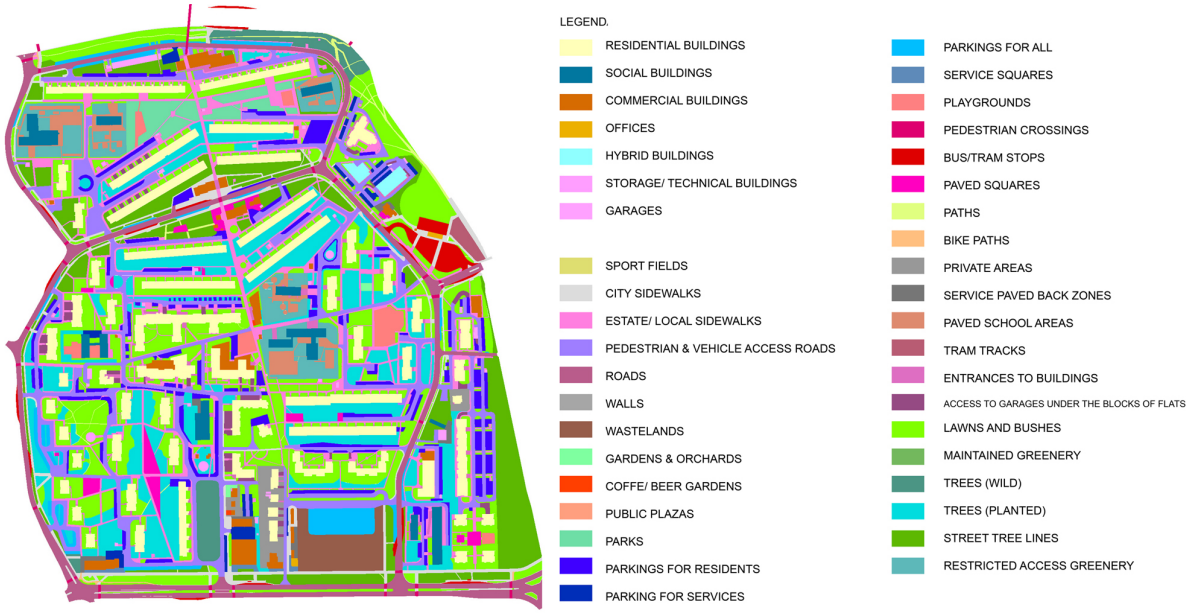


Figure 8.7: Plan of the modeled area with different area types. Source: [51]

It is estimated that 25–40% of the households in the larger cities in Poland inhabit housing blocks from the second half of the twentieth century [22, 6]. Therefore, chosen as the case study matching this data, the modeled area was a part of the city of Cracow, Poland. This housing estate is located in the northwest part of the city, with the population density over 9000 people per square kilometer (90 people / ha), not excluding greenery areas, roads, etc. The area is also relatively self-enclosed, as a variety of services are available locally, while the presence of establishments generating external traffic (public institutions, tourism-related places) is minimal. These features make this area well suited for this type of research.

The geodetic plans for the area were available and provided the data necessary to create a 2D plan shown in Figure 8.7. The legend shows all relevant area types, which were later assigned *walkability factors*. In the final grid representing the modeled environment, each cell represented a 1×1 m area, with a total of 800×960 cells representing area of size 800×960 m. Each cell was assigned the area type that occupied the largest portion of the cell, as the resolution of the original plans was significantly higher.

Information about the number of people living in the modeled area was estimated from the *gross floor area* (GFA) and the known average number of people for each building type. Four building types were selected:

– residential,

– office,

– social (schools, libraries, other cultural facilities),

– commercial (shops and other services).

Residential buildings were assigned 1 person per $30m^2$ of GFA, while for other types this value was 1 person per $35m^2$ of GFA. As some buildings are assigned several types (e.g. a residential building with shops on the ground floor), the share of the area used for each type was included in the calculation.

The most challenging part was the collection of data on the behavior and motion of people. Representative residential buildings were selected for analysis, which was carried out in accordance with the observations in [20]. It was assumed that each day is similar to the other days, and it is sufficient to perform the measurements in 15-minute intervals, four times in a two-hour period. Four representative periods were selected:

- morning peak (6:00 AM – 8:00 AM),
- noon hours (11:00 AM – 1:00 PM),
- afternoon peak (4:00 PM – 6:00 PM),
- evening hours (8:00 PM – 10:00 PM).

Two types of information were collected on two separate days, both displaying favorable weather conditions that correspond to the maximum traffic intensity. The first part was the collection of data on the distribution of people leaving the building over time, which is necessary to calibrate the creation of pedestrian agents in the model. The second part involved tracking (hidden observation) of randomly selected residents to record the distribution of types of destinations selected by pedestrians. The following types of activities were distinguished:

- Transport-related destinations:
  - a parked car,
  - a parked bicycle,
  - a bus stop,
  - a pedestrian crossing leading outside the modeled area.
- Recreational destinations:
  - playground (includes parks),
  - wandering (includes walking with a dog and jogging),
  - another residential building,
  - fitness- or sports-related place.
- Utilitarian destinations:
  - shop,
  - service,
  - healthcare-related institution,
  - education-related institution.

The experiments were conducted in three variants:

- In the first variant, people were not avoiding each other (i.e. not applying social avoidance, as in Figure 8.6a) and following the recorded distribution of destination selection. This variant represented the standard behavior of residents.

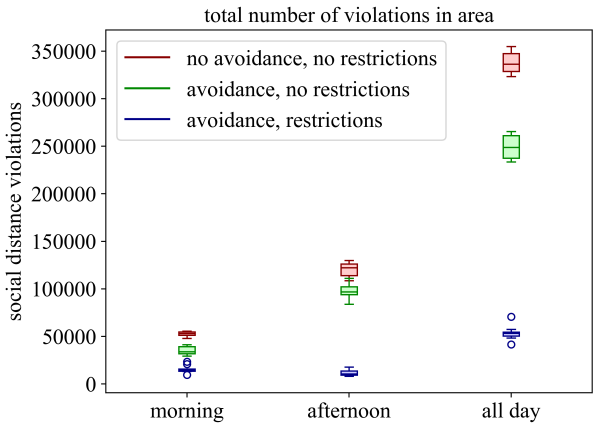


Figure 8.8: Sum of all violations for different behaviors in the time windows: morning (6:00 AM –– 8:00 AM), afternoon (4:00 AM — 6:00 PM), all day. Source: [51]

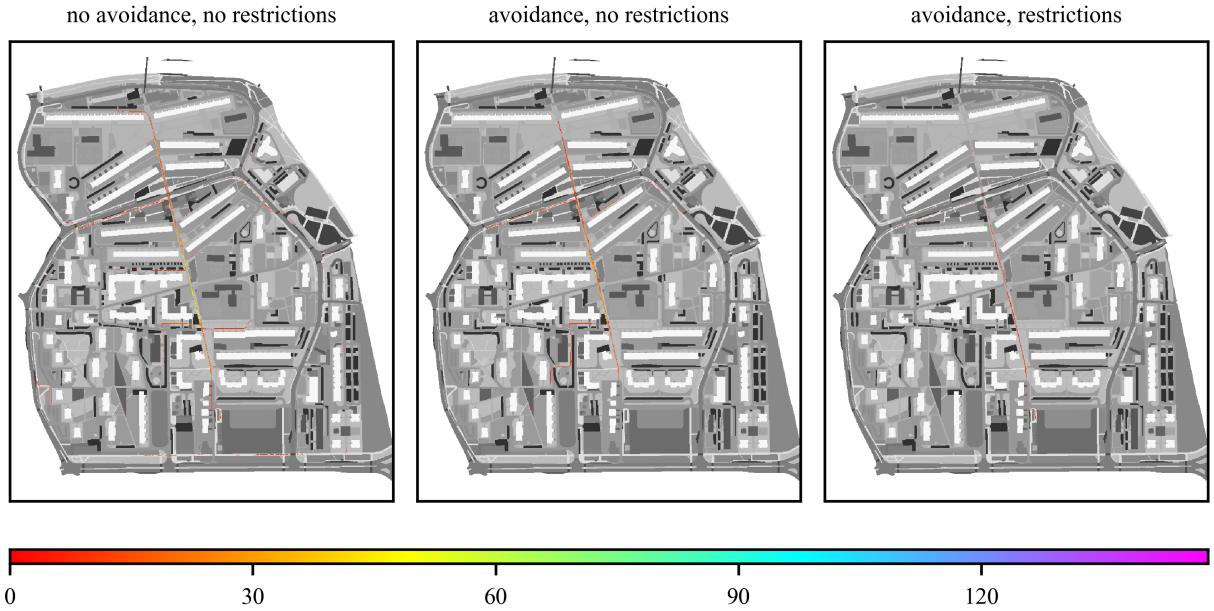


Figure 8.9: Violation locations in the morning (6:00 AM –– 8:00 AM). Source: [51]

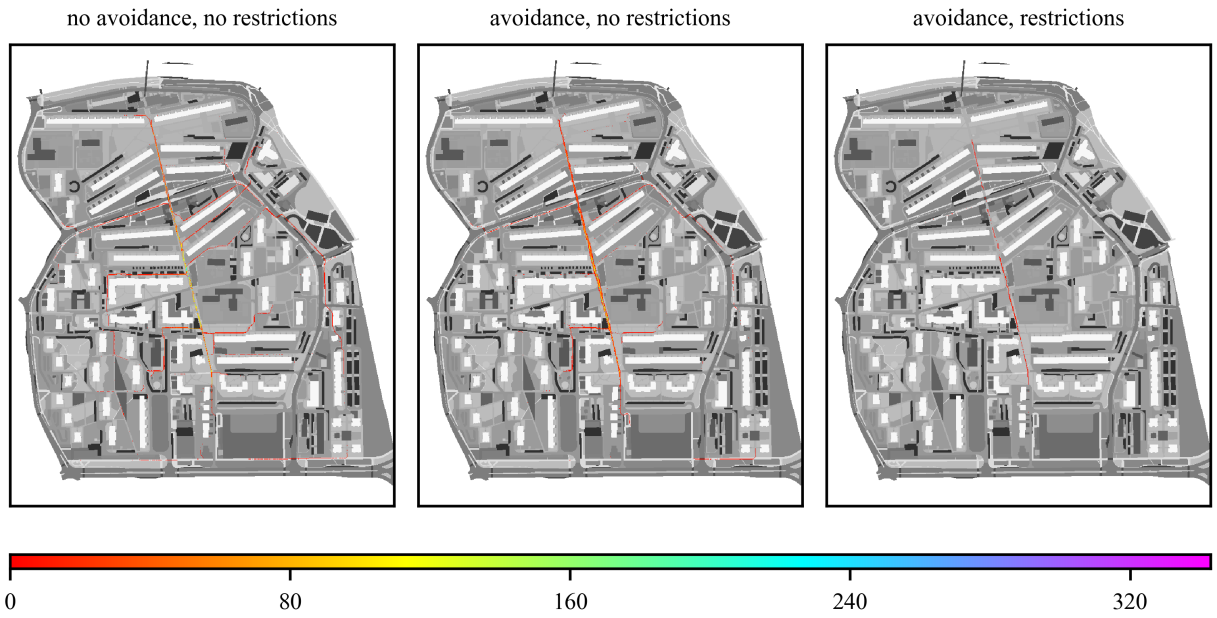


Figure 8.10: Violation locations in the afternoon (4:00 AM — 6:00 PM). Source: [51]

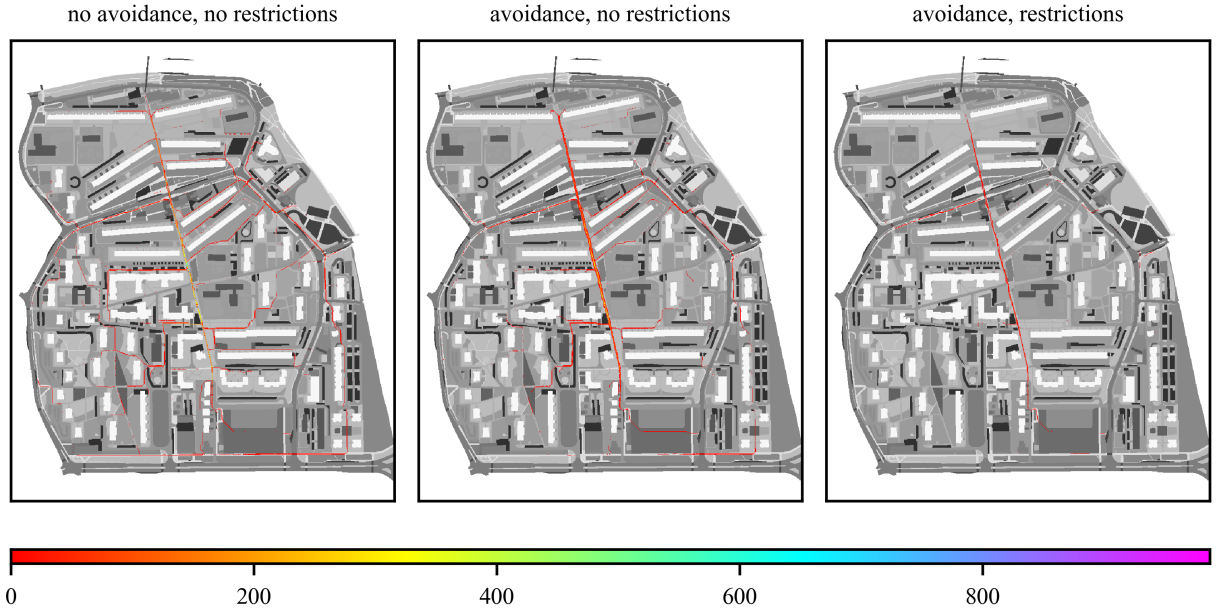


Figure 8.11: Violation location during whole day. Source: [51]

– In the second variant, people were avoiding each other (i.e. applying social avoidance, as in Figure 8.6b) and following the same distribution. This variant represented the behavior of residents following the rules enforced during the pandemic.

– In the third variant, people were avoiding each other, but additionally additional restrictions were enforced on destination selection: people that would select wandering behavior or choose playgrounds as their destinations were not created (i.e. it was forbidden to participate in such activities), while visits to other residential buildings and services were reduced to $1/3$ of the recorded distribution. This variant represented the behavior of residents following the rules enforced during the pandemic, while the functioning of some of the destinations was suspended.

Each variant was executed 10 times, utilizing 576 computing cores in parallel.

For each variant, each occurrence of an agent encountering a marker with a distance from another agent below a specified threshold (1 m in this case) during any given iteration (1 s of real time) was registered as a violation of social distance. This threshold was selected in accordance with the recommendations regarding social distancing issued by official sources. The number of violations was aggregated to show the total number for each time of the day. The results are presented in Figure 8.8, presenting the sums for the morning peak, the afternoon peak, and the whole day. The three series correspond to the three variants, as shown in the legend. An interesting observation was that although the restrictions significantly reduce the total number of violations for the whole day, the less busy times of the day (e.g. morning peak) did not record such a big difference. The conclusions were that the restriction on the possible destinations is much more effective if its absence would lead to the number of pedestrians increasing to the point in which it is not possible to avoid other pedestrians successfully.

Further analysis has shown that there are specific parts of the area that attract the majority of pedestrian traffic. Figure 8.9, Figure 8.10, and Figure 8.11 present locations in which at least 5 violations were recorded in the morning peak, the afternoon peak, and during the whole day, respectively. In all cases, the main pedestrian tracts are highlighted, with additional branching paths appearing for the less restricted variants. The conclusion is that even with the most restricted variants, some portions of the area are not wide enough to facilitate easy avoidance.

A final, yet important part was a validation of the model against the real-world data. As it is not reasonable to expect any specific behavior of an observed person to be perfectly reproduced, it was necessary to propose a validation method that focuses on statistics related to the observed phenomena. As a result, taking inspiration from [71], the selected method relied on the comparison of photos taken at predefined places with the selected time frames in the simulation. Several places in the area were chosen for the photos to be taken, as in Figure 8.12. At the selected moments in all four times of the day, photos were taken at those spots. All photos were then compared with the corresponding moments in the simulation, and the people visible in both cases were counted. An example of such a comparison is shown in Figure 8.13, where the photo on the left corresponds to the non-gray part of the grid on the right.

Table 8.1: Comparison of average number of people registered in photos and in simulation, in the same places and times of day

	Zone 1		Zone 2	
Time	**Photo**	**Simulation**	**Photo**	**Simulation**
Morning Peak	$8,5$	$7,75$	$12,5$	$5,5$
Noon Hours	$14,25$	$12,5$	16	$8,75$
Afternoon Peak	$19,75$	$24,5$	19	$21,25$
Evening Hours	6	$9,5$	8	$7,75$

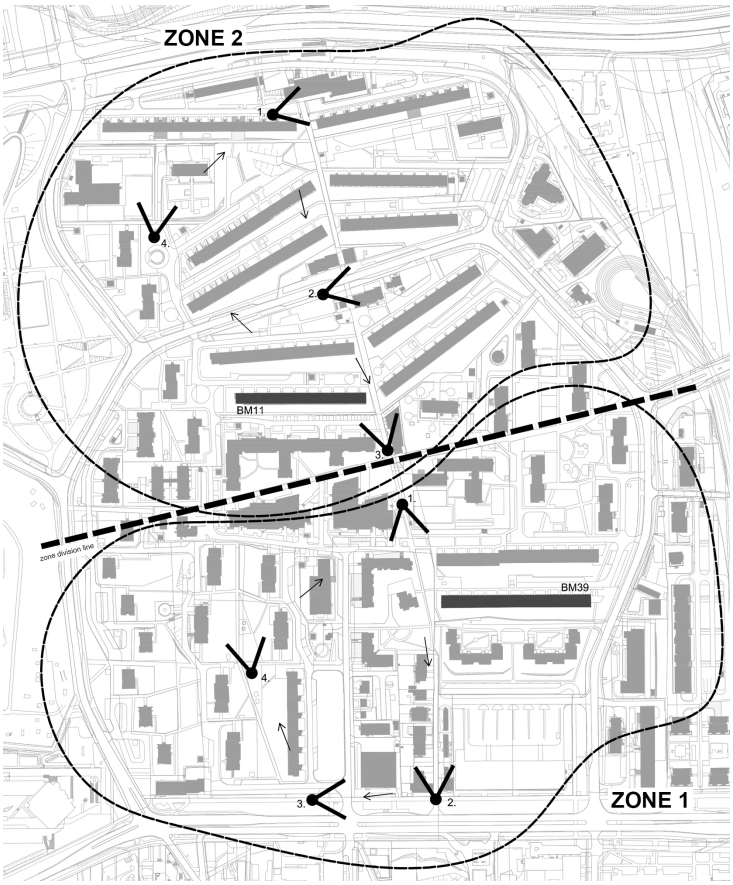


Figure 8.12: Area divided into zones with locations of photos marked. Source: [51]

Figure 8.13: Comparison of the number of people in the cameras range and in the simulation at 5:30 PM in zone 1, location 3. Source: [51]

The results of this analysis show that there is a notable difference between different times of the day, and both the reality and the simulation follow a similar trend. Table 8.1 shows the summary of the number of people registered in both methods of data collection. Some discrepancies occur, but they can be explained by the lack of sufficient data to fully account for two aspects of people behavior:

– In both zones, the afternoon peak recorded more people in the simulation. This is probably the result of the lack of data on the duration of stay of each pedestrian after reaching the destination. As no data was collected that could provide proper values, the expected delays before returning were chosen arbitrarily, with estimations based on the type of destination. This explanation is in line with the significant increase in returning pedestrians observed in the more detailed data collected during the simulation.

– In almost all other cases, the number of people is lower in the simulation. Part of the explanation lies in the shift of those people to the afternoon peak, but it is further exacerbated by the lack of any data regarding the possible pedestrians incoming from the outside of the modeled area. No such agents were present in the simulation, while the same cannot be stated with certainty about the photos. The bigger total number of people registered in the photos seems to further reinforce this explanation.

In summary, this research has proven that the method proposed in Chapter 5 can be successfully applied in modeling the behavior of pedestrians, and the agent-based approach provides the necessary tools to focus on individuals. The model of behavior used in this experiment is very simplistic but can easily be improved to account for some desired phenomena to occur. At the same time, the method efficiently utilized 576 computing cores, which makes it suitable for HPC environments and allows for a significant reduction in the time needed to obtain the results.

8.2. Support for non-discrete models

The method supports only discrete environments, which is a sufficient approximation for a large collection of models. However, some applications require a more precise representation of either the environment, especially the obstacles that interact with moving agents, or the parameters that describe the position and movement of agents. This section presents the solutions proposed for both problems.

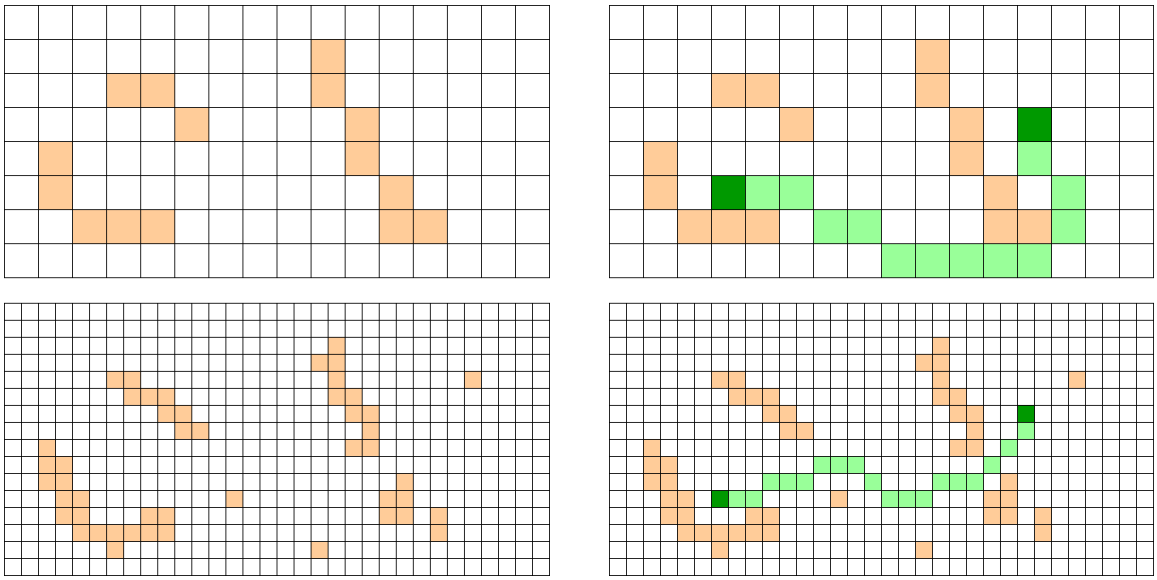
8.2.1. Continuous space model

The work summarized in this section was originally presented in a master's thesis [44] that is part of the research presented in this thesis.

The crucial element of many agent-based simulations is an accurate representation of the environment, so that agents interacting with this environment can appropriately react to its features. In some cases, a precise recreation of the shape of the obstacles is necessary for the results of simulations to be useful for their real-world counterparts (e.g. doorways or complex shapes of hallways). In such cases, discretization of the space leads to loss of precision. At the same time, the method presented in Chapter 5 is defined only for the discrete representation of the space. Figure 8.14 shows a sample environment that changes the behavior of agents significantly — Figure 8.14b shows that in coarser resolution a path through the obstacles to the right is impossible, while in finer resolution it is possible.

A naïve solution for this problem might be increasing the resolution of the grid and, as a consequence, the resolution of the time represented in the simulation. However, this solution might lead to two problems:

– The size of a single cell is no longer enough to contain an agent of the necessary size (e.g. human agent in a grid with cells 1×1 cm in size).



(a) Obstacles.

(b) Sample paths through obstacles.

Figure 8.14: The same real-world environment in different grid resolutions. Source: [44]

– The model is still not accurate enough, while the resolution of the grid has been increased to the point where it significantly reduces the performance of the simulation due to both the size of the environment and the required number of iterations for a given real-world time window.

In both cases, simple scaling of the resolution is clearly not a solution. Therefore, research has focused on providing a way to represent arbitrary polygonal obstacles without loss of precision due to the discretization of the simulated environment. Non-polygonal obstacles are not directly supported, but can usually be approximated by polygons with high accuracy.

The main goals were the following:

– The environment must remain discretized. The method relies heavily on the discretization and benefits greatly from the clear distinction of directions, as well as from the signal propagation method, both of which require a discrete environment.

– The obstacles should be observable locally, i.e. it should be possible to determine the possibility of occupying the current cell and moving to an adjacent cell without any need to scan remote areas of the environment.

– The signal propagation method should accurately reflect the possibility of traversing the environment. In other words, a signal should not attract an agent towards a cell that is not directly accessible by the agent. In particular, it should not attract an agent in the direction of an opening between obstacles that is too small for the agent to pass through.

In order to achieve all the mentioned goals, the following steps were designed to transform the geometric data describing the environment to fit the discrete simulation:

– Environment data is provided in the form of polygons.

– If any polygon contains segments that cross other segments, it is subdivided into several polygons that do not have this quality. This step simplifies future calculations by eliminating unnecessary edge cases.

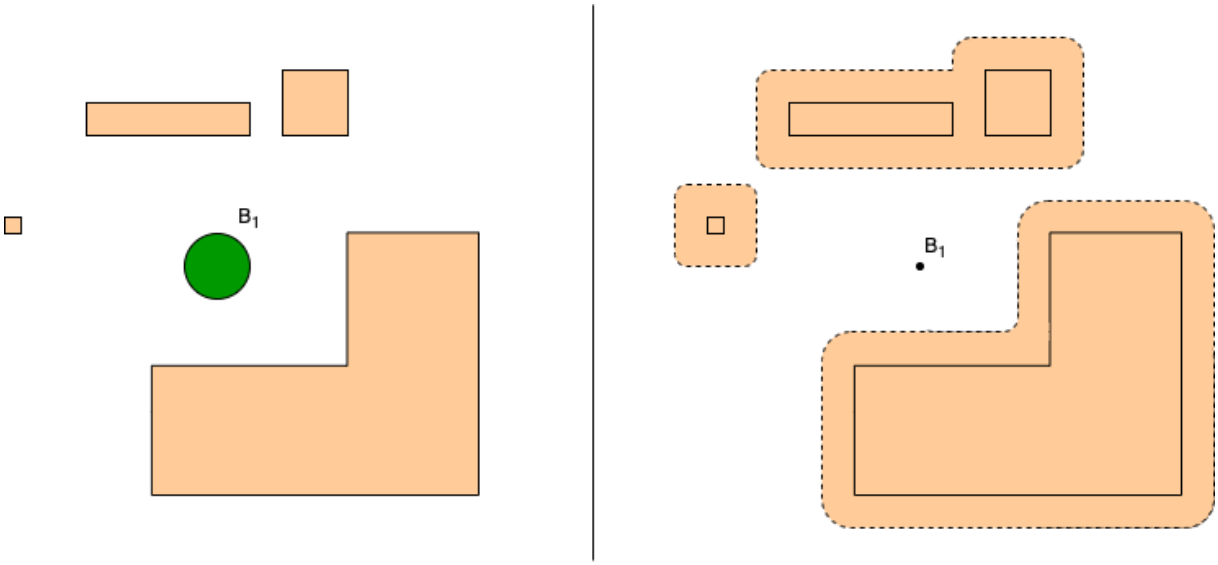


Figure 8.15: Polygon buffering applied to the obstacles, based on the radius of a person agent. Source: [44]

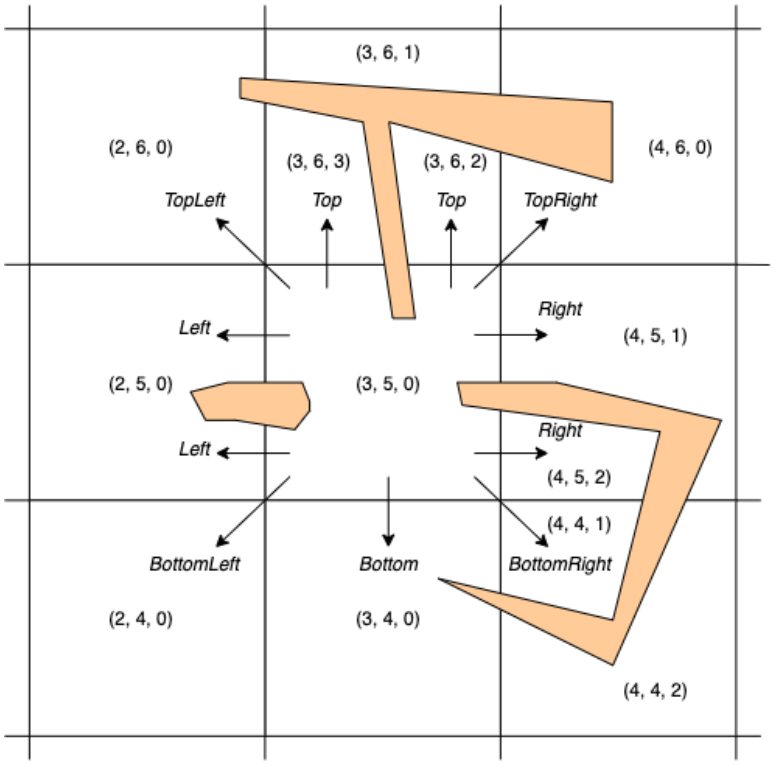


Figure 8.16: Sample result of the transformation of the polygonal obstacles into discrete environment. Source: [44]

– Each polygon is expanded using *polygon buffering* (see Figure 8.15) by a configured radius of a person agent. This step simplifies the interaction between the agents and the environment, as the agent can be treated as a point.

– All intersecting polygons (either by default or after buffering) are merged into one larger polygon (see the two rectangles at the top in Figure 8.15).

– All cells containing obstacles that divide the given cell into parts are removed and replaced by several subcells, distinguished by the additional numbering within the given coordinates. Each resulting subcell and each undivided cell are also provided with information about the contained obstacles.

– Directional neighborhood is extended, so that each direction allows for more than one neighbor, each corresponding to a separate signal.

An example of the resulting environment is presented in Figure 8.16.

The consequence of this method is a change in the rules governing the neighborhood. A direction is no longer limited to a single neighbor and a single signal. As a result, they can be treated as a collection of directions that preserve the features of the original direction (e.g. special treatment of the diagonal directions in signal propagation) but are effectively different directions for all other purposes. A signal propagation has to be changed to reduce the signal in proportion to the percentage of the original side of a cell that corresponds to that direction, e.g. if a side is split into two halves by a very thin obstacle, only half of the signal will be able to flow into the cell from those directions. This is necessary to avoid the multiplication of a signal that splits to flow around an obstacle and creates two equally strong signals, which would add up to a doubled value after joining.

The decision-making algorithm of the agents must also adapt to the changes. First, the agent in such an environment is also assigned a location in a continuous space, as well as some speed. To properly use those attributes, it should not simply follow the direction determined by the strongest signal. Instead, to take full advantage of a precise location, the attraction vector is calculated. Each possible direction is translated to a unit vector pointing in the respective direction and scaled by the value of the signal. The sum of all such vectors in a cell should give an agent a direction that accurately represents all forces acting on the agent. After reaching the edge of a cell, it is possible to determine the neighboring cell or the subcell the agent should migrate to, using the algorithm described in Chapter 5. At the same time, any collisions with locally visible obstacles can be easily resolved geometrically, since the agent is treated as a point.

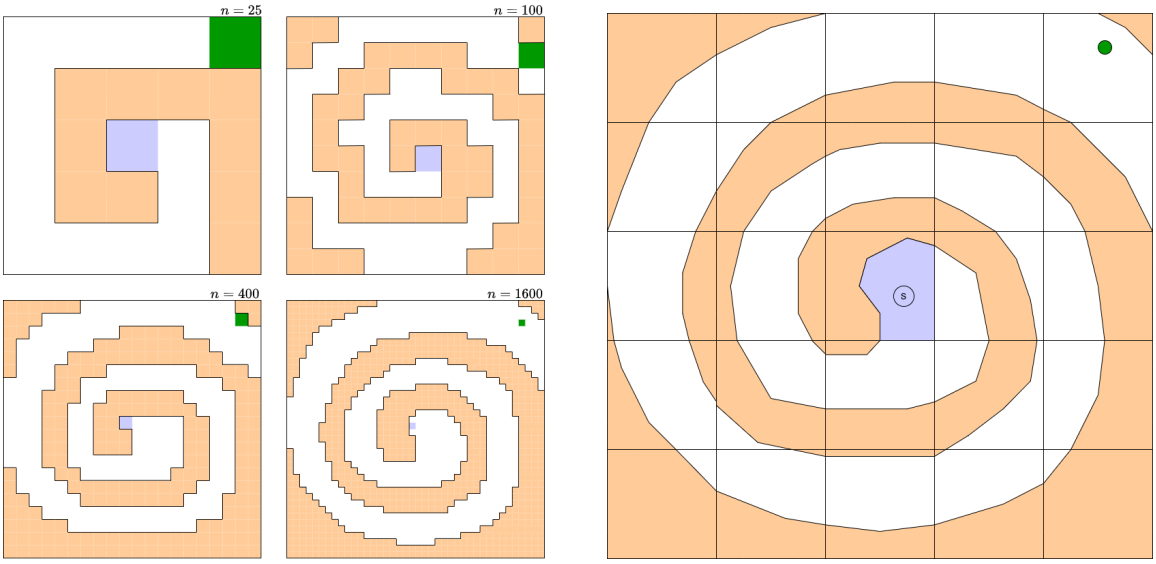
The usefulness of this approach can be demonstrated by an example of its ability to represent obstacles that are not rectangular and aligned with the axes of the grid. An extreme example is shown in Figure 8.17. The same spiral hallway can be represented with different accuracy using increasingly higher resolution of discrete obstacles, but lower resolutions lose important information about the represented shape. At the same time, it is possible to keep the lowest grid resolution while accurately representing the original grid using polygons.

The same applies for many more realistic cases: doorways of shape and size that vary within a single model, irregularly shaped hallways, and smaller obstacles. In each of those cases, it is possible to maintain a low resolution of the grid, which in turn provides much better performance, as fewer iterations are necessary to simulate the equivalent real-world period.

8.2.2. Continuous geometry and kinematics of agents

The work summarized in this section was originally presented in a master's thesis [30] that is part of the research presented in this thesis.

The original approach to agent-based simulation, borrowing heavily from the concept of cellular automata, usually represents an agent as a specific type of cell in a discrete environment. This imposes several restrictions



(a) Spiral approximated with grid of different sizes.

(b) Spiral represented as polygon in grid of size 25.

Figure 8.17: Comparison of the same spiral hallway in discrete grid and using presented approach. Source: [44]

on the dynamics of the agent, the most relevant being a discretization of agent movement. An agent can only perform fixed-length steps in any direction, usually limited to the migration to an adjacent cell. This in turn limits the possibility of the representation of the speed of an agent. Although some approximations are commonly used, e.g. introducing a delay between agent migration to decrease its speed, the resulting dynamics is nonlinear, with the total planned movement occurring in one iteration and the idleness of an agent in other iterations. One of the consequences of this approach is the lack of possibility to represent the states of an agent being in motion — the only states are ”at the source” and ”at the destination”.

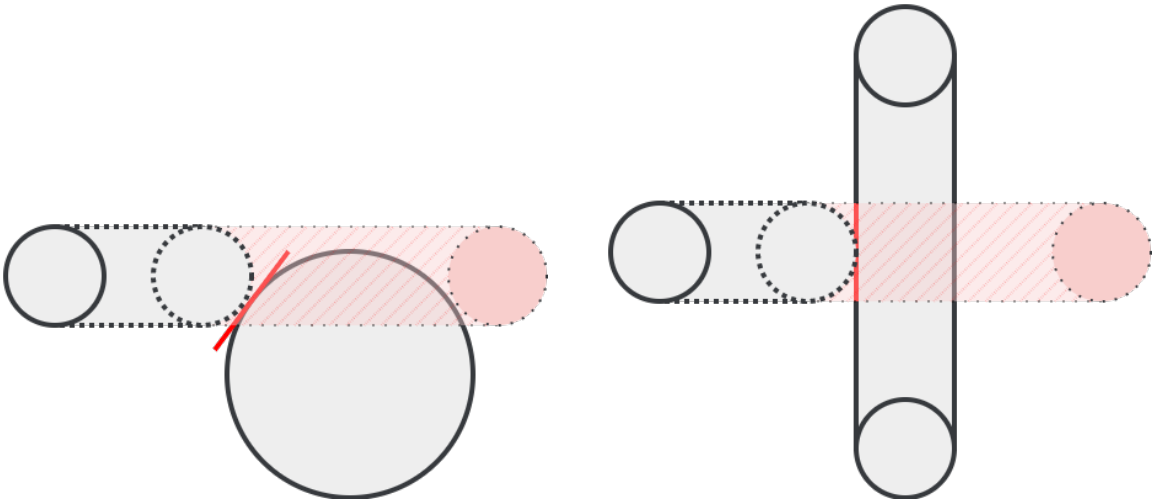
At the same time, numerous real-world phenomena require much more complex dynamics for the modeled beings: shape and size, collisions, momentum, acceleration, etc. The main goal of this part of the research was to propose a method of simulation that would allow all these mechanisms to function properly, while maintaining the high scalability of the method presented in Chapter 5.

The proposed solution preserves the discrete grid structure of the environment but adds a precise location in the continuous coordinate system to each agent. A cell is also allowed to contain multiple agents, but their positions and shapes determine how many agents can fit in the cell. The behavior of an agent, similarly to all other presented examples, is governed by the signal — each agent follows the direction of the sum of signal vectors that are visible in a cell. This signal is usually either a positive attracting signal of some objective or a negative repelling signal of other agents.

Each agent is represented as a circle of a predefined size, with its center determined by the location of the agent. The area of this circle also serves as the mass of the agent. The agent also stores the current velocity vector, which determines the position the agent will move to after the next iteration, if allowed to move freely. The movement of an agent reproduces inertia, i.e. an agent is limited in the change of its direction and must gradually accelerate to reach its regular movement speed.

The movement of an agent intending to follow a given direction is performed in several steps, which result in the creation of a plan and its resolution:

- Velocity change:
  - Agent selects a direction of movement.



(a) Partial movement due to the body of another agent.

(b) Partial movement due to the trail of another agent.

Figure 8.18: Partial movement due to movement resolution. Source: [30]

  - Agent calculates the change in velocity based on its intended direction and its mass.
  - Agent adds the velocity change to its current velocity.
  - Agent scans the area around and adjusts the direction velocity to avoid collisions with walls (adjacent obstacle cells) and other agents.
  - The resulting vector is scaled to fit within acceptable limits if necessary.
- Movement resolution:
  - If another agent resolved its movement and now is in the way of the current agent, the movement is shortened (*partial movement*) so that the agent stops before contact.
  - If another agent resolved its movement and its movement crossed the intended movement of the current agent, the movement is shortened (*partial movement*) so that the agent stops before contact.
  - The agent is placed in the location resulting from its *partial movement*.

Figure 8.18 shows examples of the situations leading to the partial movement. The red line marks the collision, and the light red shape shows the canceled part of the intended motion.

Several experiments were conducted to observe the phenomena occurring in the crowds created by the agents moving toward a common goal. However, the most important experiment involved the plotting of the *fundamental diagrams*. The fundamental diagrams represent the relation between speed and density, speed and flow (speed · density), and flow and density. These relations describe the statistical properties of individual people in a moving crowd. Some phenomena are expected to always appear: the maximal speed is achieved for the lowest density and lowers considerably, until no movement is observed at all. The specific density of the crowd varies, but some examples can be seen in Figure 8.19.

The data collected during the experiments was used to create the fundamental diagrams. After converting the values from cell size units and iterations to meters and seconds, the results have been plotted as in Figure 8.20. In each case, the density is determined by counting the number of agents occupying a set of cells chosen for observation, therefore, only specific density values are possible. Despite this limitation, the diagrams seem to follow the same trends as the ones created from the real-world data. In each case, the maximal speed is approximately $1.5\frac{m}{s}$ and drops to values close to zero in the density range of 4–$6\frac{agent}{m^2}$. The simulation involved a relatively small

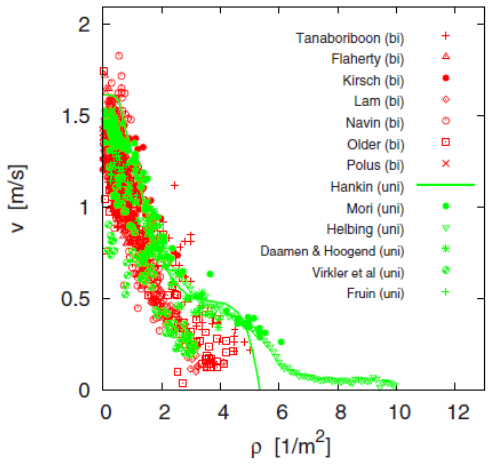
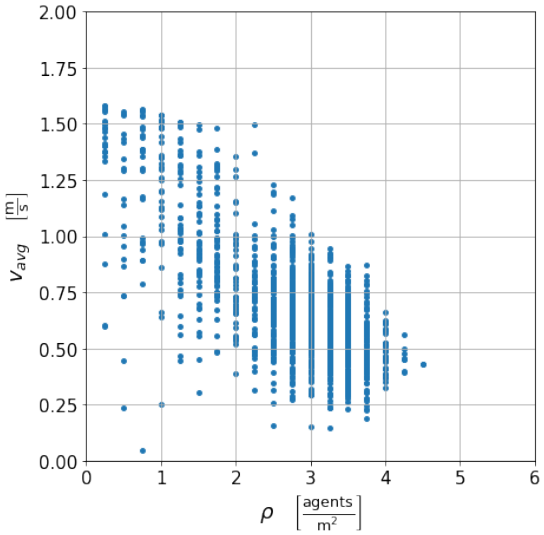


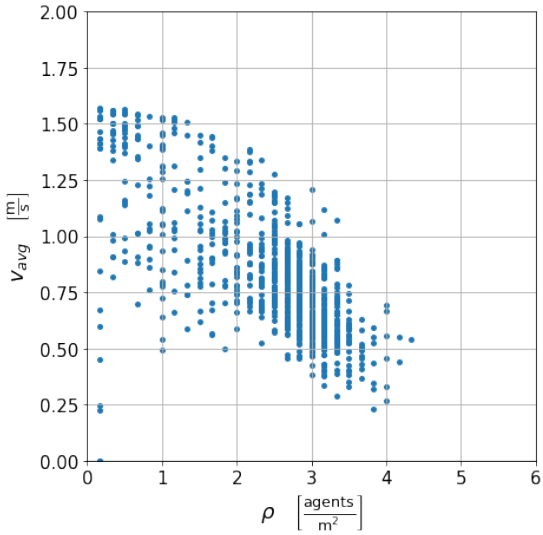
Figure 8.19: Velocity-density fundamental diagram for several studies. Source: [76]

crowd of people, therefore, the density did not reach values much higher than $4\frac{agent}{m^2}$, and for the widest corridor it only reached approximately $2.5\frac{agent}{m^2}$. Nonetheless, the collected data suggests that the behavior of the agents is realistic in terms of slowing down the movement of crowds at the points of highest density.

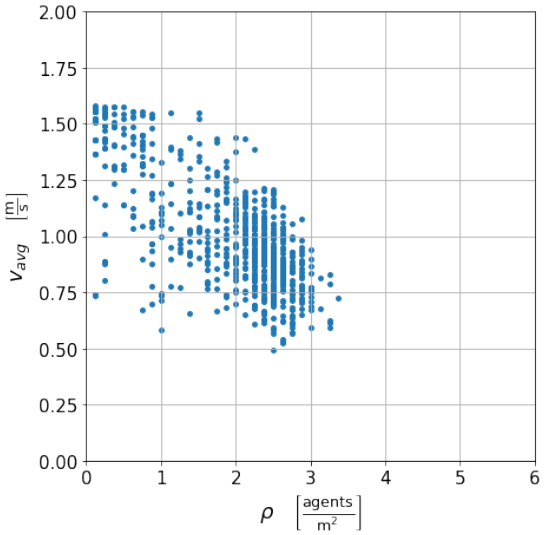
The proposed method of simulating agents in continuous space with a geometric approach to body collisions allows for the occurrence of phenomena that are impossible to obtain in a discrete grid. The resolution of simultaneous moves ensures that there are no collisions and properly changes the dynamics of the agents. At the same time, the underlying schema of synchronization and environment observation ensures good scalability, as well as eliminates any anomalies that could occur at the borders of computing workers otherwise. Further improvements in the behavior of the agents, their motivations, irregularities in movement, and reactions to collisions can make it applicable to the most complex models of social phenomena.



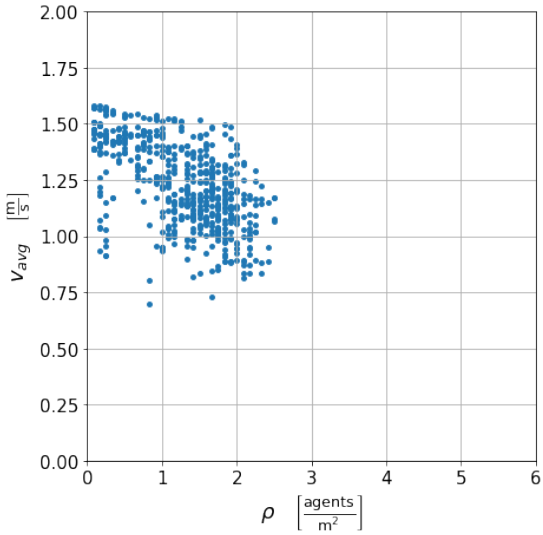
(a) Fundamental diagram for corridor 2m in width.



(b) Fundamental diagram for corridor 3m in width.



(c) Fundamental diagram for corridor 4m in width.



(d) Fundamental diagram for corridor 6m in width.

Figure 8.20: Velocity-density fundamental diagrams measured in simulation. Source: [30]

9 Conclusions

In the research presented in this thesis, the problem of parallelization and distribution of discrete spatial ABMS was considered. This problem is commonly addressed on a case-by-case basis, which essentially requires a large effort on the part of the simulation model creator to carefully analyze the possible parallelization schemes, communication between processes, data flow, synchronization, etc. The conclusions are often similar to some degree between models, which sparked research towards the creation of a common approach that would allow the model designer to focus solely on model-specific structures and algorithms. The existing solutions proved to be lacking in some aspects, especially their applicability and the impact of the method on the behaviors observed in the simulated environment. A new solution has been proposed that maintains high scalability while addressing the highlighted problems present in other works.

9.1. Summary of contributions

The main goal set for the presented research was the creation of a distributed simulation method capable of preserving the behaviors expected in a wide variety of models of social phenomena, regardless of the scale of the distribution applied, while effectively utilizing the assigned resources. This goal led to three separate contributions, which will be presented below in order of their significance. The combined outcome of this work forms a comprehensive method that fully addresses and satisfies all the goals.

Creation of the scalable simulation method. A definition of the required model structure and simulation algorithm for the new simulation method constitutes a major part of the presented research. The presented approach was designed with a focus on the scalability of the resulting system. Models representing real-world scenarios created with the use of this method were also presented, with very promising results. In each case, the results obtained from the simulation were similar to the ones gathered in the real scenario, with crucial phenomena accurately represented in the model. The method was also shown to be very extensible, as it was possible to adapt continuous models to it, despite the reliance on discrete structures. The scalability of the system was also tested, yielding excellent results — the method was able to efficiently utilize all resources in all tests. This method serves as the main contribution of this work.

Definition of simulation correctness. To better understand the risks related to the distribution of the simulations, research was carried out that led to the formulation of *simulation method correctness*. The problems initially identified in the existing solutions did not provide a full view of the desired characteristics of the simulation method. Based on research and experience, the definition was proposed, along with a scheme of verification whether a given method possesses this quality. The proposed simulation method was also verified using this method, confirming that the correctness has been preserved. The definition of correctness and its verification serve as a secondary contribution of this work.

Addressing the limitations of the core method. The method that served as a reference for the creation of the proposed solution was significantly limited with respect to the structure of the represented environment. This limitation was removed by changing the representation of the environment. This modification relieved the creator of the model from the burden of designing the environment in terms of the grid, while preserving the ability to do so for backward compatibility. Additionally, it opened the modified method to a wider variety of models that would otherwise be unable to benefit from this method. This changed representation was then used as a baseline for the new simulation method. The modifications to the original method and the guidelines regarding the process of applying such modifications to similar systems serve as a tertiary contribution of this work.

9.2. Further research directions

The models presented as successful applications of the method yielded very promising results. However, the models were based on simplistic approximations of social beings as their main actors. In general, the phenomena regarding their statistical behaviors were matching the real-world data in a satisfactory way. However, the strength of agent-based simulations lies partially in the ability to represent unique individuals. This level of precision appears to be firmly within the capabilities of the presented solution, but further improvements require interdisciplinary cooperation on a larger scale — the understanding of motivation behind the behavior of a being, especially a human being, requires expertise in sociology, psychology, and probably many other areas of research. Therefore, further extension of human-related scenarios is a very promising but also challenging direction for further research.

At the same time, there are aspects of the simulation distribution that were not considered in this research. A major factor that needs to be accounted for is the division of the simulation environment in more complex circumstances: heterogeneous resources, uneven workload in the simulated environment, or even dynamical changes in the distribution of the workload within environment. Therefore, two closely related research topics are relevant as a continuation of this work: initial environment division and dynamic workload balancing. The first topic could result in the creation of a tool capable of dividing the environment that would be aware of the structure of the computing resources and the expected computational cost of cells in the environment. The second topic could result in extending the system with a tool capable of initiating exchanges of parts of the environment between workers, while maintaining a constant low size of the worker neighborhood and other crucial characteristics of the environment division. Both directions would greatly improve the performance of the system using the proposed simulation method with heterogeneous resources.

A Test cases

The following chapters: Chapter 6, Chapter 7 and Chapter 8, analyze the method presented in Chapter 5 in terms of goals set in Chapter 1. The first two goals — correctness and scalability — require a sample model to be implemented using this method and then simulated in order to test the method in action. Therefore, this appendix describes three models that were originally created for similar purposes and presented in [10] and later in a publication [50] that is part of the research presented in this thesis.

The three models exhibit some important differences from each other that allow for better coverage of the performance of the method under different circumstances. The main differences in the context of this research are the following:

- The granularity of the environment — whether a cell can be occupied by at most one agent or by multiple agents.
- The types of actions — whether the model allows only migration or more complex actions, including affecting other agents.
- The complexity of the environment — whether the environment serves as a static containment for agents or is changing over time or according to the actions of agents.

Each of the test models is inspired by some real-world phenomenon, although the adherence to biological or physical correctness is not paramount. The models are not supported by real data, but the crucial characteristics and behaviors that are expected can be observed — the faithfulness to the modeled scenario is qualitative, but not quantitative.

Each model is given an explanation of the represented scenario, followed by:

- description of possible cell states, including the states of agents if applicable,
- description of possible actions,
- list of parameters and their meanings,
- list of metrics collected as a way of tracking the progress of the simulation, each iteration generating a separate set of values,
- a sample visualization of the state of simulation.

A.1. Rabbits and lettuce

The first sample model attempts to capture the classic predator-prey relationship [4] of two species. This type of model is often used to showcase the ability of the simulation system to successfully maintain the very sensitive

balance between populations of both types of species, which should follow the periodic oscillations described by the Lotka-Volterra equations, initially conceptualized in the field of chemistry [42] and later observed in the biological context [43]. The key features needed for such a simulation to display this behavior are the following:

- The availability of food for the prey species is always abundant and sufficient for the entire population.
- The availability of food for the predator species depends on the prey population size (i.e. predators feed on prey).
- The predator species has a limitless appetite.
- The population size changes proportionally to its current size.
- The evolution or changes in the environment have no effect on the interaction between the species.

If these features are satisfied, the changes in the sizes of both populations should follow this pattern:

- Increase in the population size of prey causes a delayed increase in the population size of predators, since the amount of food is capable of sustaining a greater number of individuals.
- Increase in the population size of predators causes a delayed decrease in the population size of prey, as the number of consumed prey is increasing.
- Decrease in the population size of prey causes a delayed decrease in the population size of predators, as competition for food causes starvation.
- Decrease in the population size of predators causes an increase in the population size of prey, as the number of consumed prey is decreasing.

This pattern loops indefinitely. Therefore, the graph of population sizes in time for both species should present two repeating oscillations with similar period, but with an offset phase. The real-world inspiration for this specific model was taken from rabbits consuming lettuce.

Cell and agent states. Each cell can be in one of the following states:

- Obstacle — stops the spreading of the signal.
- Empty — allows the spreading of the signal.
- Occupied by a lettuce agent — this agent has no additional state important for the simulation logic.
- Occupied by a rabbit agent — this agent possesses a single parameter that denotes its energy, which measures the ”healthiness” of the rabbit and is used extensively in determining its behavior.

Actions. Both types of agents can perform some actions, however, the possible actions are different for both types.

Lettuce agent is immobile, but is able to periodically attempt reproduction, which is simplified to placing a new lettuce agent in one of the adjacent empty cells.

At the same time, the behavior of rabbits is more complex and depends on the amount of energy possessed by a rabbit agent:

- If a rabbit agent has sufficiently high energy, it can attempt to reproduce. This action is simplified as well, leading to reproduction more akin to cell division, as the rabbit creates a new rabbit agent in an adjacent empty cell with some amount of energy, while also losing some amount of its own energy.

– If a rabbit has zero energy, it will die and be removed from the environment.

– Otherwise, a rabbit agent will move to an adjacent empty cell or adjacent cell with the lettuce agent, or idly stay in the original cell.

Each action is also accompanied by a loss of some energy, as a result of continuing its existence. If a rabbit moves into a cell which contains a lettuce, the lettuce is replaced by the rabbit, and the energy of the rabbit is increased due to the consumption. The direction of rabbit movement is governed by the strength of the signal, i.e. the rabbit will always prefer the direction with the strongest signal value. Attracting signal is generated by lettuce, while the rabbits generate a repelling signal.

The original implementation of the model tried to deal with the problem of conflicts in the following ways:

– If the conflict was caused by two lettuce agents creating offspring in the same cell, only one lettuce agent was created, resulting in one of the agents effectively not performing a reproduction. This behavior fitted the model, as the lettuce agent surrounded by occupied cells is not able to reproduce as well.

– If the conflict was caused by a rabbit agent and a lettuce agent appearing in the same cell, the rabbit agent was treated as if it entered the cell with lettuce, effectively allowing it to consume the lettuce.

– If the conflict was caused by two rabbit agents, there was no possibility of using existing model rules for the resolution of this conflict. As a result, the conflict led to cannibalism, in which only one rabbit remained in a cell, with the energy value being the sum of the energy values of conflicting rabbits.

Parameters. The model is configurable by a number of parameters. Different configurations allow to observe different phenomena, ranging from the desired Lotka-Volterra oscillations, through domination of one species over the other due to the high survivability of one compared to another, complete extinction of predator species, to the extinction of prey species quickly followed by the predator species. The full set of parameters is following:

RSV — rabbit signal value, the strength of signal generated by a rabbit agent,

LSV — lettuce signal value, the strength of signal generated by a lettuce agent,

RSE — rabbit starting energy, the initial value of energy possessed by a rabbit agent,

RRC — rabbit reproduction cost, the energy lost by a rabbit agent due to reproduction,

RRT — rabbit reproduction threshold, the minimal energy required for a rabbit agent to perform reproduction,

RLC — rabbit life cost, the energy lost by a rabbit agent due to continuing its existence,

LRF — lettuce reproduction frequency, the number of iterations between lettuce reproductions,

LEC — lettuce energetic capacity, the amount of energy gained by a rabbit agent by consuming lettuce.

Metrics.

– rabbit count — the number of rabbit agents present in the environment,

– rabbit reproductions count — the number of rabbit reproductions,

– rabbit total energy — the total energy of all rabbit agents present in the environment,

– rabbit deaths — the number of rabbit agents deaths,

– rabbit total lifespan — the cumulative lifespan of rabbit agents that died in this iteration,

– lettuce count — the number of lettuce agents present in the environment,

– consumed lettuce count — the number of lettuce agents consumed,

– lettuce total lifespan — the cumulative lifespan of lettuce agents consumed in this iteration,

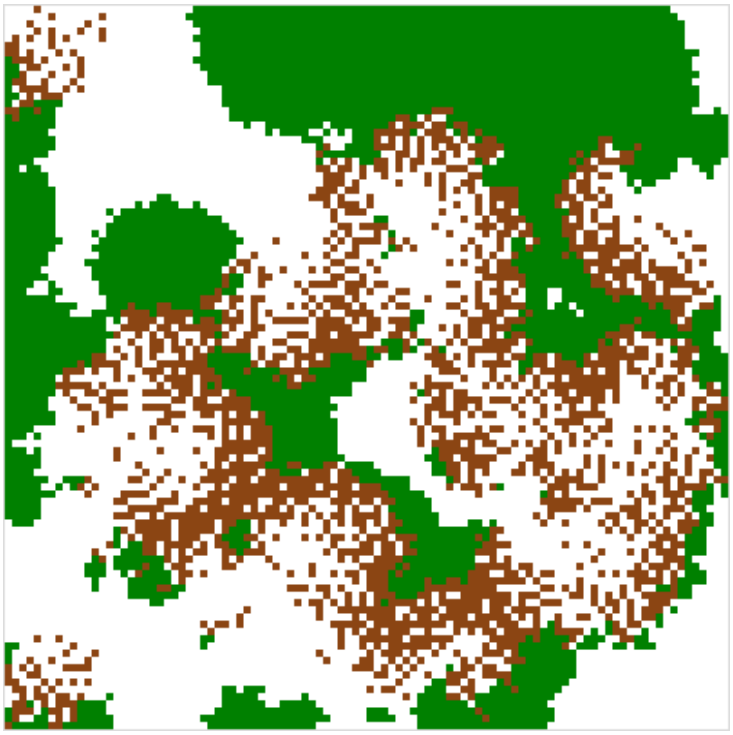


Figure A.1: Sample state of rabbits and lettuce model simulation.

Visualization. Figure A.1 presents a sample visualization of a state of the simulation of this model. White pixels represent empty cells, green pixels represent lettuce agents, and brown pixels represent rabbit agents.

A.2. Foraminifera habitat

The second sample model is a simplified simulation of single-cell benthic and pelagic eukaryotes called *foraminifera*. The choice of this specific type of creature is not accidental, as their biology is subject to extensive research. During their life cycle, they produce mineral shells that are easily fossilizable, which makes it possible to research the places they occupied and the paleoenvironmental conditions under which they existed [9]. The shells are removed in the process of reproduction and accumulate on the sea floor, which enables scientists to assess the state of the environment and the conditions in which the individual existed. This makes them a very suitable subject of research for paleoreconstructions and testing of general evolutionary hypotheses [40, 53, 17].

The attempts to model foraminifera started to appear in the literature years ago [3] and were developed for a long time, with new approaches appearing over time, such as in [65, 67]. Taking inspiration from those works, the created sample model simplifies complex biological processes to the rules outlined below, which are similar to the ones used in rabbits and lettuce simulation.

Cell and agent states. Each cell contains any number of foraminifera agents, as well as some amount of algae represented by a single real number. It is worth noting that the cell here represents a larger area that is capable of containing multiple foraminifera changes, without distinguishing their precise location within the area. Each foraminifera agent possesses some energy, similar to the rabbit agents in the previous model.

Actions. This model uses two types of actions: one related to a foraminifera agent, the other related to the changing environment.

In each iteration, algae grow in each cell at a rate proportional to the amount of algae already present in the cell.

The foraminifera agents can perform several different actions, depending on the energy possessed:

- Foraminifera agents with sufficiently high energy can reproduce, creating a new foraminifera agent in the same cell.
- Foraminifera agents with zero energy die and are removed from the environment.
- Foraminifera agents consume some amount of algae present in the cell, if the amount of algae is sufficiently high.
- Otherwise, foraminifera agents can move to an adjacent cell, following the signal emitted by algae, which is proportional to the amount of algae in the cell.

As in the rabbits and lettuce model, the foraminifera agents lose some amount of energy in each iteration as a constant cost of continuing life processes. The original resolution of conflicts in this model is trivial — as the model allows multiple agents to coexist in a single cell, no two decisions can generate conflict.

Parameters.

FSV — foraminifera signal value, the strength of signal generated by a foraminifera agent,

ASM — algae signal multiplier, the strength of the signal generated by a unit of algae amount in a cell,

FSE — foraminifera starting energy, the initial value of energy possessed by a foraminifera agent,

FRC — foraminifera reproduction cost, the energy lost by a foraminifera agent due to reproduction,

FRT — foraminifera reproduction threshold, the minimal energy required for a foraminifera agent to perform reproduction,

FLC — foraminifera life cost, the energy lost by a foraminifera agent due to continuing its existence,

ASA — algae starting amount, the initial amount of algae in a cell,

ARR — algae regeneration rate, the amount of algae that grows in a cell as a fraction of existing amount of algae in that cell,

AEC — algae energetic capacity, the amount of algae needed in a cell for a foraminifera agent to feed on algae, and the amount of energy gained.

Metrics.

- foraminifera count — the number of foraminifera agents in the environment,
- foraminifera reproduction count — the number of foraminifera reproductions,
- foraminifera total energy — the total energy of all foraminifera agents present in the environment,
- foraminifera deaths — the number of foraminifera agents deaths,
- foraminifera total lifespan — the cumulative lifespan of foraminifera agents that died in this iteration,
- foraminifera moves — the number of foraminifera agent migrations,
- algae amount — the amount of algae present in the environment,
- consumed algae amount — the amount of algae consumed.

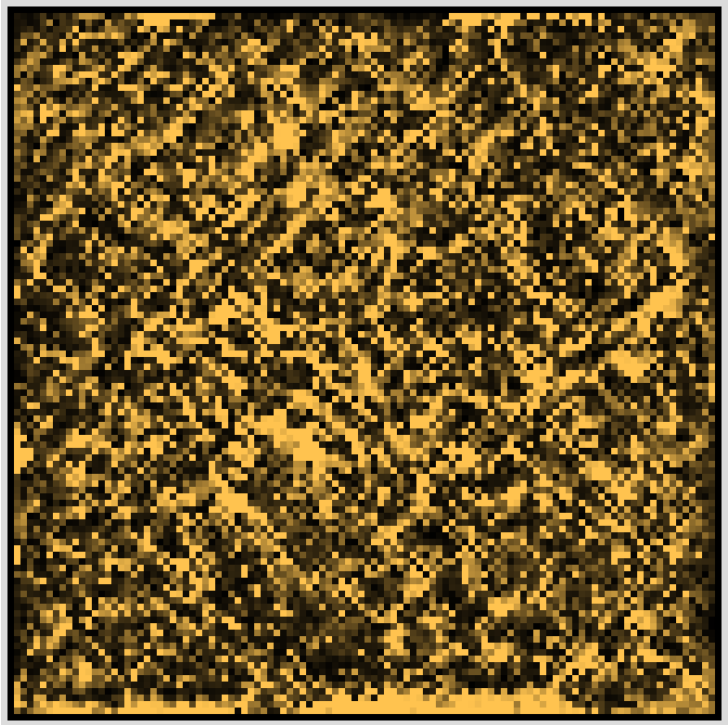


Figure A.2: Sample state of foraminifera habitat model simulation. Source: [10]

Visualization. Figure A.2 presents a sample visualization of a state of the simulation of this model. Each pixel ranges from black to bright yellow, representing the number (density) of foraminifera agents in the represented cell.

A.3. Fire emergency evacuation

The final test model is a simulation of fire emergency evacuation. The importance of this type of simulation is obvious, especially nowadays, when the process of planning the building architecture is legally required to include a proper analysis of the evacuation routes. It applies to all kinds of buildings designed for use by people —

sports objects such as stadiums, amphitheaters, shopping centers, office buildings, apartments, etc. — as well as a multitude of possible emergency situations: fires, floods, acts of terrorism, and other dangers.

Modeling human behavior is a challenging task on its own. Multiple approaches based on different ideas exist in the literature. One of the most popular are the Social Force Model [27] and Social Distances Model [69], which operate in a continuous space and represent the motivation of individuals as a set of repelling and compelling forces, not unlike the interaction between molecules. Less computationally complex solutions exist that use discretized space [7, 14], which is very useful in creating such a simulation using the tested method. The final form of the test model borrows heavily from the work presented in [74]. The sight radius of a human agent originally presented in the model translates very well into the signal, while a discrete environment with agents occupying cells is possible to be directly ported into the grid- or graph-based environment.

Cells and agent states. Each cell can be in one of the following states:

- Obstacle — stops the spreading of the signal.
- Empty — allows the spreading of the signal.
- Exit — allows the spreading of the signal, serves as a destination for human agents.
- Occupied by a fire agent — this agent has no additional state important for the simulation logic.
- Occupied by a human agent — this agent possesses a single integer parameter that represents its speed (or, more precisely, its slowness).

Actions. Obstacles, empty cells, and exit cells are stationary. Fire agents behave similarly to lettuce in the first test model, i.e. once every defined number of iterations, they attempt to spread by placing a new fire agent in an adjacent cell. This agent can be placed in an empty cell, but is allowed to overwrite the existing exit cell or the human agent, rendering the exit unusable or causing death to the person, respectively. Finally, the human agent can move to an adjacent empty cell, a cell occupied by fire, or an exit cell once every several iterations, depending on its speed. As a result, the agent can change its location, die in the fire, or successfully escape the danger by exiting the room or building.

The behavior of human agents is as simple as following the most compelling signal. The compelling signal is emitted by exit cells, while the repelling signal is generated by fire to allow avoiding it, and by other humans to avoid overcrowding a single closest exit. The resolution of conflicts in this model unfortunately introduces phenomena that are not present in the model when executed sequentially. While the spread of fire is trivial, overwriting any previous cell contents and always leading to the fire being present after the conflict resolution, the migration of human agents is more problematic. Whenever a conflict between two such agents occurs, the agents form temporary *crowd*, occupying the same cell. On the first instance of a movement opportunity, the agents will try to split into separate cells again.

Parameters.

HMS — human max speed, maximal value of the speed assigned to a human agent,

FSF — fire spreading frequency, delay between consecutive attempts to spread fire,

HSV — human signal value, the strength of the signal generated by a human agent,

FSV — fire signal value, the strength of the signal generated by a fire agent,

ESV — exit signal value, the strength of the signal generated by an exit cell.

Metrics.

- human count — the number of human agents in the environment,
- fire count — the number of fire agents in the environment,
- exit count — the number of exit cells in the environment,
- human deaths — the number of human agents that were caught by fire,
- human escapes — the number of human agents that reached an exit cell.

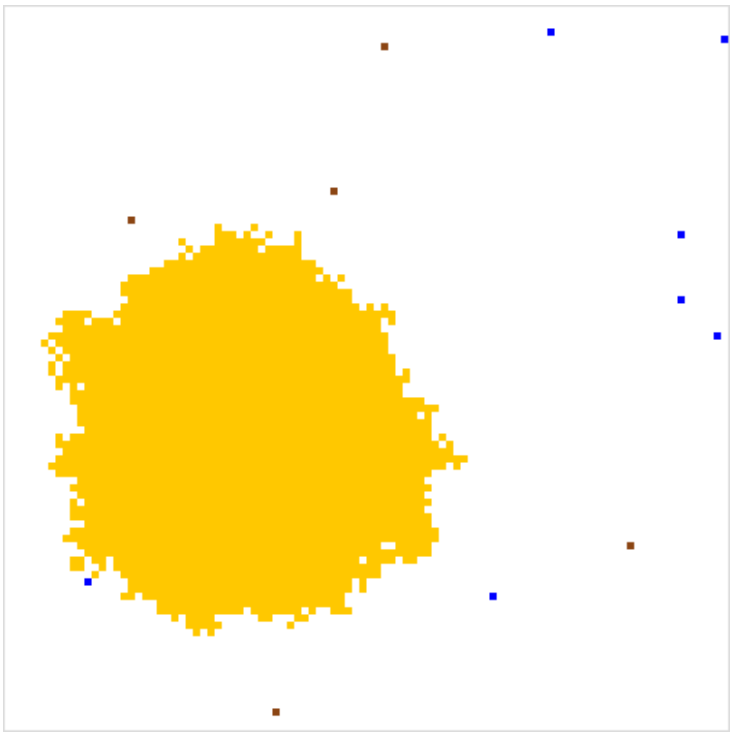


Figure A.3: Sample state of fire emergency evacuation model simulation

Visualization. Figure A.3 presents a sample visualization of the state of the simulation of this model. White pixels represent empty space, blue pixels represent human agents, brown pixels represent doors, and orange pixels represent fire agents.

List of Figures

List of Tables

Bibliography

[1] G. M. Amdahl. Validity of the single processor approach to achieving large scale computing capabilities. In *AFIPS '67 (Spring): Proceedings of the April 18-20, 1967, Spring Joint Computer Conference*, page 483–485, 1967.

[2] S. C. Bankes. Agent-based modeling: A revolution? *Proceedings of the National Academy of Sciences*, 99(suppl 3):7199–7200, 2002.

[3] W. H. Berger. Planktonic foraminifera: basic morphology and ecologic implications. *Journal of paleontology*, pages 1369–1383, 1969.

[4] A. A. Berryman. The orgins and evolution of predator-prey theory. *Ecology*, 73(5):1530–1535, 1992.

[5] M. Bezbradica, M. Crane, and H. J. Ruskin. Parallelisation strategies for large scale cellular automata frameworks in pharmaceutical modelling. In *2012 Int. Conf. on High Performance Computing & Simulation (HPCS)*, pages 223–230. IEEE, 2012.

[6] A. Białkiewicz, B. Stelmach, and M. J. Żychowska. Dobra kultury współczesnej. zarys problemu ochrony. *Wiadomości Konserwatorskie*, 2020.

[7] V. J. Blue and J. L. Adler. Cellular automata microsimulation of bidirectional pedestrian flows. *Transportation Research Record*, 1678(1):135–141, 1999.

[8] C. Bowzer, B. Phan, K. Cohen, and M. Fukuda. Collision-free agent migration in spatial simulation. In *Proceedings of 11th Joint Agent-oriented Workshops in Synergy (JAWS 2017), Prague, Czech*, 2017.

[9] M. D. Brasier and H. Armstrong. *Microfossils*. G. Allen & Unwin London, 1980.

[10] J. Bujas, D. Dworak, W. Turek, and A. Byrski. High-performance computing framework with desynchronized information propagation for large-scale simulations. *Journal of Computational Science*, 32:70–86, 2019.

[11] S. Coakley, M. Gheorghe, M. Holcombe, S. Chin, D. Worth, and C. Greenough. Exploitation of high performance computing in the FLAME agent-based simulation framework. In *2012 IEEE 14th International Conference on High Performance Computing and Communication & 2012 IEEE 9th International Conference on Embedded Software and Systems*, pages 538–545. IEEE, 2012.

[12] N. Collier and M. North. Parallel agent-based simulation with Repast for high performance computing. *Simulation*, 89(10):1215–1235, 2013.

[13] S. De Marchi and S. E. Page. Agent-based models. *Annual Review of political science*, 17:1–20, 2014.

[14] J. Dijkstra, J. Jessurun, H. J. Timmermans, et al. A multi-agent cellular automata model of pedestrian movement. *Pedestrian and evacuation dynamics*, 173:173–180, 2001.

[15] M. Dymnicka et al. *Przestrzeń publiczna a przemiany miasta*. Wydawnictwo Naukowe Scholar Sp. z o.o., 2013.

[16] C. Engelmann and A. Geist. Super-scalable algorithms for computing on 100,000 processors. In *International Conference on Computational Science*, pages 313–321. Springer, 2005.

[17] T. H. Ezard, T. Aze, P. N. Pearson, and A. Purvis. Interplay between changing climate and species' ecology drives macroevolutionary dynamics. *science*, 332(6027):349–351, 2011.

[18] M. J. Flynn. Very high-speed computing systems. *Proceedings of the IEEE*, 54(12):1901–1909, 1966.

[19] M. Gardner. The fantastic combinations of John Conway's new solitaire game "life". *Sc. Am.*, 223:20–123, 1970.

[20] J. Gehl and B. Svarre. *How to study public life*, volume 2. Springer, 2013.

[21] A. Giordano, A. De Rango, D. D'Ambrosio, R. Rongo, and W. Spataro. Strategies for parallel execution of cellular automata in distributed memory architectures. In *2019 27th Euromicro International Conference on Parallel, Distributed and Network-Based Processing (PDP)*, pages 406–413. IEEE, 2019.

[22] GUS. Demographic yearbook of Poland, 2019.

[23] J. L. Gustafson. Reevaluating Amdahl's law. *Communications of the ACM*, 31(5):532–533, 1988.

[24] S. Gwynne and D. Boswell. Pre-evacuation data collected from a mid-rise evacuation exercise. *Journal of fire protection engineering*, 19(1):5–29, 2009.

[25] M. Gyurkovich and A. Sotoca. Towards the Cracow Metropolis–a dream or a reality? a selected issues. *Technical Transactions*, 115(2):5–25, 2018.

[26] E. T. Hall. *The Silent Language*. Garden City, New York, 1959.

[27] D. Helbing and P. Molnar. Social force model for pedestrian dynamics. *Physical review E*, 51(5):4282, 1995.

[28] J. H. Holland and J. H. Miller. Artificial adaptive agents in economic theory. *The American economic review*, 81(2):365–370, 1991.

[29] A. Jasiński. Public space or safe space–remarks during the COVID-19 pandemic. *Technical Transactions*, 117(1), 2020.

[30] K. Kądzielawa. Scalable simulation of autonomous entities geometry and kinematics in a continuous space using the signal propagation method. Master's thesis, AGH University of Science and Technology, 2021.

[31] A. A. Kantarek, K. Kwiatkowski, and I. Samuels. From rural plots to urban superblocks. *Urban Morphology: Journal of the International Seminar on Urban Form*, 22(2):155–157, 2018.

[32] T. Kapecki. Elements of sustainable development in the context of the environmental and financial crisis and the covid-19 pandemic. *Sustainability*, 12(15):6188, 2020.

[33] A. H. Karp and H. P. Flatt. Measuring parallel processor performance. *Communications of the ACM*, 33(5):539–543, 1990.

[34] A. Kłusek, P. Topa, J. Wąs, and R. Lubaś. An implementation of the Social Distances Model using multi-GPU systems. *The International Journal of High Performance Computing Applications*, 32(4):482–495, 2018.

[35] K. Kollman, J. H. Miller, and S. E. Page. Adaptive parties in spatial elections. *American Political Science Review*, 86(4):929–937, 1992.

[36] W. Kosiński. Paradigm of the city of the 21st century. between the past of the polis and the future of the metropolis, 2016.

[37] M. Kowicki. *Rozproszenie zabudowy na obszarach Małopolski, a kryzys kreatywności opracowań planistyczno-przestrzennych*. Wydawnictwo PK, Kraków, 2014.

[38] J. L. Kriken, P. J. Enquist, and R. Rapaport. *City building: Nine planning principles for the twenty-first century*. Princeton Architectural Press, 2010.

[39] W. H. Kruskal and W. A. Wallis. Use of ranks in one-criterion variance analysis. *Journal of the American Statistical Association*, 47(260):583–621, 1952.

[40] D. Lazarus, H. Hilbrecht, C. Spencer-Cervato, and H. Thierstein. Sympatric speciation and phyletic change in globorotalia truncatulinoides. *Paleobiology*, 21(1):28–51, 1995.

[41] P. Lorens. *Równoważenie rozwoju przestrzennego miast polskich*. Politechnika Gdańska, 2013.

[42] A. J. Lotka. Contribution to the theory of periodic reactions. *The Journal of Physical Chemistry*, 14(3):271–274, 2002.

[43] A. Lotka-Volterra. Elements of physical biology, 1925.

[44] T. Michalski. Continuous space model for scalable simulation using the signal propagation method. Master's thesis, AGH University of Science and Technology, 2021.

[45] C. Moler. Matrix computation on distributed memory multiprocessors. *Hypercube multiprocessors 1986(A 87-37501 16-61). Philadelphia, PA, Society for Industrial and Applied Mathematics, 1986,*, pages 181–195, 1986.

[46] G. E. Moore et al. Cramming more components onto integrated circuits, 1965.

[47] M. Najdek, H. Xie, and W. Turek. Scaling simulation of continuous urban traffic model for high performance computing system. In *International Conference on Computational Science*, pages 256–263. Springer, 2021.

[48] H. S. Nwana. Software agents: An overview. *The knowledge engineering review*, 11(3):205–244, 1996.

[49] M. Paciorek, A. Bogacz, and W. Turek. Scalable signal-based simulation of autonomous beings in complex environments. In *International Conference on Computational Science*, pages 144–157. Springer, 2020.

[50] M. Paciorek, J. Bujas, D. Dworak, W. Turek, and A. Byrski. Validation of signal propagation modeling for highly scalable simulations. *Concurrency and Computation: Practice and Experience*, 33(14):e5718, 2021.

[51] M. Paciorek, D. Poklewski-Koziełł, K. Racoń-Leja, A. Byrski, M. Gyurkovich, and W. Turek. Microscopic simulation of pedestrian traffic in urban environment under epidemic conditions. *Bulletin of the Polish Academy of Sciences: Technical Sciences*, 69(4):e137725, 2021.

[52] M. Paciorek and W. Turek. Agent-based modeling of social phenomena for high performance distributed simulations. In *International Conference on Computational Science*, pages 412–425. Springer, 2021.

[53] P. N. Pearson, N. Shackleton, and M. Hall. Stable isotopic evidence for the sympatric divergence of Globigerinoides trilobus and Orbulina universa (planktonic foraminifera). *Journal of the Geological Society*, 154(2):295–302, 1997.

[54] S. F. Railsback and V. Grimm. *Agent-based and individual-based modeling: a practical introduction*. Princeton university press, 2019.

[55] K. Ramamohanarao, H. Xie, L. Kulik, S. Karunasekera, E. Tanin, R. Zhang, and E. B. Khunayn. Smarts: Scalable microscopic adaptive road traffic simulator. *ACM Transactions on Intelligent Systems and Technology (TIST)*, 8(2):1–22, 2016.

[56] P. Renc, M. Bielech, T. Pęcak, P. Morawiecki, M. Paciorek, W. Turek, A. Byrski, and J. Wąs. HPC large-scale pedestrian simulation based on proxemics rules. In *International Conference on Parallel Processing and Applied Mathematics*, pages 489–499. Springer, 2019.

[57] E. Rewers. *Post-polis: wstęp do filozofii ponowoczesnego miasta*, volume 41. Towarzystwo Autorów i Wydawców Prac Naukowych" Universitas", 2005.

[58] S. Ristov, R. Prodan, M. Gusev, and K. Skala. Superlinear speedup in HPC systems: Why and when? In *2016 Federated Conference on Computer Science and Information Systems (FedCSIS)*, pages 889–898. IEEE, 2016.

[59] J. F. Rose. *The well-tempered city: what modern science, ancient civilizations, and human nature teach us about the future of urban life*. Harper Wave, 2016.

[60] E. Schrödinger. Die gegenwärtige situation in der quantenmechanik. *Naturwissenschaften*, 23(48):807–812, 1935.

[61] Y. Shoham. Agent oriented programming. Technical report STAN-CS-90-1335. *Computer Science Department, Stanford University*, 1990.

[62] T. Sośnicki, W. Turek, K. Cetnarowicz, and M. Żabińska. Dynamic assignment of tasks to mobile robots in presence of obstacles. In *2013 18th International Conference on Methods & Models in Automation & Robotics (MMAR)*, pages 538–543. IEEE, 2013.

[63] M. Starzec, G. Starzec, and M. Paciorek. Desynchronization of simulation and optimization algorithms in HPC environment. *Computer Science*, 21(3), 2020.

[64] J. Teller. Urban density and Covid-19: towards an adaptive approach. *Buildings and Cities*, 2(1):150–165, 2021.

[65] P. Topa and J. Tyszka. Local minimization paradigm in numerical modelling of foraminiferal shells. In *International Conference on Computational Science*, pages 97–106. Springer, 2002.

[66] W. Turek. Erlang-based desynchronized urban traffic simulation for high-performance computing systems. *Future Generation Computer Systems*, 79:645–652, 2018.

[67] J. Tyszka and P. Topa. A new approach to modeling of foraminiferal shells. *Paleobiology*, 31(3):522–537, 2005.

[68] A. Varas, M. Cornejo, D. Mainemer, B. Toledo, J. Rogan, V. Munoz, and J. Valdivia. Cellular automaton model for evacuation process with obstacles. *Physica A: Statistical Mechanics and its Applications*, 382(2):631–642, 2007.

[69] J. Wąs, B. Gudowski, and P. J. Matuszyk. Social distances model of pedestrian dynamics. In *International Conference on Cellular Automata*, pages 492–501. Springer, 2006.

[70] J. Wąs and R. Lubaś. Towards realistic and effective agent-based models of crowd dynamics. *Neurocomputing*, 146:199–209, 2014.

[71] W. H. Whyte et al. *The social life of small urban spaces*. Conservation Foundation Washington, DC, 1980.

[72] M. Wooldridge. *An introduction to multiagent systems*. John wiley & sons, 2009.

[73] C. Xia, H. Wang, A. Zhang, and W. Zhang. A high-performance cellular automata model for urban simulation based on vectorization and parallel computing technology. *International J. of Geographical Information Science*, 32(2):399–424, 2018.

[74] L. Yang, W. Fang, R. Huang, and Z. Deng. Occupant evacuation model based on cellular automata in fire. *Chinese Science Bulletin*, 47(17):1484–1488, 2002.

[75] M. Żabińska, T. Sośnicki, W. Turek, and K. Cetnarowicz. Robot task allocation using signal propagation model. *Procedia Computer Science*, 18:1505–1514, 2013.

[76] J. Zhang and A. Seyfried. Empirical characteristics of different types of pedestrian streams. *Procedia engineering*, 62:655–662, 2013.