On social simulation in 4D relativistic spacetime

Lai Kwun Hang

Centre for Science and Technology Studies (CWTS), Leiden University,
k.h.lai@cwts.leidenuniv.nl

Abstract. Agent-based modeling (ABM) and simulation allow us to study social phenomena in hypothetical scenarios. If we stretch our imagination, one of the interesting scenarios would be our interstellar future. To model an interstellar society, we need to consider relativistic physics, which is not straightforward to implement in existing ABM frameworks. In this paper, we present the mathematics and algorithmic details needed for simulating ABM in 4D relativistic spacetime. These algorithms form the basis of our open-source computational framework, "Relativitization" [1].

Keywords: special relativity, interstellar society, agent-based simulation, computational framework

1 Introduction

The scientific and technological advancement in the last century greatly increases our understanding of the universe. Nowadays, we are able to build giant telescopes and observe astronomical objects billions of light years away. Apart from deepening our scientific understanding, our astronomical knowledge also stimulates our imagination of interstellar civilizations. A lot of great science fiction has been written, and scientists have proposed ideas like the Fermi paradox [2], the Dyson sphere [3] and the Kardashev scale [4]. While many of these ideas are physically plausible, it would be interesting to discuss these ideas from a social science perspective. Due to the highly hypothetical nature of the problem, we suggest that agent-based model (ABM) enables formal academic discussion on interstellar society.

An ABM is a simulation model bridging microscopic behaviours of agents and macroscopic observations. Depending on the context of the model, an agent can be an individual, an organization, or even a country. First, assumption about the social behaviours of the agents are made, then the agents are placed and evolved in a computational environment. For a research problem for which it is not viable to perform data collection and analysis, ABM can still be used for theoretical exposition [5].

To model agents in interstellar space, suppose we only consider a scale with normal stellar objects so that we can ignore the effects of general relativity, such as universe expansion and black holes, but we still have to consider the effect of special relativity. In the context of ABM, where the simulation is computed

under a set of inertial frames that are at rest to each other, we can simplify the theory of relativity into two core phenomena: speed of light as the upper bound of the speed of information travel, and time dilation relative to any stationary observer in the inertial frames. This also implies that we have to take care of four dimensions: three space dimensions, plus one time dimension.

Typically, an ABM is constructed using an ABM framework to facilitate model development and communication. There are a lot of existing ABM frameworks, to name a few, NetLogo [6], mesa [7], and Agents.jl [8], see [9] for a detailed review. While it is possible to build a 4D relativistic model in some existing ABM frameworks, those frameworks do not have native support for the necessary 4D data structures, and it can be error-prone to enforce relativistic effects via custom implementations of data structures and algorithms. Therefore, we have developed a simulation framework we call "Relativitization" [1], to help social scientists to build an ABM in relativistic spacetime. In this paper, the mathematics and the algorithms underlying the framework will be presented.

2 Definitions

In Relativitization, an agent is called a "player". Players live in a universe. Ideally, computation should be done in every local frame following all players, and the computation results can be synchronized by a Lorentz transformation. However, this will make the framework and the model substantially more complex. Therefore, all computations are done according to some inertial frames that are at rest to each other. The spatial coordinates of a player are represented by floating-point numbers x, y and z and the time coordinate of a player is represented by a floating-point number t. To simplify computation and visualization, the universe is partitioned into unit cubes. A player with floating-point coordinates (t, x, y, z) is located at the cube with integer coordinates $T = \lfloor t \rfloor$, $X = \lfloor x \rfloor$, $Y = \lfloor y \rfloor$, $Z = \lfloor z \rfloor$, note that the computations of a simulation are done at unit time steps and we can actually assume $T = t$. Denote the speed of light as c. In vector notation, define $\mathbf{s} = (t, \overrightarrow{u}) = (t, x, y, z)$, and $\mathbf{S} = (T, \overrightarrow{U}) = (T, X, Y, Z)$.

2.1 Interval and time delay

The spacetime interval between coordinates $\mathbf{s}_i$ and $\mathbf{s}_j$ is

$$\|\mathbf{s}_i - \mathbf{s}_j\| = c^2(t_i - t_j)^2 - (x_i - x_j)^2 - (y_i - y_j)^2 - (z_i - z_j)^2. \tag{1}$$

If $\|\mathbf{s}_i - \mathbf{s}_j\| < 0$, it is called a spacelike interval, and events that happen at the two coordinates are not causally connected because no information can travel faster than the speed of light c.

It is often needed to compute intervals in integer coordinates. We define the spatial distance between $\overrightarrow{U_i}$ and $\overrightarrow{U_j}$ as the maximum distance between all points in the cubes at $\overrightarrow{U_i}$ and $\overrightarrow{U_j}$

$$|\overrightarrow{U_i} - \overrightarrow{U_j}| = (X_i - X_j + 1)^2 + (Y_i - Y_j + 1)^2 + (Z_i - Z_j + 1)^2. \tag{2}$$

Suppose there is a signal sent from $\overrightarrow{U_i}$ to $\overrightarrow{U_j}$. To ensure that the information travels slower than the speed of light, the integer time delay $\tau(\overrightarrow{U_i}, \overrightarrow{U_j})$ is computed as

$$\tau(\overrightarrow{U_i}, \overrightarrow{U_j}) = \left\lceil \frac{|\overrightarrow{U_i} - \overrightarrow{U_j}|}{c} \right\rceil. \tag{3}$$

2.2 Group id

From Eq. 2, even if $\overrightarrow{U_i} = \overrightarrow{U_j}$, the time delay is non-zero. To implement zero time delay for players that are really close to each other, we divide a unit cube into several sub-cubes with edge length d_e, and information travel within the same sub-cubes is instantaneous.

To improve the computational speed when checking whether two players belong to the same sub-cube, we assign a "group id" to each sub-cube in a unit cube. A unit cube has n_e^3 sub-cubes, where $n_e = \left\lceil \frac{1}{d_e} \right\rceil$. For a player at $\overrightarrow{u}$, it belongs to the (n_x, n_y, n_z) sub-cubes, where $n_x = \left\lfloor \frac{x-X}{d_e} \right\rfloor$, $n_y = \left\lfloor \frac{y-Y}{d_e} \right\rfloor$, and $n_z = \left\lfloor \frac{z-Z}{d_e} \right\rfloor$. The group id $g(\overrightarrow{u}, \overrightarrow{U})$ of the player can be computed as

$$g(\overrightarrow{u}, \overrightarrow{U}) = n_x n_e^2 + n_y n_e + n_z. \tag{4}$$

If two players have the same integer coordinates and the same group id, then we say the players belong the same group and the time delays between the players are zero.

2.3 Player data

A player is characterized by a set of data:

- player id i,
- integer coordinates (T_i, X_i, Y_i, Z_i),
- a historical record of integer coordinates $H_i = \{(T_i', X_i', Y_i', Z_i') \mid T_i' < T_i\}$,
- floating-point coordinates (t_i, x_i, y_i, z_i),
- time dilation counter μ_i, a floating point number to keep track of time dilation (see Sec. 2.6 and Sec. 3.3),
- group id g_i,
- floating-point velocities $\overrightarrow{v_i} = (v_{ix}, v_{iy}, v_{iz})$,
- other data D_i relevant to the model.

2.4 Command

In other frameworks, interactions in ABMs are often presented as one player asking another player to do something. Because the speed of information travel is bounded by c, a player cannot simply ask other players to do something immediately. Instead, interactions are mediated by commands. Whenever player i wants to interact with player j, player i sends a command to j.

A command is characterized by:

- i_{to}, the id of the player to receive this command,
- i_{from} the id of the target player who sent this command,
- $\mathbf{S}_{\text{from}}$ the integer coordinates when the player sent this command,
- f_{target} a function to modify data of the target player when this is received.

Commands travel at the speed of light c. The amount of time needed for a command to reach the target, measured in the inertial frames we used in the simulation, depends on the trajectory of the target player i_{to} and the sender coordinates $\mathbf{S}_{\text{from}}$,

2.5 Universe data

Universe is an overarching structure which aggregates all necessary data and functionalities. An universe has:

- a current universe time T_{current},
- a 4-dimension array of maps from player id to lists of player data M_{TXYZ}, so that the data of a player residing at (T, X, Y, Z) is stored in the associated list, the "afterimages" of players are also stored in the corresponding list (Sec 3.5),
- a map M_{command} from player id to lists of commands, such that a command in the list will be executed when the player receive the command,
- other universe global data D_G relevant to the model,

2.6 Mechanism and AI

Given an instance of a universe, the dynamics of players are based on predefined rules and the state of the universe observed by the players. In our framework, we call the rules mechanisms. A mechanism takes the state of the universe observed by a player, modifies the state of a player, and generates a list of commands to send to other players.

To ease the model development to account for the time dilation effect, we further divide mechanisms into two categories: regular mechanisms and dilated mechanisms. A regular mechanism is executed once per turn, while a dilated mechanism is executed once per multiple turns, adjusted for the time dilation of the player measured in the inertial frames we used in the simulation.

3 Simulation step

The following are needed to define a model:

- the data structure of other player data D_i,
- the data structure of other universe global data D_G,
- a set of available commands,
- a function to initialize the universe data,
- a function to update the universe global data,

– a set of regular and dilated mechanisms,

Along with the universe data, it is useful to define a map M_{current} from a player id to the current player data, i.e., $T_i = T_{\text{current}}$, as an internal object of the simulation. The modifications of player data are first performed on M_{current}, and then synchronized back to the universe data at appropriate timing.

Suppose we have initialized an universe model and M_{current}, a complete step in a simulation involves:

1. update the global data (Sec. 3.1),
2. compute time dilation effects for all players (Sec. 3.2),
3. process mechanisms for each player (Sec. 3.3),
4. process the command map (Sec. 3.4),
5. move players, add afterimages, and update time (Sec. 3.5).

The simulation can be ran for a fixed amount of steps, or stop when a stopping condition is met.

3.1 Update global data

A model may rely on a mutable global data D_G to implement the dynamics. If the model depends on some player data to update D_G, and the effect is observable by players, we need to ensure that no information is transferred faster than the speed of light via the global data update.

For example, if the global data is modified if "all" player data satisfy a condition, we have to be careful about what we mean by "all" here. In the universe, the maximum time delay equals $\tau_{\max} = \tau((0,0,0),(max(X),max(Y),max(Z)))$. To fulfill the speed of light constraint, the update function has to check whether all player data in M_{TXYZ}, where $T_{\text{current}} - \tau_{\max} \leq T \leq T_{\text{current}}$, $0 \leq X \leq max(X)$, $0 \leq Y \leq max(Y)$, and $0 \leq Z \leq max(Z)$, satisfy that condition.

3.2 Compute time dilation

Relative to a stationary observer j in an inertial frame, special relativity predicts that a moving observer i experiences a time dilation effect:

$$\gamma_i = \frac{1}{\sqrt{1 - \frac{v_i^2}{c^2}}}, \tag{5}$$

$$\Delta t_i = \frac{\Delta t_j}{\gamma_i}, \tag{6}$$

where γ_i is called the Lorentz factor.

To account for the time-dilation effect, the time dilation counter μ_i is updated by Algorithm 1 every turn for every player. μ_i will then affect the mechanism processing in Sec. 3.3.

Algorithm 1: Update time dilation counter.

Input: M_{current}, map from player id to current player data
foreach player i in M_{current} **do**
  $\mu_i \leftarrow \mu_i + \sqrt{1 - \frac{v_i^2}{c^2}}$;
  if $\mu_i \geq 0$ **then**
    $\mu_i \leftarrow \mu_i - 1$;
  end
end

3.3 Process mechanisms

Before processing any mechanism for a player, we need to compute the state of the universe viewed by the player. At an instance in our discretized relativistic universe, player i sees other players located at the unit cubes closest to the surface of the past light cone of player i, while the entire cubes are still within the past light cone. The computation consists of two steps: (1) Algorithm 2 computes the view centered at a specific cube, ignoring the zero time delay when players are within the same group, (2) Algorithm 3 computes the view for players in a group. Assuming each line of these algorithms takes $O(1)$ and iterating over all (X, Y, Z), the time complexity $O(mn)$ from Algorithm 2 dominates, where $m = X_{\max} Y_{\max} Z_{\max}$ is the spatial size of the universe, and n is the number of player.

The view of the universe of a player is used by mechanisms to update the player data and generate commands to send. Regular mechanisms update the data of the player each turn, while dilated mechanisms update the player if the time dilation counter is greater than zero to account for the time dilation effect. The generated commands are executed immediately if the target player is within the same group of the sender, otherwise the commands are stored in M_{command}. Algorithm 4 shows the overall iterative process.

3.4 Process command map

The command map M_{command} is a map from player id to lists of commands that is being sent to that player. At each turn, the distance between the target player and the sent positions of all commands in the list are calculated, and the command is executed on the player if the spacetime interval is larger than zero. Algorithm 5 illustrates the process.

3.5 Move players and add afterimages

Moving players and storing their data requires additional considerations in this simulation framework. Consider the following example:

1. assume player i and player j are located in the same cube,

Algorithm 2: Compute the view of the universe at a cube, ignore the zero time delay when player are in the same group

Input:
T_i, X_i, Y_i, Z_i position of the viewing location
M_{TXYZ} 4D array of maps from player id to lists of player data
Output:
M map from player id to player data
Λ_{XYZ} 3D array of maps from group id to lists of player id

Initialize empty M and Λ_{XYZ};
foreach X_j, Y_j, Z_j **do**
  $T_j \leftarrow T_i - \tau(\overrightarrow{U_i}, (X_j, Y_j, Z_j))$;
  foreach player data in $M_{T_j X_j Y_j Z_j}[k]$ **do**
    if M has key k **then**
      if T of $M[k] < T$ of the new player data **then**
        Replace $M[k]$ by this new player data;
      end
    end
    else
      Store data of player k to $M[k]$;
    end
  end
end
Associate the player id from M to the corresponding list in Λ_{XYZ} by spatial coordinates and group id;
return (M, Λ_{XYZ})

Algorithm 3: Compute the view of the universe for players in a group.

Input:
g_i group id
T_j, X_j, Y_j, Z_j position of the viewing location
M map from player id to player data
Λ_{XYZ} 3D array of maps from group id to lists of player id
M_{TXYZ} 4D array of maps from player id to lists of player data
Output:
M' map from player id to player data
Λ'_{XYZ} 3D array of maps from group id to lists of player id

$M' \leftarrow M$;
$\Lambda'_{XYZ} \leftarrow \Lambda_{XYZ}$;
foreach player data in $M_{T_j X_j Y_j Z_j}[k]$ where $g(\overrightarrow{u_k}, \overrightarrow{U_k}) = g_i$ **do**
  if T of $M[k] < T$ of the new player data **then**
    Replace $M'[k]$ by this new player data;
    Update the corresponding position of player k in Λ'_{XYZ};
  end
end
return (M', Λ'_{XYZ})

Algorithm 4: Update all players by mechanisms.

Input:
M_{current} map from player id to current player data
Universe data

foreach (X_j, Y_j, Z_j) **do**
  Compute the view of the universe at this cube by algorithm 2;
  foreach group in this cube **do**
    Compute the view of the universe at this group by algorithm 3;
    foreach data of player k in this group **do**
      Update $M_{\text{current}}[k]$ by all regular mechanisms;
      if $\mu_k \geq 0$ **then**
        Update $M_{\text{current}}[k]$ by all dilated mechanisms;
      end
    end
    foreach generated command where target player l is in this group **do**
      Update $M_{\text{current}}[l]$ by f_{target} of the command;
    end
  end
end
Add the rest of commands to M_{command} by the target player id of the commands;

Algorithm 5: Process command map.

Input:
M_{current} map from player id to current player data
M_{command} map from player id to lists of commands

foreach key i in M_{command} **do**
  Get the integer coordinates $\mathbf{S}_i$ of player i from $M_{\text{current}}[i]$;
  foreach command C in the list $M_{\text{command}}[i]$ **do**
    if $\|\mathbf{S}_{\text{from}} - \mathbf{S}_i\| \geq 0$ **then**
      Update $M_{\text{current}}[i]$ by f_{target} of the command;
    end
  end
end
Remove all executed commands;

2. player j moves to the other cube,
3. the new information takes time to travel to player i, so player i cannot see the new position of player j,
4. player i cannot see the old information of player j either, because player j is no longer there,
5. player j disappears from the sight of player i.

This "disappearance" is caused by the problem of the integer-based coordinates used in the computation of player's 3D view.

Consider a more generic situation: suppose player i is located at $\overrightarrow{U_i}$, and player j moves from $\overrightarrow{U_j}$ to $\overrightarrow{U_k}$. Ignoring the possibility of zero time delay, the maximum time player i has to wait to see player j is bounded by Eq. 7,

$$\begin{aligned}
\Delta T &= \tau(\overrightarrow{U_i}, \overrightarrow{U_j}) - \tau(\overrightarrow{U_i}, \overrightarrow{U_k}), && (7)\\
&= \left\lceil \frac{|\overrightarrow{U_i} - \overrightarrow{U_j}|}{c} \right\rceil - \left\lceil \frac{|\overrightarrow{U_i} - \overrightarrow{U_k}|}{c} \right\rceil, && (8)\\
&\leq \left\lceil \frac{|\overrightarrow{U_i} - \overrightarrow{U_j}|}{c} - \frac{|\overrightarrow{U_i} - \overrightarrow{U_k}|}{c} \right\rceil, && (9)\\
&\leq \left\lceil \frac{|\overrightarrow{U_j} - \overrightarrow{U_k}|}{c} \right\rceil, && (10)\\
&= \tau(\overrightarrow{U_j}, \overrightarrow{U_k}). && (11)
\end{aligned}$$

Therefore, if we include back the possibility where the time delay between player i and player j can be zero, the maximum duration of the disappearance produced by the movement is bounded by $\Delta T_{\max} = \tau((0,0,0),(1,1,1))$. To prevent the unrealistic disappearance from happening, the old player data has to stay at the original position for at least $\Delta T_{\max}$ turn, we call this the "afterimage" of the player. Note that afterimages only participate in the 3D view of players, they should not be updated by commands or mechanisms.

Algorithm 6 does multiple things: it updates the universe time, it moves players by their velocities, it synchronizes time of players, it stores old coordinates to the history of player, it cleans the history if the stored coordinates is too old, and it adds the current player and afterimages to the latest spatial 3D array in the 4D data array M_{TXYZ}. Since the universe time has been updated, this simulation step has finished, the universe should go to the next step and loop over all algorithms in Sec. 3 again.

4 Discussion

The presented algorithms form the backbone of our computational framework, "Relativitization" [1]. There are technical subtleties that are not discussed here, such as creating new players, removing dead players, introducing randomness

Algorithm 6: Move player and add afterimages.

Input:
M_{current} map from player id to current player data
Universe data

$T_{\text{current}} \leftarrow T_{\text{current}} + 1$;
Initialize a 3D array of maps from player id to lists of player data M_{XYZ};
foreach data of player i in M_{current} **do**
  $t_i \leftarrow T_{\text{current}}$;
  $x_i \leftarrow x_i + v_{ix}$;
  $y_i \leftarrow y_i + v_{iy}$;
  $z_i \leftarrow z_i + v_{iz}$;
  $T_i \leftarrow T_{\text{current}}$;
  $X_i \leftarrow \lfloor x_i \rfloor$;
  $Y_i \leftarrow \lfloor y_i \rfloor$;
  $Z_i \leftarrow \lfloor z_i \rfloor$;
  $g_i \leftarrow g(\overrightarrow{u_i}, \overrightarrow{U_i})$ by Eq. 4;
  if coordinates or group is new **then**
    Save the previous coordinates to history H_i;
  end
  foreach (T_i', X_i', Y_i', Z_i') in H_i **do**
    Remove from H_i if $T_{\text{current}} - T' > \Delta T_{\max}$;
  end
  Save the new data to $M_{X_i Y_i Z_i}[i]$;
  foreach (T_i', X_i', Y_i', Z_i') in H_i **do**
    Find the old player data from $M_{T_i' X_i' Y_i' Z_i'}[i']$;
    Add the old player data to $M_{X_i' Y_i' Z_i'}[i']$;
  end
end
Drop the oldest 3D spatial array from M_{TXYZ};
Add M_{XYZ} as the latest spatial array to M_{TXYZ};

to models, parallelization of the algorithms, generating deterministic outcomes from parallelized simulations with random number generators, interactive human input to intervene in a simulation, etc. Nevertheless, the framework implements the major part of the technical subtleties, and provides a suitable interface to ease the development of any 4D, relativistic ABM.

It can be interesting to implement a classical ABM into the framework. Spatial ABMs with non-local interactions, such as the classical flocking model [10], are particularly suitable. These models are naturally affected by the time delay imposed by the speed of light limitation. Simulating such a model in the Relativitization framework allows us to explore the effects of time delay on the model.

Ultimately, existing ABMs might not be suitable to describe interstellar society. A solid understanding of social mechanisms and physics, together with some artistic imagination, are needed to build inspiring interstellar ABMs. As a first step, we have integrated a few social mechanisms to build a big "model", which is also a game. The "model" can be found on the GitHub[1] repository of our framework.

Apart from the possibility of implementing different models using the framework, the algorithms may also be optimized further. For example, the iteration in Sec. 3.3 has a time complexity of $O(mn)$. A naive alternative implementation to iterate over all the combinations of players could change the complexity to $O(n^2)$, which could have better performance when the density of players is low. We leave these potential improvements to future research.

5 Conclusion

In this paper, we have presented a set of algorithms to implement ABM simulations in a 4D, relativistic spacetime. Based on these algorithms, we have developed a simulation framework we call "Relativitization" [1]. Our framework will lower the barrier of entry for social scientists to apply their expertise to explore the interstellar future of human civilization. We hope our framework can be used to initiate meaningful and academically interesting discussions about our future.

Acknowledgement

We thank Diego Garlaschelli, Alexandru Babeanu, Michael Szell, and all QSS members of the CWTS institute for useful discussion.

References

1. Lai, K. H. *Relativitization* https://github.com/Adriankhl/relativitization. 2022. https://doi.org/10.5281/zenodo.6120765.

[1] https://github.com/Adriankhl/relativitization

2. Gray, R. H. The Fermi paradox is neither Fermi's nor a paradox. *Astrobiology* **15,** 195–199 (2015).
3. Wright, J. T. Dyson spheres. *arXiv preprint arXiv:2006.16734* (2020).
4. Gray, R. H. The Extended Kardashev Scale. *The Astronomical Journal* **159,** 228 (2020).
5. Edmonds, B. & Meyer, R. *Simulating social complexity* (Springer, 2015).
6. Wilensky, U. *NetLogo* http://ccl.northwestern.edu/netlogo/. 1999.
7. Kazil, J., Masad, D. & Crooks, A. *Utilizing Python for Agent-Based Modeling: The Mesa Framework* in *Social, Cultural, and Behavioral Modeling* (eds Thomson, R., Bisgin, H., Dancy, C., Hyder, A. & Hussain, M.) (Springer International Publishing, Cham, 2020), 308–317. ISBN: 978-3-030-61255-9.
8. Datseris, G., Vahdati, A. R. & DuBois, T. C. Agents.jl: a performant and feature-full agent-based modeling software of minimal code complexity. *SIMULATION* **0,** 00375497211068820. eprint: https://doi.org/10.1177/00375497211068820. https://doi.org/10.1177/00375497211068820 (2021).
9. Pal, C.-V., Leon, F., Paprzycki, M. & Ganzha, M. A review of platforms for the development of agent systems. *arXiv preprint arXiv:2007.08961* (2020).
10. Reynolds, C. W. *Flocks, herds and schools: A distributed behavioral model* in *Proceedings of the 14th annual conference on Computer graphics and interactive techniques* (1987), 25–34.