Global Climate Forum

Working Paper 3 / 2018

Distributed Agent Graph

A proposal for an ABM platform for distributed Global Systems Science simulations

Steffen Fürst[α*] · **Andreas Geiges**[α]

[α]Global Climate Forum

[*]E-mail: steffen.fuerst@globalclimateforum.org

Distributed Agent Graph

A proposal for an ABM platform for distributed Global Systems Science simulations

Steffen Fürst · Andreas Geiges

September 07, 2018

Abstract In the Global System Science (GSS) field simulations of Agent Based Models (ABM) can be challenging regarding the needed computer resources, so that it's necessary to run the simulation on computer clusters. The existing HPC ready ABM frameworks does not fit well to the structure of GSS models. This paper proposes a parallelization strategy that take GSS needs into consideration and tries to find a good balance between the needs of the modeler and restrictions that enables a good scalability. It represent the model structure as a directed multi-graph, where spatial information is added as nodes. Existing hypergraph partitioning software can then be used the distribute the agents to the different processing elements.

Keywords Agent-based modeling · Parallelization strategies · Global System Science · Graph partitioning · Communication schema

1 Introduction

Global Systems Science (GSS) is an emerging field that studies global systems, like the financial system, and tries to close the gap between science, policy and society. "The vision guiding GSS is to make full use of progress in ICT to improve the way scientific knowledge can stimulate, guide, be used by, and help evaluate policy and societal responses to global challenges like climate change, financial crisis, pandemics, global growth of cities."[1]

Models developed for GSS can be highly computationally expensive. For example, the current resolution of grid data from the Socioeconomic Data and Applications Center (SEDAC) for the world population is 43200 x 17400 cells (Socioeconomic Data and Applications Center (SEDAC) 2015). To store this data in a 2D array of a 32bit long data type, it is necessary to allocate more than 2,5 GB for a single attribute. And after removing all cells which are covered by water, there are about 1 billion cells left. So it is not feasible to run a simulation of this level on a normal desktop computer.

GSS models are often so called "Agent-Based Models" (ABM). This approach simulates the behavior of autonomous agents on a micro level and observes the emerging patterns of the macro level. This is computational more intensive then many other classical approaches like "System Dynamics", where aggregated variables are used for representing the underlying subjects.

This work has been supported by the CoeGSS project and has been partly funded by the European Commissions ICT activity of the H2020 Programme under grant agreement number: 676547. This paper expresses the opinions of the authors and not necessarily those of the European Commission. The European Commission is not liable for any use that may be made of the information contained in this paper.

[1] `http://global-systems-science.org/about-gss/`

There exist many platforms for developing ABM, like NetLogo[2], Repast[3], Swarm[4], MASON[5], just to name some important players in this field[6]. However, only a small subset of ABM platforms allow to parallelize a simulation, so that it is executed on multiple nodes of a computer cluster. In the cases that they support spatially explicit ABM out of the box, the load balancing is usually based on a rectangular decomposition of the spatial domain. Also there exists a "sphere of influence" in most cases, often with the restriction that the size of this sphere cannot be bigger then the dimension of the smallest rectangle in the decomposition (Cosenza et al (2011), Shook et al (2013), Rubio-Campillo (2014). This limits the granularity of the decomposition by not being flexible enough in the case that the agents are distributed unevenly in the spatial domain, e.g. when modeling the world population where many agents are living in megacities and no one in the oceans, or the classical flocks model (Reynolds 1987) where the uneven distribution of agent/birds is additionally highly dynamical. Further, this decomposition scheme may not be appropriate for models based on interaction networks in a globalized world, as the "sphere of influence" could then be the whole world for some individual agents.

So to be useful for implementing a GSS model, an ABM platform:

– should support GIS raster data (like the SEDAC data set mentioned above)
– allows agents to be assigned to a (discrete 2D) location
– does not cause imbalances because of an uneven distribution of the agents' location in this space
– should let agents store a variable number of traits and variables depending on the problem under consideration and/or the level of detail of the system description
– should provide a way to implement long distance communication, e.g. to implement advertisement or normative agents
– but should still have some optimized functions for short distance communication like: select randomly an agent in the radius of 3 km

This paper presents a proposal for a load-balancing method based on graph partitioning and MPI[7], which tries to find a balanced features set that is sufficiently enriched for the typical GSS models but has enough restrictions to allow the parallelization of a simulation. Section 2 gives the mathematical definition for a matching structure, which we call "Agent Graph" in the following. Section 3 describes the communication pattern between the processing elements (PEs) of the parallel simulation and therefore introduces the idea of "Ghost Agents". The GIS raster data itself is part of the graph, Section 4 describes two different integration approaches. For partitioning the graph, an existing graph partitioning tool can be used. In Section 5, we propose to use hypergraph partitioning for this task, in Section 6 we discuss the possibility to add weights to the edges. And finally, in Section 7 we give a short introduction to an implementation based on this proposal.

[2] https://ccl.northwestern.edu/netlogo/
[3] https://repast.github.io/index.html
[4] http://www.swarm.org/wiki/Swarm_main_page
[5] https://cs.gmu.edu/~eclab/projects/mason/
[6] See https://www.comses.net/resources/modeling-platforms/ for a good but still incomplete overview
[7] Message Passing Interface (MPI) is a standard used widely in the HPC world.for programming parallel computers.

2 The Distributed Agent Graph

2.1 Overview

In the following, an agent[8] is represented by a vertex. Vertices can have different types, e.g. household or location. The state of an agent can not be changed by another agent, and decisions of agents are based only on the previous state of other agents connected via vertices and the state of the agent itself.

Edges between the agents are directed, if a decision of agent a depends on the state of agent b, a edge from b to a is added to the graph. Edges have also different types. These types are representing the single agent decisions like `move` or `updateBeliefs`. The next time-step of a simulation is thereby composed by a sequence of agent decisions. As each of this decisions can change the agent graph, we call those decisions "transition functions" when discussing the properties of the agent graph. Transition functions can also involve the addition or deletion of edges, on the conditions that are described in Section 2.2.3.

To describe an ABM as a specialized graph is not the only possible approach. Especially spatially explicit ABMs often only construct a graph implicitly, if at all, like the flock model mentioned above construct implicitly a network of adjoining birds by calculating the distance between each pair of birds.

In the current form, the described approach is limited to time-stepped simulations of parallel dynamic systems(Laubenbacher et al 2008), as discrete event simulations and sequential dynamical systems would need a lot of small grained messages between the PEs. This means that mechanism like trading must be broken in substeps like `demandGoods, calculateDelivery` and `transferGoods`.

2.2 Mathematical definition

2.2.1 Agent Graph

Let $\varDelta_A$ be a set of agent types, $\varDelta_E$ be a set of edge types, and let $\varPi_{\delta_a}$ be the set of agent states for the agents of type δ_a. The set of Agent Graphs is defined by $\gamma \in \varGamma \Leftrightarrow \gamma =< \varPsi, \varPhi, \varXi, \varOmega >$, where $\varPsi$ is the set of vertices/agents, $\varPhi \subseteq \varDelta_E \times \varPsi \times \varPsi$ is the set of directed edges, $\varXi : \varPsi \rightarrow \varDelta_A$ is a typing function assigning a type to each agent, and $\varOmega : \varPsi \rightarrow \varPi_{\varXi(\psi)}$ is a valuing function assigning a state π to any vertex/agent ψ.

$\varPsi$ together with $\varPhi$ describes a directed multi-graph (multiple directed edges between a pair of vertices), we can split this graph into $|\varDelta_E|$ directed subgraphs (without multiple edges).

We will often use terms like $proj_\varOmega(\gamma)$. This means, that we use a "projection" of an agent graph γ, e.g. using $proj_\varOmega(\gamma)$ we are only interested in the valuing function. So the term $proj_\varOmega(\gamma)(\psi)$ gives the state of an agent ψ of the agent graph γ. In the same spirit we use $first, second$ and $third$ on edges ψ to map ψ to the edge type or one of the adjoint vertices.

2.2.2 Subsets of adjacent edges and agents

For a given agent ψ, we define some subsets of $\varPsi$ and $\varPhi$ as follows:

- $\varPhi^{\mathrm{to}}_{\delta_E,\psi}$ are subsets of $\varPhi$ that describe all the edges of type $\delta_E \in \varDelta_E$ that point to ψ: $\varPhi^{\mathrm{to}}_{\delta_E,\psi} = \{\phi \in \varPhi |\, first(\phi) = \delta_E \wedge third(\phi) = \psi\}$.
- Correspondingly $\varPhi^{\mathrm{from}}_{\delta_E,\psi} = \{\phi \in \varPhi |\, first(\phi) = \delta_E \wedge second(\phi) = \psi\}$.

[8] There is no universal definition of an "agent" in the context of agent-based models, see, e.g. Salamon (2011) and Grimm et al (2010). "Agents", here, include not only those software elements that represent acting entities in the real world (such as people, households, firms, etc.) but also, for example, locations.

- So we can construct the set of all adjacent edges of type $\delta_E \in \Delta_E$ for an agent ψ: $\Phi_{\delta_E,\psi} = \Phi^{\text{from}}_{\delta_E,\psi} \cup \Phi^{\text{to}}_{\delta_E,\psi}$.
- If we are only interested in the set of agents where edges of type $\delta_E \in \Delta_E$ point to/from a given agent ψ, we use $\Psi^{\text{to}}_{\delta_E,\psi} = third(\Phi^{\text{to}}_{\delta_E,\psi})$, correspondingly $\Psi^{\text{from}}_{\delta_E,\psi} = second(\Phi^{\text{from}}_{\delta_E,\psi})$.
- Finally, we have the set of all agents that are connected with a given agent ψ via edges of type $\delta_E \in \Delta_E$: $\Psi_{\delta_E,\psi} = \Psi^{\text{from}}_{\delta_E,\psi} \cup \Psi^{\text{to}}_{\delta_E,\psi}$.

2.2.3 Transition function

For each $\delta \in \Delta_A \times \Delta_E$ we have a transition function $f_\delta : (\Gamma, \Psi) \to \Gamma$, that constructs a new Agent Graph $\gamma^*_\psi = f_\delta(\gamma, \psi)$.[9]

For any given γ and ψ, the transition functions have the following restrictions:

$$\text{if } \Xi(\psi) \notin first(\delta) \Rightarrow \gamma^*_\psi = \gamma \tag{1}$$

$$\forall \hat{\gamma} \in \Gamma : \text{if } \forall \hat{\psi} \in \Psi^{\text{from}}_{\delta_E,\psi} : proj_\Omega(\gamma)(\hat{\psi}) = proj_\Omega(\hat{\gamma})(\hat{\psi}) \Rightarrow f_\delta(\gamma,\psi) = f_\delta(\hat{\gamma},\psi) \tag{2}$$

$$\forall \hat{\psi} \in proj_\Psi(\gamma) : \text{if } \hat{\psi} \in proj_\Psi(\gamma^*_\psi) \Rightarrow proj_\Xi(\gamma)(\hat{\psi}) = proj_\Xi(\gamma^*_\psi)(\hat{\psi}) \tag{3}$$

$$\forall \hat{\psi} \in proj_\Psi(\gamma) : \text{if } \psi \neq \hat{\psi} \Rightarrow proj_\Omega(\gamma)(\hat{\psi}) = proj_\Omega(\gamma^*_\psi)(\hat{\psi}) \tag{4}$$

$$\forall \hat{\psi} \in proj_\Psi(\gamma) : \text{if } \psi \neq \hat{\psi} \Rightarrow \hat{\psi} \in proj_\Psi(\gamma^*_\psi) \tag{5}$$

$$\forall \phi \in proj_\Phi(\gamma) : \text{if } \psi \notin proj_\Psi(\gamma^*_\psi) \wedge (second(\phi) = \psi \vee third(\phi) = \psi) \Rightarrow \phi \notin proj_\Phi(\gamma^*_\psi) \tag{6}$$

$$\forall \phi \in proj_\Phi(\gamma) : \text{if } \phi \notin proj_\Phi(\gamma^*_\psi) \Rightarrow second(\phi) \in \hat{\Psi} \wedge third(\phi) \in \hat{\Psi}; \hat{\Psi} = \Psi_{\delta_E,\psi} \cup \psi \tag{7}$$

$$\forall \phi \in proj_\Phi(\gamma^*_\psi) : \text{if } \phi \notin proj_\Phi(\gamma) \Rightarrow second(\phi) \in \hat{\Psi} \wedge third(\phi) \in \hat{\Psi}; \hat{\Psi} = \Psi_{\delta_E,\psi} \cup \psi \tag{8}$$

The equations have the following (sometimes trivial) meaning:

- (1): Only agents of a type that match the transition functions δ can change the graph γ, for all other agents the transition function is the identity function.
- (2): The transition function does not depend on the state of agents that the agent ψ cannot observe. The projection on Ω also implies that the transition function does not depend on properties of Ψ and Φ, like e.g. the density of the graph.
- (3): The type of an agent $\hat{\psi}$ is immutable.
- (4): An agent ψ cannot change the state of another agent $\hat{\psi}$.
- (5): An agent can only be removed from the graph by herself.
- (6): If an agent is removed, all adjacent edges must be removed too.
- (7): An agent ψ can only remove edges between vertices which are adjacent to ψ (or between himself and an adjacent vertex, e.g. for a different type δ_E or a different direction).
- (8): In the same way an agent ψ can only add edges between vertices which are adjacent to ψ (or between herself and an adjacent vertex).

Many restrictions (2, 5, 7, 8) are added to define and reduce the necessary communication in a parallel simulation, but can be also justified by the idea of autonomous decisions and limited information in ABMs.

[9] E.g. in a predator-prey model we have three agents types: *predator*, *prey* and *loccation*. When it comes to the movement of the agents, we have the three transition functions $f_{predator,move}$, $f_{pray,move}$ and $f_{location,move}$ where $f_{location,move}$ is the identity function.

3 Communication schema

3.1 Ghost agents

To avoid unnecessary communication between the PEs, we introduce ghost agents. In the case that an agent on one PE need access to the state of an agent on another PE, such a ghost agent is added to the first PE:

a) $\beta : \Psi \to P$ assigns each agent to a PE, where P is the set of PEs.
b) $\forall \hat{\psi} \in \Phi^{\text{from}}_{\delta_E,\psi}$ with $\beta(\hat{\psi}) \neq \beta(\psi)$, there exists a ghost agent $\mathring{\psi}_i$ on PE $i = \beta(\psi)$.
c) $\forall \hat{\psi} \in \Phi_{\delta_E,\psi} : \beta(\hat{\psi}) \in \Omega(\psi)$, each agent knows the PE of all adjacent agents.

There is the following correspondence between the ghost agents and the hyperedges [10] which are the result from the proposed hypergraph partitioning in Section 5: For each hyperedge that belongs to n subsets of the partition, we need $n-1$ ghost agents. The ghost agents are not part of the agent graph itself, but a result of the partitioning of the agent graph.

So the restriction to the transition function have the following background:

– (2) ensures that $\hat{\psi} \in \Psi^{\text{from}}_{\delta_E,\psi}$ in the case that $\hat{\psi}$ has an influence in the decision δ_E of agent ψ. In this case b) ensures that an ghost agent of $hat\psi$ exists on $\beta(\psi)$ in the case that $\beta(\psi) \neq \beta(\hat{\psi})$.
– (5) In the case that $\hat{\psi} \notin proj_\Psi(f_\delta(\gamma, \psi))$ the PE $\beta(\psi)$ must instruct PE $\beta(\hat{\psi})$ to remove the agent $\hat{\psi}$, but it possible that PE $\beta(\psi)$ can not resolve $\beta(\hat{\psi})$. Therefore the restriction forbid this case.
– (7) c) ensures that the PE of the vertices of the removed edges are known by the agent who removes the edge.
– (8) Is similar to (7).

3.2 A single simulation step

The transition functions described above are only an isolated piece of a single step, here is the proposal how to combine those functions to calculate a single step of a simulation in parallel, so that we have an overall transition from γ to γ^*. It is important to understand that each agent (and even each PE) knows only a subgraph. After the agents transition functions are called in parallel, the individual agents have individual graphs γ^*_ψ, so that e.g. it can be the case that $proj_\Omega(\gamma^*_\psi)(\psi) \neq proj_\Omega(\gamma^*_{\hat{\psi}})(\psi)$.

To calculate a step we:

– iterate serially over Δ_E (the order of this iteration must be determined by the model implementation). For each $\delta_E \in \Delta_E$ we do in parallel $\forall p \in P$:
  – Update the state of the ghost agents in $\Psi^{\text{to}}_{\delta_E}$.[11]
  – Barrier (Wait till all PEs have finished the step above)
  – Apply for each agent ψ the transition function $f_{(\Xi(\psi),\delta_E)}$. This will result into a different new agent graph γ^*_ψ for each agent.
  – Barrier
  – Each agent updates its internal state.
  – Each agent ψ removes herself from γ, if $\psi \notin proj_\Psi(\gamma^*_\psi)$.
  – Each agent ψ removes the edges from the set $proj_\Phi(\gamma) \setminus proj_\Phi(\gamma^*_\psi)$ from γ.
  – Each agent ψ add the agents from the set $proj_\Psi(\gamma^*_\psi) \setminus proj_\Psi(\gamma)$ to γ. We can now set $\Psi^* = proj_\Psi(\gamma)$ and $\Xi^* = proj_\Xi(\gamma)$, and we can also construct Ω^* as $\Omega^*(\psi) = proj_\Omega(\gamma^*_\psi)(\psi)$.

[10] A hyperedge is a subset of Ψ, which means that it connect an arbitrary number of agents.
[11] This can also imply the creation of new ghost agents

- Barrier[12]
- Each agent ψ add the edges from the set $proj_{\Phi}(\gamma^*_{\psi}) \setminus proj_{\Phi}(\gamma)$ to γ. We can have here the conflict that $\hat{\psi}$ does not exist anymore, in this case just don't add the edge. Now we have also $\Phi^* = proj_{\Phi}(\gamma)$, and in overall $\gamma^* =< \Psi^*, \Phi^*, \Xi^*, \Omega^* >$.

4 Raster support

As GSS models often use geographic data, a GSS framework should support the import of raster/grid data, like the world population data set from SEDAC mentioned in the Introduction. We are assuming in the following, that different data sources are all converted to rasters of the same size $n \times m$, and that spatial agents are assigned to locations via a function $loc : \Psi \rightarrow n \times m$.

We see two possible solutions to introduce this grid data to the Agent Graph. In the first solution, a subgraph is added to the agent graph, that represents agents which contains the grid data as there state. We call those agents environmental agents.

The second approach combines the usual rectangular decomposition of the rasters with the agent graph partitioning.

4.1 Solution A

We create an environmental agent for each non empty[13] cell of the raster. For each of the environmental cell we have an edge from and to each agent that "lives" on this cell. Additionally the environmental agents are connected with all other environmental agents in its "sphere of influence".

To construct the raster subgraph inside the agent graph, the following steps have to be done:

- Create an environmental agent for each non empty cell of the raster. The state of this agent contains all the raster values for this cell.
- Add edges between environmental agents if there distance is smaller then a given threshold δ.
- Each agent that has a spatial position adds an edge from and to the environmental agent with the same position.

Agents can move, but only within a range $\leq \delta$. If an agent a moves, the edges between a and the current environmental agent e_c are removed by a or e_c, and the current environmental agent also adds the new edges between a and the new environmental agent e_n. Due to limitation (8), the moving agent a can not add the new edges by herself, but e_c "knows" both a and e_n, as e_n is in the δ-range of e_c.

4.2 Solution B

An alternative approach would be to create a rectangular partitioning of the rasters, with the same number of partitions as for the Agent Graph. So we have beside the function β as defined in Section 3.1, also a function $\beta_r : R \rightarrow P$, where R is the set of rectangles.

In this approach the agent spatial locations is ignored in the graph partitioning, but as it is likely that in the model nearby agents are more often connected to the nearer neighbourhood, the partitioning result will still reflect some spatial information.

[12] This barrier resolve the conflicted of adding and removing the same edge at the same time by different agents. In the case that this happens, the conflict will be solved by keeping the edge.

[13] The definition of *empty* depends on the model. E.g. in the case that the non-environmental agents are only "living" on land, *empty* would be equal to all cells that are covered complete by water.

As usual for the rectangular decomposition approach, the border of the rectangles must be send to the neighbours. Additional for an Agent ψ with $\beta_r(loc(\psi)) \neq \beta(\psi)$, the state of the cell $loc(\psi)$ must be send to PE $\beta_r(loc(\psi))$. So β_r should be constructed in a way that minimize the necessary communication induced by the location PE vs. graph PE mismatch. Also, in line with the Agent Graph approach in general, agents can only read from but not write to the rasters.

5 Partitioning

Using Graph partitioning software for load balancing began to appear in the early 90s (Hendrickson 2000). The goal is to partition vertices so that each PE has roughly equal total vertex weight while minimizing the total weight of cutted edges, where the vertex weight represent the number of CPU instruction needed by the vertex (e.g. the decision functions of different agent type can have a different complexity), and the edge weight represent the amount of data to be send (e.g. the state of different agent can vary in size). But standard graph partitioning is not suited for the concept of ghost agents, as in the case that agent a on PE p_1 has edges to agent b and c on PE p_2, two edges would be cut, but it's only necessary to transmit the state of agent a once(Devine et al 2006).

This problem is solved by using hypergraph partitioning. We have then for each node an hyperedge which contains all vertices $\bigcup_{\delta_E \in \Delta_E} \Phi^{\text{to}}_{\delta_E,\psi} \cup \psi$. This graph is equivalent to a matrix of size $|\Psi| \times |\Psi|$, where $r_{ij} = 0 (i \neq j)$, if there is no directed edge from the ith node to the jth node. In the hypergraph partitioning problem a hyperedge is cut if it contains at least two vertices belonging to different partitions, so this problem matches the implementation with ghost agents as described in Section 3.1.

A possible library that supports hypergraph partitioning is Zoltan (Boman et al 2012), which also supports dynamic load-balancing, what could be utilized to improve the load-balancing at runtime by e.g. monitoring the amount of CPU-time used by the different agent types.

But there is also a problem with the vertex weight, which can not be solved with dynamic load-balancing alone, as a single time step is a multiple stages calculation[14] as described in Section 3.2. Assume the following number of CPU instructions for the trade example from Section 2.1:

$\delta_a \setminus \delta_e$	`demandGood`	`calculateDelivery`
Household	100	0
Firm	0	1000

If all Households would be assigned to PE p_1 and all Firms to PE p_2, p_2 would be completely idle in the `demandGood` transition and p_1 in the `calculateDelivery` transition. The same imbalance can also happen for the communication at different stages. Multi-objective graph partitioning algorithms exists, that extend the single weights of the vertices and edges to weight vectors (Schloegel et al 1999). A partition is then only considered balanced, if each of the components of the weight vector is balanced. One of the most recent development in this direction is PuLP (Slota et al 2014), an interface to Zoltan2 is available in the Trilios package[15].

6 Weighted edges

The edges in the Agent Graph structure are not weighted[16]. This ensure that edges can be added without creating a conflict, which can arise in the case for weighted edges, as one agent could create the edge with a different weight then a second one.

[14] We are assuming here, that we have more then one edge type.

[15] `https://fastmath-scidac.llnl.gov/software/pulpxtrapulp.html`

[16] The weights mentioned in the Partitioning Section above are added to improve the Partitioning, but are not part of the Agent Graph.

It is possible to represent weighted edges by adding the weight of the edge to the state of the agent of the source of the edge. This resolves the conflicts via the rule that only the agent on the source of an edge can set the weight.

A second solution would be to allow that for some edge types the edges could be weighted. Rule (8) must be then more strict and allow only the Agent on the tail of the edge to add/modify weighted edges:

$$\forall \phi \in proj_{\Phi}(\gamma^{*}_{\psi}) : \text{if } \phi \notin proj_{\Phi}(\gamma) \Rightarrow second(\phi) = \psi \tag{9}$$

Assuming that the transition functions for an weighted edge type only depends on the weight of the incoming edges and the state of the agent itself, it would be possible to skip the "update the complete state of ghost agents in $\Psi^{\text{to}}_{\delta_E}$" step in Section 3.2.

7 Implementation

We started to work on an Open Source ABM framework, which is based on the Agent Graph ideas. It is implemented in Python, using the mpi4py package for the MPI communication and the NumPy package for the agent states. The current implementation does not support unweighted edges but has weighted edges instead, where the weight can be an arbitrary tuple of NumPy Data Types, with the constraint that only the agent on the tail of the edge can add edges and set the weight (see also (9)). The edge information is only available for the agent, who has created the edge.

The elements from the Agent Graph are implemented in the following way:

- Spatial information is added by location agents, as described in Section 4.1.
- Δ_A and Δ_E are lists, each PE has the complete list.
- Ψ is implemented as a list too, but each PE i only add the agents ψ with $\beta(\psi) = i$ to the list. Additionally each PE has a list of all ghost agents $\{\hat{\psi} \in \Phi^{\text{from}}_{\delta_E,\psi} \mid \beta(\hat{\psi}) \neq \beta(\psi)\}$ existing on this PE.
- Φ is implemented via structured arrays from the NumPy package. Each PE i has such an structured array per edge type Δ_E which contains all the edges created by agents ψ with $\beta(\psi) = i$. As an agent can only access edges she has created herself, a PE j does not have any information about edges ϕ with $\beta(second(\phi)) \neq j$.
- Also the valuing function Ω is implemented via a structured arrays from the NumPy package. For each agent type Δ_A each PE has a corresponding structured array that contain the states of all agents and ghost agents that are distributed to this PE. As each agent is represented by a row in this array, the valuing is just index operator on the table.

This framework is still work in progress, for more details please watch out for forthcoming papers and the release of the framework itself.

References

Boman EG, Catalyurek UV, Chevalier C, Devine KD (2012) The Zoltan and Isorropia parallel toolkits for combinatorial scientific computing: Partitioning, ordering, and coloring. Scientific Programming 20(2):129–150

Cosenza B, Cordasco G, De Chiara R, Scarano V (2011) Distributed load balancing for parallel agent-based simulations. In: Proceedings - 19th International Euromicro Conference on Parallel, Distributed, and Network-Based Processing, PDP 2011, DOI 10.1109/PDP.2011.22

Devine KD, Boman EG, Heaphy RT, Bisseling RH, Catalyurek UV (2006) Parallel hypergraph partitioning for scientific computing. In: 20th International Parallel and Distributed Processing Symposium, IPDPS 2006, DOI 10.1109/IPDPS.2006.1639359

Grimm V, Berger U, De Angelis DL, Polhill JG, Giske J, Railsback SF (2010) The ODD protocol: a review and first update. Ecological Modelling 221:2760–2768

Hendrickson B (2000) Load balancing fictions, falsehoods and fallacies. Applied Mathematical Modelling DOI 10.1016/S0307-904X(00)00042-1

Laubenbacher R, Jarrah AS, Mortveit H, Ravi SS (2008) A mathematical formalism for agent-based modeling. Enczclopedia of Complexity ans System Science

Reynolds CW (1987) Flocks, herds and schools: A distributed behavioral model. ACM SIGGRAPH Computer Graphics DOI 10.1145/37402.37406, 0208573

Rubio-Campillo X (2014) Pandora: A Versatile Agent-Based Modelling Platform for Social Simulation. In: SIMUL 2014: The Sixth International Conference on Advances in System Simulation

Salamon T (2011) Design of Agent-Based Models : Developing Computer Simulations for a Better Understanding of Social Processes. Academic series, Bruckner Publishing, Repin, Czech Republic

Schloegel K, Karypis G, Kumar V (1999) A New Algorithm for Multi-objective Graph Partitioning. In: Amestoy P, Berger P, Daydé M, Ruiz D, Duff I, Frayssé V, Giraud L (eds) Euro-Par'99 Parallel Processing, Springer Berlin Heidelberg, Berlin, Heidelberg, pp 322–331

Shook E, Wang S, Tang W (2013) A communication-aware framework for parallel spatially explicit agent-based models. International Journal of Geographical Information Science 27(11):2160–2181, DOI 10.1080/13658816.2013.771740, URL https://doi.org/10.1080/13658816.2013.771740

Slota GM, Madduri K, Rajamanickam S (2014) Pulp: Scalable multi-objective multi-constraint partitioning for small-world networks. In: 2014 IEEE International Conference on Big Data (Big Data), pp 481–490, DOI 10.1109/BigData.2014.7004265

Socioeconomic Data and Applications Center (SEDAC) (2015) Gridded Population of the World, v3. http://sedac.ciesin.columbia.edu/data/collection/gpw-v3/maps/services, visited December 2015