Performance, Scale & Time in Agent-based Traffic Modelling with NetLogo

Christoph Mayrhofer

Z_GIS, University of Salzburg, Austria · mayrhoferchr@stud.sbg.ac.at

Short paper

Abstract

Agent-based modelling has proven as an effective approach to analyze, reconstruct, and predict systems with many individual entities. Typical use cases include segregation processes, predator/prey models, or supply chain optimization (TISUE 2004). Most of these models work within the framework of a "patch-dominated" world, where agents move within a world consisting of square-cells named patches (AL-DMOUR 2011). The development of network specific extensions for simulation software like the "network" and "gis" extensions of NetLogo have allowed to apply agent-based theories to spatial data (CROOKS 2012). This allows the shift from a patch to a network concept. However, there are still some shortcomings that are not satisfyingly solved within the NetLogo software that may hinder an efficient and realistic simulation of agent movements along spatial networks.

This work evaluates the role of scale and time in agent-based traffic models and offers suggested implementations to allow for easier and more realistic parameterization and runtime analysis of traffic models. It will also discuss performance issues with the shortest path calculation in NetLogo, and provide a possible solution to increase the simulation speed.

The model and analysis derived in this project are based on a polyline shapefile that contains the street network of Salzburg city.

1 Spatial Scale

All conventional GIS systems work with coordinate systems in order to place information in its correct spatial context. Simulation models on the other hand are usually independent of spatial constraints and operate within an "abstract" spatial setting (AL-DMOUR 2011). It is now necessary to combine those two worlds for traffic modelling since the agents need to move at certain speeds and it is therefore necessary to have an actual measure (meters), rather than patches.

GIS coordinates (map units) vs. NetLogo world coordinates: NetLogo uses its own coordinate system. The map is called "world" and the coordinates correspond to the number of patches in each dimension. This is suitable for "abstract" simulations with no spatial context, but infers certain limitations for traffic modelling where it is crucial to know NetLogo's actual scale. Since speed is an inherent parameter to any traffic model we also need its main components – distance and time – in order to simulate accurate movement of the agents within the given spatial setting (LANSDOWNE 2006).

© Herbert Wichmann Verlag, VDE VERLAG GMBH, Berlin/Offenbach. ISBN 978-3-87907-558-4.
© ÖAW Verlag, Wien. ISSN 2308-1708, doi:10.1553/giscience2015s567.

Connecting GIS and NetLogo coordinates: It is a rather trivial process to implement scale in a NetLogo model. Nonetheless only few models apply this and make use of the added functionalities that scale implies. The GIS extension provides functions that read the actual coordinates from the input dataset. These coordinates are optimally already in UTM or any other CRS that uses meters as its basic map unit. This number can now be used to resize the NetLogo world accordingly. Our test dataset spans over 14500 x 14500 meters. Thus, we would resize to a world of 14500 x 14500 patches to achieve a simple 1:1 scale between the NetLogo and GIS coordinates.

However, it is important to consider performance once again, and keep in mind that the software will slow down significantly with such huge patch numbers. It is therefore advisable to implement a maximum size (e.g. 1000 x 1000 patches) and use the ratio (1:14.5) as a scale-factor that is later used for all calculations that involve distance.

2 Temporal Scale

The second necessary dimension is time. NetLogo's logic is built on the concept of ticks. Each tick represents one simulation step and equals to one cycle of the entire code. The time representation of one tick is completely up to the developer and may be hard to comprehend by the users since they normally think in terms of seconds or hours, etc.

NetLogo computes the ticks as fast as possible and only provides a slider that allows users to vary the amount of time that is waited between each tick (WILENSKY 2014). Thereby, it completely ignores the variation within the ticks itself. The computation time of each tick depends on the number of active agents in the model at each distinct time step. This number may vary greatly during a simulation, which results in visualizations that continuously change from several seconds per tick to dozens of ticks per second. The user is not able to accurately follow the process and the visualization loses a lot of its purpose to support the understanding and interpretation of the simulation.

This demonstrates that the concept of the speed control in NetLogo is not suitable for traffic models that require constant timings during the simulation. However, models that only focus on the end result are not negatively affected, since the simulation process itself is not of interest to them.

It is necessary to "abandon" NetLogo's native speed control in order to synchronize real-time with the time passed within the simulation. This is possible using the "timer" variable in NetLogo. This keeps track of the actual time passed since the start of a simulation and allows the developer to time the duration of the ticks. The waiting time between the ticks can now be dynamically changed to add up to a user-defined value.

⇨ Variable tick duration + variable sleep time = constant simulation step time

An additional measure to improve usability is to implement a time-factor, which represents the ratio between real-time and simulation time. A factor of 20 would cause the model to simulate 20 seconds during every second the user watches the model run. It is advisable to keep track of the past tick durations to automatically adjust that factor for the user in case that a too large factor was chosen which cannot be computed fast enough by the system. This control can now replace the native speed control.

3 Performance

Performance is not a major issue with most basic operations like changing agent statuses or moving an agent. The NetLogo software can easily handle several thousand agents on a modern computer. However, there is one bottleneck in most traffic models – The shortest path calculation. The street network needs to be evaluated and analyzed for each agent that is added to the model in order to create a route from its location to its destination. This is computationally expensive and may have a big impact on the model performance with larger networks of many thousand vertices.

Shortest path algorithm: NetLogo uses the Dijkstra algorithm to find the shortest path between two points in the network. This algorithm builds a tree of distance values between the location and its linked vertices. It then continues to expand until the tree includes the desired destination. The time complexity of this algorithm is nonlinear and performance slows down significantly with larger networks (BARBEHENN 1998).

NetLogo saves those trees to reuse suitable parts of them for later path calculations. While this improves performance it may also cause physical memory issues on the computer. All the saved trees add up quickly and they will eventually cause a java heap space overflow (the computer runs out of RAM). It is therefore advisable to disable this temporary tree storage and use another approach to improve performance: **Reduce the network complexity to the necessary minimum.**

The main objective is to eliminate all vertices that are not essential for the path calculation and then only pass this smaller subset of vertices to the shortest path algorithm. The first step is to remove all "duplicate" vertices (i.e. vertices with the same location). The topology of the test dataset divided all streets in segments that would reach from one junction to the next. The meeting points of those segments would then consist of two points (the street ends) or several points if more than two streets meet and create a junction. It is now possible to link all streets to a single common junction vertex and delete the remaining unused junction vertices.

Most of the improvement can be achieved by excluding all vertices that are not a junction. There is no decision to be made if there is no junction, since the agents can only go forward or backwards. (a shortest path will never include a move back). The reduction in network complexity can be very significant if the network contains many intermediary vertices (e.g. to follow the shape of curvy roads). The resulting "junctions only" network may only consist of a few percent of the initial vertices (Fig. 1).

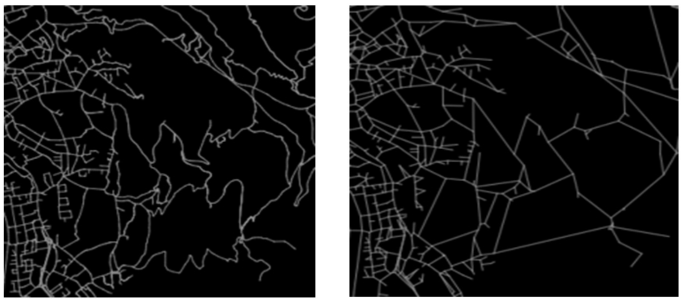

Fig. 1:
Subset of the network before (top, 76894 vertices) and after (bottom, 7589 vertices)

The effectiveness of this approach depends on the overall number of vertices and the share of junctions and intermediary vertices in the dataset. The performance improvement (comparison of full and reduced network calculation) increases with larger networks and may not be noticeable with small ones of several hundred vertices.

Table 1: Calculation times for 1000 agents using different types of network reduction

Generate 1000 cyclists	Time (min)	#vertices (shortest-path)	#intermediary vertices	#vertices (total)
Full network	12:30	76894	0	76894
Simplified network	00:51	7589	0	7589
Simplified network 2	00:57	7589	53428	61017

The above table shows the times necessary to calculate the shortest paths for 1000 agents on the test computer. The origin and destination points are randomly selected and the simulations were repeated 40 times. The time listed is the mean of the second and third quartile of the result values. The second network type (simplified network 2) adds the vertices between the junctions to create the actual path along the street as it would also be computed by the shortest-path algorithm if the entire network was used with it. This approach is almost as fast as only using the junctions and seems to be the best compromise for the route calculation. However, there may be further potential to improve the model performance by choosing more advanced routing algorithms like a dynamic Landmarks-A* router as implemented by MEISTER et. al. (2010) in the simulation tool MATSIM-T.

References

AL-DMOUR, N. A. A. H. (2011), TarffSim: Multiagent traffic simulation. European Journal of Scientific Research, 53(4), 570-575.

BARBEHENN, M. (1998), A note on the complexity of Dijkstra's algorithm for graphs with weighted vertices. IEEE transactions on computers, 47 (2), 263-263.

CROOKS, A. T. & CASTLE, C. J. (2012), The integration of agent-based modelling and geographical information for geospatial simulation. In: Agent-based models of geographical systems. Springer Netherlands, 219-251.

LANSDOWNE, A. (2006), Traffic simulation using agent-based modelling. University of the West of England.

MEISTER, K., BALMER, M., CIARI, F., HORNI, A., RIESER, M., WARAICH, R. A. & AXHAUSEN, K. W. (2010), Large-scale agent-based travel demand optimization applied to Switzerland, including mode choice. 12th World Conference on Transportation Research, Lisbon.

TISUE, S. & WILENSKY, U. (2004). NetLogo: Design and implementation of a multi-agent modelling environment. Proceedings of the Agent 2004 Conference on Social Dynamics: Interaction, Reflexivity and Emergence, Chicago, IL.

WILENSKY, U. (2014), NetLogo Users Manual. http://ccl.northwestern.edu/netlogo/. Center for Connected Learning and Computer-Based Modeling, Northwestern University, Evanston, IL.