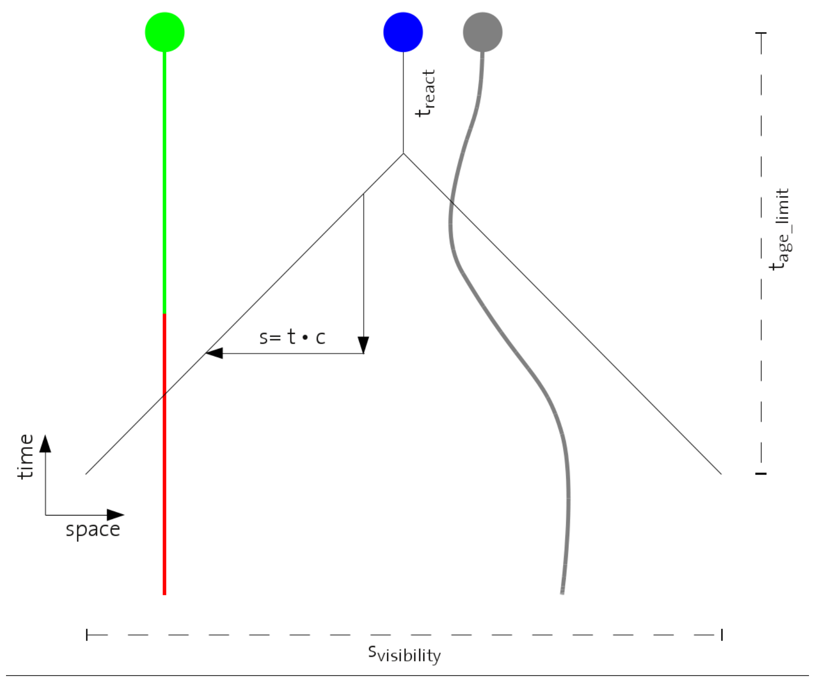


Integrating two Simulation Modules with a General Parallelization Framework

David Charypar

Fabian Märki

Kay W. Axhausen

2011

Institut für Verkehrsplanung und Transportsysteme
Institute for Transport Planning and Systems

Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Integrating two Simulation Modules with a General Parallelization Framework

David Charypar
Institute for Transport Planning and Systems
ETH Zurich
CH-8093 Zurich
phone: +41-44-633 35 62
fax: +41-44-633 10 57
charypar@ivt.baug.ethz.ch

Fabian Märki
Institute for Transport Planning and Systems
ETH Zurich
CH-8093 Zurich
phone: +41-44-633 33 25
fax: +41-44-633 10 57
maerki@ivt.baug.ethz.ch

Kay W. Axhausen
Institute for Transport Planning and Systems
ETH Zurich
CH-8093 Zurich
phone: +41-44-633 39 43
fax: +41-44-633 10 57
axhausen@ivt.baug.ethz.ch

2011

Abstract

This paper describes a pedestrian walking simulator coupled with a simple destination choice model. First, a straight forward implementation is presented together with some performance figures. Afterwards, we discuss efficiency issues and describe several improvements in the algorithms and data structures employed. These improvements are based on the concept of a simulation framework that imposes limitation on the communication range and speed of the simulated entities (i.e. walking agents). As we discuss in the paper, these limitations speed up the computations necessary to simulate the interaction between virtual persons. The resulting implementation shows clear performance gains and will simplify the final parallelization process of the simulation.

Keywords

transport simulation, framework, concept, microsimulation, integrated model

1 Introduction

Integrated microscopic transport simulation models show at least two desirable properties: their inherent high resolution and the relatively simple models describing the behavior of the simulated entities. One of the key ideas behind this class of models is to represent a complex system (such as our society and its travel behavior) by describing the relevant parts of the local interactions and to make use of computers to simulate a large number of simple objects to let the large scale effects emerge. These results can then be used by e.g. policy makers to make better decisions on various propositions.

There are, however, downsides of the microscopic modeling approach. Perhaps the most important of them is the often substantial computing time needed to run the corresponding simulations. A result of this is a serious limitation of the problem size solvable with this approach.

There are various ways of increasing the feasible problem size of any model used for simulation. When we try to speed up an integrated microsimulation perhaps the method requiring most foresight, as well as modeling and programing skills, is to increase the efficiency of the simulation employed. This includes trimming the model down to the point where only minimal overhead is generated while retaining all features relevant for the problem at hand. Usually there are narrow limits on this road.

A very basic approach is to increase the speed of the simulation by increasing the speed (GHz) of the CPU core used. Progress in this field has, however, mostly vanished in the last years: Gigahertz figures are stagnating, and therefore current CPU development is focusing on multi core architectures.

This leads to the very simple idea of making use of the additional CPU cores. Unfortunately, implementing this idea proves to be more difficult than it seems as sequential software (i.e. software designed to be run an a single CPU core) hardly ever runs efficiently in parallel out of the box. To the contrary, it generally proves to be necessary to develop new software for that purpose. Care must be taken to carefully balance the load between CPU cores and to avoid interference between cores due to data dependencies. One goal of the framework proposed in this paper is to substantially reduce the burden of developing parallel microscopic travel behavior simulation programs.

Microscopic integrated demand models are being successfully applied to large real world problems. Often, the average work day is under investigation—a problem that can be addressed with equilibrium models. One example for an available model able to deal with such a problem is (Balmer *et al.*, 2009a) even though they use substantial computing power and simulation time to get the solution.

There are, however, different modeling task that cannot be solved using equilibrium models. One important example is the simulation of unpredictable events like accidents, emergencies, or disasters. Since, in such scenarios, the focus lies on reactions in a quickly changing environment with a lot of uncertainty about the progression of the situation, equilibrium models cannot be used. Models taking into account the continuous development of the environment are needed.

To produce instantaneous reactions, at any time of the simulation the relevant parts of the current state of the system must be made available to the agents in the simulation. By providing this information the proposed simulation framework creates the necessary infrastructure for interaction between entities. It shows that this infrastructure can be used for the simulation of long periods as well. (see e.g. Märki *et al.*, forthcoming)

2 Related Work

The following is a short overview about other work that was performed either in the field of parallel microsimulations or in the design of frameworks for transport or urban modeling.

2.1 Parallel Transport Simulations

There are numerous examples of parallel implementations of traffic simulations. Barceló *et al.* (1998) showed a parallel implementation of their micro simulator AIMSUN achieving a parallel speedup of 3.5 when run on 8 processors. The parallelization concept was to make all data globally accessible. PTV's VISSIM traffic micro simulator also has the capability to run in parallel using a multi-threaded concept. (PTV America, 2010)

Nagel and Rickert (Nagel and Rickert, 2001; Rickert and Nagel, 2001) showed a parallel version of a cellular automaton used for traffic flow simulation in TRANSIMS (Nagel *et al.*, 1998). They used message passing between processors and achieved a speedup of 10 with 32 processors. They reported latency problems due to Ethernet data communication.

There has been some work on parallel queue-based models (e.g. Cetin, 2005; Cetin *et al.*, 2003; Charypar *et al.*, 2007, 2009). Using message passing between cluster nodes, the queue-based model presented in (Cetin, 2005; Cetin *et al.*, 2003) achieved a speed-up of 32 using 64 CPUs when simulating a peak period. In (Charypar *et al.*, 2009) the authors report a parallel speedup of 53 when using 64 processors for simulating a large scenario.

A number of parallel implementations of mesoscopic transport models have been presented in the past. METROPOLIS(Marchal, 2001; de Palma and Marchal, 2002) is able to simulate large scenarios efficiently by using a parallel implementation based on up to 16 threads. DynaMIT(Ben-Akiva *et al.*, 1998; DynaMIT, 2006) does not parallelize the traffic flow simulation itself but uses task parallelization i.e. different modules are run in parallel. Unfortunately, this limits the number of usable processors to the number of modules. DYNEMO(Schwerdtfeger, 1984; Nökel and Schmidt, 2002) was run in parallel (Nökel and Schmidt, 2002) by using a message passing technique on 19 CPUs for simulating small scenarios. Larger numbers of CPUs were reported to be inefficient.

2.2 Frameworks for Integrated Modeling

Ferreira *et al.* (2008) present a framework (MAS-T2er) for integrated multi-agent systems. Their focus is on control strategies and intelligent transport systems. The intended use of their software is "...for cooperative design, visualization and engineering, allowing for the cooperative decision-making by different traffic and transport experts". Their framework is designed to run on distributed systems.

The goal of UrbanSim(Waddell, 2002) is to model and simulate urban development by modeling the interactions of many different actors that make decisions in the markets for land, housing, non-residential space and transportation.

The Multi-Agent Transport Simulation Toolkit (MATSim-T) (Balmer *et al.*, 2009b) is a simulation framework for modular development of an integrated transport simulation for large-scale applications. Several modules that where written for/in MATSim run in parallel, e.g. certain versions of the traffic flow simulator(e.g. Charypar *et al.*, 2009), the processing of simulation events, and the activity planning module. In general MATSim aims to run all the modules in parallel but there is no direct support by the toolkit for parallelization efforts by the developers.

3 Framework

3.1 Motivation

It seems that one of the problems hindering the application of microsimulation models to large scale problems is their computational demand. While parallelization could in principle solve this problem, developing the necessary parallel computer programs is not trivial. We therefore propose a software framework that facilitates the aspect of parallel communication between computer nodes in spatial simulation software. The framework is intended for simulations with an open time horizon where information about the past is available to the simulated entities (potentially with some delay), but no knowledge about the future is provided. Consequently, it is not intended to be used with iterative approaches. The feed-forward design of the framework makes it particularly interesting for investigating the immediate response to unexpected events.

3.2 General Idea

The described framework assumes that different tasks in a large integrated simulation are encapsulated in individual modules. Simulated entities should communicate with other entities only through an interface provided by the framework, whether they are part of the same module or different modules. With communication we refer to any exchange of information about what exists and happens in the simulated environment.

3.3 Contracts

The general idea is that there exists a contract between user developed modules and the simulation framework. The modules submit all information that is (or might be) relevant to other parts of the system to the frame work and the frame work in turn provides all relevant information to the entities of the various modules. What information is to be submitted and what to be provided is specified by the following set of rules.

3.3.1 Radius of Perception

Modules must specify what is there radius of perception. This is a distance measure. Information (events, objects, etc.) outside this radius cannot be perceived by objects of the corresponding module, irrespective of whether the information comes from the same or a different module. While a large radius improves the vision of simulated entities it also slows down the resulting simulation. That is, the developer has to trade vision versus simulation speed.

3.3.2 Delayed Perception

In the framework, information travels at a limited speed and objects in the simulation have a certain reaction time that must elapse before a new event can be perceived. The user must specify the speed of information propagation as well as the reaction time of objects. High speeds and low reaction times enable quicker processes inside the simulation (measured in simulated time) while they slow down, at the same time, the simulation itself (measured in CPU time). Again, there is a trade-off between desirable properties.

3.3.3 Limited Speed of Motion

Similar to the limited speed of information there exists also a limited speed of motion. Objects in the simulation are not allowed to move faster than a certain predefined speed which once again must be selected by the developer employing the framework. Higher maximum speeds will result in slower simulations and vice versa. Usually the maximum speed of motion should be selected lower than half the speed of information propagation. If the maximum speed of motion is selected higher, strange effects may happen: In such a setting, two objects moving in opposite directions passing by each other might not "see" each other until after a potential collision. While this is not totally unthinkable, most simulations will not work this way.

3.3.4 Information Age Limit

For the sake of memory efficiency it is necessary to also specify an information age limit. Information older than this limit will be forgotten by the system and will no longer be available to objects in the simulation. Higher information age limits will mainly increase the memory consumption of the resulting simulation, while the running time will not be largely affected. Please note: In order to be consistent, the information age limit should be chosen larger than the radius of perception divided by the speed of information propagation. Otherwise the effective radius of perception will be reduced.

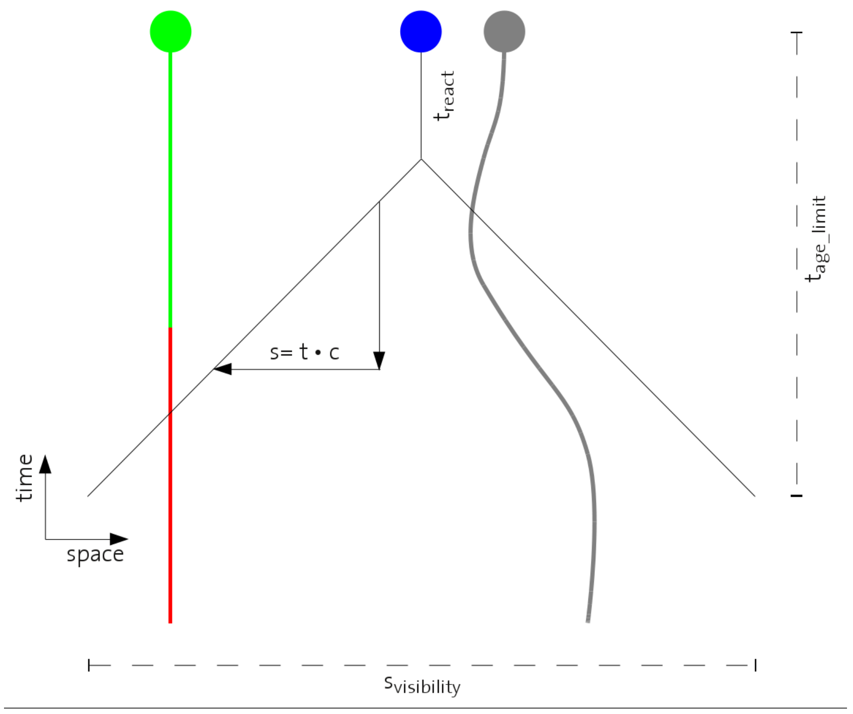
3.4 Final Set of Limits

The final set of limits is illustrated in Figure 1, where the perception of an object (blue) is shown in a space time diagram. $s_{visibility}$ is the visibility range, t_{react} is the reaction time, c is the speed of information propagation, t represents time, and s space. The thin black line represents the front of information just becoming available to the blue object. The green object is not moving and changed its state from red to green at a certain point in the past. Since the blue object is relatively far away this change can not yet be perceived and hence the red state is still relevant to the blue object. The gray object is moving and the position where its path intersects the information front is the latest position visible to blue. All older positions of its trajectory are visible up to the age limit t_{age_limit}.

4 Model

Different transport simulation modules have different needs for the simulation framework. For instance the way information is queried varies significantly between continuous space simula-

Figure 1: Limited Perception of Information Based on Specified Limits



tions (e.g. pedestrian simulations) and network-based simulations (e.g. link based traffic simulations). It is, however, difficult to address all the different requirements to such simulation frameworks at once, and so we decided to first implement one particular simulation model and learn the various requirements for the framework on the way. The following therefore describes the various steps we took so far to develop a pedestrian simulation that runs in parallel using our simulation framework.

4.1 Pedestrian Walking Simulator

The decision was made to develop a pedestrian walk simulator suitable for large scale applications. The simulation is time-step based, simulating the position and velocity of all virtual persons (agents) in the scenario. There is a (repulsing) social force between all agents that depends on the distance between the involved agents. Furthermore, all agents have a destination they want to reach and a desired walking speed. The destination of each agent can change during the simulation as a function of time and the performance of the agent and all its coexisting agents. The simulation is designed with a time-step loop. In each time-step, the force acting between each pair of agents is computed; then, if the desired velocity differs from the current velocity of an agent a restoring force is added. From the resulting force the new velocity of the agent is computed by explicit time integration. Finally, the new velocity is used to integrate the new position of the agent. This corresponds to the Euler-Cromer method. The set of position, velocity, and current acceleration of each agent, together with a current time stamp, is stored as the current state of the agent. The list of all (available) states of each agent then represents the pool of information that can be used by other agents for their decisions.

4.2 First Implementation

In the first implementation the interaction force, $\vec{F}_{rep}$, form one agent on another agent was chosen as follows:

$$\vec{F}_{rep} = a\frac{\vec{r}}{|r|^4}, \tag{1}$$

with a, a scaling factor, and $\vec{r}$, the distance vector *from* the neighbor *to* the agent.

During the time-step computation, the force acting between each possible pair of agents is computed in a double loop over all agents. The forces are summed up in one variable (a force vector) per agent. To compute the vector $\vec{r}$ the current position of the agent is used and the most recent visible state of the neighboring agent. In the first implementation, each agent stores all its past states itself. This is done using a double linked list. Information requests from other agents are answered directly, without a specified interface taking care of it.

There is a desired velocity restoring force, $\vec{F}_{vel}$, defined as follows:

$$\vec{F}_{vel} = b\left(\vec{v}_{des} - \vec{v}_{cur}\right), \tag{2}$$

with b, a scaling factor, $\vec{v}_{des}$, the desired velocity of the agent, and $\vec{v}_{cur}$, its current velocity. The desired velocity is selected in such a way as to lead to the desired destination with a desired walking speed. The target location is selected such that the direct path to it leads through a small area in the center of the simulated domain. As soon as an agent reaches its destination a new target is selected such that the path there leads—once again—through the center of the domain. With this procedure we make sure that a dynamic bottleneck is formed in the center of the domain.

The performance of the described implementation is about 20 time-steps per second, with 100 agents simulated. Clearly, the inefficient handling of the pair interactions is the main reason for the high computational demand. The presented algorithm has a run time complexity of $O(n^2)$, with n, the number of agents. This means that simulating 10 times as many agents results in 100 times the simulation time.

4.3 Steps Forward

At this point, the focus was to increase the simulation speed and at the same time introduce some of the data structures necessary for later parallelization. The first modification is to introduce a spacial search data structure. Assuming that agents interact only over short distance (only pedestrians nearby have an influence on the walking behavior) it is unnecessary to iterate over all agents in the simulation to find the resulting force on one agent. The idea is to subdivide the simulation domain in grid cells, give each cell a unique number (and index), compute the hash value of this index, and store all agents belonging into the same hashed grid cell in the same data-container. Now, to compute the interaction forces an agent has to iterate over all agents in its own cell and all 8 neighboring cells. This algorithm is correct, if the range of the agent to agent interaction force is equal to (or less than) the size of the grid cells. To allow for that the interaction force has to be modified in such a way that it is zero outside the force range. This corresponds also to the radius of perception described in the last chapter. The new repulsive force, $\vec{F}_{rep2}$, we use is:

$$\vec{F}_{rep2} = \begin{cases} c\,\vec{r}\frac{(|r|-R)^2}{|r|^2}, & if\,|r| < R \\ \vec{0}, & else \end{cases} \tag{3}$$

with c, a scaling factor and R, the interaction force range.

The agents are put into the new data structure at the end of each time-step using their respective current position.

A substantial part of memory consumption and computing time of the simulation can be attributed to the state storing mechanism of the individual agents. As noted in the last chapter, there needs to be an age limit at which old states are forgotten by the simulation. Consequently, the size of the state memory of each agent was limited and old stated are now “pushed out” by new states.

The described changes have sped up the pedestrian simulation substantially. Now, about 10 times as many agents can be simulated in the same time as before. This corresponds to a 100 times faster simulation, as the old pair iteration algorithm would have treated many interactions between agents that are actually outside their respective interaction force ranges.

4.4 Next Steps

The next steps in the course of the software development will be to first continue to work on the simulation performance. Currently, the main loop in the force calculation algorithm runs over individual agents. From an efficiency point of view, it would make sense to instead process whole hashed cells at once since this would avoid many unnecessary search operations. Furthermore the hashing algorithm is not yet optimized to the problem and could be made more efficient.

Once we will have reached this point we will start working on separating the representation of the states of an agent and the agent itself. This will be also in preparation of the parallelization scheme.

After that we will have to expand the concept of the cells and make them individual objects in the simulation framework. By doing so, we will be able to use them as entities that are assigned to different processors and moved around from machine to machine to allow for a proper load balancing scheme to be implemented. We will start with static load balancing, i.e. the assignment of cells to processors will be fixed throughout a simulation run. Then, we will investigate how simple statistics could be used to decide wheater a grid cell should be virtually moved to a different, less loaded CPU.

5 Discussion and Conclusion

The described pedestrian walking simulator shows significant performance gains with the described improvements. In course of the development process the employed cell-based data structure will show useful also for the parallelization process and the distribution of information across collaborating CPUs. Also will it be utilized to evenly distribute the workload among processors of computer cluster.

After the first parallel implementation, one of the big challenges will be to extract the described interfaces from the simulation and to make it easily usable for other simulation tasks. Furthermore, providing good guidelines on how tho chose the proper communication limits based on the simulation purpose will be crucial for the success of the framework. To find these guidelines it will be necessary to systematically study the effects of such limits on both the result of the simulation and its performance.

References

Balmer, M., A. Horni, K. Meister, F. Ciari, D. Charypar and K. W. Axhausen (2009a) Wirkungen der Westumfahrung Zürich: Eine Analyse mit einer Agenten-basierten Mikrosimulation, *Final Report*, Baudirektion Kanton Zurich, IVT, ETH Zurich, Zurich, February 2009.

Balmer, M., M. Rieser, K. Meister, D. Charypar, N. Lefebvre and K. Nagel (2009b) MATSim-T: Architecture and simulation times, in A. L. C. Bazzan and F. Klügl (eds.) *Multi-Agent*

Systems for Traffic and Transportation Engineering, 57–78, Information Science Reference, Hershey.

Barceló, J., J. L. Ferrer, D. Garcia, M. Florian and E. Le Saux (1998) Microscopic traffic simulation, in P. Marcotte and S. Nguyen (eds.) *Equilibrium and Advanced Transportation Modelling*, chap. 1, 1–26, Kluwer, Dordrecht.

Ben-Akiva, M. E., M. Bierlaire, H. Koutsopoulos and R. Mishalani (1998) DynaMIT: A simulation-based system for traffic prediction, paper presented at the *DACCORS Short Term Forecasting Workshop*.

Cetin, N. (2005) Large-scale parallel graph-based simulations, Ph.D. Thesis, ETH Zurich, Zurich.

Cetin, N., A. Burri and K. Nagel (2003) A large-scale multi-agent traffic microsimulation based on queue model, paper presented at the *3rd Swiss Transport Research Conference*, Ascona, March 2003.

Charypar, D., K. W. Axhausen and K. Nagel (2007) An event-driven queue-based traffic flow microsimulation, *Transportation Research Record*, **2003**, 35–40.

Charypar, D., M. Balmer and K. W. Axhausen (2009) High-performance traffic flow microsimulation for large problems, paper presented at the *88th Annual Meeting of the Transportation Research Board*, Washington, D.C., January 2009.

de Palma, A. and F. Marchal (2002) Real cases applications of the fully dynamic METROPOLIS tool-box: An advocacy for large-scale mesoscopic transportation systems, *Networks and Spatial Economics*, **2** (4) 347–369.

DynaMIT (2006) Intelligent transportation system program, webpage, `http://mit.edu/its/dynamit.html`.

Ferreira, P. A. F., E. F. Esteves, R. J. F. Rossetti and E. C. Oliveira (2008) A cooperative simulation framework for traffic and transportation engineering, in L. Yuhua (ed.) *Cooperative Design, Visualization, and Engineering*, vol. 5220 of *Lecture Notes in Computer Science*, 89–97, Springer, Berlin.

Marchal, F. (2001) Contribution to dynamic transportation models, Ph.D. Thesis, University of Cergy-Pontoise, Cergy-Pontoise.

Märki, F., D. Charypar and K. W. Axhausen (forthcoming) Continuous activity planning for a continuous traffic simulation, paper presented at the *90th Annual Meeting of the Transportation Research Board*, Washington, D.C.

Nagel, K. and M. Rickert (2001) Parallel implementation of the TRANSIMS micro-simulation, *Parallel Computing*, **58** (2) 1611–1639.

Nagel, K., D. E. Wolf, P. Wagner and P. M. Simon (1998) Two-lane traffic rules for cellular automata: A systematic approach, *Physical Review E*, **58** (2) 1611–1639.

Nökel, K. and M. Schmidt (2002) Parallel DYNEMO: Meso-scopic traffic flow simulation on large networks, *Networks and Spatial Economics*, **2** (4) 387–403.

PTV America (2010) PTV America, webpage, `http://www.ptvamerica.com`.

Rickert, M. and K. Nagel (2001) Dynamic traffic assignment on parallel computers in TRANSIMS, *Future Generation Computer Systems*, **17** (5) 637–648.

Schwerdtfeger, T. (1984) DYNEMO: A model for the simulation of traffic flow in motorway networks, in J. Volmuller and R. Hamerslag (eds.) *Proceedings of the Ninth International Symposium on Transportation and Traffic Theory*, chap. 4, 65–87, VNU Science Press, Utrecht.

Waddell, P. (2002) Urbansim: Modeling urban development for land use, transportation, and environmental planning, *Journal of the American Planning Association*, **68** (3) 297–314.