

**A DDDAS FRAMEWORK FOR MANAGING ONLINE
TRANSPORTATION SYSTEMS**

A Dissertation
Presented to
The Academic Faculty

by

Philip K. Pecher

In Partial Fulfillment
of the Requirements for the Degree
Doctor of Philosophy in the
School of Computational Science and Engineering, College of Computing
H. Milton Stewart School of Industrial and Systems Engineering

Georgia Institute of Technology
May 2018

COPYRIGHT © 2018 BY PHILIP K. PECHER

A DDDAS FRAMEWORK FOR MANAGING ONLINE TRANSPORTATION SYSTEMS

Approved by:

Professor Richard M. Fujimoto, Advisor
School of Computational Science and
Engineering
Georgia Institute of Technology

Professor David Goldsman
H. Milton Stewart School of Industrial
and Systems Engineering
Georgia Institute of Technology

Professor Michael P. Hunter
School of Civil and Environmental
Engineering
Georgia Institute of Technology

Dr. Brian Swenson
Georgia Tech Research Institute
Georgia Institute of Technology

Professor Michael O. Rodgers
School of Civil and Environmental
Engineering
Georgia Institute of Technology

Date Approved: March 26, 2018

ACKNOWLEDGEMENTS

I would like to thank my advisor Prof. Richard Fujimoto for his knowledgeable guidance and patient support through my doctoral study. Furthermore, I would like to extend thanks to my other committee members – Michael Hunter, Michael Rodgers, David Goldsman, and Brian Swenson – and other members of the Georgia Tech faculty – including Bistra Dilkina, Richard Vuduc, Guin Angshuman and many others - for their valuable advice during my time here.

I would also like to thank the Air Force Office of Scientific Research and the National Science Foundation for their financial support, as well as my fellow graduate students SaBra Neal, Aradhya Biswas, Mark Jackson, Holly Williams, Prakalp Sudhakar, Felix Lagarde, Michael Palli, and many others for the great memories during our study and research.

Last but not least, I'm thankful to Sara, and to my family, for their warm support and encouragement.

Table of Contents

ACKNOWLEDGEMENTS	iii
LIST OF TABLES	viii
LIST OF FIGURES	ix
SUMMARY	xii
1 INTRODUCTION	1
1.1 Dynamic Data Driven Application Systems	4
1.2 Microscopic Traffic Simulation	7
1.3 Trajectory Prediction	11
1.4 Simulation Cloning	12
1.5 Research Contribution and Thesis Organization	13
2 TRAJECTORY PREDICTION WITH PAST AND FUTURE TREES ¹	15
2.1 Introduction	15
2.2 Motivating Context: DDDAS	16
2.2.1 FIND-OFFSET: Determining the critical lag	18
2.2.2 PROBABILITY-MAP: Path Speculation	19
2.3 Related Work	20
2.3.1 Krumm's Prediction Model	21
2.4 Past and Future Trees	22
2.4.1 Conceptual Structure and Trajectory Insertions	22
2.4.2 The Method of Past Trees: Predicting Trajectories and Destinations	25

2.4.3	Generating Simulation Trajectories	27
2.4.4	Space-Efficient Storage.....	28
2.5	Experimental Evaluation.....	30
2.5.1	Synthetic Experiment: Setup.....	30
2.5.2	Real-World Traces: Setup and Results	34
2.6	Conclusion and Future Work	35
3	Data-Driven Vehicle Trajectory Prediction ²	37
3.1	Introduction.....	37
3.2	Related Work	41
3.3	Prediction Models	43
3.3.1	Krumm's Destination Prediction Algorithm Based on Efficient Routes.....	44
3.3.2	Artificial Neural Networks.....	45
3.3.3	Markov Models and Trajectory Storage	47
3.3.4	Generating Routes from Probability Maps	49
3.4	Experimental Evaluation.....	50
3.4.1	The Efficient Route Model and Extensions	51
3.4.2	Markov Models.....	55
3.4.3	Feed-Forward Artificial Neural Network.....	59
3.5	Comparison of the Models.....	60
3.6	Displacement Inference Approach.....	62
3.7	Conclusion and Future Work	71

4	Execution of Replicated Transportation Simulations with Uncertain Vehicle Trajectories ³	73
4.1	Introduction.....	73
4.2	Related Work	75
4.3	Simulation Algorithm	77
4.3.1	Superimposed Execution Before Hazards Arise	79
4.3.2	Speculative Execution with Hazards.....	81
4.4	Experimental Evaluation.....	85
4.4.1	Experimental Setup	85
4.4.2	Speedup vs. Number of Replications and Traffic Intensity	86
4.4.3	Speedup vs. Relative Speed Differences.....	87
4.4.4	Tag Memory Footprint vs. Traffic Intensity and Speed Limit.....	88
4.5	Conclusion and Future Work	89
5	Granular Cloning: Intra-Object Parallelism in Ensemble Studies ⁴	90
5.1	Introduction.....	90
5.2	Related Work	92
5.3	Background: Cloning in Parallel Simulations.....	96
5.4	Granular Cloning	98
5.4.1	Intuition.....	98
5.4.2	Formulation.....	101
5.4.3	Optimizations and Heuristics	109
5.5	Expected Performance for Agent-Based Granular Cloning.....	111

5.6	Empirical Evaluation	112
5.6.1	Conceptual Model: CA Traffic Simulation.....	113
5.6.2	Conceptual Model: Land Use Model	114
5.6.3	Conceptual Model: ITLUM	115
5.6.4	Evaluation: CA Traffic Simulation on a Single Process	116
5.6.5	Evaluation: ITLUM on Two Processes.....	118
5.6.6	Evaluation: 2x4 XOR-HOLD (Event-Scheduling on Eight Processes).....	119
5.7	Conclusion and Future Work	120
6	Conclusions and Future Work.....	122
	APPENDIX A. PERMISSION REQUESTS TO REUSE MATERIALS	124
	References.....	127

LIST OF TABLES

Table 1	Speedup for cellcount/timesteps = 100 and start of decision points commences at timestep=0	118
Table 2	Speedup for cellcount/timesteps = 100 and start of decision points commences halfway through	118
Table 3	Speedup for cellcount/timesteps = 10 and start of decision points commences at timestep=0	119
Table 4	Speedup for cellcount/timesteps = 10 and start of decision points commences halfway through	119

LIST OF FIGURES

Figure 1	Classification of Transportation Simulations (Jorge Laval, Creative Commons)	9
Figure 2	High-level location prediction procedure	18
Figure 3	Krumm visualization	22
Figure 4	Past & Future Tree visualization	24
Figure 5	Probability map generation algorithm	26
Figure 6	The Future Tree is used to retrieve the potential locations of the vehicle Δt into the future. In this case, we identify all vertices that are reached from the current location within a time span of 23 seconds. The shaded line illustrates the possible locations at that time.	27
Figure 7	Route generation algorithm	28
Figure 8	Illustration of the space-efficient storage scheme. Trajectory 292 is stored (only once) at location 0x46b3. Since it contains Intersection 4 and 5, the lists of these two intersections contain pointers to the trajectory's memory location.	30
Figure 9	Error plots of the mean absolute difference between actual and predicted rectilinear distance to destination	33
Figure 10	Error plots of the mean rectilinear distance between actual and predicted destination.	33
Figure 11	Error plots of the mean absolute difference between actual and predicted rectilinear distance to destination.	34
Figure 12	Error plots of the mean rectilinear distance between actual and predicted destination.	35
Figure 13	The red cells are assumed to be more likely because they are consistent (in efficiency) with the partial route taken so far (rectilinear distance).	44
Figure 14	Krumm's efficient route prediction algorithm.	45

Figure 15	The edges have weights associated with them that set the strength of the value being sent. The circles are neurons that typically send values close to 0 or close to 1.	47
Figure 16	All trajectories that contain the cell 24 are being pointed to by the column 24. Empirical distributions of any Markov order can be obtained quickly by traversing this column.	48
Figure 17	The next zone visited by the taxicab is uncertain, but is restricted to eight possible zones.	51
Figure 18	Modified efficient route algorithm (changes in bold)	52
Figure 19	Cumulative likelihood for recapturing the target vehicle at the next cell transition using the path efficiency models (Krumm) with a $p=0.75$ parameter.	53
Figure 20	Cumulative likelihood for recapturing the target vehicle at the next cell transition using 1st, 2nd, and 3rd-order Markov models (Past Tree implementation).	55
Figure 21	Cumulative likelihood for recapturing the target vehicle at the next cell transition using 1st, 2nd, and 3rd-order Markov models with a cascading fallback mechanism towards lower orders.	57
Figure 22	Cumulative likelihood for recapturing the target vehicle at the next cell transition using the modified Krumm models and dynamic data.	58
Figure 23	Cumulative likelihood for recapturing the target vehicle at the next cell transition using a feed-forward artificial neural network with one hidden layer, consisting of seven nodes.	60
Figure 24	Cumulative likelihood for recapturing the target vehicle at the next cell transition using the best-performing submodels.	61
Figure 25	Peachtree Street from 10th Street to 14th Street (Map data: Google, 2018)	68
Figure 26	Flow graph of trajectory generation procedure for a single segment	69
Figure 27	Actual and Predicted Trajectories in the Space-Time Diagram	70
Figure 28	The traditional execution approach (left) and the proposed approach (right)	80
Figure 29	Modified collision detection logic.	84
Figure 30	Execution time results by scenario	86

Figure 31	Speedup for different relative speed tendencies	88
Figure 32	Memory requirements for storing the tags	88
Figure 33	Rather than duplicate the entire LP, granular cloning only duplicates the state that changes.	99
Figure 34	Time-space diagram for granular cloning split.	100
Figure 35	Lower growth rates yield better performance for granular cloning.	112
Figure 36	ITLUM Federation Sequencing Pipeline	115
Figure 37	Speedup of the transport model of three different cell-to-timestep ratios	116
Figure 38	Speedup of the transport model if uniform scenario alterations are drawn in restricted time intervals	117

SUMMARY

The proliferation of sensors and data is the catalyst for traffic simulators to help monitor and manage transportation systems in real-time to improve key performance metrics and overall efficiency. As a specific instance, consider vehicle tracking. The future path of an arbitrary driver on a road network is uncertain (without any further information), while the benefit of using this knowledge grows each year. The **D**ynamic **D**ata-**D**riven **A**pplication **S**ystem (DDDAS) paradigm applied to transportation simulation mitigates this uncertainty.

One contribution of this thesis is to define a DDDAS framework for managing transportation systems online. Real-time data is fed into a traffic simulation, which generates future states of the road network. Because the results of the system need to be both faster than real-time and accurate, the acceleration of the simulation execution and the accuracy of the prediction models are critical. The system can be used by both decision makers, such as traffic control agencies and law enforcement, as well as drivers with access to a mobile computer. Applications are abundant, as accurate transportation simulation results not only help decision makers from a top-down perspective (e.g., analyzing the impact of unexpected congestion), but also the end users themselves from a bottom-up perspective (e.g., automatic suggestions of user-preferred routes in on-board technology).

This thesis *proposes a framework for vehicle tracking* and trajectory prediction, as well as the efficient execution of simulations for transportation and other applications. The vehicle tracking framework follows a vehicle under investigation by detecting the vehicle from sensors, predicting likely future locations of the vehicle, and relocating the sensors in order to reacquire the vehicle.

With regard to *trajectory prediction*, a *data structure* that stores previously observed vehicle paths in a given area in order to predict the forward trajectory of an observed vehicle at any stage is proposed. Incomplete vehicle trajectories are looked up in a Past Tree, to predict future trajectories in another tree structure - a Future Tree. For a substantial portion of the trip, the Past Tree scheme was found to outperform a competing scheme which only utilizes the driver's partial trajectory before the point of prediction.

Vehicle trajectory prediction models are compared in the context of a relatively large real-world dataset. The various approaches to route prediction have varying degrees of data required to predict future vehicle trajectories. Several extensions to previously proposed approaches were developed. Three approaches to vehicle trajectory prediction are examined to assess their accuracy on a realistic urban road network. These include an approach based on the intuition that drivers attempt to reduce their travel time, an approach based on neural networks, and an approach based on Markov models. These results highlight the benefit of exploiting dynamic data to improve the accuracy of transportation simulation predictions. A cascading Markov model scheme outperformed all other tested models.

In the context of accelerating transportation simulations, there are often substantial similarities among these different simulation replications. In other cases, output statistics are independent of certain simulation computations. *This thesis contributes computational methods to exploit these properties to speed up the simulation execution time*, when the output statistics of interest focus on one or more attributes such as the trajectory of a certain “target” vehicle. By focusing on correctly reproducing the behavior of the target vehicle and its interaction with other modeled entities across the different replications and modifying the event handling mechanism, the execution time can be reduced. Up to 16-fold speedup were obtained compared to the conventional approach of running one instance per scenario.

An application-independent method for computation sharing is proposed as well. Many runs of a computer simulation are needed to model uncertainty and evaluate alternate design choices. Such an *ensemble* of runs often contains many commonalities among the different individual runs. Simulation cloning is a technique that capitalizes on this fact to reduce the amount of computation required by the ensemble. Granular cloning is proposed that allows the sharing of state and computations at the scale of simulation objects as small as individual variables, offering savings in computation and memory, increased parallelism and improved tractability of sample path patterns across multiple runs. The ensemble produces results that are identical to separately executed runs. Whenever simulation objects interact, granular cloning will resolve their association to subsets of runs through binary operations on tags. Algorithms and computational techniques required to efficiently implement granular cloning are presented. Results from an experimental study using a cellular automata-based transportation simulation model and a coupled transportation and land use model are presented providing evidence the approach can yield significant speed ups relative to brute force replicated runs.

1 INTRODUCTION

With the advent of pervasive data feeds from mobile or static sensors, the proliferation of data, the widespread use of mobile applications by the public, and input/output devices used in the Internet of Things (e.g., augmented reality systems), real-time instrumentation data can be directly used in various applications to improve various key performance metrics. The abundance of real-time data has implications for a wide range of sectors, including public health, finance, and transportation systems. There exist many applications in these areas and the third among the listed examples – that is, transportation systems – serves as the focal point of much of this thesis.

For transportation systems, real-time data can be used to improve the safety, fuel-efficiency, and travel time of drivers at various levels. Some broad example applications include:

- *Travel time estimation.* Cameras can detect a vehicle at different points and times on a freeway and compute travel time estimates to other drivers who may wish to take an alternative route.
- *Safety.* Many modern vehicles already include crash avoidance or automatic emergency vehicle notification systems. Detailed information can immediately be communicated to nearby first responders, even when the driver is incapacitated.
- *Fuel efficiency.* Modern car engines use an array of sensors and controllers for fuel-efficient operations. An example is the recent trend of equipping combustion engines with direct injection. Various sensors dynamically determine the suitable amount of air and fuel to spray into the combustion chamber in order to improve fuel efficiency.
- *Sustainable urban development.* Although the transportation application discussed in this thesis mainly concerns vehicle tracking, this thesis will touch upon other areas. For

example, new computational techniques to accelerate investigations of the impact of transportation on sustainable urban development are examined.

(Darema, 2004) characterizes Dynamic Data-Driven Application Systems (DDDAS) as follows:

In DDDAS instrumentation data and executing application models of these systems become a dynamic feedback control loop, whereby measurement data are dynamically incorporated into an executing model of the system in order to improve the accuracy of the model (or simulation), or to speed-up the simulation, and in reverse the executing application model controls the instrumentation process to guide the measurement process.

DDDAS presents opportunity to create new capabilities through more accurate understanding, analysis, and prediction of the behavior of complex systems, be they natural, engineered, or societal, and to create decision support methods which can have the accuracy of full-scale simulations, as well as to create more efficient and effective instrumentation methods, such as intelligent management of Big Data, and dynamic and adaptive management of networked collections of heterogeneous sensors and controllers. DDDAS is a unifying paradigm, unifying the computational and instrumentation aspects of an application system, extends the notion of Big Computing to span from the high-end to the real-time data acquisition and control, and it's a key methodology in managing and intelligently exploiting Big Data.

In the context of this thesis, the *application system* component of DDDAS concerns vehicle *traffic*; however, one could envision other systems (not necessarily rooted in transportation) where the contributions may prove useful – including in three spatial dimensions. Microscopic traffic simulations are often used to gain insights into physical road systems where one wishes to investigate various operational scenarios. The *dynamic data-driven* component of DDDAS may be used to initialize the traffic simulation. Even if rush hour data had been available for use in a conventional traffic simulation, there is no guarantee that the intensity of the traffic

and/or the road context (with potential blockages and detours) would be representative of the physical system during the time window under investigation. The strength of DDDAS, in the context of microscopic traffic simulations that focus on the behavior of an individual entity, is that the initial state, and subsequent states, if the simulation state is updated in an on-line fashion, reflects an abstraction of the ground truth of the system under investigation (SUI). Even a slightly mischaracterized initial state may lead to dramatic inaccuracies as the model evolves. Whereas analytical methods *alone* tend to fail when evaluating unfamiliar scenarios (e.g., rare events such as traffic accidents at certain locations), traffic simulations with inaccurate data are likely to produce unusable output statistics - especially if the output statistics focus on observations of a single entity, rather than system-level aggregate metrics. The DDDAS paradigm offers an effective answer to both issues.

Despite the potentially useful information supplied by mobile geospatial applications, only a miniscule fraction of intended trips is specified by drivers. Developers have not leveraged these elements into an ecosystem that allows stakeholders and drivers to make more informed decisions, without the need of troublesome user interaction in the form of manual data entry. While there has been research on static trajectory prediction - without accommodating functionality for temporal queries - this thesis proposes a dynamic framework, where data and simulation are used together to make dynamic (temporal) predictions on the future trajectory of a driver. This framework does not assume that any personal data for the vehicle under investigation (VUI) has been previously collected. This allows the prediction models outlined in this thesis to be used for any VUI in an area where trajectory data is available. The drawback is that VUI's who differ significantly from the behavior of the general population are difficult to track. As an example, consider a VUI which exhibits evasive and inefficient actions. A further limitation is that VUI's with a small and usual set of trajectories are not exploited by the models. Related works that address these scenarios will be summarized in Chapter 2. Depending on the objectives

of a simulation study, the marginal benefit of more precise and larger input data sets can range from nonexistent to critical. The application of this thesis examines the dynamic behavior of a specific VUI, which is sensitive to the accuracy of the data supplied to the application system. Suppose we wish to predict the position of a VUI at a time when a busy local event is taking place, and a conventional microscopic traffic simulation is initialized with estimated road conditions. Not only might the VUI have a high propensity to alter his/her path to a less busy path or detour (assuming he/she is informed of such paths), but the temporal estimates within typical road sections are likely to be inaccurate due to unanticipated areas of congestion.

Because the analysis approach, in this context, is a *microscopic* traffic simulation, the computational load of even a moderately sized road network is significant. In order to tackle this challenge, a scheme to speed up the simulation execution will be presented. This scheme can be used in both serial and parallel computing architectures.

This thesis builds upon the following concepts: DDDAS, microscopic traffic simulation, trajectory prediction, and simulation cloning. These concepts will be discussed in the following subsections.

1.1 Dynamic Data Driven Application Systems

In **Dynamic Data-Driven Application Systems** (Darema, 2004), real-time measurement data of a physical system are unified with the simulation system in a dynamic control loop. Data from sensors can be used by the simulation to improve the accuracy of its working model or speed up the execution of the system in order to improve the operation of the physical system or to adapt the measurement process itself. For example, one might use sensor data to reconfigure the sensor network or organize and filter the obtained data. Overall, the intent is to enhance the monitoring of the system under investigation (SUI) – e.g., for real-time decision support – and leverage the resulting benefits (including improvements in the SUI).

The paradigm has spawned fruitful research in many areas of science and engineering since its inception in 2000. Example DDDAS applications include:

- *Monitoring the spread of wildland fires.* In (Mandel J. e., 2008) differential equations characterizing spread based on available fuel are derived using coefficients derived from historical wildfire data. A method that incorporates Kalman filters allows accurate tracking of fire propagation (temperatures) in the affected region, even when the initial location of the fire is chosen incorrectly. An extension of the simulation model also incorporates wind as a factor (Babak, 2009) and another extension of the data assimilation process using Sequential Monte Carlo methods is proposed in (Xue H. F., 2012). In another extension of the original model, the coupled fire spread model starts at a perimeter rather than a point (Mandel J. J., 2012). A DDDAS system for tracking specific fires using unmanned aerial vehicles is described in (Peng, 2014).
- *Emergency and disaster management.* (Madey G. R., 2007) shows how cell phones can be used in an emergency or disaster situation as ad hoc mobile sensors. Subsequent work also investigated similar data for uses in efficient urban management (Steenbruggen, 2015).
- *Construction and waste management.* (Song, 2012) proposes a scheme where real-time construction operation parameters are captured with sensors and used in a simulation of heavy construction operations. As the simulation receives data from the tracking sensors, the state of the models is constantly updated to reflect the changing physical system in order to improve the accuracy of future schedules. (Vahdatikhaki, 2014) presents a framework for a detailed capturing of the construction equipment. The authors of (Parashar, 2006) applied their DDDAS software stack - which was originally developed for their oil field DDDAS project (specifically, reservoir management) - for the management of the Ruby Gulch Waste Repository.

- *Monitoring weather conditions and ocean forecasting.* (Brotzge, 2006) discusses a DDDAS to forecast surface weather conditions with a network of small, low-power, and inexpensive radars that operate in a collaborative and adaptive (based on user needs and weather conditions) fashion. (McGovern, 2011) describes an approach to identify key features from spatiotemporal data for severe weather prediction (tornadoes). (Patrikalakis, 2004) introduces a DDDAS framework for modeling the evolution of the ocean state (surface water temperature, biological systems etc.). In climate science and meteorology, real-time (or near real-time) data acquisition has been coupled with relevant application systems prior to the introduction of DDDAS. For example, (Kunkel, Changnon, Lonquist, & Angel, 1990) discuss how the Midwestern Climate Information System (MICIS) uses near-real-time precipitation and temperature data to predict soil moisture and crop yield risks (collected daily). (Reynolds & Marsico, 1993) augment the analysis of global sea surface temperatures (which is driven by in-situ sensors and satellite data) with the simulation of temperatures in ice-covered regions. These simulated temperatures become external boundary conditions for the updated solution, in order to remove high-latitude satellite sensor biases.
- *Optimizing supply networks.* (Celik, 2010) describe a DDDAS framework where the fidelity of a simulation model is updated (*selective data update*) on-the-fly based on the computational resources available. The system is tested on a semiconductor supply chain as a case study.
- *Improving healthcare operations.* (Barjis, 2011) surveys healthcare simulation and future research. As one of the non-traditional simulation efforts, Barjis highlights the benefits of running simulation models in parallel with routine operations and connecting the simulation model with the information systems of the facilities where data are collected dynamically (Gaba, 2004) and (Darema, 2004)). Real-time simulations are also investigated in (Bahrani, 2013) to determine the impact of resource allocation on wait times in a case study of an Ontario hospital.

DDDAS has also been used in transportation systems (see (Fujimoto R. R.-K., 2006) and (Fujimoto R. M.-K., 2007)), which is the area discussed in this thesis. As a specific application, a vehicle tracking framework will be discussed where a vehicle under investigation is tracked by detecting the vehicle from sensors, predicting likely future locations of the vehicle, and relocating the sensors in order to reacquire the vehicle. Here, the prediction step makes use of real-time trajectory data, which is used by the microscopic traffic simulation to give temporal estimates of the VUI's future location.

1.2 Microscopic Traffic Simulation

Much of the work described in this thesis is related to traffic simulation in some fashion, whether it is the acceleration of the simulation execution, the evaluation of a trajectory prediction model, or the calibration of a model. Researchers have modeled transport and vehicle behavior on computers since the 1950's (Gerlough, 1955) for the purpose of enhancing the design and operation of traffic systems. For example, one could evaluate proposed infrastructure alternatives and inspect the impact on the mean travel time through the affected area. The impact of different signal timing procedures on congestion is another example application. Sometimes, there are analytical approaches (e.g., queueing theory) that could be used as an alternative to computer simulation. However, if numerous real-world entities and their interactions are incorporated, the benefits of using the analytical approach (simplicity, development speed, compactness and tractability) vanish. On the other hand, using real-world experiments on a road or road network may be prohibitively expensive or cause unacceptable conditions for drivers.

At the conceptual level, the ideas of this thesis mainly focus on ground vehicle traffic without bicycles and pedestrians, although one could modify the concepts to apply them to other transportation modes as well. Another conceptual distinction concerns the granularity in the model used. *Microscopic* models view the entities at a detailed level where vehicles are characterized individually and their behavior is explicitly described. In a *macroscopic* model, the

vehicles are not individually separated, but incorporated in a more coarse-grained aggregate element. *Mesosopic* models fall in between the previous two extremes and look at clusters of vehicles as the most atomic entity in the system. The objectives of a simulation study determine what level of detail is the most prudent. More details will usually increase the development time as well as the execution time of the simulation, while they can lead to more accurate output statistics, depending on what is analyzed. This thesis will focus on microscopic traffic simulation.

Simulation models are distinguished by whether time and space are modeled as continuous or discrete quantities. Figure 1 illustrates simulation model examples that are used at the different intersections of these aspects. For example, partial differential equations can be used to model the system if both time and space are assumed to be continuous. The algorithms presented in this thesis are most easily implemented in discrete-time and discrete-space simulation models. However, they can be incorporated in other categories as well. In the trajectory prediction models (Chapters 2 and 3) one could encode road segments, as opposed to cells, as decision points, and in the accelerated simulation algorithm of Chapter 4, the simulation execution is modified when a certain type of vehicle changes its behavior upon an encounter with another type of vehicle. These properties are independent of whether time and space are modeled as continuous or discrete.

Types Of Simulation In Transportation

Time	State	Space		
		Continuous	Discrete	N/A
Continuous	Disc.	<u>Real Transportation Systems *</u> Traffic flow, pedestrians Dynamic traffic assignment		<u>Discrete Event Systems *</u> queueing inventory manufacturing
	Cont.	<u>PDE</u> Traffic flow models Pedestrian models		<u>ODE</u> vehicle motion car suspension queueing (fluid approx)
Discrete	Disc.		<u>Cellular Automata *</u> Traffic, pedestrians Land use Urban sprawl Random Number Generation	<u>Discrete Event Simulation *</u> queueing inventory manufacturing
	Cont.	<u>Car-following models *</u> <u>Microscopic traffic flow models *</u>	<u>Numerical PDE methods</u> Godunov, Variational	<u>Numerical ODE methods</u> Euler, Runge-Kutta <u>time-series *</u> ARIMA
N/A	Disc. or Cont.	<u>Monte Carlo method *</u> : use of pseudo-random number Simulation of static probabilistic problems Integration, Optimization		<u>Econometric models</u> trip generation, distribution, modal split <u>Optimization</u> static traffic assignment

Figure 1: Classification of Transportation Simulations (Jorge Laval, Creative Commons)

Because the methods of this thesis are most easily implemented and efficiently executed in discrete-time and discrete-space models, it is worthwhile to analyze the model listed at the relevant intersection in Figure 1: *Cellular Automata (CA)* models were developed by John von Neumann and Stanislaw Ulam in the 1940's at Los Alamos National Laboratory (Schiff, 2008). CA's are *grids* that consist of *cells*. Each cell assumes a particular state and might change its state, as the simulation *clock* advances, subject to a set of rules. Conway's Game of Life (Gardner, 1970) is perhaps the most well-known CA, where cells are either alive or dead and each cell changes its state based on the state of its neighbor cells. CA's find application in many fields of science (Wolfram, 2002), engineering, mathematics, and computing. Many of the ideas presented in this thesis are evaluated on a CA model for road traffic (Nagel & Schreckenberg, 1992) that

was developed by the German physicists Kai Nagel and Michael Schreckenberg and will be described next.

The Nagel-Schreckenberg model (NS) is a CA that is used to model roadway traffic. Emergent phenomena such as congestion appear in the model, just as they do in the real world. In addition, it has been shown (Daganzo, 2006) that NS is equivalent to other vehicle-following models. The NS model (Nagel & Schreckenberg, 1992) maps an area typically occupied by a car into a cell. A cell encodes whether it is occupied by a car or not and cars have an integer speed attribute associated with them. Additionally, a global maximum speed is usually defined. At each simulated timestep, four compact steps are repeated, in order, simultaneously to all cars. In order to model the desire of drivers to reach their destination in an efficient manner, the speed attribute of the cars is first incremented. In order to avoid collisions, cars will then reduce their speed attribute if they were to collide with the car in front of them at the current speed, assuming the vehicle in front of them remains stationary. To model the uncertainty in the ability of human drivers to maintain a constant speed, the speed attribute of each car is reduced with probability p (a fixed parameter). Finally, the positions of the cars is updated in the grid. Modifications to NS have also been developed since the publication of the initial paper, e.g., multiple lane models, models with more traffic elements such as traffic lights, models with adaptive parameters that change during runtime etc. Section 4.3 will discuss the NS algorithm in more detail as it is central for the ideas presented in Chapter 4.

Tangentially related to this thesis are Intelligent Transportation Systems (ITS). ITS use communication, information, and computation technology on transportation infrastructure, vehicles, and users to offer services with the objective to enhance the safety (including environmental aspects), mobility, information awareness, efficiency, and productivity of users and stakeholders (e.g., traffic management) (Technology, n.d.) (EU Council, 2010). These technologies include smartphones, cameras, audio sensors, digital message boards, RFID, and

other tools. Example applications with respect to safety include a mechanism to automatically call an emergency hotline, notifications for drivers on the road about how to deal with hazardous road conditions, and general surveillance for homeland security.

The framework described in this thesis can be integrated in intelligent transportation systems (ITS) to improve general driving safety and efficiency, as not only decision makers (e.g., law enforcement and traffic control agencies) can act on supplied information by regulating traffic, but also the drivers themselves may act in a favorable fashion as they are informed on a mobile computer. Applications are abundant, as accurate transportation simulation results not only help decision makers from a top-down perspective, but also the end users themselves from a bottom-up perspective (e.g., fuel-efficient, fast, convenient, or safe routes without the user needing to input his/her destination into mobile applications or on-board augmented reality systems).

The vehicle tracking framework developed for this thesis is an example ITS that not only has uses in security and planning, but also for end users, as the examples at the beginning of this chapter illustrate.

1.3 Trajectory Prediction

Trajectory prediction refers to predicting any future location of an object based on its previous or current position(s), velocity, or other contextual data in different dimensions. Trajectory prediction finds use in the perception of animals (Gallistel, 2007), in air traffic to avoid collisions (Gong, 2004), in interactive environments to avoid excessive communication (Cai W. F., 1999), in localization (Lin & et al., 2016), and many other areas. In the context of this thesis, the object of interest is a vehicle. A related, but easier, problem is destination prediction, where only the final position of the vehicle is relevant - not the path to reach any of the candidate destinations. Several interesting ideas have been proposed for the destination prediction problem. One example is based on the intuition that drivers pick efficient routes (Krumm J. , 2006).

Another destination prediction model that recently won a Kaggle competition is based on recurrent bidirectional neural networks (de Brébisson, 2015). Chapters 2 and 3 will discuss existing and new trajectory prediction methods in more detail.

1.4 Simulation Cloning

Trajectory prediction models usually only estimate spatial values in the future. One could associate temporal information to these values with analytical models. However, to capture rare and context-specific information, traffic simulation may generate data that would otherwise be difficult to extrapolate from analytical models. At the scale of granular trajectory prediction, one would degrade the accuracy if the traffic simulation model were too coarse. At the same time, the execution of *microscopic* traffic simulation may be computationally expensive for large road networks and several simulation replications. Various algorithms exist to accelerate simulations and a suitable technique depends on the specific problem characteristics. A family of acceleration techniques use the concept of *simulation cloning*, which allows multiple alternative futures to be evaluated in parallel from a common decision point in the simulation execution; a mechanism based on *virtual logical processes* allows cloning to occur in the background in a computationally efficient manner where common computations are shared within a single physical process (Hybinette M. a., 2001). In time-sensitive situations, analysts may use simulation cloning in an interactive manner to evaluate different alternatives in a simulation model. Oftentimes, the number of possible scenarios is unbounded or many scenarios are not feasible based on real-world constraints. In these situations, it is useful for a domain expert to enumerate a small set of key scenarios and quickly evaluate them based on metrics that are important in the particular domain. The possibly nonlinear relationships among the different metrics and how they need to be prioritized may also need to be evaluated in an ad-hoc manner by the domain expert. Section 4.2 and Section 5.2 will discuss related works in this area in more detail before delving into the main computational contributions of this thesis. Section 5.3 discusses simulation cloning in detail.

1.5 Research Contribution and Thesis Organization

The main contributions of this thesis are as follows:

- A real-time DDDAS framework for monitoring transportation systems in an urban environment is proposed. In this application, the goal is to predict future system states of the road network. The framework tracks a vehicle under investigation by detecting the vehicle from sensors, predicting likely future locations of the vehicle, and relocating the sensors in order to reacquire the vehicle. The procedure can also be utilized for other transportation entities (e.g., pedestrians and bicyclists) and, more generally, for different application areas at various scales and dimensions.
- A fast-access and compact data structure that stores previously observed vehicle paths in a given area in order to predict the forward trajectory of an observed vehicle at any stage is proposed. Across a synthetic and real data set, the algorithm outperforms a competing scheme in accuracy and access time; the competing scheme only uses the driver's partial trajectory before the point of prediction instead of using historical path data. The data structure can be used for other trajectory data (including higher dimensional data with multiple attributes) if the space that the object displaces is mapped to discrete points. If the space is continuous, one can superimpose a grid or define suitable clusters to discretize it.
- To highlight the value of computationally efficient and accurate trajectory models to drive on-line data-driven simulations, trajectory prediction models are compared against each other in the context of a relatively large real-world dataset. The approaches to route prediction have varying degrees of data required to predict future vehicle trajectories. Three approaches to vehicle trajectory prediction, *along with original extensions*, are examined to assess their accuracy on an urban road network. These include an approach based on the intuition that drivers attempt to reduce their travel time, an approach based on neural networks, and an approach based on Markov models. These results highlight the benefit of exploiting dynamic

data to improve the accuracy of transportation simulation predictions. A cascading Markov model scheme (an original contribution) outperforms all other tested models. It may be possible to utilize the cascading mechanism in completely different areas where predicting sequence patterns is useful (for example, to leverage cache hierarchy access patterns in computer systems).

- A computational method to accelerate transportation simulations is developed. By exploiting similar properties among different simulation replications, the simulation execution time can be reduced, when the output statistics of interest focus on one or more attributes such as the trajectory of a certain “target” vehicle. Depending on the model characteristics, it is possible to approach a speedup proportional to the number of replications, compared to the conventional approach of running one instance per scenario. With some mild assumptions, the scheme can also be used for other agent-based models where objects travel within a network.
- A general computational method, termed granular cloning, is proposed to accelerate ensemble studies. State sharing occurs at any scale allowed by the computer system and only the state that changes from the shared state is physically cloned.

Chapter 2 discusses an efficient Markov model trajectory prediction algorithm and motivates trajectory prediction in the context of DDDAS, Chapter 3 compares several trajectory prediction models on a large dataset, Chapter 4 introduces an algorithm to accelerate the execution certain Monte-Carlo simulation models where the output statistics are collected on a single object that travels in a network, and Chapter 5 outlines the granular cloning approach to expose parallelism at the scale of any simulation object.

2 TRAJECTORY PREDICTION WITH PAST AND FUTURE

TREES ¹

¹This chapter includes content from (Pecher P. H., 2014) and (Fujimoto, et al., 2014). © 2014 IEEE. Reprinted, with permission, from reference found under (Pecher P. H., 2014) In reference to IEEE copyrighted material which is used with permission in this thesis, the IEEE does not endorse any of Georgia Institute of Technology's products or services. Internal or personal use of this material is permitted. If interested in reprinting/republishing IEEE copyrighted material for advertising or promotional purposes or for creating new collective works for resale or redistribution, please go to http://www.ieee.org/publications_standards/publications/rights/rights_link.html to learn how to obtain a License from RightsLink.

2.1 Introduction

Computer simulation of road networks and transportation entities is a valuable tool for managing traffic and designing new construction projects. Some transportation models are small in scale and make many simplifying assumptions; others can be very sophisticated and computationally demanding. Many dynamic factors are present in the real world that are difficult to model and predict. For example, say, a particular event takes place that attracts one specific demographic; a simulation model can be made more accurate if it takes into account recently observed trajectories, rather than static turn frequencies at decision nodes. One way to view this information is via decision trees holding relative frequencies at each branching point. Observed trajectories can be stored in a tree data structure (Past Tree) where nodes represent past trajectories up to the given decision point and hold references to other trees (Future Trees) that store observed frequencies of the remaining trajectory. If the target entity is currently in an area

where trajectories have been observed in the past, population path similarity can be used to predict where a vehicle will turn or to generate realistic trajectories. For reasons that will become obvious in Section 2.4.1, we will subsequently refer to Past and Future Trees as simply Past Trees. We will also sometimes use the expression Method of Past Trees (or simply, Past Trees in the charts of Section 2.5) to specifically refer to the querying operation on Past Trees that outputs location estimates (see Section 2.4.2).

Past Trees are useful for dynamic data-driven application systems - DDDAS (Darema, 2004). The ability to modeling individual vehicles accurately is needed in a variety of applications. The analysis of emergent, macro-scale phenomena in real-time data-driven types of simulations can be of great interest to various stakeholders. To optimize systems for efficiency, for example, it is possible to model traffic congestion (Fujimoto R. M.-K., 2007) and evacuation models (Chaturvedi, 2006) with a high degree of validity. The data centers of web search providers have clustered equipment based on the expected workload in different domains (Marin, 2013). Past Trees can model consumer preferences and assist in parameter tuning (Ye, 2008) and bottleneck analysis of computer networks. Past Trees may also be used in online simulations to determine an accident from cell phone data (Madey G. R.-L., 2006) (Madey G. R.-L., 2007) or in law enforcement applications (Fujimoto R. A., 2014).

2.2 Motivating Context: DDDAS

In order to set the problem context and motivation, a computational framework to model the movement of a mobile target vehicle in a road network (without access to perfect information) is first described. The objective is to predict the final destination of the target. This framework dynamically selects an appropriate prediction model at runtime based upon what data is currently available.

Consider a directed graph for the road network $G(V, E)$ with nodes representing intersections and links road segments. A *target entity* E_T , resides at vertex n_0 at time t_0 . E_T travels

through G on directed edges and through vertices. If E_T is on an edge, it is assumed, for simplicity, that E_T is also mapped to the vertex to which the edge points. $\mathbf{n}=[n_0, n_1, n_2, \dots, n_f]$ and $\mathbf{a}=[a_0, a_1, a_2, \dots, a_f]$ are used to denote the node-vector and the arc-vector of E_T 's true, realized path. $n(t)$ and $a(t)$ return the nearest vertex that is being approached by E_T and E_T 's current edge for a given time t , respectively. $s(t)$ returns a scalar between 0 and 1 denoting E_T 's position at t on $a(t)$.

At some later time $t=t_f$, we wish to find E_T 's exact location (i.e., $a(t_f)$ and $s(t_f)$). In order to accomplish this final goal, *probing entities* $E_1, E_2, E_3, \dots, E_S$ are intermittently mobilized within the network, and each may test whether any one point in G contains E_T at timestamps $t_1, t_2, t_3, \dots, t_k$.

The lag (in time units) between two consecutive rediscovery attempts is determined by ensuring that the likelihood of losing E_T at the next test timestamp does not exceed a user set tolerance probability p_{loss} . At the same time, we wish to maximize the lag within a test cycle, Δt_i and assume this likelihood increases monotonically as the candidate test interval decreases.

For a fixed p_{loss} , the critical lag Δt_i^* is determined so that the probing entities may attempt to observe E_T after the sum of Δt_i^* and the previous test timestamp. This is accomplished by enumerating all paths that could potentially be reached by E_T after Δt_i^* time units. This is implemented with a microscopic, and data-driven traffic simulation that is called by a tree construction algorithm; see FIND-OFFSET below.

If at any test timestamp, none of the S sensors observes E_T , either E_T is lost or E_T stopped in the maximal hull that connects all vertices that can be reached after Δt_i^* time units from the time of E_T 's last observed vertex. The search can occur in structured fashion, from the last observed point to the edge of the hull with S search paths that occur in parallel (along regions that

have the highest probability mass). In this final mode, each sensor has a fixed speed of testing a segment on an edge (length per time unit).

Once S and p_{loss} are determined, the S mobile probing entities are placed at the S locations most likely to contain E_T at each rediscovery attempt, the third phase of the DDDAS loop described earlier. Therefore, the key is to construct an accurate conditional probability map of E_T 's location in G at time t_{i+1} given the last known positions of E_T at time t_i, t_{i-1}, \dots, t_0 . Several methods are presented to estimate $(a(t_{i+1}), s(t_{i+1}))$.

To summarize, the high-level framework proceeds as follows:

```

1  Fix S and ploss
2  While( E_T was detected in the last test )
3      Determine the critical lag, Δt_i* (FIND-OFFSET)
4      Determine the probability map @t_i+Δt_i* (PROBABILITY-MAP)
5      Place the E_1, E_2, E_3, ..., E_S at the S most likely locations
6      Test at t_i+ Δt_i*
7  Find E_T's stopping point within the reachable hull

```

Figure 2: High-level location prediction procedure

2.2.1 FIND-OFFSET: Determining the critical lag

The objective of FIND-OFFSET is to find the critical lag to the next test, the maximum time that E_T can go unobserved, while still satisfying p_{loss} . This is done with a tree construction algorithm, where the vertices represent intersections and arcs represent roads between those intersections. The algorithm first starts in a global search (roughly, level-by-level) and then proceeds to engage in a local search with an S -element window (node-by-node) until the critical lag is identified. If p_{loss} is 0, the algorithm can be sped up substantially by simply checking for cardinality and side-stepping the call to the evaluation function PROBABILITY-MAP (which is not redundant if p_{loss} is 0 because the final test phase also invokes it).

2.2.2 PROBABILITY-MAP: Path Speculation

In order to properly assess the likelihood of different locations, all relevant and available data must be considered. In accordance with the DDDAS paradigm, the system will automatically switch prediction modes, or 'tiers,' based on the data that becomes available during runtime (crowd-sourced or collected from other sources). There are three prediction tiers that this approach uses, with higher tiers utilizing more specific data, but with more promising accuracy.

- Tier 1: No Data or Only Static Data. If no historical data is available, or only general information (popularity or population density of different zones), inferences can still be made from E_T 's trajectory that has been observed thus far. One particular clue to consider is efficiency. Krumm developed a static prediction model (Krumm J. , 2006) that assigns higher likelihoods to areas that are consistent with the path taken thus far. Thus, inefficient potential trajectories are given less weight. For use in the overall procedure, one additional factor can be conditioned against – time. Since we know when the next test will occur, all cells except the ones that could be reached after the critical lag can be excluded.
- Tier 2: Past Trajectories of the Population. If E_T is currently in an area where trajectories have been observed in the past, population path similarity information can be used. One way to store this information is via decision trees holding relative frequencies at each branching point. This method is discussed in Chapter 2.4.
- Tier 3: Past Trajectories of E_T . The ideal situation is to have actual data available from the person who is being tracked. (Laasonen, 2005) developed a model where similar routes of an individual are clustered. If an observed path is similar to an existing composite path, the composite path is merged with the observed path. Merging, in this case, refers to finding an alignment between two routes (via the dynamic programming procedure to the edit distance problem) and updating the position of intermediate locations with a new average.

A similarity index is used for inserting into the appropriate cluster when a new path is observed and when the future path is to be predicted from a current location. Instead of using a quadratic time edit distance algorithm, Laasonen suggests using a heuristic, *inclusion similarity*:

$$I(r, t) = T/|t|$$

where r refers to the composite path, t refers to the path whose similarity is being computed, and T refers to the number of elements in t that appear in r in the same order. Laasonen further makes use of factors such as the time of day, the particular weekday, relative frequency, and the previously observed *base* (a base refers to a place that is deemed important to a person and is crystallized when the person spends a significant amount of time at such a place).

Our DDDAS can be decomposed into an analytics component and a simulation component. With regard to the former, a data structure that stores previously observed vehicle paths in a given area in order to predict the forward trajectory of an observed vehicle at any stage is proposed (it is compared against one other model in Section 2.5 and against further methods in Chapter 3). With regard to the acceleration of transportation simulation, there are often substantial similarities among these different simulation replications. This thesis explores computational methods to exploit these properties to speed up the simulation execution time in Chapters 4 and 5.

2.3 Related Work

With a motivating context for trajectory prediction discussed, we now reference some related literature. Destination estimation is an active field of research. The constraint on the models selected here, however, is that they must be at a conceptual scope that can be mapped directly to a general transportation simulation model. Prediction models for a highly domain-specific area such as handoffs of a mobile phone from antenna-to-antenna (Cheng, 2003), or

where the target is at an unknown position, but stationary (Sinclair, 2008), for example are inappropriate within general simulation frameworks. Those that extrapolate from direct vehicular motions (e.g., dead reckoning models; see (Cai W. F., 1999)) are too granular and impose an unrealistic behavior on a given vehicle entity. We also ignore models that depend on the identification of a *particular* vehicle and its historical data (Laasonen, 2005).

In Section 2.3.1, we summarize a prediction model based on efficient routes (Krumm J. , 2006). The model predicts based on the idea that drivers tend to make choices that are fairly efficient when attempting to reach their destination. Unlike Past Trees, Krumm's prediction model does not use any historical trajectory data. Nevertheless, we will use it as a benchmark in Section 2.5.

2.3.1 Krumm's Prediction Model

Krumm developed a static prediction model that assigns higher likelihoods to areas that are consistent with the path taken thus far. Thus, inefficient potential forward trajectories are given less weight than those that are consistent with efficient behavior. It is important to note that this model does not utilize previously recorded trajectories, outside the partial trajectory of the vehicle currently being observed. Krumm assigns each cell, c_i , in a region an estimated probability, based on the observed path, S , taken thus far, using Bayes' Law:

$$p(c_i|S) = \frac{p(S|c_i) \times p(c_i)}{\sum_{j=1}^{N_c} p(S|c_j) \times p(c_j)}.$$

Krumm has empirically determined that the probability of an efficient cell transition is 0.625 in a particular experiment in Seattle, WA. For a particular cell c_i in the grid, each cell of the observed trajectory is multiplied by 0.625 if it brings the target closer to c_i , and by 1-0.625 if not. This is then multiplied by an overall cell bias term $p(c_i)$ and normalized. Figure 3 shows how the

algorithm updates the probability map as more information about the route can be used; the cells visited by the actual route are colored green, while the predicted future cells are colored yellow, orange, and red. The higher the red color, the higher the likelihood assigned by the Krumm model. In the rightmost image the leftmost cells are assigned the lowest likelihood as they would be inconsistent with efficient routing behavior.

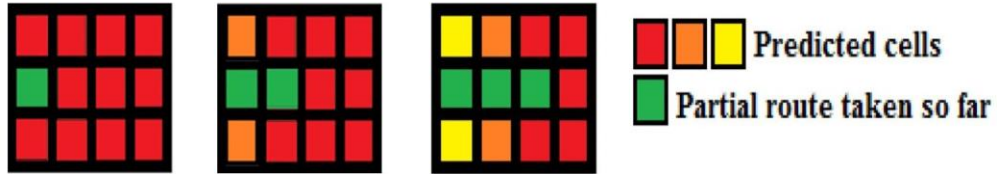


Figure 3: Krumm visualization

2.4 Past and Future Trees

2.4.1 Conceptual Structure and Trajectory Insertions

In order to make proper use of Past and Future Trees, a number of population trajectories must be sampled first. In the following conceptual description of the structures, we assume that this data has been collected already and has been inserted into the structures. A discussion on trajectory insertion (see Figure 4) follows this description. Querying Past Trees to make predictions (i.e., the Method of Past Trees) is discussed in the next section.

Every decision point in a region (e.g., the road on the western side on a particular intersection) is associated with an instance of a Past Tree and each node in a Past Tree links to an instance of a Future Tree (see Figure 4). The vertices of a Future Tree contain traversal counts as well as destination counts. If a partial trajectory, $T_p=(e_0, e_1, \dots, e_t)$, of a vehicle up to a decision point has been observed, the corresponding Past Tree is traversed in the reverse order of T_p (the root corresponds to e_t). Traversing the Past Tree in this fashion is equivalent to conditioning against the

observed trajectory. This means that if no partial trajectory up to e_r , had been observed, we would simply access the Future Tree associated with the Past Tree's root node. If the currently observed partial trajectory does not exist in the tree, the traversal only goes down to the node that does not have the requested child. Once the corresponding Future Tree is discovered, it is traversed to yield the measure of interest. Some of these metrics are discussed in the next two subsections (2.4.2 and 2.4.3). For example, if one wishes to construct an estimated destination probability map, a breadth-first search for positive destination counts can construct a frequency distribution, leading to the desired map.

In Figure 4 below, we assume no previous trajectories have been observed (i.e., this is the first trajectory being observed). Further, this trajectory will cause updates on every Past Tree adjacent to the realized path. However, for illustrative purposes, we will only focus on the updates to the Past Tree corresponding to the western side of Intersection 3, $PastTree_{3,w}$. A vehicle has approached Intersection 3 by first traversing Intersection 1 and then Intersection 2. This partial trajectory is inserted into $PastTree_{3,w}$ (the solid black arcs). After that point, the remaining path is inserted in the Future Trees of all vertices in the path up to Intersection 3 (the lightly dotted arcs), so that any future read request on the Past Tree may access this particular observation for any partial trajectory that follows the same pattern up to the current decision point (irrespective of the length). The Future Tree's traversal counts are incremented if the vehicle traverses the given intersection and the destination counts are incremented if the vehicle stops in the given intersection's proximity (i.e., terminal point along the trajectory). As mentioned before, we assume this is the first trajectory being observed. Had the Past Tree already been partially populated, it could have been the case that no additional arcs would have had to be inserted. However, the counts would still need to be incremented.

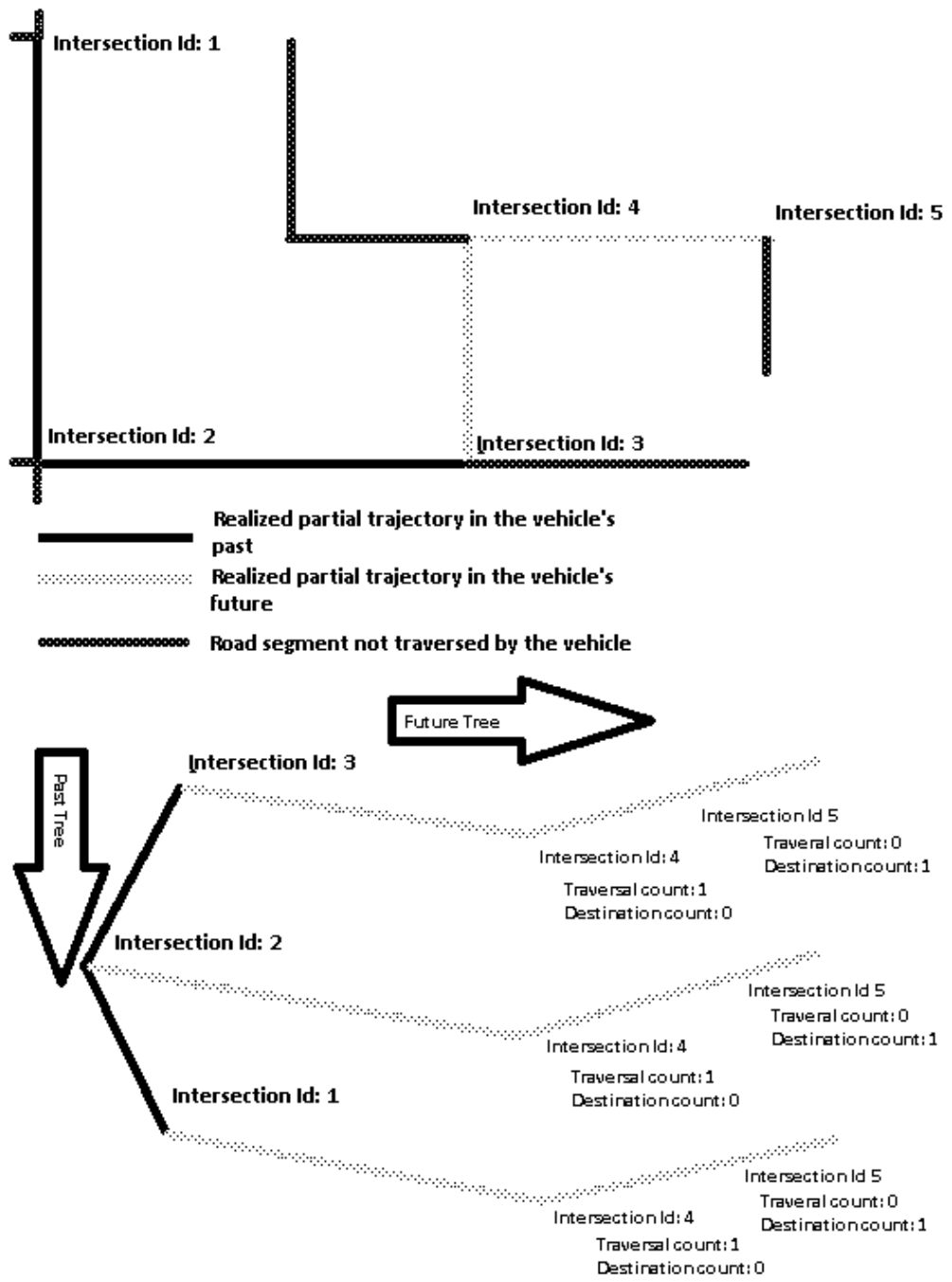


Figure 4: Past & Future Tree visualization

2.4.2 The Method of Past Trees: Predicting Trajectories and Destinations

Once a Past Tree for a given decision point is populated, it can be used to make predictions for a vehicle approaching the given decision point. If the vehicle's trajectory up to the point has been observed, the given Past Tree can be queried to retrieve the corresponding Future Tree.

Predictions regarding any future decision point can be made by considering the corresponding nodes at that depth of the Future Tree. It should be noted that this yields the n^{th} decision into the future and, therefore, has limited use: These points could be asynchronous in the expected arrival time (e.g., the example travel times to the Future Tree nodes in Figure 6). A prediction for Δt time units into the future requires locations to be tagged with point estimates of the travel time it takes to reach the respective points. A general search algorithm can then return the set of feasibly reachable locations (the intersection of the Future Tree and the shaded line cut in Figure 6). The frequency distribution of the traversal counts in this set can then be used to build an estimated probability mass on the map. The following pseudocode (for a parallel shared-memory machine), applied on the Future Tree of an observed vehicle, yields this prediction (a *probability map*):

```

Input: Future Tree (this instance),  $\Delta t$ 
1 i:=0
2 root.travelTime := 0
3 while (there is at least one non-leaf vertex of travelTime <  $\Delta t$  at depth  $D_i$ ) {
4   Par for all ( non-leaf vertices  $u \in D_i$  with  $u.travelTime < \Delta t$  ) {
5     Par for all w that are adjacent to u {
6       <edgeTravelTime, positionAt $\Delta t$ > := simulateTravel((u,w, $\Delta t$ ))
7       w.travelTime := u.travelTime +edgeTravelTime
8       if (w.travelTime >  $\Delta t$ ) {
9         w.late := true
10        frequencyMap.populateCountAtPosition(positionAt $\Delta t$ , u, w)
11      }
12    }
13  }
14  sync()
15  i := i+1 //master process only
16 }
17 probabilityMap := frequencyMap.transformToProbabilities()
18 return probabilityMap

```

Figure 5: Probability map generation algorithm

In general, this depth-synchronous implementation is not efficient, even if the underlying work-scheduler balances optimally: A time-parallel simulation (Kiesling, 2005) may simultaneously evaluate delays across road segments on differing depths and, therefore, offers the flexibility to saturate all the available processors with work. Nevertheless, we focus on this implementation for ease of exposition. The procedure expands all simple paths from the root node (Lines 3-4) until all the terminal vertices are tagged with a travel time greater than Δt ; at this point, the potential locations of the vehicle are found. Pursuant to this goal, vertices are tagged with a point estimate of the travel time to reach the respective intersections by invoking the transportation simulation, *simulateTravel*, on Line 6; this subroutine returns the travel time on the road segment from vertex u to vertex w . Unless this subroutine is implemented in a trivial fashion (for example, by simply retrieving a point estimate from historical data and not considering live data), the execution time of this call will dominate the total runtime of the program. (The time used by this bottleneck primarily depends on the granularity of the simulation model; a computationally efficient microscopic traffic simulation model can be found in (Nagel & Schreckenberg, 1992).) If the cumulative travel time up to w is greater than Δt , *simulateTravel* will also return the predicted position of the vehicle at Δt (*positionAt Δt*). At this point, the observed traversal count is queried

from the corresponding Future Tree (from vertex w is the vehicle is closer to w than to u ; otherwise the traversal count from u is used). This count is inserted into a map representing the frequency distribution at Δt (Line 10). After this map is populated, it is converted to a probability mass on the map and returned (Lines 17-18).

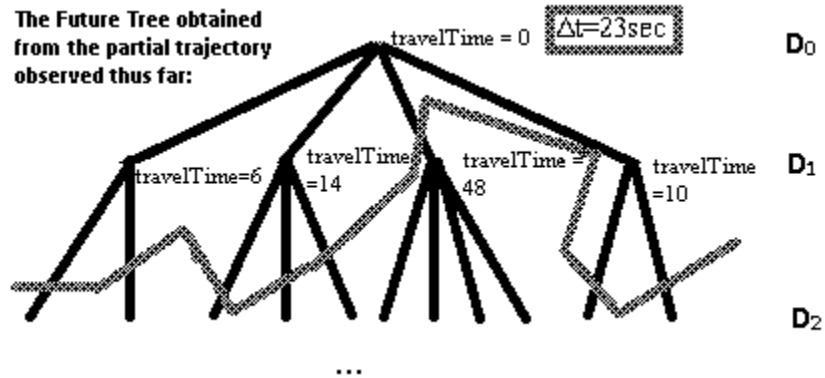


Figure 6: The Future Tree is used to retrieve the potential locations of the vehicle Δt into the future. In this case, we identify all vertices that are reached from the current location within a time span of 23 seconds. The shaded line illustrates the possible locations at that time.

If the question is what *destination* the vehicle is likely to approach, the destination counts described in Section 2.1 are fetched with the help of any tree search procedure (e.g., breadth-first search). The non-zero *destination count* of *every* node in the future tree is inserted into the appropriate location in the frequency map, which is subsequently transformed into a probability map and returned.

2.4.3 Generating Simulation Trajectories

Suppose we have constructed a subgraph that represents some road-connected intersections and now we wish to generate and route vehicle entities within. The following pseudocode illustrates these steps at a high level:

```

Input: Future Tree (assume code is applied to this instance), vehicle entity e
1 totalCount := 0
2 for all ( vertices u ∈ D1 ) {
3   totalCount := totalCount + u.traversalCount
4   traversalCountBoundaryList.add(totalCount)
5 }
6 U0 := realization from a discrete uniform between 0 and totalCount
7 toNode := computeToNode( traversalCountBoundaryList, U0 )
8 e.setNextNode( toNode )
9 sampleSpaceCardinality := toNode.destinationCount + toNode.traversalCount
10 e.stopAtNextNode := Bernoulli(toNode.destinationCount / sampleSpaceCardinality)
11 return e

```

Figure 7: Route generation algorithm

Vehicle instances will still be created as usual (e.g., from a non-homogeneous Poisson process). Once the entity is in the model and reaches a decision point, the simulation engine will query the past tree corresponding to a decision point for the trajectory taken up to that point. This, then yields a reference to the corresponding Future Tree. At Depth 1 of the tree, the observed frequency counts of all sibling nodes are read and inserted into a list, *traversalCountBoundaryList* (Lines 2-5). A random number (Line 6) is then transformed to a decision from this empirical frequency distribution via its c.d.f. (Line 7) and assigns it as the next location of the vehicle entity (Line 8). However, the vehicle entity may stop at that point. To account for this, we first realize the next intersection (Lines 1-8) and then condition against it (Lines 9-10). Both the destination and the traversal counts are known, so whether the vehicle stops can be inferred from a Bernoulli realization with success probability equal to the destination count divided by the sum of the destination count and the traversal count of the realized vertex (Line 10).

2.4.4 Space-Efficient Storage

The naive storage mechanism (i.e., a tree for each decision point) has much redundancy, but offers fast access to the future likelihoods (in fact, linear time in the trajectory length taken thus far). One observed trajectory will cause insertions into every tree along its path. This redundancy causes much wastage of storage. More precisely, suppose the length of the typical

trajectory is n and the total number of observed trajectories is T . This vector of size $O(n)$ is pushed into n Past Trees, but in each Past Tree it is replicated n times because it is recorded for each partial trajectory up to the point represented by the tree. The total storage requirement in this representation is thus $O(T \times n^3)$.

A sensible approach would be to cut off the tree at a certain depth – usually one should expect diminishing returns on accuracy when going down the Past Tree at significant depths. The results in Section 2.5 truncate the Past Trees at Depth 3.

A better approach, reducing the storage requirement from $O(T \times n^3)$ to $O(T \times n)$, is to use an array indexed by directed intersection identifiers (this is with almost no loss of generality as a different identification scheme can be coupled with a hashing rule with no performance loss in the average case). Each cell in the array will hold a reference to the head of a linked list. Such a linked list will simply contain pointers to trajectories containing the corresponding directed intersection identifiers (see Figure 8). Every time a trajectory is observed, all linked lists that contain a directed intersection of that trajectory will be augmented by one more reference. Thus, the storage requirement and the total insertion costs are limited to $O(T \times n)$. The potential for a highly skewed length distribution (on the collection of trajectories on directed intersection identifiers) motivates the use of lists, rather than arrays. Once a prediction is requested (when a vehicle has entered a particular directed intersection), a significant number, k , of previously observed trajectories are queried: The corresponding header can be accessed in constant time (random access with a specific directed intersection id) which allows for the extraction of the k trajectories of interest. Building a corresponding tree (by parsing and inserting the k trajectories) takes linear time in the size of the trajectories if partial trajectories outside of the currently observed one are omitted, leading to a total cost of $O(k \times n)$.

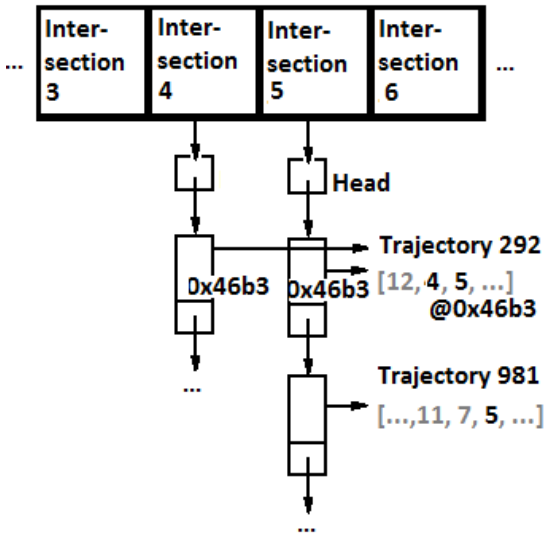


Figure 8: Illustration of the space-efficient storage scheme. Trajectory 292 is stored (only once) at location 0x46b3. Since it contains Intersection 4 and 5, the lists of these two intersections contain pointers to the trajectory's memory location.

2.5 Experimental Evaluation

To evaluate the accuracy of Past Trees, separate experiments were performed on randomly generated input as well as real-world data. Both experiments are run on cellular automata (CA) and generate two different error plots each (lower quantities are more desirable). In both cases, the directed graph representation introduced in Section 2.4.1 can be mapped in one-to-one correspondence with the CA if the appropriate cell transition rules are added.

The Method of Past Trees is compared to the method described in 2.3.1 (Krumm J. , 2006), which does not require previously collected trajectories. Whether the data acquisition and storage costs for Past Trees are acceptable depends on the added value from the increased accuracy for a given application. These costs scale with the desired sample size and the size of the target area.

2.5.1 Synthetic Experiment: Setup

A computational experiment was performed to sample the degree of accuracy provided by Past Trees. Here, the input (i.e., the trajectories of the drivers) is generated synthetically.

The model is based on a 20-cell-by-20-cell cellular automaton that approximates a two-dimensional metric space; this discretized plane serves as a conceptual model of a real-world region with pervasive road network coverage. 200 trajectories are sampled as follows: The origin and destination are first sampled by the pseudo-random selection of two cells. In this selection, some cells are more likely to be realized than others because cell popularity is assigned - at the beginning of the simulation - from a probability distribution that mimics the population distribution of US cities. Each cell is assigned a probability by first sampling the lognormal distribution with $\mu=7.28$ and $\sigma=1.75$ (which is a good fit for the population distribution of the United States - see (Eeckhout, 2004)), evaluating the density at those realizations, and then normalizing these values into a probability mass. Further, positive correlation is forced upon the popularities of neighboring cells ($\rho=0.3$): Less (more) popular cells are more likely to be near less (more) popular cells.

After the origin and destination are determined, the cells in between them are generated. At each intermediate cell, the vehicle will transition to a cell that is in accordance with the most efficient route with probability 0.625, and to one that is not with probability 1-0.625; we call this behavior *the desire for efficiency*. Each cell also has a statically assigned 10% chance of being an entry point that leads into a predetermined tunnel of neighboring cells, *a preferred path*. The length of a preferred path is geometrically distributed with a mean of five cells. A cell that leads into a preferred path is privileged to temporarily override the desire for efficiency until the vehicle exits the preferred path. Preferred paths (and their entry points) are randomly generated at the beginning of the simulation experiment and don't change during the execution. In this scenario, preferred paths have a mean length of five cells. The rationale of including preferred paths is to model unexplained trends within the population. Of the 200 simulated trajectories, 80% are first used to populate the trees; the remaining 20% are used to predict destinations and compute errors. The depth of Past Trees is limited to three cells.

2.5.1.1 Synthetic Experiment: Results

Figure 9 shows the mean accuracy as a function of the fraction of the trip completed (ensemble averages across the $0.2 \cdot 200 = 40$ trajectories). At each transition of any trajectory in the test set, the error model queries the Past and Future Trees of the current cell for all potential destinations of the vehicle. It then computes the rectilinear distance from the vehicle's current position to the center of the estimated probability mass (the predicted destination) as well as the rectilinear distance from the vehicle's current position to the actual destination. The error in Figure 9 is the absolute value of the difference of these two values.

Across all the simulated trajectories, for each cell transition, an error is computed and inserted into the bin corresponding to the tenths precision bin of the trip completion fraction. Then, for every completed trip fraction bin, the average error is selected. $0.8 \cdot 200 = 160$ trajectories were used to populate the trees.

As the vehicle entity approaches its destination, more data can be used because the partial trajectory up to the given cell is available: In the case of Krumm, it points towards efficient behavior, and in the case of Past Trees it points towards previously observed trajectories that are similar. This explains the steady improvement to a trip completion of 0.7. The increase of the plot at the end of the trip is a result of the vehicle approaching its destination relatively quickly and thus decreasing the first term mentioned above (distance to actual destination), while the second term (distance to predicted destination) is exposed to diminishing returns.

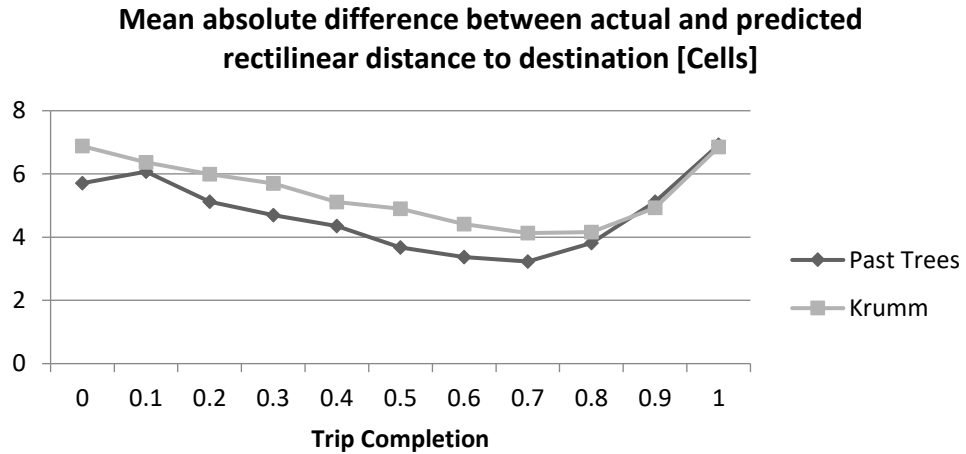


Figure 9: Error plots of the mean absolute difference between actual and predicted rectilinear distance to destination

Figure 10 simply shows the rectilinear distance between the predicted and actual destination. This quantity hovers around 10 cells for both prediction models (with a slight improvement towards the end of the trip). This is reasonably close in a 20-by-20 grid; no miracles are to be expected in a scenario where the trajectories of vehicles *almost* exhibit the patterns of 2D random walks with cell transitions that are independent of one another.

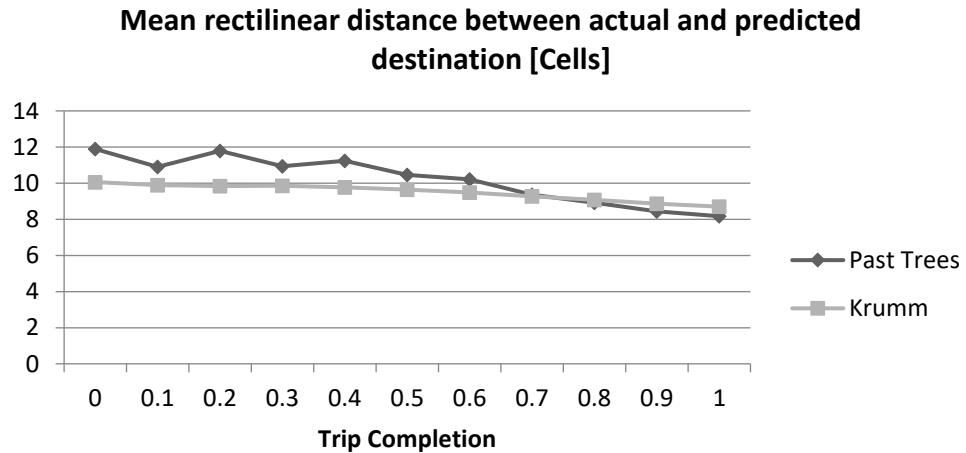


Figure 10: Error plots of the mean rectilinear distance between actual and predicted destination.

2.5.2 Real-World Traces: Setup and Results

A deterministic simulation was performed using the detailed NGSIM Peachtree Street dataset, collected between 4p and 4:15p on November 8th, 2006 in Atlanta, GA (Community, 2014). The road segment was modeled as a 30-by-3 cellular automaton and the first 80% of the 461 observed trajectories were used to populate the trees (the remaining 20% were used to test the model).

The following plot illustrates the error computed in the same fashion as illustrated in Subsection 2.5.1.1 for Figure 9. It should be noted here that Krumm's model is not well-suited for an almost one-dimensional environment as the estimated destination probabilities will just be uniform across the entire road. Because the sample size is small, the first 80% of each trajectory use superimposed Past Trees for each cell.

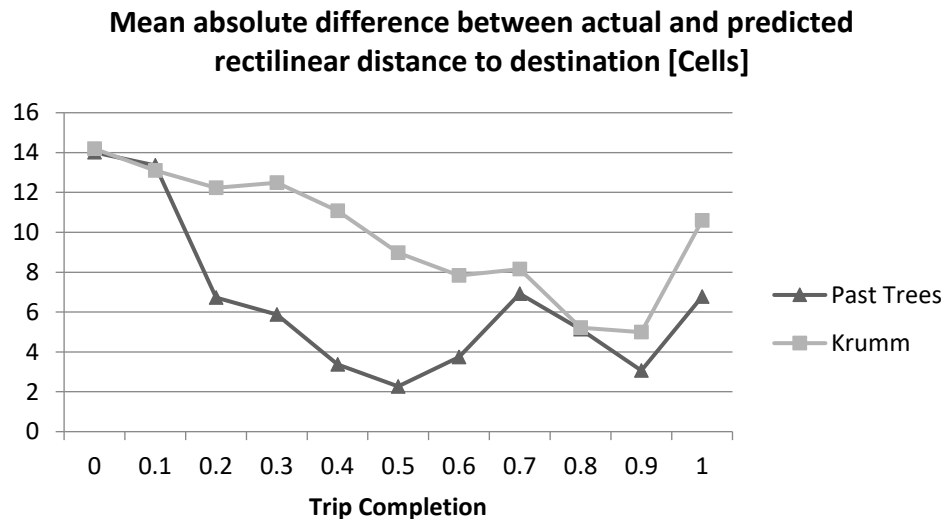


Figure 11: Error plots of the mean absolute difference between actual and predicted rectilinear distance to destination.

Figure 12 presents errors in the same way as Figure 10. For the Past Tree errors, the increased values towards the end of a trip are the result of time not being considered in the models: As the vehicle entity is in one of the final cells, the model assumes that the partial

trajectory up to the current point is indifferent from one that is near the center of the trajectory, which leads to an estimated cell much farther away. This may be avoided by not only incrementing destination counts, but also noting the progress along the trajectory during the data collection phase.

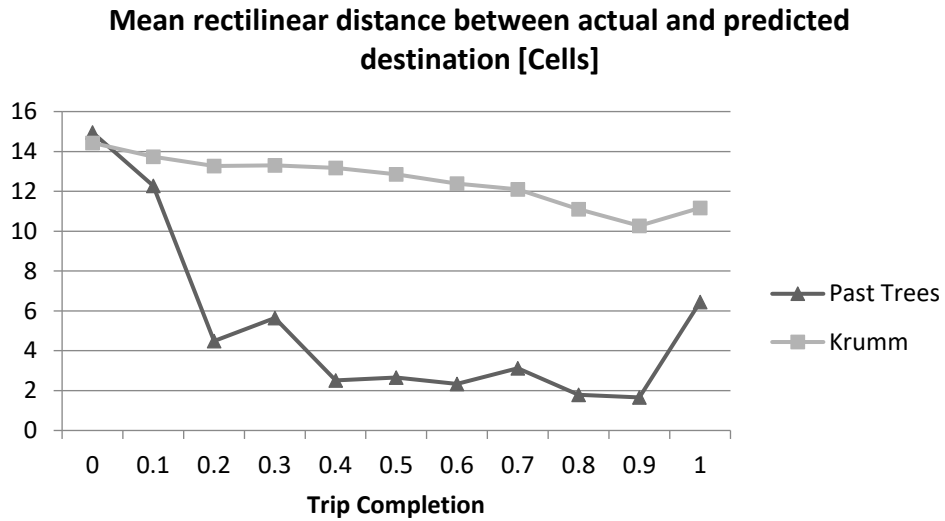


Figure 12: Error plots of the mean rectilinear distance between actual and predicted destination.

2.6 Conclusion and Future Work

A structure to store trajectories along road segments was introduced. Past Trees can be used to either generate trajectories for simulation entities or to predict likely destination of a vehicle whose path is being observed. The experimental results show that the structure gives a solid prediction framework. Modeling vehicular movement is clearly of great economic value as driver behavior and traffic management can be made more efficient.

One could enhance Past Trees by conditioning against more factors as well as weighing more recently observed trajectories more heavily; applying statistical regression and exponential smoothing to the Method of Past Trees has the potential to improve accuracy. Although it was

mentioned above that every decision point has one past tree associated with it, it can be very useful to have multiple trees to account for factors such as seasonality (the same tree can be used if it is 'colored') or other factors like congestion level. Traffic exhibits seasonal characteristics; for example, drivers might opt for a particular pathway during rush hour in order to avoid congestion in other areas.

It seems reasonable that more recent collections are most heavily correlated with present and near-future trajectories. For example, a popular sports game may tilt historical behavior into a particular direction. If paths are proactively collected in a particular area, exponential smoothing can adapt to recent trends.

3 DATA-DRIVEN VEHICLE TRAJECTORY PREDICTION²

²*This chapter includes content from (Pecher, Hunter, & Fujimoto, 2016) and (Hunter, et al., 2016).*

3.1 Introduction

Route or trajectory prediction algorithms attempt to predict the path that a vehicle will follow assuming its current position is known, but the vehicle's final destination is unknown. These algorithms assume other information is available such as the trajectory taken by the vehicle thus far and/or the routes taken by other vehicles in its vicinity or derived from historical information. As discussed later, several algorithms have been developed to predict future trajectories.

Trajectory and destination prediction finds applications in many areas. For example, in military applications focusing on aerial attack of stationary and unknown ground target (Sinclair, 2008) and to determine mobility patterns of cell phone users for desirable antenna handoffs for moving vehicles (Cheng, 2003). It has been suggested that advertising signs can be targeted for nearby locations based on trajectories of approaching travelers (Miklušćák, 2012). Similarly, mobile apps may suggest restaurants, hotels, or other services based on the driver's expected trajectory (Krumm J. , 2006). Law enforcement could make use of these models as decision support tools to track the movement of individuals by iteratively sensing the location of a target vehicle, estimating its future location, and redeploying mobile probing entities to locations where it is likely to reside in the future (see Section 2.2). The fuel efficiency of hybrid vehicles can be improved if the system knows the planned route of the driver in advance; automated generation of this information eliminates the need for the driver to explicitly specify this information (Johannesson, 2007) (Deguchi, 2004). This is especially true for routine trips, where the accuracy

of personalized trajectory prediction schemes can be particularly good (Laasonen, 2005) (Ashbrook, 2003) (Patterson, 2004) (Simmons, 2006). Further, the need for such capability is seen in that it has been observed that in practice, in only approximately 1% of all trips does the user specify his/her intended destination (Krumm J. , 2009).

Here, we are concerned with the use of trajectory prediction algorithms in the context of dynamic data-driven applications systems (DDDAS) (Darema, 2004). Recall that DDDAS systems involve the use of a control loop that iteratively (1) senses the current state of the system, (2) predicts future system states, and (3) relocates or reconfigures monitoring devices or deploys changes in the operational system to optimize its behavior along one or more dimensions. One use of trajectory prediction is to predict routes to be used by faster-than-real-time microscopic transportation simulation. Such simulations may predict temporal and spatial properties of congestion likely to arise in the future in order to provide travelers with useful information for decision-making, e.g., to determine if an alternate route should be chosen. Today, more drivers are taking advantage of applications that use real-time data to inform them of traffic information such as existing congestion or obstructions (e.g., the mobile app Waze). Accurate simulation results may advise these users of suggested routes and departure times that reduce congestion (Miklušcák, 2012) or reduce travel time globally or individually; we note that these objectives may conflict, and do not necessarily result in identical policies (Roughgarden, 2002).

Similarly, online simulations are often used by transportation engineers to manage the transportation network itself through means such as varying traffic signal timing, ramp metering, or providing travelers with information or recommendations to plan their travel activities. Transportation engineers routinely use microscopic transportation simulation in order to gain insights in traffic, e.g., to determine desirable traffic control policies or logistical considerations of construction projects under a variety of scenarios (Fujimoto R. H.-K., 2007). Desired output

statistics for these models may include the level of congestion and travel times on roads of interest.

Among other factors, the accuracy of these simulation results depends on the validity of the simulation model data that they use. Specifically, these simulators depend on knowledge of what routes will likely be taken by vehicles in the transportation network. Data-driven transportation simulations require dynamically collected data both to capture the current state of the system as well as to predict future system states. An important question concerns what data and how much data needs to be collected to make reliable predictions. This research specifically examines the value of utilizing past vehicle trajectory information to provide information for routes likely to be taken by vehicles in the future. Trajectory predictions allow the simulators to use more sophisticated vehicle entity behavior than independent turn probabilities at each intersection.

Sources of dynamic traffic data are numerous. The government uses toll collection transponders, embedded roadway loop detectors (Miklušćák, 2012), automatic license plate recognition, satellite imagery etc., and companies use information collected from drivers who carry smartphones (e.g., Google Maps) and other technologies. Technologies such as Bluetooth and WiFi detectors offer the ability to obtain partial trajectory information of individual vehicles in real-time in an operational transportation network.

Our hypothesis is that dynamic data significantly improves route prediction models. In the evaluation of the prediction models, our aim is to highlight the performance improvement that dynamic data-driven models yield over those that do not utilize such data. The predictive power of dynamic data in urban computing is seen in other areas as well. For example, (Yuan J. Z., 2011) proposes a cloud-based scheme to predict the travel time of a driver along candidate routes using dynamic data: traffic conditions, time of day, weather, driver behavior, etc. As a result, this

system allows GPS navigation systems to find the fastest route for a user, outperforming models that do not utilize this type of dynamic data.

Here, we compare methods to predict a driver's future trajectory given the observation of the partial trajectory traversed by the vehicle thus far. For the vehicle whose travel is being predicted we do not assume that we have historical travel information, i.e., common destinations used by that driver in the past. Reliance on such data may be problematic due to privacy concerns. This assumption allows the models to be easily applied to different vehicles in a DDDAS application, but will offer lower accuracy than personalized models since the preferred routine destinations of a particular driver are not known. Work by other authors have been used to predict a vehicle's future destination, irrespective of the future route taken to reach it. For example, Krumm's destination prediction model, described later, is based on efficient routes (Krumm J. , 2006). A modified form of Krumm's model is evaluated in this study.

Here, we discuss different methods to predict the trajectory (or route) of a driver on an urban road network and compare their accuracy utilizing the T-Drive trajectory data set. This data set consists of GPS trajectories of over ten thousand taxicabs (15 million data points) in Beijing, China. The performance of the models is measured by how often the next zone (a 1.25 km x 1.25 km square) visited by the vehicle falls within a predicted zone. Our results demonstrate that using route data collected for other drivers substantially improves the accuracy of forward trajectory prediction in the T-Drive data set.

While our primary goal is to show the value of trajectory prediction to drive on-line data-driven simulations, our contributions extend further. First, several proposed trajectory prediction algorithms are compared using a common dataset. Second, this chapter examines the benefit of utilizing dynamic data from other vehicle trajectories in trajectory prediction algorithms. Third, several extensions to previously published prediction algorithms are developed and evaluated.

The next section reviews the destination prediction literature. This is followed by a description of the models examined in this study. The performance of the models using the T-Drive data set is presented, followed by a discussion of future work.

3.2 Related Work

Early work in destination prediction was developed by Krumm (Krumm J. , 2006). This work utilizes a Bayesian model that uses the immediate past trajectory taken by a vehicle to predict the vehicle's intended destination. An underlying assumption used in this work is that drivers utilize efficient routes in order to reach their intended destination.

An efficient Markov model for destination and trajectory prediction is presented in Chapter 2. When a prediction is requested, a data structure is traversed that holds the partial trajectory observed for the vehicle thus far. This data structure subsequently provides the empirical distribution of the forward trajectory.

Trajectory prediction using artificial neural networks is described in (Miklušćák, 2012). The previous five locations visited by the target vehicle are used to estimate the future trajectory by training a feed-forward artificial neural network with two hidden layers consisting of 500 neurons each. These three approaches – Krumm's approach, Markov models, and neural networks are discussed in greater detail later.

In 2008, a Carnegie Mellon University group published the PROCAB model (Probabilistic Reasoning from Observed Context-Aware Behavior) (Ziebart, 2008). This model uses the current context (e.g., accidents or congestion) and the user's preferences (e.g., fuel efficiency or safety) in order to predict his/her actions, rather than just focus on previous actions for prediction. The model maps actions into a Markov Decision Process (MDP), where intersections are encoded as states and road segments are encoded as transitions. State transitions are associated with a cost, namely the sum-product of road segment features and cost weights

obtained from collected training data. The model assumes that drivers attempt to minimize the cost to reach their destination. For destination prediction, it has been reported that PROCAB outperformed Krumm's Predestination algorithm for the first half of the trip. The PROCAB implementation used in (Ziebart, 2008) takes into account specific road features such as the number of lanes and speed limit. A further challenge in the context of the work discussed here is we assume the destination is unknown; one of the inputs used by (Ziebart, 2008) for trajectory prediction is the actual destination of the target vehicle.

In (Gui, 2009) the authors suggest using a personalization profile of smartphone users for localized searches. User activities and on-device probing entities are queried to build a context profile, including demographical features and previous user activities. The weights of this information along with an environmental profile (weather, temperature, etc.) are trained via an artificial neural network. These profiles are used to rank personalized queries (e.g., local businesses). One could envision utilizing the same model to predict destinations or routes, similar to PROCAB.

In (Amini, 2012) the authors use Krumm's destination prediction model based on efficient routes, but truncate the destination sample space to locations reachable within 30 minutes from the trip's starting location, citing a study (Hu, 2001) that concluded that most driving trips end before that amount of time.

In (Simmons, 2006) a hidden Markov model (HMM) is used to estimate a particular driver's destination and route, by using his/her previous trajectories along with driving time. Because personal identification is not specified in the data files of the T-Drive trajectory sample, we apply our Markov model (see Section 3.3.3) without discrimination towards any particular driver. That is, we train our model using the trajectories of many different drivers, rather than rely on individualized routes.

In (Persad-Maharaj, 2008), historical trajectories are converted into polygons by adding a 10 meter radius around the observed path. This data is then stored in a database. As soon as a prediction is requested, a polyline is formed through the requesting driver's observed GPS points and the database is checked for any intersecting polygon. If there are multiple results, further filters are applied to obtain the prediction. For example, match of the driver id would rank that particular polygon higher than one from another driver.

Sub-Trajectory Synthesis (SubSyn) (Xue A. Z., 2013) addresses the data sparsity problem, as well as privacy concerns, by synthetically generating trajectories from existing routes. These existing routes are decomposed into two-pair nodes and a 1st order Markov Model is used to generate the new trajectories.

Some of the approaches described above are not well suited for use of the T-Drive data set for evaluation. In some cases, necessary information such as source-destination information for trips are not known. Inference of this information from taxi trajectories is not straightforward. Further, the T-Drive sample data is not at a suitable temporal granularity for effective evaluation of some methods; for example, speed data cannot be easily derived.

3.3 Prediction Models

The three prediction models compared in this study are described next. These include Krumm's model along with several extensions developed during the course of this study, a model based on artificial neural networks, and a Markov model. Some of these approaches use trajectory information collected from other vehicles. An approach to store this trajectory information, termed Past and Future Trees, is also briefly discussed.

3.3.1 Krumm's Destination Prediction Algorithm Based on Efficient Routes

Krumm developed a *destination* prediction model based on the intuition that drivers reduce their minimum remaining travel time to their destination as the trip time increases (Krumm J. , 2006), that is, drivers tend to the shortest path to a destination rather than more circuitous routes. As already shown in Figure 3 previously, Figure 13 again illustrates this intuition. In (Krumm J. , 2006) Seattle was divided into cells measuring 1 km x 1 km. For the study, it was observed that when drivers transitioned between cells they reduced the potential minimum travel time to their destination 62.5% of the time. Krumm utilizes this driver tendency to seek efficient paths to estimate the probability that a given location (i.e. cell) is the driver's destination given the current partial path (i.e. the partial route of the traveler to their current location). That is, suppose a driver has now traversed a partial trajectory P and we wish to determine the estimated probability of the driver's destination being a cell c_i . Each probability $Pr(P/c_i)$ is computed by a product of $\{p, (1 - p)\}$, where $p = 0.625$ and the number of factors is equal to the partial trajectory length (there is a surjective mapping from the cells of the partial trajectory to $\{p, (1 - p)\}$). A factor of p is selected when the partial trajectory cell brings the vehicle closer to the candidate destination c_i and $(1-p)$ otherwise. All of these candidate $Pr(P/c_i)$'s are then multiplied with the prior probability $Pr(c_i)$, if the cell bias is known, and normalized, so that their sum equals 1 for a valid probability mass of the $Pr(c_i/P)$'s.

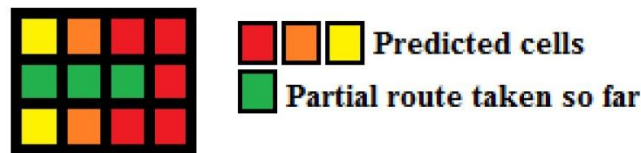


Figure 13: The red cells are assumed to be more likely because they are consistent (in efficiency) with the partial route taken so far (rectilinear distance).

In algorithm format, the prediction algorithm operates as follows:

```

1 For each candidate cell  $c_i$ 
2    $p_c = 1$ 
3   For each cell  $c$  in partial route  $P$ 
4     if  $c$  is closer to  $c_i$  than any previous cell in  $P$ 
5        $p_c = p_c * 0.65$ 
6     else
7        $p_c = p_c * 0.35$ 
8    $P(c_i) = p_c$ 
9 Normalize all  $P(c_i)$  to get a valid probability mass

```

Figure 14: Krumm's efficient route prediction algorithm.

In the above algorithmic description, $Pr(P/c_i)$ is abbreviated as p_c and the cell bias is assumed to be unknown. If the cell bias is known, the probability estimate on Line 8 can be scaled accordingly.

In Section 3.4.1 we modify this algorithm to use it for *trajectory* prediction, or more precisely, for the *next transition* in the trajectory.

3.3.2 Artificial Neural Networks

Artificial neural networks (ANN's) map input values to output values and are often used to approximate complex functions (e.g., recognizing handwritten digits). For trajectory prediction, the input layer may encode some number of previously visited locations and the output layer may represent one or more projected future locations. ANN's consist of nodes called neurons (as they are inspired by the brain) and edges that connect them. In a feed-forward ANN (see Figure 15), information flows from the input layer to the output layer, via one or more hidden layers. The nodes in any layer are fully connected to the nodes of their preceding layer. Edges have weights associated with them that determine the strength of the signal sent by the preceding neuron. Typically, the product of the weight and the signal is then received by the

neuron to which that the edge points. The signal sent by a neuron is determined by its *activation function*. Sigmoid functions are commonly used in feed-forward ANN's. One of them is the logistic function, $(1+\exp(-x))^{-1}$, where x is the total input signal to the receiving neuron. For most values of x , the sigmoid function evaluates to a real number either close to 0 or close to 1. A popular, supervised learning algorithm for training feed-forward ANN's (i.e., determining the edge weights) is the backpropagation algorithm. Usually, the initial weights are randomized after which they are modified to minimize the error via gradient descent.

Since ANN's are often used in prediction to map relevant factors to an estimated output when training data is available, it seems reasonable to assume that we can utilize them for trajectory prediction as well. Just as in the other trajectory prediction models, the partial trajectory of the target vehicle can be used as the input and the output layer can encode a future location of the vehicle. The design of the ANN, including the encoding of the input and output neurons and the architecture of the hidden layers is not trivial. (Mikluščák, 2012) outlines three neuron encoding schemes and opts for the third one in the experiment: (i) A single neuron encodes the location, (ii) each possible location value is associated with a dedicated neuron, and (iii) a neuron encodes a bit in the binary representation. The learning parameters must also be selected carefully, in order to avoid oscillations, high errors, and/or under-training. (Mikluščák, 2012) evaluates the trajectory prediction accuracy of a feed-forward artificial neural network (Figure 15) with sigmoid activation functions on artificially generated data. The input layer consists of the five previously visited locations in the trajectory where each location has multiple input neurons representing the integral location id in binary form. Two hidden layers of 500 neurons are used. Using a training set of 2000, the authors attain a success rate of almost 99% and 64% in their synthetically generated datasets, where the former quantity is tested on a dataset described as an easy data set with very little stochasticity while the latter quantity is tested on a dataset described to contain a considerable amount of random decisions.

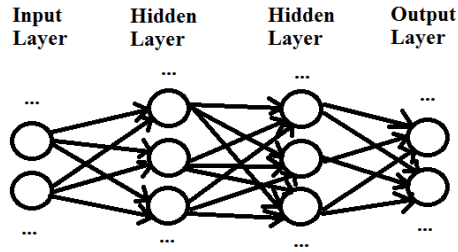


Figure 15: The edges have weights associated with them that set the strength of the value being sent. The circles are neurons that typically send values close to 0 or close to 1.

In order to reduce the training time to an amount comparable to the other tested models, we reduce the hidden layer count to just one and the hidden layer size to just seven neurons. With our fixed learning parameters, ANN's with more hidden layers and/or larger hidden layer sizes performed worse in our profiling tests.

3.3.3 Markov Models and Trajectory Storage

Chapter 2 proposes a data structure that serves as the fast backbone of an arbitrary-order Markov model. This data structure can be used for a simple trajectory prediction algorithm if it is used to store historical trajectories. If queried order is sufficiently high and/or the data is sufficiently sparse, the future trajectory distribution is obtained from the maximal order for which data is available. A tolerance for minimum sample size can be set by the user. The emphasis on speed makes this implementation especially useful for embedded applications and/or simulation engines where realizations have to be drawn quickly. Historical route data is stored in Past and Future Trees and the data are queried on demand. The Past Tree of a location is queried with the partial trajectory traversed by the target vehicle up to that location. A Future Tree that contains the future trajectory distribution is then returned by the Past Tree. If the user wishes to get an estimate of the location of the target vehicle at some defined time in the future, the model invokes

a microscopic traffic simulation. If storage is more important than access time, a hashtable implementation may be used instead (see Figure 16).

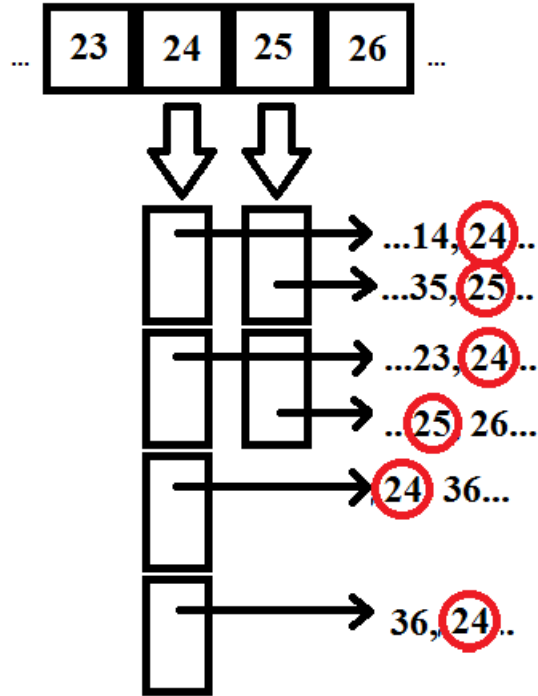


Figure 16: All trajectories that contain the cell 24 are being pointed to by the column 24. Empirical distributions of any Markov order can be obtained quickly by traversing this column.

Suppose the target vehicle is at cell 24, a prediction for the next cell is requested, and the partial trajectory so far is supplied. The 24th column can be accessed in constant time and its elements can be traversed comparing the elements with the partial trajectory observed so far. An empirical distribution can now be constructed by collecting a sufficient number of partial-trajectory matches along this column.

The models discussed thus far – in their current implementation – statically predict the future route with some spatial discretization, but do not specify point estimates for the time needed to reach the points along that route and do not return the projected location at a specific future time. If, rather than the next cell, the projected location of the target vehicle at Δt time units

into the future is requested, one can construct a decision tree, T , that enumerates all possible paths in a breath-first fashion. Leaf nodes are tagged with the expected travel time to the intersections represented by those nodes until they exceed Δt . Once all leaf nodes exceed Δt , all locations expected to be reached at Δt time units into the future are tagged with the likelihoods stored in the Past Trees (or with those returned by the models of Section 3.3.1 and 3.3.2), and normalized to yield a valid probability mass. For example, if Past Trees are used and all the relevant leaf nodes of T have been enumerated and flagged, the Past Tree column of last observed location of the target vehicle is retrieved (Figure 16), a suitable number of matches of the partial trajectory observed thus far is queried from this column, and used to build an empirical distribution of future locations. Each path from the root node of T to a flagged node can now be used to retrieve the estimated probability from the empirical distribution. All these estimated probabilities are then divided by their sum in order to scale them to a valid probability mass.

3.3.4 Generating Routes from Probability Maps

The prediction models described above generate probability maps indicating the likelihood the vehicle will reside at particular locations in the future. It is straightforward to use the probability maps to generate future routes. These models can also be used to simulate entity routing in microscopic traffic simulation. For example, if an ANN estimates the next-transition probabilities, one can input a random number into the inverse c.d.f. of the empirical distribution and generate a next location. This next location, along with the previously visited locations, can be used as an input to the ANN and this process can be repeated. In the same fashion, the probability map returned by Krumm's prediction model based on efficient routes can be used to generate a cell. Lastly, Past Trees can be traversed quickly to construct an empirical distribution function of the next transition and random numbers can be used to obtain a sample of the next decision. In all these cases, a destination should be realized at some point of the simulated trip.

3.4 Experimental Evaluation

The data set used for evaluating the accuracy of the models is the T-Drive sample dataset published by Microsoft Research (Yuan J. Z., 2011) (Yuan J. Z., 2010). It consists of 15 million GPS points collected from over 10,000 taxicabs from February 2 to February 8, 2008, in Beijing. The sampling interval between two consecutive GPS coordinates is variable with a mean of 177 seconds (average distance: 623m), but can be as long as 600 seconds. It is important to note that the dataset consists of tour-based data and not origin-destination pairs. In other words, all the origin-destination pairs of sub-trajectories (taxi trips) are hidden in the tours of each taxicab file. The taximeter data of the T-Drive sample data set is not published (as of January, 2016). In our experimental evaluation, we discretize the road network into a two-dimensional grid.

From the T-Drive Sample Dataset, we focus on an area that is approximately $100\text{km} \times 100\text{km}$: longitude from 116.0 to 117.1 (inclusive) and latitude from 39.6 to 40.5 (inclusive), in decimal degrees. This area was selected by isolating grid cells that have over 100,000 data points at the resolution of 0.1 decimal degrees in both dimensions (approx. $10\text{km} \times 10\text{km}$). This region is overlaid with a 96-wide and 80-high cell grid, where each cell maps to a $1.25\text{km} \times 1.25\text{km}$ area. The relatively coarse grid size was chosen because of the long median sampling interval in the dataset. Finer resolutions would require one to speculate on the path taken by the taxicab between successive reported locations. This is also the reason, in general, why the turn ratio at intersections cannot be determined with certainty in this data sample. To obtain consistent results, we use the same size for all cells; using variable cell sizes (e.g., determined by the variable sample intervals) could result in an arbitrary grid schema. Taxicab files are only considered if they fall within the above mentioned region. Taxicabs frequently visit certain, popular hubs to drop-off or pick-up passengers. In order to avoid utilizing these “artificial” trips we use a simple heuristic of ignoring the top 5% of the most popular cells, which are more likely to be hub locations, from the results.

We use the first 80% of filtered data points to train the models, and the last 20% of filtered data points to test the models. All interior cells in the grid contain eight neighbors (see Figure 17).

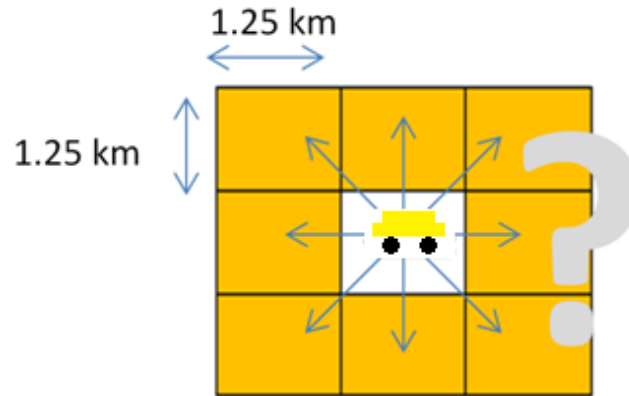


Figure 17: The next zone visited by the taxicab is uncertain, but is restricted to eight possible zones.

We consider the partial trajectory of the taxicab in question in order to predict the next cell visited. To evaluate the accuracy of the models, we assume that we can probe i grid cells, ranging from one to eight, for the presence of the vehicle. For each model, let $A(i)$ denote the measured cumulative probability of capturing the vehicle by greedily selecting the i most probable cells as estimated by the respective model. The objective is to achieve a high probability of detection for small i . In this section, if a model $m2$ is described to be $x\%$ *better* or *improved* than model $m1$ (for a specific number of observed grid cells i), x is derived from the distance $A(i)_{m2} - A(i)_{m1}$ - not from a ratio of the two terms.

The test system uses an Intel i7 6700 CPU, 32GB of DDR4-2133 main memory, and a Samsung 840 Evo 1TB solid state disk. Neuroph v2.92 is used for the artificial neural networks in section Feed-Forward Artificial Neural Network.

3.4.1 The Efficient Route Model and Extensions

We modify the efficient route algorithm from Section 3.3.1:

```
1 For each candidate cell  $c_i$ 
2  $p_c = 1$ 
3 For each cell  $c$  in partial route  $P$ 
4 If  $c$  is closer to  $c_i$  than the previous cell in  $P$ 
5  $p_c = p_c * 0.75$ 
6 else
7  $p_c = p_c * 0.25$ 
8  $P(c_i) = p_c * \text{Markov}(P(P.Length-2), P(P.Length-1))$ 
9 Normalize all  $P(c_i)$  to get a valid probability mass
```

Figure 18: Modified efficient route algorithm (changes in bold)

We use a less strict condition, on Line 4, to determine whether the vehicle opted for efficient transitions, in order to achieve better accuracy of the prediction of the next transition in the T-Drive sample. In most applications, the sample space of the destination is orders of magnitude greater than the immediate trajectory, so a more discriminatory algorithm is likely to perform better for the prediction of the destination. We also tune the p parameter (Line 5 and 7), denoting the fraction of time a transition is efficient, with respect to the T-Drive sample training set. Lastly, rather than use just the static cell popularity or no adjustment at all, the candidate $Pr(P/c_i)$'s are scaled by the 2nd order Markov estimates on Line 8. $Markov(start, end)$ uses the subsequence of P , which starts with cell $start$ (inclusive) and ends with cell end (inclusive), as arguments, and returns the corresponding Markov probability estimate. The indexing into P is assumed to be zero-based. We will refer to the condition on Line 4 with the term *isCloser* in the subsequent discussion.

For the *modified* Krumm route efficiency model without any historical data for the prior, we noticed very minor changes in the prediction accuracy for low i , as the efficiency likelihood parameter p is varied from 0.55 to 0.95 (using 0.1 increments). Among these, the best results

were obtained with $p=0.75$. In Figure 19, the dark blue plot visualizes the performance, $A(i)$, of this model. This plot, as well as all other plots in Section Experimental Evaluation, is derived from evaluating the test set - the last 20% of the data points.

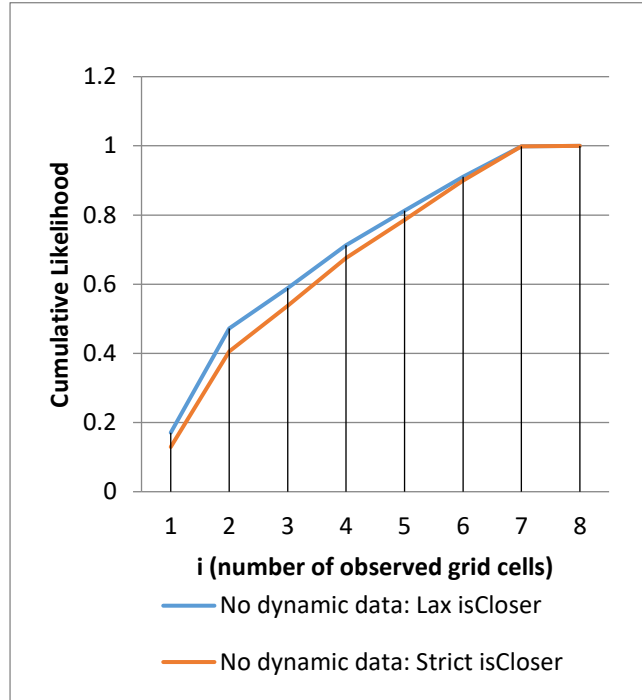


Figure 19: Cumulative likelihood for recapturing the target vehicle at the next cell transition using the path efficiency models (Krumm) with a $p=0.75$ parameter.

In the Seattle dataset, used in the original study, a value of p equal to 0.65 offered the best results. However, due to the different road network (Seattle vs. Beijing), the different driving population (Microsoft employees vs. cab drivers), slightly different grid size (1.25 km vs. 1.25 km), different objective function (distance to destination vs. next transition) or simply random noise (due to hidden factors) we found that setting p equal to 0.75 performed slightly better in this context. We experimented with dynamically varying the strength of the p parameter for the observed partial trajectory, using higher p parameter values for more current cell transitions than in older transitions. This results in slightly worse performance than a constant $p=0.75$ for all transitions. Thus, for all subsequent variations of the Krumm route efficiency model, we use

$p=0.75$. We also note that truncating the partial route traversed by the taxicab to two or three cells did not yield any benefits. In other words, using the entire partial trajectory of the taxicab yielded superior results than just considering a limited horizon. Lastly, when deciding if a cell transition is efficient, different distance functions can be used. When we opted for Euclidian distance, rather than rectilinear distance, we obtained slightly worse results. As a result of these two considerations, partial trajectories for the efficient route models will not be truncated and rectilinear distances will be used.

One important algorithm modification that led to the previously mentioned results was the replacement of the efficiency random variable. In the original model, a transition is considered efficient if it brings the vehicle closer to the candidate cell relative to *any* other previously visited cell of the partial trajectory traversed thus far. Instead, we consider a transition efficient if it brings the vehicle closer to the candidate cell relative to the *previously visited cell* only. The original random variable results in slightly worse reacquisition likelihoods as the red-colored plot in Figure 19 illustrates. To be clear, the term *strict isCloser* exclusively refers to the conditional expression in Line 4 of the first algorithm listed in Section 3.3.1.

The original efficiency standard is stricter, as the comparison is relative to *all* previously visited cells. The sample space of potential destinations is much larger than just the next transition and if the efficiency standard is strict, a large area of potential destinations can be virtually eliminated (with very small assigned probabilities). Our objective is to predict the immediate trajectory, rather than the destination, a likely reason for the better performance of the lax conditional (the conditional on Line 4 of the algorithm listed at the beginning of this section). For all subsequent variations of the Krumm route efficiency model, we only consider a cell transition efficient if it brings the vehicle closer to the candidate cell compared to the previously visited cell only.

3.4.2 Markov Models

To evaluate the Markov models, we use the Past Tree implementation from Chapter 2. We do not set a sample size limit; in other words, we use relevant observations from the entire training set to populate the Past Trees. In our discussion, the term *order- i* refers to using the previous i locations of the target vehicle to make the next prediction. The results of Figure 20 are obtained from the pure order 1, 2, and 3 Markov models:

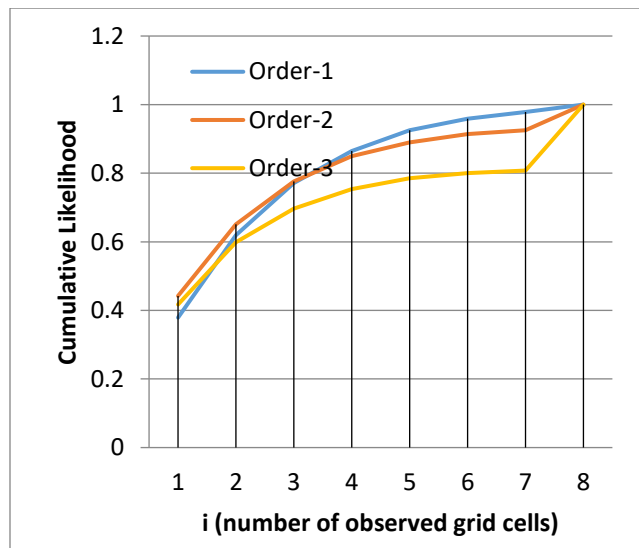


Figure 20: Cumulative likelihood for recapturing the target vehicle at the next cell transition using 1st, 2nd, and 3rd-order Markov models (Past Tree implementation).

Due to over-fitting or data sparsity, the order-3 model performs worse than the order-2 and order-1 model. The order-2 Markov model performs slightly better than the order-1 variant for $i=1,2,3$ because directional information can be inferred from the previous two destinations. However, there is a slight drop off in this advantage as i increases and the implementation of the

order-1 model is easier. Below $i=6$, every Markov model outperforms the models that do not have historical data (the blue and red plots in Figure 19) and the $i=1$ accuracy is over twice as high in the Markov model results. Even though the models without historical data directly infer the immediate future trajectory from the *behavior* (route to current location) *of the target vehicle itself*, the data of *other vehicles* that engaged in the same behavior offers better point estimates on average.

Figure 21 shows the results of Markov models that attempt to use estimates from the next lower orders, after a candidate cell could not be found in the higher order model (due to lack of data). In other words, the estimate is extracted from the highest order model that has the candidate location available. If there is still a miss of the candidate cell in the order-1 model, the static cell popularity is used. There is little benefit of this cascading fallback mechanism at order-1, compared to the plain order-1 model, as the data available at order-1 is quite abundant (i.e., rare fallbacks to static cell popularities). However, the order-2 fallback estimates are on average 1% better from $i=1$ to $i=3$ and 4% better from $i=5$ to $i=7$, compared the plain order-2 model. The order-2 fallback model performs slightly better than the order-3 fallback model.

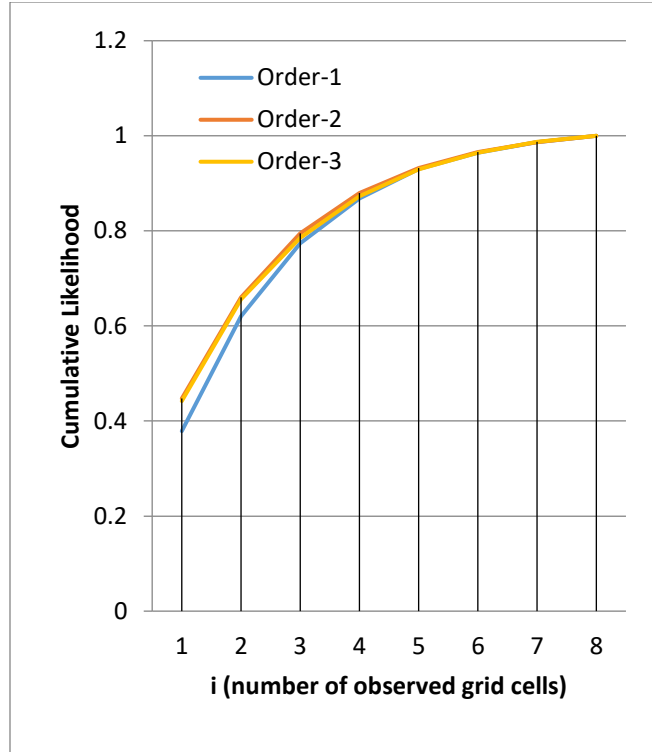


Figure 21: Cumulative likelihood for recapturing the target vehicle at the next cell transition using 1st, 2nd, and 3rd-order Markov models with a cascading fallback mechanism towards lower orders.

If we define the prior (i.e., the cell bias $Pr(c_i)$) in the Krumm route efficiency model as the static cell popularity (how much has each cell been visited by a vehicle in the training dataset), rather than just a uniform constant (in the case where no vehicle density data is available), and then normalize these probabilities for only the eight possible cells, we obtain a significantly improved (relative to the basic route efficiency model) distribution function, as can be seen in the dark blue-colored plot of Figure 22.

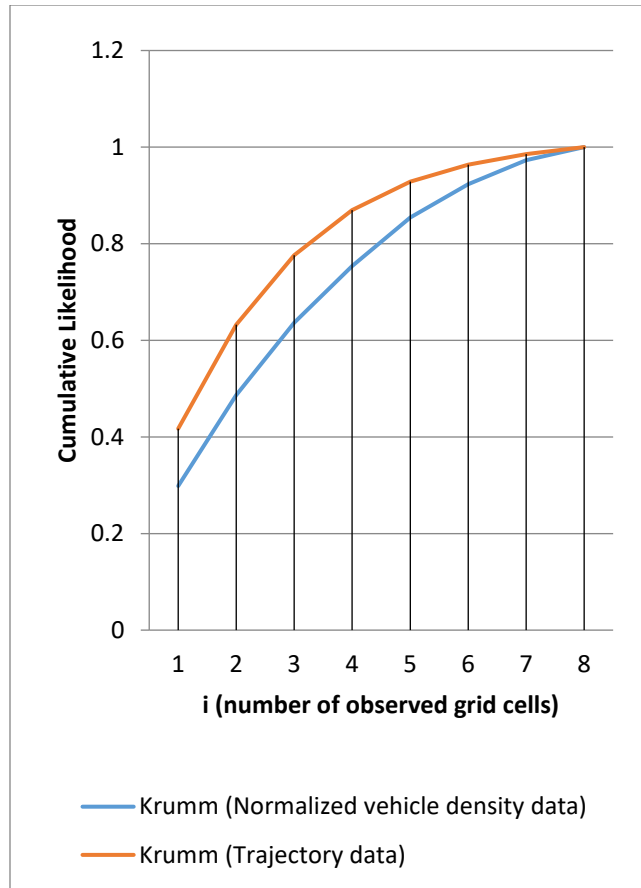


Figure 22: Cumulative likelihood for recapturing the target vehicle at the next cell transition using the modified Krumm models and dynamic data.

When we replace the prior (or the cell bias) in the Krumm route efficiency model with order-2 fallback probability estimates, we obtain reacquisition likelihoods that are improved further (the red-colored plot in Figure 22).

These results illustrate the power of using dynamic data in this particular application. Compared to the original route efficiency model, the probability of correctly predicting the target vehicle's location more than doubles at $i=1$. This advantage is crucial if there are several trials (reacquisition attempts), translating into savings in tracking resources and/or the more likely successful tracking of the target vehicle.

3.4.3 Feed-Forward Artificial Neural Network

For the feed-forward artificial neural network results, we use a single hidden layer with seven neurons in order to reach a training time comparable to the other models. Our profiling experiments showed that adding significantly more or less than seven neurons to the hidden layer and adding more than one hidden layer worsened the results, fixing the learning parameters described shortly. Omitting the hidden layer completely worsened the results. The activation functions are hyperbolic tangent and the edge weights are learned with the backpropagation algorithm. Because we opt for binary encodings of the input (of the five previously visited nodes, as in the setup in (Miklušćák, 2012)) and our cells are identified with 12 bits, we have 60 ($=5 \times 12$) input neurons in total. The first six binary values of a binary cell identifier encode the row index, and the last six values encode the column index. The output consists of eight neurons, denoting the prediction of the next cell visited (direction relative to the current cell of the vehicle; see Figure 17), a higher output value being interpreted as a higher likelihood of the next cell visited matching with the cell associated with the given output neuron. Given a step size of 0.2 (based on our results, just slightly better than 0.8), a maximum error of 0.1 (based on our results, as good as 0.01), at most five learning cycles (better than lower values and just as good as most higher values), and including the 5% most popular cells, we obtain the results depicted in Figure 23. The value(s) in parentheses, after the abbreviation *ANN*, refer to the number of neurons in the hidden layer(s).

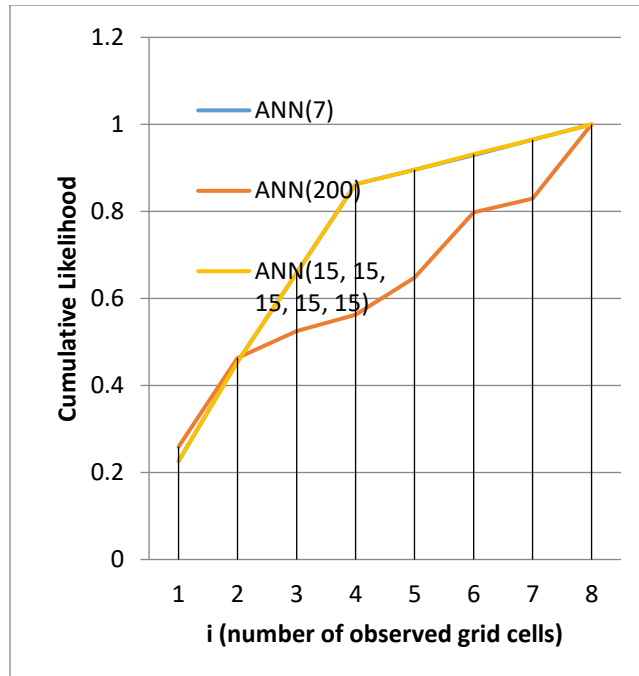


Figure 23: Cumulative likelihood for recapturing the target vehicle at the next cell transition using a feed-forward artificial neural network with one hidden layer, consisting of seven nodes.

The learning parameters are fixed for each ANN model. The performance of ANN(7) and ANN(15,15,15,15,15) is virtually the same. The ANN(7) results are superior to random guessing and the models that are only aware of the partial trajectory traversed by the target vehicle. Training and testing ANN(7) took 25.7 sec and 2.3 seconds of CPU time, respectively. One possible explanation of the change at $i=4$ is that most of the variation in the output is captured by the most significant bit of the most recent node's row and column index.

3.5 Comparison of the Models

Figure 24 shows the best performing members of the discussed model families. The light blue plot shows the performance of the ANN with one hidden layer of seven neurons

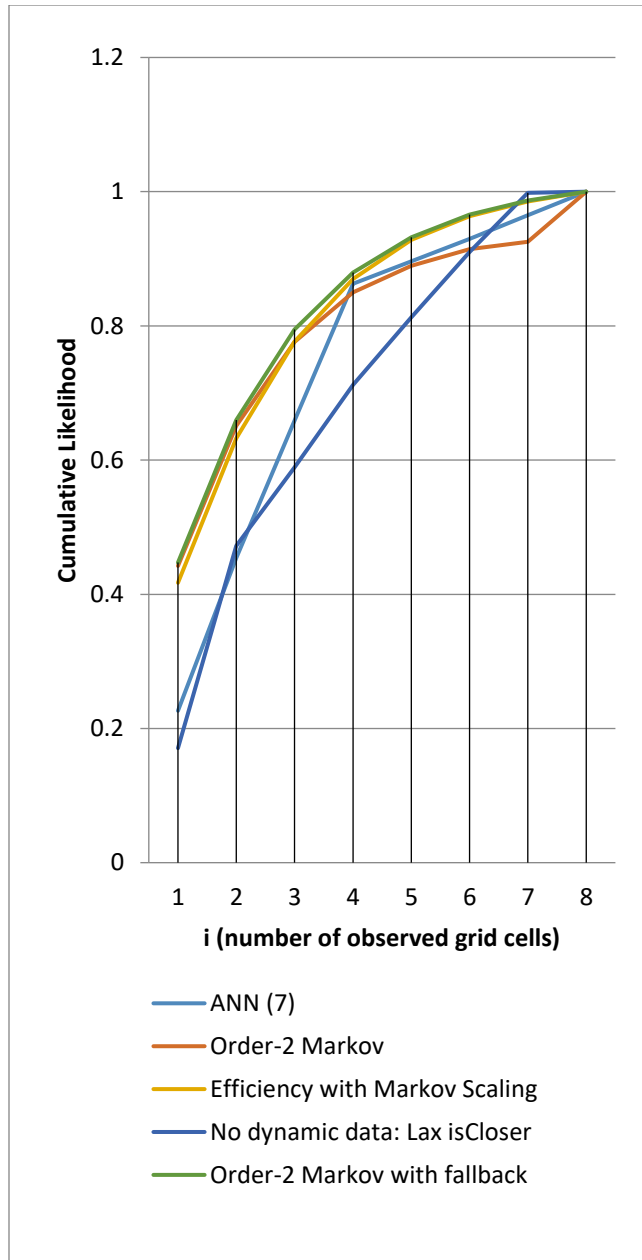


Figure 24: Cumulative likelihood for recapturing the target vehicle at the next cell transition using the best-performing submodels.

The order-2 Markov model with the fallback mechanism outperforms all other models for $i < 7$. The preference for efficiency appears to already be implicit in the data. We have also evaluated a tournament predictor (not shown) that uses the estimates from either the hybrid model (efficiency scaled by order-2 fallback estimates) or the order-2 fallback model, depending on how

well each model has performed in the current context (the trip progress and specific last two cells visited). For $i < 5$, the tournament predictor performs worse than just the order-2 fallback model. While the pure efficiency prediction does not take into account specific features of the Beijing road network or other dynamic factors (dark blue), it is – in its modified form (yellow) - able to leverage the information that is implicit in the historical order-2 Markov estimates. The modified Krumm model, using the order-2 Markov fallback model in the cell bias term, performs slightly worse than just the pure order-2 Markov model at $i=1$, perhaps due to the power law nature of popular locations and thus the strong preference of drivers to transition to certain cells. With higher i , however, the skewness of the transition distribution reduces and drivers are likely to still be in transit, when efficiency is still very much relevant. Further investigation is needed to determine appropriate hyper parameters, architectures, and training heuristics to train deep neural networks for trajectory prediction. Our simple ANN architecture is able to outperform the Order-2 Markov model (without fallback) only after $i=3$, but is dominated by the efficiency-Markov hybrid model across all i .

As a final section, an array of granular-displacement real-time prediction methods are compared against each other. Here, the assumption is that one is given the route of the driver, but wishes to construct the continuous displacement of the vehicle over time. The displacement function can then be used to compute a variety of other observables (velocity, acceleration, jerk, and - with other information - momentum, energy, etc.).

3.6 Displacement Inference Approach

The Displacement Inference approach seeks to utilize historic and near-real time data to construct a predicted trajectory of the client. In the current implementation, the Displacement Inference approach implementation is on a segment-by-segment basis, where a segment consists of the section of roadway between two adjacent intersections and the downstream intersection. The overall vehicle trajectory is then the accumulation of the predicted vehicle trajectories over

each traversed segment. In this current effort only arterial corridors are considered, although other segment definitions could be considered, such as between freeway ramp junctions. In the least-informed approach, trajectory prediction uses simple averaging or sampling of the segment historic or near-real time data. However, alternatives are explored where a given segment trajectory prediction may be conditioned on other predicted or near-real time behavior such as stopping (e.g., was the vehicle predicted to stop at an upstream intersection?), path similarity (e.g., sample only vehicles with similar upstream or downstream routes), or other trip variables.

Implementation. In the implementation of the Displacement Inference models, a routing table defines what road segments a given vehicle will traverse for a given route, in time order. An example of a routing table is $R = [4, 3, 2]$, where the numbers encode roadway segments. The routes are determined based on the driver input origin-destination pair.

In several of the prediction models, a trail (also, called history, window, or sub-sequence in other literature) of realized random variables is used in the prediction of the vehicle trajectory for a segment. For example, if a vehicle is predicted to stop at a segment intersection the likelihood of stopping at the downstream segment intersection will be conditioned on the realization of a stop at the upstream intersection. In some prediction models, the trail is multidimensional and contains multiple random variables (stops and time at crawl speed, for example). The *key type* in each model defines the random variable(s) used for the respective model.

The variable *segment_trail* ($-k$) expresses a past trail of segment identifiers of length k , up to, but not including the current segment. For example, an instance of *segment_trail* (-2) for Segment 3 may be [1,2,3]. In other words, the vehicle has traversed Segments 1 and 2 and now needs information about its potential behavior on Segment 3, conditioned on its previous path. As another example an instance of *segment_trail* ($+2$) for Segment 3 may be [3,4,5]. In other words,

the vehicle will traverse Segments 4 and 5 in the future, and now some information about its potential behavior on Segment 3 is desired, conditioned on its future path.

Different models may use different trail lengths, referred to here as a *lag*; this is sometimes referred to as *depth* and *order*. For instance, we may condition the likelihood of stopping on a given segment by the predicted stopping on the previous k segments. In the subsequent discussions, $M(k)$ will refer to a *Markovian* predictor that has a lag of k . Although most of the investigated random processes are unlikely to be memoryless, in this effort models are referred to as being Markovian by collapsing the current trail into the current state, even if the Markov property is still violated for lower lag values. Hash tables are used for some of the models, which are keyed by the exact match of the trail (of some random variable(s) over a window of size *lag*). Currently the returned value is a mean value obtained from observations in the training set that share the same trail.

The current prediction models make use of a kinematics tuple in the format $\langle \textit{mean running time}, \textit{mean running speed}, \textit{mean crawl time}, \textit{mean crawl speed} \rangle$ (abbreviated in variable format as $\langle t_r, v_r, t_c, v_c \rangle$), defined below:

- *Mean crawl time* refers to the average time the vehicle spends on a segment at a speed of less than 5 mph, including time stopped.
- *Mean crawl speed* refers to the average speed of the vehicle on the segment while its speed is less than 5 mph.
- *Mean running time* and *mean running speed* are defined in terms of the velocity complement (over 5 mph) of the preceding definitions, respectively.

It is important to distinguish these categories to generate accurate predictions of emissions. A vehicle trajectory is constructed by accumulating the predicted speeds v_i for time Δt_i , for all i over trip duration \mathbf{T} . Where v_i is the speed over Δt_i at time step t_i of the total trip time \mathbf{T} . Thus, the sum of Δt_i over all i is \mathbf{T} . For example, the predicted segment vehicle trajectory

represented by kinematic tuple $\langle 22, 35, 12, 3 \rangle$ would be the vehicle traveling at 35 mph for 22 seconds followed by 3 mph for 12 seconds. The next traversed segment's kinematic tuple represents the next portion of the vehicle trip.

Global Segment Estimation. This method seeks to predict vehicle trajectories using the global (historic) average travel times and speeds of prior vehicles that traverse a segment during a similar time period (e.g., peak travel period data). During the training phase, the kinematics tuple is updated for each observed segment. There is no further conditioning; the key consists solely of the segment identification number (id). During the training phase, the data structure being populated is *segment_to_kinematics_tuple*, which is keyed by segment id and has values $\langle t_r, v_r, t_c, v_c \rangle$. During the testing phase, a vehicle's segment routing table is supplied to the prediction model in its sequential ordering. To produce a trajectory for each segment the segments kinematics tuple is retrieved, i.e., the mean running speed and the mean running time, followed by the mean crawl speed and the mean crawl time. The trip trajectory is then constructed through the accumulation of the individual segment trajectories.

Local(n) Segment Estimation. This method is similar to the Global Segment Estimation, although it reduces data needs by using only the data from the n -most-recent vehicles to traverse the segment, rather than all historical data. During the training phase, the predicted kinematics tuple is estimated from the preceding n vehicles that traversed each particular segment. The *segment_to_kinematics_tuple* is again keyed by segment id and also has floating point values $\langle t_r, v_r, t_c, v_c \rangle$. During the testing phase, the vehicle trip trajectory is then constructed in the same manner as in the Global Segment Estimation, however, using the Local(n) Segment Estimation kinematics tuples.

Segment $M(-k)$ Estimation. This method seeks to improve the prediction for a client vehicle by sampling only those vehicles that travel the same k prior segments. During the training phase, the kinematics tuple $\langle t_r, v_r, t_c, v_c \rangle$ over the considered segment is estimated for each

segment trail of lag k . An example (for $k=2$) is $\langle [1, 2, 3] \rangle$ which means that the vehicle has traversed Segment 1 and 2; the relevant segment for the kinematics information extraction and tuple update is Segment 3, as this is the last element (i.e., the segment of interest) in the segment trail. The data structure is `segment_trail_to_kinematics_tuple`, which is keyed by `segment_trail(k)` (a vector) and has values $\langle t_r, v_r, t_c, v_c \rangle$ (i.e., the kinematics tuple). During the testing phase, the segment routing table of each tested vehicle is supplied to the prediction model. The vehicle trip trajectory is then constructed in the same manner as in the Global Segment Estimation, however, using the Segment $M(-k)$ Estimation kinematics tuples.

Segment $M(+k)$ Estimation Ignoring Stops. This is identical to the previous model, with the exception that the conditioning for each segment occurs over the future routing table. For example (if $k=2$), $[3, 4, 5]$ supplied as a key from a test vehicle means that we wish to obtain the kinematics tuple for Segment 3 given that the vehicle will traverse Segments 4 and 5 immediately thereafter.

Segment $M(-k)$ Estimation Including Stops. Here stop information, is included in the conditioning, attempting to further improve prediction by not only conditioning on similar paths but also on similar stop histories on those paths. During the training phase, *the kinematics tuple* $\langle t_r, v_r, t_c, v_c \rangle$ over the considered segment is estimated for each stopped-segment trail of lag k . An example (for $k=2$) is $\langle [1, 2, 3], [\text{true}, \text{false}] \rangle$, which means that the vehicle has traversed Segment 1 and 2, and has stopped in Segment 1 (the first element of the stop list is true), but not Segment 2 (the second element is false); the relevant segment for the kinematics information extraction and tuple update is Segment 3, as this is the last element (i.e., the segment of interest) in the segment trail. The data structure is `segment_stop_trail_to_kinematics_tuple`, which is keyed by $\langle \text{segment_trail}(k), \text{stopped_trail}(k) \rangle$ (each of these is a vector) and has values $\langle t_r, v_r, t_c, v_c \rangle$. During the testing phase, the segment routing table along with the stop history of each tested vehicle is supplied to the prediction model. The dimensionality of the input is larger as the

conditioning occurs over not only the visited segments, but also the stop history of the visited segments. Upon querying, only the point estimates generated from the training set that share the exact segment trail and stop history of the visited segments are considered. During the testing phase, the vehicle trip trajectory is then constructed in the same manner as in the Global Segment Estimation.

Evaluation. For the evaluation of these models, we used the Federal Highway Administrations Next Generation Simulation (NGSIM) data. This data contains high-resolution vehicle trajectory data collected for enhancing existing traffic flow models and developing new microscopic models (NGSIM Community Home., n.d.). The current efforts utilize NGSIM data collected on the Peachtree Street in Atlanta, Georgia, between 10th Street and 14th Street. The section is approximately 2,100 feet in length, with five intersections and two to three arterial through lanes in each direction. Figure 25 depicts the section. The speed limit on this corridor is 35 mph with multiple midblock driveways. The data were collected from 4:00 p.m. to 4:15 p.m. on November 8, 2006, at a resolution of 10 frames per second. These data provide complete coverage of all vehicle trajectories on this section of roadway, during the specified time, resulting in one of the most robust data sets available. In the current evaluation of the models, only NGSIM vehicle trip trajectories that cover the entirety of all segments traversed are considered. This avoids the complication of incomplete trajectories in the tables. Most of these trajectories have [1, 2, 3, 4, 5]-realized segment routing tables (i.e., the entire corridor is visited). Future efforts will expand to include vehicles that with partial segment data.



Figure 25: Peachtree Street from 10th Street to 14th Street (Map data: Google, 2018)

The NGSIM data was used to “train” the model (to establish conditional performance distributions), allowing each model to run and predict vehicle trajectories. While robust, the NGSIM is limited in size so this effort does not reserve a subset of the data for comparison to the predictions. Current efforts are limited to a comparison of the quality of the prediction with the training data. However, ongoing efforts are using simulated data sets and probe vehicle data to further explore the robustness of the predicted trajectories.

Data Preprocessing. The NGSIM data were first loaded into client memory by using a table data structure. To provide the segment prediction models with the required point estimates, dictionaries are first populated to map segment identifiers to mean velocities and mean travel times. The latter is calculated by counting the number of rows (in the table) along each specific segment id, for each vehicle id (each row corresponds to a tenth-of-a-second). Further, the minimum and maximum y -coordinates for each segment are extracted from the dataset.

Vehicle Trajectory Generation. Before control is given to the main training and testing loops, the predicted and actual trajectory containers, as well as the Markov tables are initialized. In the first main loop structure, the first and last row for each considered vehicle in the NGSIM data are first obtained. The direction of travel and all visited intersections along a trajectory are saved for each vehicle. The relevant Markov tables are populated and the actual trajectory is stored in a container. As mentioned previously, the kinematics tuple is the central piece of information used for the models. It contains point estimates for the stop duration, the stop velocity, the running duration, and the running velocity. The trajectories are built segment-by-segment. The following subroutine shows how this is accomplished for a given segment. The inputs are *kinematics*, the kinematics tuple, and *trajectory*, a reference to the working trajectory being built. Figure 26 shows how the trajectories are generated for a specific segment using the variable names from the kinematics tuple.

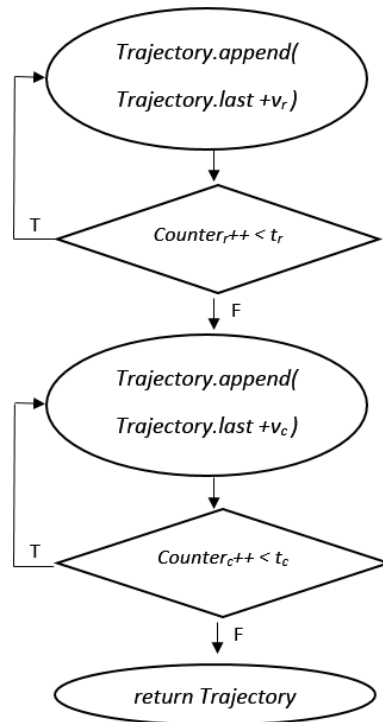


Figure 26: Flow graph of trajectory generation procedure for a single segment

First, the trajectory is augmented, for each tenth-of-a-second (0.1 second) timestep, with the displacement values inferred from the run speed point estimate of the kinematics tuple (v_r) over the expected run duration (t_r). Second, the vehicle's trajectory is extended with the crawl movement over the expected crawl duration. The model assumes that vehicles first travel in an unobstructed fashion, and then slow down in response to intersection queues. For each tenth-of-a-second within each interval, the procedure takes the last accumulated displacement that was appended and adds the next value to it. In this case, the value is based upon the historical travel speeds (v_r and v_c) across the segment. Both crawling and running could occur on the same segment. Further, the stopping speed (i.e., speed less than 5 mph) is inferred from historical information across that segment.

Figure 27 shows the observed NGSIM trajectory of NGSIM Vehicle 17 in red, along with the predicted trajectories; the line colors for the respective prediction models may be decoded by the included legend.

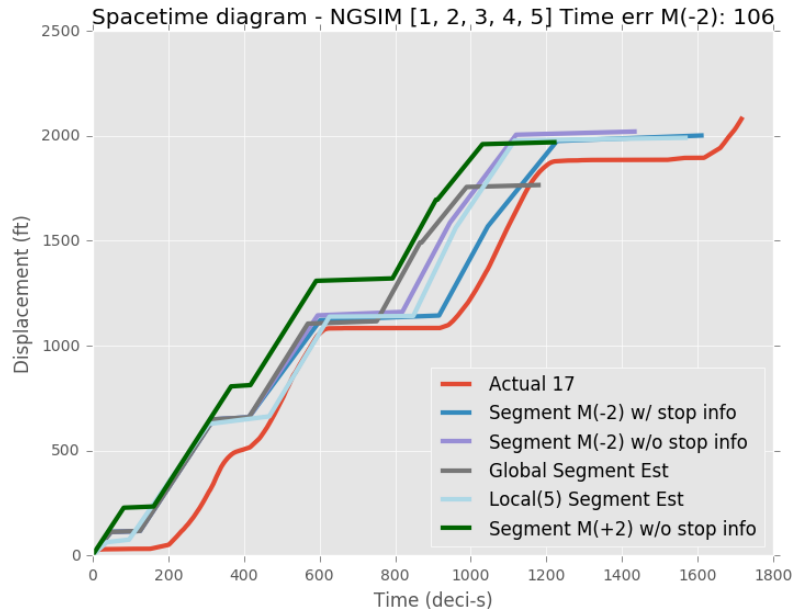


Figure 27: Actual and Predicted Trajectories in the Space-Time Diagram

The accumulated displacement of the vehicle is plotted against the elapsed travel time. The given vehicle traversed all five segments of the data set. The predicted trajectories appear to capture the overall behavior of Vehicle 17 from 40 seconds to 120 seconds. The underpredicted crawl duration, during the first 20 seconds, in Segment 1 results in overestimation of the displacement throughout the trajectory (for all the models). The deceleration curves of the actual trajectory are relatively smooth compared to the predicted trajectories. Future research is aimed at resolving both these issues: the accurate modeling of intersection stop times, as well as the smoothing of the deceleration curves.

3.7 Conclusion and Future Work

This chapter examined different route prediction models on the real-world T-Drive trajectory data sample collected in Beijing, China. All of the models tested can be executed reasonably quickly, even on mobile devices and/or simulation engines. For models making use of large data-sets, queries may be performed in the cloud. Some of the models are provided with the current trajectory of the driver, up to just before the tested zone, while others are provided with a relatively large training set of previously observed trajectories. In all cases where the model is permitted to use historical route data, rather than just the observed partial route, we observe substantial improvement in the forward-trajectory prediction accuracy on the T-Drive trajectory data sample - especially the order-2 Markov models and the Krumm-Markov hybrid model.

We would like to emphasize the value of exploiting dynamic data, in accordance with the DDDAS paradigm, for the vehicle tracking problem (see Section 2.2). It becomes especially important to improve the single trial reacquisition probability, when the scheme is repeated for a tracked vehicle. In any case, the economies of scale of the data acquisition costs should be compared to the tracking equipment acquisition and maintenance costs combined with the utility of recapturing the target vehicles. The value of successful trajectory prediction is observed in many commercial and non-commercial applications and is likely to grow as the penetration of

smartphones and augmented reality devices increases. The models may also be used to generate accurate trajectories for entities in microscopic traffic simulation.

Areas of future work include examining more factors and blended models on potentially different conceptual models; for example, roads being modeled rather than zones by randomly selecting among possible subsections in trajectories whose sampling interval may be relatively long. Another area of future work is to investigate the impact in the accuracy of simulation output statistics, when these prediction models are used instead of the traditional inputs. Rather than just predict the forward trajectory statically, one can envision predicting the forward trajectory by incorporating time.

4 EXECUTION OF REPLICATED TRANSPORTATION SIMULATIONS WITH UNCERTAIN VEHICLE TRAJECTORIES³

³*This chapter includes content from (Philip Pecher, 2015).*

4.1 Introduction

In some Monte-Carlo simulation applications, the output statistics are constrained to observations collected on a single entity that moves in a network. This type of simulation is usually executed several times in order to capture uncertainties of the underlying system. Across these replications, there are often substantial similarities in state and behavior. The method described here tries to leverage these similarities in order to reduce the computation time, while still maintaining the same results as the traditional approach of “brute force” replications. To illustrate this concept, we focus on microscopic traffic simulation and motivate the need for computational acceleration in the context of Dynamic Data-Driven Application Systems (DDDAS).

There are many applications that utilize microscopic traffic simulation models driven by online data to analyze and optimize operational transportation systems (e.g., see (Suh, Hunter, & Fujimoto, 2014)).

As an example, we again posit a motivating DDDAS application concerned with tracking the location of a vehicle where continuous surveillance is either not possible or not desirable. Recall such a DDDAS deployment may repeatedly execute a cycle that includes (1) detecting the current location of the vehicle, (2) predicting the likelihood of possible locations some time into the future, and (3) focusing surveillance efforts on the most likely future locations, e.g., by re-

positioning probing entities or concentrating data analysis efforts in certain areas (Fujimoto R. A., 2014). Realizing this DDDAS application may require the completion of many replicated simulation runs. For instance, the second step involves executing a set of replicated simulation runs where each run models the target vehicle following a different potential route to reach some (potentially unknown) destination. In addition, to further optimize sensor repositioning other unknowns may need to be captured such as changing traffic conditions and travel time variability within a traffic condition. In this chapter we propose to exploit the similarities among these runs to optimize execution time or minimize needed computational resources.

For example, in runs that only differ according to the route taken by the target vehicle the trajectory of vehicles not impacted by the target vehicle will remain the same across the different runs. Further, the output statistics of interest may not require portions of the simulation to be completed at all as simulation computations concerning vehicles far away from the target as the target approaches its destination will not affect statistics related to the target vehicle. The focus of this research is to develop and evaluate techniques to reduce the amount of computation required given these similarities, thereby reducing the time required to gain future estimates and/or reduce the computational resources that are required for the DDDAS application.

Here we present an algorithm to accelerate the generation of results from N similar, replicated simulation runs without loss of accuracy. In other words, the results that are produced should be identical to those produced by a brute-force approach of completing N independent simulation runs. The problem is stated as follows. Let E_T denote the target vehicle, i.e., the vehicle of interest. Let us assume its position is known at time t_0 and that a prediction model has determined a set of possible future paths and their likelihoods. The simulation now aims to determine the time at which E_T reaches certain user-specified points along each of these paths. After extracting real-time information such as current traffic conditions throughout the region, the traditional approach would perform N replications each simulating E_T using a different route. Our

goal is to minimize the amount of computation that must be performed while still computing the same results as the traditional replicated approach. An extension of this work is to consider multiple replications using a single route, e.g., to consider the impact of stochastic or other variations, however, we will not explore this direction here.

4.2 Related Work

An application such as this requires one to predict a set of likely destinations for the target vehicle. There have been numerous efforts that have developed methods to predict this destination set. For example, in Chapter 2 the routes the vehicle might take and aspects such as traffic congestion that will impact its travel time are considered. A prediction model that assigns destinations a higher likelihood if they are consistent with the path taken thus far can be found in (Krumm J. , 2006). Neural networks have also been investigated for use in destination prediction (Miklušćák, Gregor, & Janota, 2012). If limited data are available, the SubSyn algorithm from (Xue A. Z., 2013) can be used to synthesize new trajectories from previously collected ones. A summary of relevant models can be found in (Pournajaf, Xiong, & Sunderam, 2014). Given these and other efforts the purpose of this chapter is not to modify a previous prediction method. Rather, the aim is to illustrate how one can *quickly* execute a traffic simulation that projects the most probable positions of the vehicle at some point in time into the future.

Sharing computation among replicated simulation runs in the context of cloning running simulations is described in (Hybinette & Fujimoto, 2001), which served as motivation and the starting point for this work. The approach described here differs in three ways. First, the approach described here focuses on transportation simulations though many of the ideas can be generalized to other applications. Second, we introduce speculative execution as a means to improve performance. This involves considering the output statistics produced by the simulation in order to determine those simulation computations that must be completed. Third, like the approach described in (Hybinette & Fujimoto, 2001) portions of the simulation are replicated as needed

during the computation as the different replications diverge using a technique called incremental cloning. Here, incremental cloning is applied to selected state variables as needed rather than an entire simulation process, yielding a more efficient implementation.

Another related approach is called updateable simulations (Ferenci, Fujimoto, Ammar, Perumalla, & Riley, 2002). This is a technique that first executes a baseline simulation run, creates a log of the entire execution, and then computes the results for other replications by determining the computations that are different and updating the log, thereby reusing results from the baseline run. The updateable simulation approach analyzes the events of the baseline simulation for potential speedups in the management of the Future Event List (FEL) for subsequent runs. The approach described here does not require completion of a baseline run and the creation of an event history log, nor the management of an FEL. Further, for each different scenario, the updateable simulation algorithm must explicitly execute another run (albeit with reduced computation), which is avoided with the algorithm presented here. Lastly, the reuse discussed here does not stem from unchanged events, but rather from the invariance in certain state variables across the different scenarios.

The approach described here uses the concept of minimum propagation delays, i.e., the minimum amount of simulation time that must elapse before one object in the simulation can affect another. This is a familiar one in the parallel discrete event simulation literature. Stemming from its origins captured in a term called lookahead (Chandy & Misra, 1979), this concept is generalized as the distance between objects and used to synchronize parallel simulations, e.g., see the work described in (Ayani, 1988), among others. Here, distance between objects is exploited to help determine those computations that may affect the statistics produced by the simulation in order to improve the efficiency of the proposed algorithm, especially the tagging mechanism described later. For example, if statistics are needed for the target vehicle E_T at simulation time T ,

one can use the distance between objects concept to determine those vehicles that might affect the statistics concerning E_T at time T .

4.3 Simulation Algorithm

For this study we use the (Nagel & Schreckenberg, 1992) model for traffic based on cellular automata. Although the concepts presented here generalize to other temporal and spatial resolutions, it is straightforward to illustrate the concepts used in the simulation algorithm with the Nagel-Schreckenberg model. In this model, a road is mapped to an array of neighboring cells. Each cell either contains a vehicle or is empty. A vehicle, at any given time, includes a speed attribute that is represented as an integer. A global maximum speed is also defined. At each time step, the following actions are applied to every cell-occupying vehicle:

1. The speed of the vehicle is incremented if the vehicle is not at the maximum speed.
2. The current vehicle speed is compared to the number of empty cells in front of the vehicle. If the number of empty cells is smaller, the speed is reduced to the empty cell count (in order to avoid a potential collision).
3. If the vehicle's speed is positive, it is decremented with probability p (this is a predefined parameter).
4. The vehicle advances forward by the number of cells equal to the corresponding speed.

Since 1992, multi-lane models have been proposed where it is possible for vehicles to overtake each other. This aspect is not modeled here, however, the algorithm proposed below easily generalizes to multi-lane models.

The approach used here is summarized as follows. The source of variation among the replications stems from different routes taken by the target vehicle E_T . To simplify the discussion, we assume the statistics of interest are concerned with properties of E_T , e.g., its temporal-spatial trajectory through the road network over time. We superimpose N computational sequences, each

modeling one trajectory of E_T onto a single run where E_T is absent. Each such trajectory is produced by what is referred to as a virtual instance of E_T , i.e., a virtual vehicle (VV). The other non-target vehicles in the simulation are called model vehicles (MV's). Whereas the traditional approach simulates one instance of E_T and all of the MVs for each of the N replications, here the *single* superimposed replication simulates N VVs, and initially one set of MVs. Each VV is therefore logically associated with one replication in the traditional, brute force approach.

If an MV interacts with a VV and as a result behaves differently compared to the case where E_T is absent, the simulated trajectory of MV is erroneous; for example, MV may need to reduce its speed to account for the VV. In this case the MV is referred to as a *hazardous* MV. A hazardous MV is at risk of causing errors in the output statistics, however, a hazardous MV may not actually affect these statistics. For example, if the correct behavior is the MV should slow down, its corrected behavior may not have any impact on the trajectory of VV, which is the focus of the simulation. Additional MVs that interact with a hazardous MV may also become hazardous. In order to determine if the output statistics have been compromised, a tagging mechanism is used to flag “hazardous” MV's and their actions. Once a hazard is confirmed to have impacted the output statistics, a new execution path is created (a physical "clone," using the terminology from (Hybinette & Fujimoto, 2001) that explicitly computes the events for the specific VV that originally caused its first tag to appear. During certain timesteps, the simulation may clear tags that are guaranteed not to cause any errors in the output statistics. It should also be noted that in most cases, *one* superimposed replication may not capture sufficient variation in the environment. In order to capture not only variation among the attributes of E_T , but also the simulation environment, the modeler may use a batching scheme for a set of superimposed replications. Section 4.4 shows how many VV's should be realized within a single superimposed replication for a particular case study.

4.3.1 Superimposed Execution Before Hazards Arise

In the following illustration, we adapt the original Nagel-Schreckenberg model for our purposes. We assume that the parameter p is only global for MV's. Each E_T instance realizes its own p_i parameter from a predefined density before it spawns. Further, each cell may contain any number of vehicles in the proposed approach, but there may at most be one MV contained within one cell (the reason for this will be explained shortly). Each vehicle samples a routing sequence from a probability table before it spawns and owns this sequence throughout its lifetime. All vehicles (MV's and VV's) will execute all four steps outlined in the previous section during each time step. In the examples below, we assume that the output statistics of interest are limited to measurements of the target entity E_T , namely the time taken to reach the final point of the routing sequence.

The left part of Figure 28 illustrates the traditional replicated approach executing the model. Each array of cells is a subsection of the road in a given replication at timestep 44. The speed reduction probability attribute p_i of each E_T instance is depicted; the routing sequence attribute, which governs the vehicle's behavior at intersections, is not displayed. One instance of the target vehicle, E_T (the cells with light orange shading), is shown for each replication. The model vehicle (light grey shading) in the rightmost cell has not been influenced by E_T in all three replications. It is important to note that the behavior of the environment does not change across replications, unless the environment is influenced by E_T . Note that the model vehicle shown in the leftmost cell in replications 2 and 3 has reduced its speed in Replication 1, prior to Timestep 44, due to the presence of a rather slow instance of E_T . Thus, the leftmost cell in replication 1 is empty while it is occupied in replications 2 and 3.

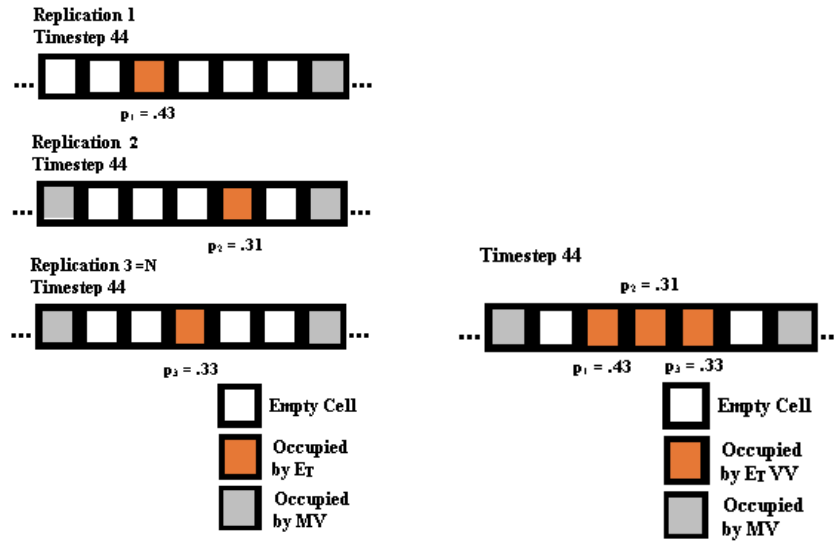


Figure 28: The traditional execution approach (left) and the proposed approach (right)

The right part of Figure 28 shows a snapshot of the simulation where the three replications are superimposed into one execution. Note that all three VVs appear in the superimposed execution. During collision detection (Step 2 of the Nagel-Schreckenberg algorithm) all vehicles will react to MV, but MVs will ignore virtual vehicles. This implies that it is possible for a cell to now contain both zero or one MV along with any number of VV's. Virtual vehicles do not interact with other virtual vehicles because they logically reside in different replications. VVs do interact with MVs, however. Lastly, MVs react to other MVs as in the traditional approach. The fact that they ignore VV's can create hazards. For example, the leftmost cell of the superimposed simulation is shown to contain a vehicle, even though that cell is empty in replicated simulation 1 (shown on the left). This vehicle represents a hazard for replication 1 in the superimposed simulation. In the following we refer to this vehicle as model vehicle A.

4.3.2 Speculative Execution with Hazards

As noted above, in replication 1 shown in left part of Figure 28 MV A was missing from the leftmost cell at Timestep 44, but is present in the superimposed simulation as well as the other two replications; the superimposed simulation represents the location of A in the absence of the virtual vehicle. This invalid state does not directly pose a problem because the desired output statistics are concerned with the virtual vehicle, and the virtual vehicle's behavior is not affected by A. However, it is possible A does affect another model vehicle B, e.g., a fast moving vehicle that overtakes A and then interacts with the VV. Unless precautions are taken the VV's output statistic may be incorrect if the behavior of B is not taken into account in the superimposed simulation. To address this problem, A is flagged as a hazard. When B interacts with A it too is also flagged as a hazard, and when B interacts with the VV the error is detected, and a corrective action, termed a rollback, must be executed.

The approach used here is termed *speculative execution* because VV's behavior is modeled assuming VV is oblivious to the impact that VV has on the rest of the simulation. This will not impact the results of the simulation so long as VV's impact on the rest of the simulation do not affect VV itself. If this assertion turns out to be incorrect, as exemplified by the above example, corrective actions are required to ensure the output statistics produced by the simulation match the brute-force replicated approach.

Once a model vehicle alters its behavior because of a virtual vehicle in front of it, there are now two states to consider:

- *State A.* The model vehicle does not consider the virtual vehicle and checks if there is another vehicle present in the road segment it tries to traverse in the current timestep. This is the valid state for all virtual vehicles except the one that was ignored.

- *State B*. The model vehicle considers the virtual vehicle during the collision detection step. This is an invalid state for all virtual vehicles except the one that was considered.

Because there are more virtual vehicles beyond the one being considered, it is beneficial to simulate using state A. In order to confirm whether this causes an error for the virtual vehicle that was ignored by the model vehicle, two possible sources of error must be considered:

- Error-Source 1. Had the model vehicle been blocked by the virtual vehicle, it could have caused model vehicles behind it (upstream) to slow down. In response, those model vehicles could alter their behavior (routing sequence, aggression etc.) and impact the virtual vehicle at some point in the future.
- Error-Source 2. The model vehicle behavior that ignores the virtual vehicle can also cause errors. While the virtual vehicle can ignore the specific model vehicle that passed through it, the model vehicle can now influence the behavior of other model vehicles in front of it that can impact the passed virtual vehicle.

In either of these situations two error-causing events may occur:

- Error I. The virtual vehicle encounters a model vehicle that it should not have encountered.
- Error II. The virtual vehicle does not encounter a model vehicle that it should have encountered.

If a model makes Error-Source 1 possible, one can simply clone the model vehicle and consequently run the scenario of the virtual vehicle along with the superimposed scenario, rather than to use the tagging rules (which are discussed shortly) for both the passing model vehicle *and* an alternative (blocked) instance of the same model vehicle. Which mechanism is more efficient (explicit cloning vs. two tagging sources) depends on the specific application. If the traffic intensity is high, it is likely that explicitly cloning all vehicles could be computationally intensive.

At the same time a high traffic intensity is likely to trigger a rollback when the tagging mechanism is used.

If only Error-Source 2 is possible, a forward tagging mechanism may alternatively be used to detect errors (explicit cloning could still be the better option for certain models). As soon as a model vehicle overlaps or overtakes a virtual vehicle, a tag is added to the model vehicle. A tag contains the *id* of the virtual vehicle, an occurrence count, and a flag that specifies whether this is the first model vehicle affected (this would be set to *true* as this is the first time the particular occurrence *count-id* combination has been observed). Whenever another model entity slows down - in the collision detection step - as a result of a tagged model entity in front of it, the tag is copied (with the flag field now being *false*). Additionally, when a tagged model entity reaches an intersection, the tag is added to the intersection. As soon as the virtual vehicle encounters one of its prior tags (at an intersection or in the collision detection step), the simulation explicitly executes the replication with the given virtual vehicle as the sole instance of E_T (as in the traditional approach) and the virtual vehicle along with all its tags are removed from the superimposed model. If other statistics from the virtual vehicle were collected, they must be rolled back. A virtual vehicle that encounters its tag on a model vehicle in front of it implies an Error I, while an encounter with its tag on an intersection could imply an Error II. The listing in Figure 29 shows the full algorithm, which replaces step 2 of the Nagel-Schreckenberg algorithm.

```

collision_detect(vehicle e){
    if (e.type == virtual)
        1. e doesn't collide with other virtual vehicles
        2. e collides with model vehicle that are not tagged with e's id but...
        3. ... e doesn't collide with model vehicles that have previously illegally overtaken e
or overlapped with e (check flag if the model vehicle is tagged with e's id) and...
        4. ...if e collides with a tagged (of its own id) model vehicle, remove e from current
model (along with all its tags) and physically clone e on a separate replication.
    if (e.type == model)
        1. e collides with model vehicles but...
        2. ... if e collides with a tagged model vehicle, e will copy the tag(s)
        3. e doesn't collide with virtual vehicles, but if a collision with a particular virtual
vehicle occurs for the first time, e is tagged with the virtual vehicle's id and flag.
}

```

Figure 29: Modified collision detection logic.

As model vehicles may own any number of tags, a list (of tags) is associated with them. For any tag, if *flag* is true, it means that the model vehicle overtook or overlapped with a virtual vehicle at some point. If it is false, it means that the model vehicle came into contact with another tagged model vehicle at some point. If the model is more sophisticated and contains traffic signals for example, tags must be copied every time there is a conditional link. For example, if a tagged model vehicle reaches an intersection, and, because of its presence causes the light to switch from green to red for another lane, the tag will have to be copied to every model vehicle in the stopped lane.

4.4 Experimental Evaluation

In order to evaluate the speedup of the superimposed approach, a synthetic simulation experiment (with randomly generated input) was performed with varying numbers of virtual vehicles on a road network. These experiments do not flush redundant tags using the minimum propagation delay techniques. The implementation is sequential; a work-optimal parallel version could easily be created by partitioning the cell regions and mapping them to logical processes (LP). Each LP would execute the algorithm listed previously for the cell transitions of its vehicles. Once a vehicle exits the region of a given LP, a message is sent to the LP that owns the neighboring region of cells.

4.4.1 Experimental Setup

The simulated transportation network has a Manhattan style topology consisting of nine north-south and nine east-west two-way roads. The intersection-to-intersection distance measures 100 cells. Each vehicle has a randomly selected intersection as its destination and will take a shortest path to it. Model vehicles are generated at the end points of each two-way road. Two scenarios will be considered. The "low traffic" scenario will generate a model vehicle with probability 0.1 for each source cell during each timestep, while the "high traffic" scenario will use a probability of 0.5. Model vehicles have a maximum speed of 3 cells per timestep and a $p=0.5$ deceleration parameter. The simulation runs for 500 timesteps and uses the superimposed Nagel-Schreckenberg algorithm with the tagging mechanism. The target vehicle E_T , is released at Timestep 3 and varies its deceleration parameter (0.2, 0.25, 0.3, 0.35, 0.6, 0.65, 0.7, and 0.75), its maximum speed (2, 3, 4, and 5 cells per tick), and its destination (uniformly selected among all intersections).

4.4.2 Speedup vs. Number of Replications and Traffic Intensity

Figure 30 shows the execution times for the naive approach of running one replication per realization of E_T (**B**rute **F**orce **E**xecution **M**ode) and the superimposed approach with the tagging mechanism (**V**irtual **V**ehicle **E**xecution **M**ode). The green and grey lines cover data points from the high and low traffic scenarios, respectively. As the number of replications increase, the VVEM execution time grows more slowly than the BFEM execution times. The E_T travel paths were verified to yield identical results for both execution modes. At 64 replications, the speedup of VVEM (relative to BFEM) is 5.8 for the low traffic scenario, while it is 1.65 for the high traffic scenario. This is intuitive because in highly congested systems, many model vehicles will be able to overtake the virtual vehicles. As a result, more model vehicles will be tagged, and their tag propagation rate is also much higher. Subsequently, the probability of having to spawn physical clones increases. The results show that one should opt for larger batch sizes when using VVEM - especially when modeling a highly congested network. The test computer is an Intel Core i5-3550 with 16 GB DDR3 RAM running Windows 7 64bit.

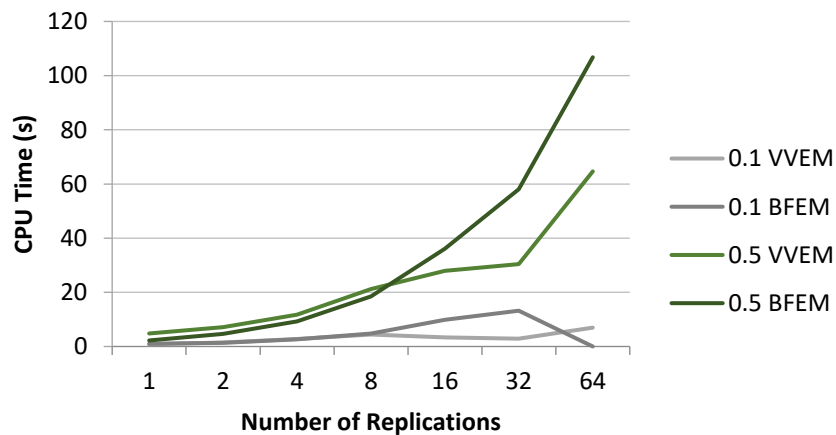


Figure 30: Execution time results by scenario

4.4.3 Speedup vs. Relative Speed Differences

Because the main VVEM run is, in most cases, only slightly more computationally intensive than a traditional replication, the VVEM speedup is equal to approximately $N/(I+C)$, where N is the number of replications (all N replications are executed by BFEM) and C is the number of physical clones triggered by VVEM. The number of triggered clones is primarily dependent on how many model vehicles are able to overtake the virtual vehicles. This metric, in turn, is directly related to the traffic intensity and the speed of the general traffic relative to the virtual vehicles. This is a result of model vehicles being tagged as they overtake the slow virtual vehicles and propagating the tag further. This increases the probability of a clone being triggered. Figure 31 shows the VVEM speedup for different values of the global speed limit parameter V (used by MV's) as well as the traffic scenario (high vs. low). Each scenario is run five times (in both VVEM and BFEM) for $N=16$ virtual vehicles for all permutations of low & high VV deceleration probability set ($\langle 0.2, 0.25, 0.3, 0.35 \rangle$ vs. $\langle 0.6, 0.65, 0.7, 0.75 \rangle$) and a low & high VV maximum speed ($\langle 2, 3 \rangle$ vs. $\langle 4, 5 \rangle$). The average speedup is displayed for each scenario. It is clear that a lower speed limit results in a higher speedup as the model vehicles are less likely to overtake the virtual vehicles (independent of the traffic intensity) and, thus, less likely to trigger physical clones. It is also noticeable that (comparatively) fast virtual vehicles result in a very high speedup (although it is more extreme in the low traffic scenario) as model vehicles are less likely to have to slow down and be tagged.

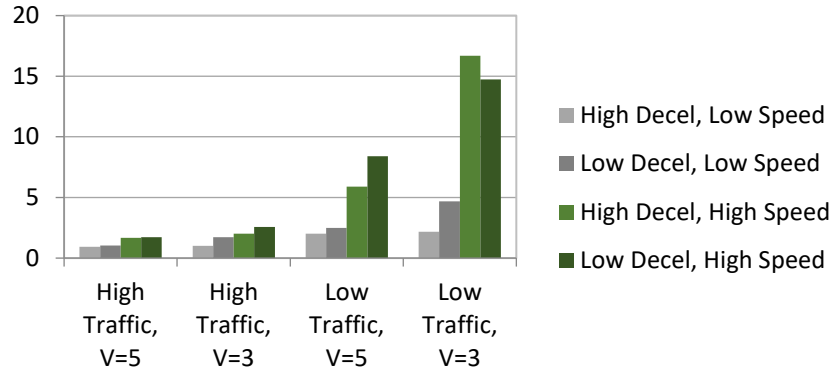


Figure 31: Speedup for different relative speed tendencies

4.4.4 Tag Memory Footprint vs. Traffic Intensity and Speed Limit

A VVEM replication uses more memory than a single BFEM replication because the tags from VV-MV collisions need to be stored as they are spread throughout the simulation environment. Figure 32 shows the rate of growth in memory requirements as the traffic intensity increases; no tag flushing procedure was used in those iterations. One could clear tags of VV's that have reached their destination as well as tags of MV's that cannot reach the corresponding VV based on the minimum propagation delay. The values were computed at Timestep 350 using 64 VV's.

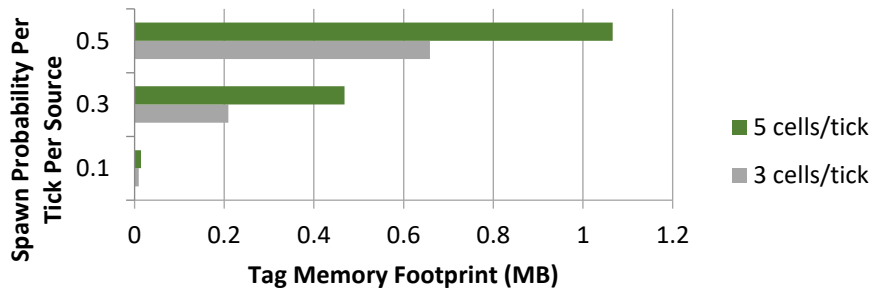


Figure 32: Memory requirements for storing the tags

4.5 Conclusion and Future Work

An algorithm to speed up replicated traffic simulations was presented. The concepts can also be applied to other models, but the benefits are particularly well-suited for models with similar characteristics. Three of these characteristics are the local movement of entities (triangle inequality), low connectivity (vertices with low degree), and simple output collection (travel time of E_T). The experimental results show that the execution time grows slowly as the number of E_T instances increases.

An area of future research is to investigate what heuristics can be used to determine how and when tags should be cleared. There is an overhead from performing a check, but there are also gains from fewer comparisons and reduced storage requirements. One can also envision a hybrid between execution with and without error detection where heuristics will compute the probability of a certain error occurring (e.g., given a macroscopic analysis of the traffic intensity). It is also worthwhile to study what models benefit from the superimposed approach and its performance using real-world network configurations. Under what circumstances is it worthwhile to physically clone, rather than verify whether an error occurs via the tagging rules? There is also a bias-variance tradeoff present. The analyst may simplify the model and speed up the execution. This will cause a reduction in the half-width for fixed computation time. However, using a more sophisticated model may reduce the bias at the expense of computational time (because certain optimizations cannot be used).

5 GRANULAR CLONING: INTRA-OBJECT PARALLELISM IN ENSEMBLE STUDIES⁴

⁴*This chapter includes content from (Pecher, Crittenden, Lu, & Fujimoto, 2018).*

5.1 Introduction

Computer simulation shows its strength by its flexibility to model a vast set of domains. However, its power is limited by the accuracy of the underlying model and the computational performance of the simulations. This paper focuses on alleviating the latter constraint. State changes are either driven by event-scheduling or by time-stepping through simulated time. The method described in this paper, termed granular cloning, can be utilized in both time-traversal modes, in both serial and parallel computer architectures, and – for parallel systems – in both optimistic and conservative synchronization protocols. Granular cloning exploits redundant computations across multiple runs of a model at the scale of individual simulation objects. Since it only affects common computations the results are equivalent to explicitly enumerating these runs. The runs are differentiated by random inputs and/or variations in the model scenarios.

In stochastic simulation, randomness is injected in input variables to model uncertainty of the system under investigation (SUI). The performance of granular cloning is improved with the use of common random numbers because the pseudorandom number streams can be shared by all the logical runs. If one wishes to introduce variability, granular cloning can be further improved by altering individual elements of the stream or augmenting the stream with more elements.

In many simulation studies, the objective is to find a feasible policy that leads to desirable output statistics. The number of feasible configurations often grows exponentially with the number of controllable discrete factors one is considering. Consider a network with ten nodes where one wishes to place one of three different router models at each node. The number of

possible configurations is 3^{10} . While there are methods for efficiently searching the often nonconvex configuration space, many runs will still be required. For each scenario, it is also desirable to complete many runs to obtain a point estimate of the output statistic with low bias and spread.

When variation is introduced into a set of runs of a certain model, the collective is referred to as an ensemble. An ensemble study may have a set of objectives, including characterizing (e.g., for validation) or optimizing certain model parameters (Fujimoto, Bock, Chen, Page, & Panchal, 2016). Whatever the way variation is introduced, random input or scenario variation, granular cloning superimposes all runs of an ensemble into one run and only explicitly stores the actual and most granular state differences across these runs, rather than larger supersets of state. Ensembles may be batched together based on a partitioning that is favorable for granular cloning. These conditions are discussed later. For example, suppose we model a particle p in a neighborhood of other particles and some gaps of “empty” space. We wish to investigate the evolution of the neighborhood for two different polarity values of p (two different *versions*). If most of the simulation state remains the same for both versions, it would be inefficient to explicitly store two sets of the entire state. We only allocate memory for neighboring particles if they are affected differently for a version compared to the other version. The number of objects with state differences for a given version in this example tends to grow monotonically with simulated time. Explicit enumeration of all runs would be wasteful because of the common states and behaviors across the runs. The technique is generally applicable for Monte-Carlo simulation, although one could construct adversarial models where the runs are completely different at model initialization, in which case the approach collapses into explicit enumeration of runs. At the same time, the technique incurs a small overhead whenever different simulation objects interact with each other. However, there are heuristics that can be used to reduce the overhead cost, which depend on the object interactions of the underlying application.

The next section describes related work. In Section 5.3, we define the notation and terminology from the simulation cloning literature (Hybinette & Fujimoto, 2001) along with cloning concepts. Section 5.4 describes how cloning can be extended to state objects within LP's. Section 5.5 brings attention to the quantitative conditions under which computational performance can be harvested without loss in output accuracy; this is expressed in a simple mathematical inequality that is intuitive from a geometric view. Experimental results of use cases are given in Section 5.6, varying key parameters that affect the computational performance.

5.2 Related Work

The concept of cloning memory segments has been used extensively since Von Neumann introduced it for fault tolerance in 1956 (Neumann, 1956). Since then, it has influenced modern relational database systems, distributed memory management, and other areas of computer systems. In addition to describing this historical context in detail, (Hybinette & Fujimoto, 2001) contains a survey of domain-specific simulation models where cloning has been used previously. Common themes in the relevant literature include computation sharing (reuse of common state or events) and incremental simulation.

Reducing the execution time and memory requirements of replicated simulation instances without affecting the accuracy of output statistics has been investigated in both domain specific and general (yet, typically, simulation-engine-specific) settings. Domain-specific use cases generally allow more state to be logically shared by grouping together similar replications at suitable times during the execution, and/or otherwise exploit the underlying application's specific characteristics for space/time efficiency. Relevant examples include the following.

In Chapter 4, we presented a lazy evaluation and speculative execution scheme for physical cloning of vehicle objects in microscopic traffic simulation. Vehicle objects that are independent of output statistics propagate tags containing the replication numbers they "touch." Their numbers are compared to vehicle objects that have a dependency on output statistics. If

there is a version match between the vehicle types, a physical clone must be launched via a rollback to a relevant saved state for explicit computation. (Lentz, Manolakos, Czeck, & Heller, 1997) apply incremental cloning to test different signature paths in a digital logic simulation for potential faults. The “offspring” (clone) of a “parent” is limited to four logical output values. (Vakili, 1992) uses a so-called standard clock scheme (an implementation of single-clock multiple systems, or SCMS) approach to execute – on a SIMD computer system - the same events simultaneously across multiple synchronized replications, which are parametrized differently (e.g., different service disciplines in a queue). The injected positive correlation makes this approach especially useful in simulation-based optimization (ranking & selection), albeit for a restricted class of applications.

Unlike these prior efforts, the granular cloning approach described here does not rely on domain-specific properties. Although domain-specific methods can exploit knowledge of the given application, general techniques are by definition more flexible. As discussed next, more general techniques have been developed. It is possible that combinations of these techniques complement each other if they exploit different aspects of the underlying application.

An example of a more general technique is the parallel cloning scheme introduced by (Hybinette & Fujimoto, 2001) that applies to the distributed LP event-scheduling paradigm. It is summarized in Section 5.3 because our granular cloning scheme will be described with similar terminology and notation. Just as with parallel cloning, granular cloning also makes heavy use of set-manipulation algorithms. Furthermore, the underlying mechanism for both algorithms consists of sharing state logically, while incrementally allocating physical memory. However, granular cloning offers a much finer grained mechanism that focuses on cloning individual objects rather than entire logical processes, and is agnostic to the underlying simulation executive paradigm. (Chen, Turner, Cai, Gan, & Low, 2005) have extended the distributed cloning algorithms for the

High Level Architecture (HLA) along with HLA's data distribution management (DDM) in (Chen, et al., 2010).

(Ferenci, Fujimoto, Ammar, Perumalla, & Riley, 2002) present an incremental scheme termed updatable simulation, where one first logs the events and sample path of a baseline run and hopes to reuse the state modifications from it for subsequent runs. Subsequent runs then determine what horizon of logged events can be reused at a given timestep where an event is to be processed. In the ideal case, a sequence of $r \gg I$ events can be composed together, rather than separately, to act on the current state. If the runs are sufficiently different, the event horizon r will be zero at each event-processing timestep and nothing can be reused. The staged simulation technique discussed in (Walsh & Simer, 2003) caches previous event invocations, decomposes them, reuses previously computed and/or similar results, and reorders restructured events for their efficient scheduling. (Stoffers, Schemmel, Dustmann, & Wehrle, 2016) extend automatic memoization to impure functions where side effects are permitted (subject to a few constraints). Granular cloning differs from these approaches in that it does not rely on reusing previously computed results.

Granular cloning is perhaps most closely related to recent work by (Li, Cai, & Turner, 2017) which focuses on tree-based cloning algorithms for agent-based models. A cloning tree contains nodes that map to simulation instances (*versions*) that have a different parameter than their parents. Each such instance is associated with two data structures: *AgentPool* and *Context*. The former contains agents whose characteristics are unique to that instance, while the latter contains references to shared agents in ancestor nodes. During the execution of an instance, the relevant agents from a child's ancestors are copied. A child instance performs clone condition checking as follows. If a *Context* agent (shared with an ancestor node) senses a parameter variation or an agent in the respective *AgentPool*, a clone is generated and moved to the *AgentPool*. Execution of simulation instances within the clone tree occur level-by-level in a

breadth-first manner. In contrast to this simulation instance-hierarchical scheme, granular cloning does not use a tree hierarchy; rather, it associates tags with object realizations and clone condition checking occurs by comparing the tags of interacting objects. Granular cloning accesses relevant objects in arrays rather than by tree traversal. A bit masking scheme is used to efficiently implement version management. Superficially, granular cloning applies to granular subsets of state, while by (Li, Cai, & Turner, 2017) applies to agent objects, though potentially one could adapt their algorithm to encompass the same scale. Furthermore, at any given timestep, the parallelism in (Li, Cai, & Turner, 2017) is limited by the available nodes at any level of the clone tree (batch-by-batch); in granular cloning, all objects may be executed concurrently at any timestep.

Granular cloning utilizes a kind of data dependency detection for interacting objects (or state variables thereof). Granular cloning compares properties of versions associated with these objects. Detecting various properties of data dependencies with the end goal of accelerating simulation execution has been studied in other works as well. (Quaglia & Baldoni, 1999) investigate data dependencies associated with a simulation object over several events. The scheme can be used to accelerate certain optimistic simulations by increasing event-level parallelism. The definition of “intra-object parallelism” in (Quaglia & Baldoni, 1999) differs from the usage in this thesis and in (Henriksen, 1995). In (Quaglia & Baldoni, 1999), the parallelism refers to the state transitions (i.e., events) that an object is exposed to, while here (and in (Henriksen, 1995)) it refers to the variations in the object’s state. (Marziale, Nobilia, Pellegrini, & Quaglia, 2016) measure the amount of data dependencies across processes in order to create suitable groups of these processes (“granular LPs”) dynamically in a Time Warp-based environment. The grouping that is derived from dependencies in (Marziale, Nobilia, Pellegrini, & Quaglia, 2016) applies to subsets of state (logical processes), while they apply to subsets of runs in this chapter. Furthermore, the grouping sequence for granular cloning is unique.

5.3 Background: Cloning in Parallel Simulations

(Hybinette & Fujimoto, 2001) introduce the concept of simulation cloning in the context of distributed and interactive simulations where the analyst can intervene and dynamically evaluate alternative futures (clones, identified by a version number that is incremented chronologically: $i = 1, 2, 3, \dots$) at decision points and find a policy that offers advantages over others. A decision point must specify how a given clone differs in its state from a given reference version and at which timestep it occurs. However, different courses of action may also be defined a priori to evaluate different scenarios and inject uncertainty into certain input variables; they can also be triggered by certain conditions at runtime. Clones are generated by specifying them at decision points or through certain interactions between logical processes (LPs). The underlying system architecture used in (Hybinette & Fujimoto, 2001) is based on distributed LPs and the executive operates with the event-scheduling paradigm. The physical system is mapped into a model in the computer and the state of this model is partitioned into LPs (identified with $j = A, B, C, \dots$). During runtime, LPs exchange time-stamped messages to change their state and schedule new events; an LP only changes its state after processing an event (scheduled by itself or with a message received from another LP). Consider a simulation that evaluates two scenarios where, at some timestep $t_b > 0$, some partial component of the state differs among them.

Simulation cloning makes use of computation sharing because:

- the sample path up to the decision point is only computed once. The set of virtual logical processes (VLP) being mapped to by each clone are shared by a shared memory region in the computer system, namely the corresponding physical logical processes (PLP), and
- only the VLP's that differ in their state after the decision point are explicitly duplicated (as a PLP) in memory and processed separately and in parallel; the VLP's that are identical continue to be shared within a corresponding PLP. As the given clone starts affecting state in other

VLP's, further PLP's will be incrementally spawned. An important point with respect to one key contribution of this chapter is that the algorithm in [1] clones an entire VLP, even if there is just a small difference in a state object within the VLP.

VLP's exchange virtual messages among each other, but each virtual message maps to a single physical message and each VLP maps to a single PLP. Whenever several virtual instances are mapped to a particular physical instance (that is, if they have the same state), computations are essentially shared. A given VLP is identified by (i) the LP identifier and (ii) the version of the clone it refers to. A given version is associated with a set of VLP's for every LP in the model.

It is useful to define some notation prior to delving into the actual cloning algorithm. As mentioned before, the entire state of a model is partitioned into LP's (identified by $j = A, B, C, \dots$) and each version (identified by $i = 1, 2, 3, \dots$) maps to a set of VLP's. Subsequently, we will refer to a particular VLP as $V(i, j)$. If, for fixed j , several VLP's map to the same PLP (only if they share the same state), the corresponding PLP is denoted $P(i, j)$ and i is the minimum version number of the VLP's pointing to $P(i, j)$.

To implement simulation cloning, one must alter the simulation executive to incorporate (i) message cloning and (ii) process cloning. Message cloning simply refers to a simple mechanism whereby a message sent from a PLP (sharing several versions in a set called $VSendSet$) and received by a PLP (sharing several versions in a set called $VRcvSet$) must be forwarded to other PLP's mapped to from versions contained in $VSendSet$ that are not in $VRcvSet$. Process cloning occurs when a PLP is mapped to by versions not contained in a receiving message. The respective message may cause a modification in the state of all versions that the PLP represents, which would clearly be invalid for the states that are not in the message's $VSendSet$. The full process cloning algorithm from (Hybinette & Fujimoto, 2001) is (S^C denotes the complement of set S with respect to the universe of versions):

- (1) Get VSendSet from message
- (2) Get VRcvSet from local state
- (3) $VPsUnaffected = VSendSetC \cap VRcvSet$
- (4) $VPsMoveToClone = VPsUnaffected$
- (5) $VPsRequested = VSendSet \cap VRcvSet$
- (6) If $(!(PhysicalReceiver \cap VPsRequested))$ then

$$VPsMoveToClone = VPsRequested$$
- (7) If $(VPsMoveToClone \neq \{\})$ then

$$Cloned\ PLP = \min(VPsMoveToClone)$$

5.4 Granular Cloning

5.4.1 Intuition

A straightforward extension to simulation cloning approach from (Hybinette & Fujimoto, 2001) is to replace the process cloning step with one that only clones state differences. An associative map is maintained for each PLP that maps a given version i to a record that contains information regarding where and how a PLP differs from shared state. In Figure 33, an LP modeling an airport initially shares the state for versions 1 and 2 and receives an airliner arrival event that only occurs in version 1. To maintain correctness for version 2, simulation cloning now copies the entire LP state prior to handling of the arrival. Granular cloning instead only duplicates the state that the arrival event handler would modify. These duplicated state variables are now valid for version 2 (airliner absent), while the variables they were copied from may now be safely overridden for the version 1 arrival event.

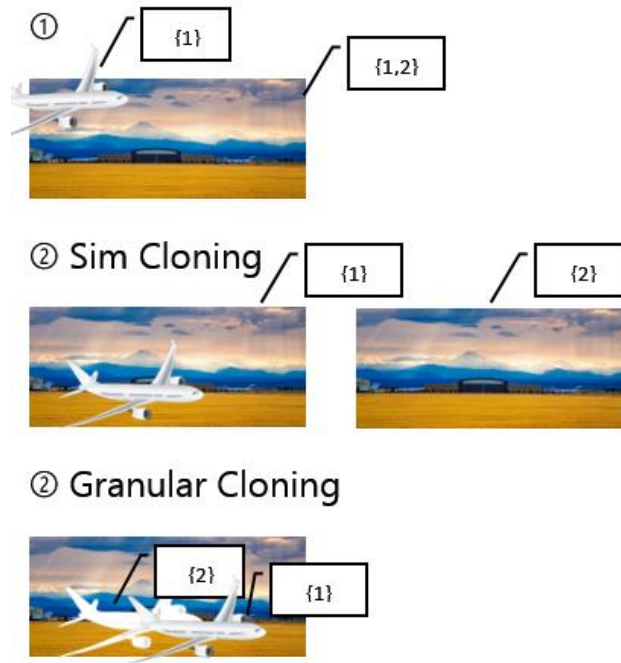


Figure 33: Rather than duplicate the entire LP, granular cloning only duplicates the state that changes. ©Free-Photos, Creative Commons 0

Relative to the parallel cloning scheme from Section 5.3, granular cloning should perform well if a given LP owns

1. a sufficiently large partition of the entire state space,
2. event handlers where state changes tend to be fragmented over the state variables, rather than over its entire state,
3. event handlers where unmodified state variables are insensitive to the changes to the state variables that are modified.

One could, of course, simply take the parallel cloning scheme and redefine each LP to encompass only a single state object. In doing so, however, one potentially increases the communication between the LP's which can significantly degrade performance.

The horizontal axis measures simulated time and the vertical axis contains discrete state space objects (in memory). The lines within the quadrants represent sample paths, here denoting simply the presence of a given physical state object. The state objects use a similar notation as that used for PLP's in Section 5.3. Here, the superscript refers to the versions mapping to the state object and the subscript denotes the identifier for the state object. At $t=15$, the *kernel* of A for its sole version set $\{1, 2\}$ is executed, depicted as c_1 . A kernel is a sequence of instructions computed for a version set of a single agent (called the kernel object) and is not permitted to write to any other object. If any write operation in the kernel depends on a read operation of other objects, the intersection of the read sources' version sets is compared to the version set of the kernel object. If the kernel object contains members not present in the intersection of the read sources, the kernel object splits. If one were to not split the kernel object, the state of the kernel object would be overwritten for the versions that are contained in it, but not in the read sources, potentially leading to errors.

5.4.2 Formulation

As summarized before, a timestepped ABM - from a high-level view - typically has the following operational scheme:

```

for s in scenarios
  initialize_model(s)
  for t in timesteps
    for a in agents
      a.kernel /*also called transition
                or step */

```

Algorithm 1: Agent-based model - operational outline (traditional)

When granular cloning is applied to these kinds of models, it obviates the need to explicitly enumerate the scenarios (since all of them are superimposed). However, at runtime, granular cloning needs to perform some bookkeeping, in order to trace dependencies between objects that are now associated with versions (*versioned objects*). As the simulation progresses, versioned objects that originally shared state across all versions start to diverge from the shared state to a version-specific state. From a high-level view, the operational scheme for granular cloning applied to a timestepped ABM looks like:

```
initialize_model(scenarios)
for t in timesteps
  for a in agents
    for vo in a.versionedobjects
      a.GC_interaction /*kernel with
                       granular cloning*/
```

Algorithm 2: Agent-based model - operational outline (granular cloning)

In some sense, the outermost loop in Algorithm 1 has been logically replaced with the innermost loop in Algorithm 2. However, we hope to iterate over fewer than `scenarios.number` (the total number of runs, or versions) in the innermost loop of Algorithm 2 by exploiting common state across the `versionedobjects`. The degree of reduction (or state-sharing) needs to be sufficiently high to offset the additional overhead that occurs in `a.GC_interaction` (in Algorithm 2) relative to `a.kernel` (in Algorithm 1). Most of what follows in this subsection is concerned with the details of the last line (`a.GC_interaction`) in Algorithm 2. In fact, `a.GC_interaction` is presented as Algorithm 3 and represents the high-level granular cloning kernel management by ensuring that versioned objects that diverge from any shared state have the kernel applied to them as well. In order to (i) report potential diverging splits in versioned objects and to (ii) dispatch the actual kernel function on versioned objects, Algorithm 3 calls Algorithm 4 (`GC_transition()`). (i) is computed in Algorithm 5 (`handle_conflicts()`). The following diagram summarizes this granular cloning mechanism:

Algorithm 2

- ABM executive with granular cloning
- Iterates over timesteps, agents, and intra-agent realizations
- Calls Algorithm 3 to execute transitions correctly

Algorithm 3

- GC_interaction()
- Dispatches granular cloning-based transitions for existing versioned objects and for discovered divergent versioned objects
- Calls Algorithm 4 to determine whether divergence occurs and to execute the kernel

Algorithm 4

- GC_transition()
- Traces dependencies among versioned objects and creates additional versioned objects if necessary
- Executes the kernel for non-divergent versioned objects
- Calls Algorithm 5 to track dependencies and determine if divergence occurs.

Algorithm 5

- handle_conflicts()
- Tracks dependencies and determines whether versioned objects with shared state would be overridden by another version object which is mapped to fewer versions

Granular cloning for agent-based models is differentiated from conventional execution in that it needs to preserve the integrity of the superimposed sample path. Before executing any transition function, granular cloning first ensures that no version-specific state of the kernel object is being wrongly overwritten for a dependency that does not match that version (this check is done in a function called `handle_conflicts()`). If there is a version conflict, the current kernel object needs to be split. The split object is handled after the current kernel object. At a high level, the kernel management algorithm for a set of kernel objects – here, `versionedobjects`, is as follows:

```
VersionObj[] split_versionedobjects
for VersionObj versioned_obj in versionedobjects
    GC_transition(versioned_obj,
        split_versionedobjects)
while split_versionedobjects.has_elements
    GC_transition(split_versionedobjects.pop,
        split_versionedobjects)
```

Algorithm 3: High-level granular cloning kernel management

The variable `split_versionedobjects` maintains potential split kernel objects that fragment from the currently considered kernel object and still need to be handled by the kernel. For example, suppose a given kernel object, `versioned_obj`, is associated with versions 1 and 3, and the granular cloning transition manager `GC_transition()` determines that `versioned_obj` needs to be split from the version set $\{1,3\}$ into $\{1\}$ and $\{3\}$. In this case, `GC_transition()` will only override the kernel object corresponding to the intersection of the versions sets of all read sources. Suppose, in our example, the intersection is $\{1\}$. Now, the kernel object for $\{3\}$ is still unhandled and still needs to be processed. In the while-block of the granular cloning kernel manager, we simply execute the kernel for all these split objects until they are all handled.

A given object's version set is encoded in unsigned integers where bits are set for corresponding version indices; granular cloning makes heavy use of bit manipulation. For example, the following binary value encodes a mapping to versions 1 and 3 (1-based indexing) among 32 total versions:

```
0000 0000 0000 0000 0000 0000 0000 0101
```

The granular cloning-wrapped kernel function `GC_transition()` performs the following steps:

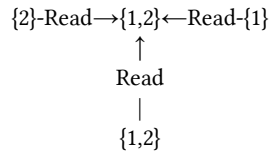
```
GC_transition() (VersionObj vo, VersionObj[]
splits)
    VersionObj[] potential_splits =
        vo.handle_conflicts()
    Versions unaffected_versions
    if potential_splits.has_splits
        splits.emplace_back(potential_splits)
        unaffected_versions =
            potential_splits.get_versions()
    vo.skernel(unaffected_versions)
```

Algorithm 4: Low-level granular cloning kernel wrapping

As previously mentioned, `splits` is an output argument that will store unhandled versioned objects in case we need to split the current kernel object `vo`. Whether splitting will occur depends on the comparison of the version set of `vo` and the versions of all objects we read in the kernel (i.e., all dependencies of the kernel object `vo`). The `handle_conflicts()` method determines if the version set intersection of all objects that are being read in the kernel (all dependencies) are a strict subset of the kernel object `vo`'s version set. If they represent a strict subset, the unhandled split version objects need to be stored in `splits`. The version set of the unhandled split version objects are maintained in the `unaffected_versions` variable. Finally, we execute the superimposed kernel, `skernel()`, for the kernel object `vo`, but only for the versions that are not included in the unhandled splits – that is, the logical complement of `unaffected_versions`. For the complement of `unaffected_versions` we can be sure

that we can read the corresponding version sets from all the read sources because `handle_conflicts()` already filtered out the maximal version set we can manage (by the definition of set intersection).

In `handle_conflicts()`, a tie breaker rule will determine what version number serves as the foundation of integrity checking. An arbitrary selector, like the filtering for the minimum version, will denoted with `SEL`. Once that version is determined, one must filter neighboring objects for corresponding sets. Consider the following example:



In this case, the min-selector uses version 1. The intersection of $\{1,2\}$ and $\{1\}$ is $\{1\}$ and so we split $\{1,2\}$ into $\{1\}$ and $\{2\}$. We update $\{1\}$ with the kernel and leave the state mapped to version set $\{2\}$ unaffected; $\{2\}$ will be handled in the while-loop of Algorithm 3 later. The state of the kernel object (after the split, only associated with $\{1\}$) does not only depend on the state of the versioned object to the right, but also on one on the bottom. The while-loop of Algorithm 3 ensures that any fragment of the current kernel object will be handled by the read sources that are omitted (those fragments will potentially be split further).

Without an automated runtime that can inspect all object identifiers of a kernel invocation, it is necessary for the user to specify what objects the kernel accesses. In many ABM's the read sources consist of some neighborhood – in a given coordinate system - around the kernel object. For example, in Conway's Game of Life (Conway, 1970), a given kernel object (a cell that can assume a state of 1 or 0), which is located in a 2D grid, reads from its eight neighbors. In the general `handle_conflicts()` method (listed in Algorithm 5), it is assumed that one can access the entire state of the previous timestep (`OLD`) as well as the read sources relative to the kernel object (`NBR`) by some defined addressing scheme (i.e., coordinate system for

most ABM's). There are more general variants for `handle_conflicts()` and some of them may be more efficient for a particular application. One can also engineer optimized versions for specific application where certain characteristics are exploited. For example, one fairly common characteristic among cellular automata is that all the versions are present (with mutual exclusivity) for each element of NBR.

```
VersionObj handle_conflicts()
    Version ref_version = SEL(this.versions)
    VersionObj[] read_sources =
        get_sources(this, OLD, NBR)
    VersionObj[] ref_vo = filter(read_sources,
        ref_version)
    Version working =
        intrsct(getversions(ref_vo), this.versions)
    For Version v in
        working.get_onehots_except(ref_version)
            working = intrsct(working,
                getversions(filter(read_sources, v)))
    if working != this.versions
        VersionObj res(this.state, working)
        res.set_splits(true)
        return res
    else
        VersionObj res(WILDCARD_S, WILDCARD_V)
        res.set_splits(false)
        return res
```

Algorithm 5: One possible general variant of `handle_conflicts()`

Algorithm 5 first arbitrarily picks a reference version from the kernel object's version set with the selector function `SEL`. It then proceeds to collect all the kernel dependencies and filters those objects with the reference version that was just selected. A working version set is then used to determine whether there is version-consistency between the dependencies' (`ref_vo`) and the kernel object's respective version sets. The intersection ensures that we enter `skernel()` with the maximal version set that is still common among all dependencies (and the kernel object itself) and preserves correctness. The for-loop ensures that transitive dependencies of non-reference versions are captured. To crystalize this in a small example:

$\{2,4\}$ -Read $\rightarrow\{1,2,3\}$ \leftarrow Read- $\{1,2\}$

Using a min-selector, we would have a reference version of 1. `working` would capture $\{1,2\} \cap \{1,2,3\} = \{1,2\}$ before entering the for-loop. If we did not have the for-loop intersections, the `skernel()` would later only read the dependency on the right (associated with version set $\{1,2\}$). However, the kernel object is also associated with version 2. Version 2 depends on the left-versioned object that is associated with $\{2,4\}$. Therefore, to preserve correctness, one must include transitively linked versions as well. The for-loop grabs all non-reference versions from `working` and then tries to find these transitive dependencies. The conditional branch at the end simply reports back to the caller whether a split needs to occur. If there are elements in the kernel object that are not present in the relevant read sources, a split occurs. Otherwise, the version sets are synchronized which is indicated by a flag in the returned object.

The `skernel()` method behaves exactly like the traditional `kernel` with the exception that we have to filter for the version set of the kernel object among the read sources, namely for `vo.get_versions() \ unaffected_versions` (where `\` is the set minus operator) and separate out the split version set from the kernel object. Read sources that are associated with version sets that represent a strict superset of the kernel object version set are treated exactly like those that are associated with `vo.get_versions() \ unaffected_versions`. The additional versions in the superset simply refer to state being shared with other versions. Since we do not write to the read objects, the presence of the additional versions is irrelevant.

For event-scheduled simulation executives, granular cloning is compatible with both optimistic and conservative synchronization protocols. For the latter, each LP will proceed to execute local events up to the synchronization point without violating local causality. The only difference compared to the traditional approach is that messages and objects are associated with versions and, whenever events are processed, the comparison of “version set”-tags occurs (via the

granular cloning algorithms described). For situations in which the lookahead value is state-dependent, local causality is guaranteed if the lookahead values are derived across all versions of a given LP (there may exist further optimizations whereby correctness can be ensured if this is relaxed). For optimistic synchronization protocols, the same granular cloning algorithms are used during normal execution. However, during rollbacks one needs to ensure that anti-messages annihilate messages with the same version set and that state saving captures the version sets associated with simulation objects (message acknowledgements are also version set specific). A straggler message may trigger a “global” rollback (i.e., for all versions) even if it only affects a few versions. Although forward progress is guaranteed with a global rollback (as in traditional Time Warp), it may cause rollback thrashing. To counteract this, it may be possible to modify the granular cloning rollback mechanism to preemptively query a rollback’s version dependencies across LPs and only affect an isolated subset of the versions, while the execution of unaffected versions can proceed in parallel (since the rollback does not apply to them). These extensions are beyond the scope of this thesis.

5.4.3 Optimizations and Heuristics

As previously mentioned, it is possible to optimize the general approach for `handle_conflicts()` for a specific kernel. For example, sometimes a particular state value of the kernel object determines that few other objects need to be read than if the state of the current kernel object is assumed to be unknown. However, it should be emphasized that this is not necessary for granular cloning - unlike some other methods for computation sharing. Algorithm 6 shows a simplified version of an optimized `handle_conflicts()`. It exploits the application-specific fact that the kernel reads from at most one object and that we can “look ahead” which object will be read based on the current state of the kernel object.

```

VersionedObj handle_conflicts()
  if this.state == STATE0
    VersionedCell[] neighbor =
      OLD[get_index()+NBR[0]]
    for VersionedObj lvn in neighbor
      /*assume neighbor is sorted by min
      Version*/
      if versions.has_a_match(
        lvn.get_versions()) and
        !versions.is_contained_in(
          lvn.versions)
        return VersionedCell(
          this.state,
          this.versions.diff(
            this.versions.intersection(
              lvn.versions
            )
          )
        )
      )
    else if this.state == STATE1 /* remaining cases */
      return VersionedCell() //no conflict

```

Algorithm 6: An application-specific variant of the `handle_conflicts()` method

(applied on the kernel object) that exploits the property that there is at most one object dependency by peeking at the kernel object's state and then filtering the read sources.

Algorithm 6 first decides where in the old state of the model environment, OLD, it needs to look for the relevant object. As mentioned, the correct location depends on the state of the kernel object. In order to avoid displaying redundant code, only one of the conditional branches is shown completely (the one for STATE0). The algorithm then iterates over the versioned instances of the object and tries to discern whether the relevant instance is in (version-)conflict with the kernel object's version set. If so, it returns the split object. If not, the end of the function will return an empty object. The caller will then be able to discern whether the returned object is split or not.

This is just a small sample of possible optimizations. There are many properties that can be exploited, ranging from (i) dense and globally-sensitive version sets to (ii) sparse, localized, and clustered version sets. In applications that fall under the former category (i), it may be the

case that explicitly enumerating sample paths corresponding to single versions after the branching point outperforms the general `handle_conflicts()` scheme; in this case, the computations are only shared up to the branching point. For applications in the latter category (ii), where one wishes to simulate many versions, it may prove fruitful to abandon the unsigned integer representation and instead adopt a compressed representation of version sets. Indices could be used to map to particular version sets in aggregate data structures.

5.5 Expected Performance for Agent-Based Granular Cloning

Intuitively, one expects better performance (more shared computation) if the state variation is introduced late and if there is a slow spread of version-differentiated sample path relative to the entire state space. Better performance is also expected if the overhead from granular cloning is much smaller than the transition function.

Consider an agent based simulation (ABS) that simulates E agents over $T (>0)$ timesteps (t is a particular timestep) over R runs (r is a particular run). At a given run, timestep, and for a given agent, a kernel (or, transition function) with work W_{kernel} is executed. Although the W_{kernel} for granular cloning is slightly more complex than the one for traditional execution, the same factor will be used for both: The only additional granular cloning overhead is a potential split of the kernel object (which is a constant-time operation). Both the traditional and the granular cloning kernels access interacting objects in constant time. The key for the former is the object id, while for the latter it consists of both the object id and relevant version set. After $p\%$ of these timesteps, decision points are injected whereby the state $S(t)$ differs from the baseline sample path (which is identified with version id, $r = I$). At each timestep after $\lfloor p * T \rfloor$, each agent reads the state of $E_{\text{read}} (\leq E)$ agents. $I \leq q(t) \leq R$ is a function that – for granular cloning - returns the average number of version sets at timestep t across all agents E . $\kappa_{\text{conflicts}} (> 1)$ scales W_{kernel} by the additional overhead from `handle_conflicts()`.

The speedup of granular cloning vs. traditional replicated execution is approximated by:

$$\frac{E \times W_{kernel} \times T \times R}{E \times \kappa_{conflicts} \times W_{kernel} \times \sum_{t=0}^T \varrho(t)} = \frac{T \times R}{\kappa_{conflicts} \times \sum_{t=0}^T \varrho(t)}$$

Without a mechanism for re-merging split objects, $\varrho(t)$ is monotonically increasing.

Furthermore, $\varrho(t)$ cannot exceed R for any t because a version number is the most atomic element of an object and, by assumption, no more than R runs are simulated. Therefore, $T \times R \geq \sum_{t=0}^T \varrho(t)$. Figure 35 shows three example growth rates of $\varrho(t)$ with overlaid trendlines. The fastest-rising curve $\varrho'(t)$ would not be expected to yield a significant speedup and, depending upon the implementation of `handle_conflicts()`, the scaling with $\kappa_{conflicts}$ may even dominate the payoff expression above the curve $T \times R - \sum_{t=0}^T \varrho(t)$. The growth rates primarily depend on the percentage of the total state space that is being read in an object's kernel at a point in time. A constant number (without any noticeable bias for detecting certain types of state variables) would likely yield the linear $\varrho''(t)$ divergence pattern in Figure 35.

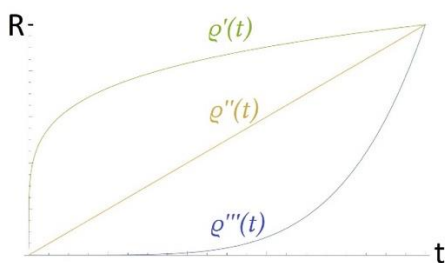


Figure 35: Lower growth rates yield better performance for granular cloning.

5.6 Empirical Evaluation

The performance of granular cloning is evaluated with two models, both of which are described in more detail later. In brief, they consist of:

(i) a cellular automaton- (CA-) based traffic simulation on a 1D grid, which is a boundary case of the Nagel-Schreckenberg traffic model (Nagel & Schreckenberg, 1992).

(ii) a distributed setting that simulates a basic integrated transportation and land use model (ITLUM), where one process executes (i) and another process executes a land use model from (Lu, Noonan, Crittenden, Jeong, & Wang, 2013). The execution of both processes occurs in a synchronous fashion per timestep.

The ITLUM benchmark is a useful test case because in many usage scenarios replicated runs will often only modify one model, e.g., the land use or the transportation model, but not both. As such, this is an interesting test case to explore the benefits of cloning in general, and granular cloning in particular. Further, ITLUM represents a class of simulations of practical interest to communities concerned with urban planning and the sustainable growth of cities.

For both models, the number of runs is limited to 32; the version tags assigned to objects are 32-bit unsigned integers. In settings with more runs, one could assign wider tags for even greater speedups. The machine that generated these results uses an Intel Core i7 6700™ (4 physical cores, 8 logical) and 32GB DDR4 memory. All programs were compiled from C++ and the distributed implementation uses MSMPI. Unit tests were successfully completed to ensure that states of the granular cloning match the explicit execution of the same runs.

The next three subsections discuss the conceptual models, and the two subsections thereafter report the performance results. Opportunities for attractive applications as well as limitations in adversarial problems due to tradeoffs are highlighted.

5.6.1 Conceptual Model: CA Traffic Simulation

A 1D CA models one road in a neighborhood. Each cell of the array is either set or cleared, the former representing a cell by a vehicle and the latter the absence of a vehicle. At

discrete timesteps every vehicle will move forward (towards the right) by one cell if the cell ahead is clear. Vehicles “wrap around” when they reach the rightmost cell. The initial array state is populated randomly. This ruleset is referred to as Wolfram Rule 184 which is also used to model several other systems. However, there is a slight modification relative to this behavior that is described by Wolfram Rule 184. A vehicle will sample random cells in front of it and not move forward if the perceived traffic intensity is deemed too high, namely with probability $p = 1 - [0.9 + 0.1 * ESTIMATED_TRAFFIC_INTENSITY]$, where *ESTIMATED_TRAFFIC_INTENSITY* is determined by the ratio $[number\ of\ sampled\ cells\ that\ are\ occupied] / [number\ of\ sampled\ cells]$. In the experimental design, the number of sampled cells is varied with powers of two to illuminate its impact on the speedup. The length of the 1D array and the number of timesteps are also varied in the performance section.

5.6.2 Conceptual Model: Land Use Model

The land use model used is a translation from the Netlogo model of (Lu, Noonan, Crittenden, Jeong, & Wang, 2013) into C++. A detailed description of the model is beyond the scope of this chapter, but can be found in the reference guide of (Lu, Noonan, Crittenden, Jeong, & Wang, 2013). At a high level, the ABM maintains a 24x24 grid that represents 9 square miles of greenfield, which is developed with single-family dwellings over a 30-year period. It also contains on the order of 100 input, output and state variables. The agents in the model include homebuyers and developers, which interact with the local government (taxation and infrastructure improvements) and the grid. In each of the 30 time steps:

1. 1000 homebuyers bid on up to 10 properties,
2. property transactions are settled,
3. new properties are constructed, and

4. local taxes are collected to improve the infrastructure.

The model was developed to compare two stormwater development policies against each other to find out how many apartment homebuyers they would incentivize.

5.6.3 Conceptual Model: ITLUM

The two models from the previous subsections were coupled into an ITLUM. During the 4th phase of the land use model of (Lu, Noonan, Crittenden, Jeong, & Wang, 2013), namely the tax collection and infrastructure improvement phase, the transportation cost savings of a neighborhood are derived from the degradation of the road infrastructure and depreciation of public transit. However, these variables were fixed in the original model. Using an actual transportation simulation to determine these variables dynamically may improve the credibility and validity of the results. The traffic intensity of the neighborhood roads is understood to be proportional to the population density of the neighborhood. These communication ideas inspire the federation sequencing pipeline for the ITLUM shown in Figure 36.

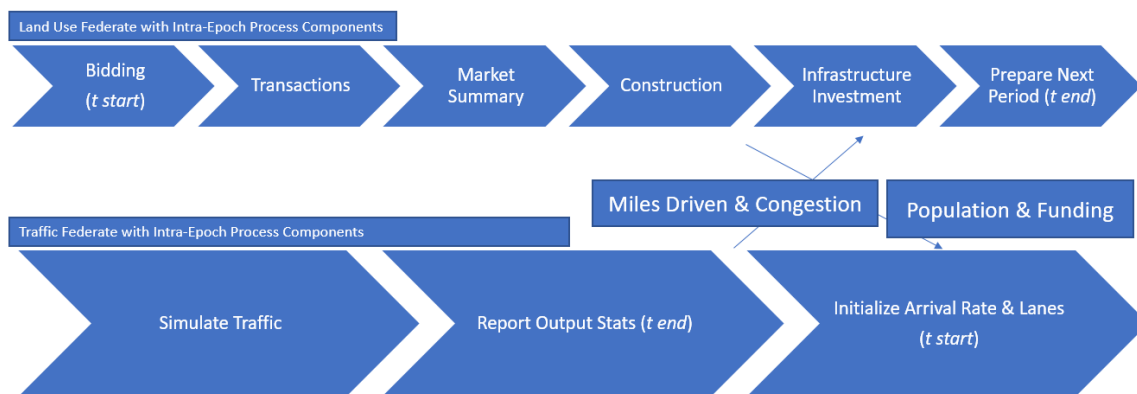


Figure 36: ITLUM Federation Sequencing Pipeline

As the ratio between (i) the execution time of one epoch of the land use node and (ii) the execution time of one epoch of the transportation node approaches 0, the speedup of the ABS dominates the overall speedup.

5.6.4 Evaluation: CA Traffic Simulation on a Single Process

Figure 37 shows the speedup of the granular cloning approach executing 32 versions in one single replication, compared to the traditional approach of executing a single version for each of 32 replications. Each of the 32 replications alters the state at a uniformly drawn timestep at a uniformly drawn cell. The speedups were determined using measured CPU times for a single process executing the transport model only.

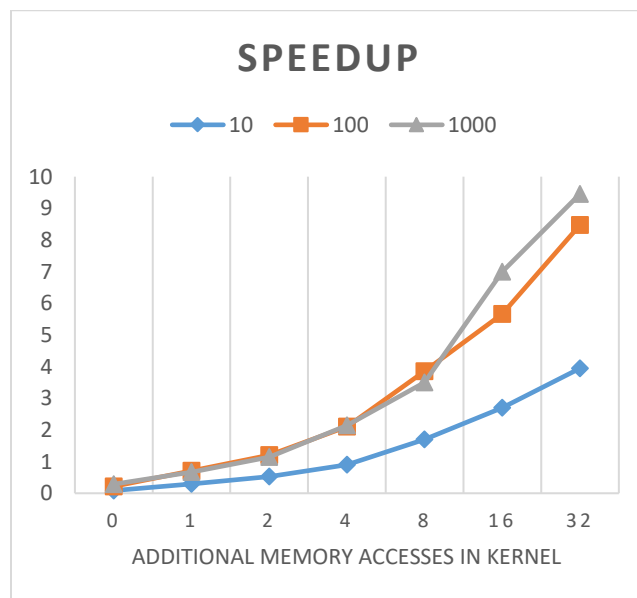


Figure 37: Speedup of the transport model of three different cell-to-timestep ratios

The three lines refer to three different scenarios differentiated by the space-to-time ratio (the cell-count divided by the time-step count). The speedup is less impressive if there are only a few timesteps relative to the numbers of cells because a given version has less time available to propagate its specific behavior into the global state. The speedup increases with the computational load in the transition function as the relative overhead of the granular cloning algorithm diminishes. The computational load is injected by enhancing the traffic intensity sampling resolution of the given driver (see the description of the transportation conceptual

model above). It should be noted that merging versions that share the same state together at certain intervals in the granular cloning mode did not improve the performance significantly.

Figure 38 shows, for the 10 space-to-time ratio, the speedup if one realizes the differences among the replications late. As mentioned above, the differences among the 32 replications are uniformly drawn in the 1D grid space and across all timesteps. However, if one restricts the realized timesteps to only be drawn in the last 20% or 50% of timesteps, the version-specific local state perturbations have fewer opportunities to propagate through the global state space. This directly translates into computational savings in memory and execution time. As can be seen in the figure, the jump from the 50% scenario to the 20% scenario is less dramatic. Early branches pose a significant computational burden on granular cloning. In an adversarial scenario, the speedup is less than 1 because the physical sample path is equivalent to that in the traditional approach and one cannot reap the benefits of shared computations up to the decision point. The shared computations after the decision point are out-shadowed by the granular cloning overhead. With regard to the performance model from Section 5.5, $\varrho(t)$ for this particular model would exhibit a linear growth trend based on the kernel which attempts to detect collisions.

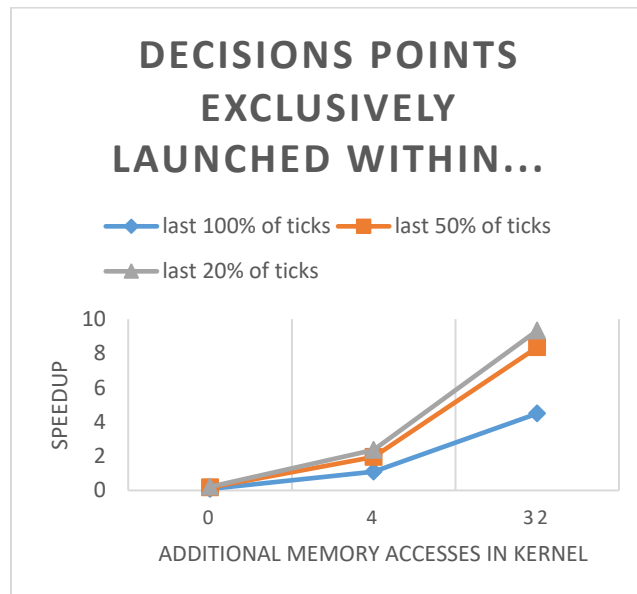


Figure 38: Speedup of the transport model if uniform scenario alterations are drawn in restricted time intervals

5.6.5 Evaluation: ITLUM on Two Processes

The following tables show the speedup results of the combined ITLUM model described above for 30 simulated years each. The speedup quantities are not measured relative to sequential execution of the two processes separately; rather, they are relative to the parallel execution of the two processes where only the transport model uses the traditional approach of explicitly enumerating all the replications: $S = T_{\text{parallel,traditional}} / T_{\text{parallel,granular}}$. A column headed by the expression $S_{\text{TI=IC}}$ denotes speedup figures that were obtained from models where IC instructions in the transition function for each object's update in the next timestep were computed. The data exchanged between the two simulators in a synchronized/blocking fashion. This is also why the speedup quantities in the following tables are derived from wall-clock intervals. If they were based on CPU time, they could be misleading as a result of the platform context-switching from a blocking process. However, the drawback is that platform-specific factors may pollute the times. Nevertheless, a second set of replications confirmed that the numbers were close to those in the first set.

Table 1: Speedup for cellcount/timesteps = 100 and start of decision points commences at timestep=0

Cell count	$S_{\text{TI=4}}$	$S_{\text{TI=32}}$
1,000	0.93	1.03
10,000	1.04	3.39

Table 2: Speedup for cellcount/timesteps = 100 and start of decision points commences halfway through

Cell count	$S_{\text{TI=4}}$	$S_{\text{TI=32}}$
1,000	1.01	1.07
10,000	1.13	3.29

Table 3: Speedup for cellcount/timesteps = 10 and start of decision points commences at timestep=0

Cell count	$S_{TI=4}$	$S_{TI=32}$
1,000	0.99	1.35
10,000	0.92	3.07

Table 4: Speedup for cellcount/timesteps = 10 and start of decision points commences halfway through

Cell count	$S_{TI=4}$	$S_{TI=32}$
1,000	1.00	1.66
10,000	1.11	4.11*

In the entry marked with * was also evaluated with a transition function count of 64 and 128, which resulted in speedups of 4.89 and 5.96, respectively. Since the only speedups that can be reaped in take place in the transport simulator, it determines the overall speedup provided it takes longer per epoch than the land use simulator. Because of reasons discussed before, the instruction count of the driver’s transition function has a significant impact on the overall speedup. In general, especially high performance is expected when there is a large number of runs, the variation is introduced late, the behavior (transition function or event handler) is complex in relation to the overhead, and/or when individual objects do not have the opportunity to quickly affect the entire state space in a unique fashion.

5.6.6 Evaluation: 2x4 XOR-HOLD (Event-Scheduling on Eight Processes)

In addition to these agent-based and timestepped models, granular cloning was also compared against traditional simulation cloning in a distributed event-scheduling setting (with 8 processes). The benchmark used here is 2x4 XOR-HOLD (2 LPs, 4 versions each) using a

conservative barrier synchronization based approach. In addition to simply processing and scheduling events (HOLD), LPs in XOR-HOLD also own state variables (represented as an integer array) and modify them in their (sole) event handler. More specifically, they access their array at an index randomly sampled by the sender and apply logical exclusive OR of the sender's corresponding value to their own value at that index. Versions are realized as random variations in individual state variables (integers) at the start of the simulation. When the array sizes are set to 100,000 and the number of total events processed per LP set to 300,000, the obtained average speedup is 1.68 (standard error = 0.08). The speedup was again based on wallclock time until the simulation terminates using each method.

5.7 Conclusion and Future Work

The granular cloning framework offers performance improvements without loss in accuracy for a broad range of simulation applications where one wishes to execute several runs that share similar sample paths. Instead of being restricted to the scale of extended partitions of the state space, the explicit state representation occurs at the exact minimal scale where state differs. Our experimental results show order-of-magnitude speedup in some cases and illustrate that performance is increased when:

- the number of replications is large,
- the transition functions (in timestepped models) or the event handlers (in event-scheduling models) are relatively complex, and
- the state among runs is similar, especially at the beginning of the run, so that the framework may reuse more shared computations, rather than duplicate redundant work. In many applications, the replication-specific variation tends to explode throughout the state space as simulated time increases.

There are interesting further research questions related to granular cloning. For example, how should one batch together runs of various scenarios to apply simulation-based optimization methods (scenarios vs. random input)? Another research topic lies in approximate computing: One could reduce the overhead of granular cloning interactions selectively, which relaxes correctness; in this fashion, bias is traded off for variance. It is also an open question of whether there are more efficient algorithms to keep track of the state-version associations, especially for common applications. Another open research area is how different simulation acceleration techniques contribute to speedup when they are used in groups across a broad range of benchmarks.

We plan to release a general-purpose library that allows users to encapsulate their simulation objects into granular cloning objects as well as the interactions among these objects into granular cloning interactions. As in the simulation cloning library of (Hybinette & Fujimoto, 2001), the granular cloning runtime will manage the bookkeeping of the varying state of a given object to the replication number (version) internally.

6 CONCLUSIONS AND FUTURE WORK

This thesis discussed how DDDAS can be used to manage transportation systems by integrating trajectory prediction with accelerated simulation models (including land use and transportation models). With these tools, it is possible to accurately project future traveler states both spatially and with temporal estimates. Acquiring these results faster-than-real-time has the potential to benefit both regulatory stakeholders and the travelers themselves.

In the application direction, it is possible to expand into further areas of relevant research and integrate heterogeneous modules together (by using different assumptions and focusing on different urban subsystems). Application-focused research could investigate best practices and key challenges in integrating various sub-models. For example, beyond land use and transportation models, one could incorporate utility supply systems and air pollution models.

In terms of methodology, several ways to accelerate simulation models have been presented, along with possible heuristic enhancements. Beyond the aspect of computational acceleration, improved tractability and transparency is another benefit of these methods. For certain models, it can be highly advantageous to explore a breadth of scenarios before executing a small number of runs in depth. Defining suitable real-time application programming interfaces (API) into these acceleration schemes may have implications for a whole host of other areas in the simulation model construction life cycle. Simulation output analysis routines, ranking & selection algorithms, and simulation-based optimization can access more scenarios concurrently than by sequentially enumerating individual runs, while simultaneously preserving common sample path data that would have been unsaved. Moreover, validation methods can be enhanced as state-access to a large set of runs can expose germane factors in the physical system.

Simulation verification is more credible if more test cases can be pushed through over a fixed time period.

For some more concrete cases of possible future research that could be explored in the context of this thesis material:

- The trajectory prediction models could be enhanced further by designing metamodels that use different “submodels” that have displayed superior performance for particular types of queries. Based on the data availability, it could also be possible to include more factors - such as time, weather, and special events – in the prediction models. Investigating a suitable interface between trajectory prediction analytics and path planning algorithms may also prove fruitful.
- For both the specific “virtual entity” and the more general “granular cloning” execution schemes, one can incorporate relaxations at various points of the respective algorithms which would sacrifice correctness in order to provide further speedups. These relaxations could occur selectively based on certain computationally inexpensive heuristics, so that a small amount of bias may be traded in for tighter confidence intervals in the output statistics. In the other direction, it is also possible to include heuristics for both algorithms which maintain correctness, but trigger preemptive cloning in situations where the density of entities and their interactions are high. This would allow the simulation execution to sidestep possible overheads in simulation time frames where the respective acceleration scheme would yield little benefit.
- The “virtual entity” scheme may also incorporate heuristics to flush tags that are guaranteed or unlikely to be used later. Future work in “virtual entities” can also include the generalization of the scheme to multiple target vehicles simultaneously.
- A significant amount of research could be conducted on coupling granular cloning with output analysis algorithms. Certain subsets of runs upon which key performance indicators depend upon can be investigated.

APPENDIX A. PERMISSION REQUESTS TO REUSE MATERIALS

This thesis includes own materials from (1) Springer (ICCS), (2) ACM (SIGSIM-PADS), (3) IEEE (WSC). Regarding (1), the paper cites the relevant papers and the following response was received via email:

Dear Philip,

Here is the official statement from Springer:

"The author may include his LNCS paper in his PhD thesis provided acknowledgement is given to the original source of publication."

We hope this is of help.


With best regards,


ICCS Secretariat

Regarding (2), the rights management form for the conference grants the relevant permissions.

Regarding (3), an order was placed through the Copyright Clearance Center's RightsLink® service:

Order Summary

Licensee:	Philip Pecher
Order Date:	Apr 3, 2018
Order Number:	
Publication:	IEEE Proceedings
Title:	Past and Future Trees: Structures for predicting vehicle trajectories in real-time
Type of Use:	Journal/book cover
Order Total:	0.00 USD

Regarding (3), the following email was also received (and the requests incorporated):

“The IEEE does not require individuals working on a thesis to obtain a formal reuse license however, you must follow the requirements listed below:

Textual Material

Using short quotes or referring to the work within these papers) users must give full credit to the original source (author, paper, publication) followed by the IEEE copyright line © [Year of publication] IEEE.

In the case of illustrations or tabular material, we require that the copyright line © [Year of original publication] IEEE appear prominently with each reprinted figure and/or table.

If a substantial portion of the original paper is to be used, and if you are not the senior author, also obtain the senior author’s approval.

Full-Text Article

If you are using the entire IEEE copyright owned article, the following IEEE copyright/ credit notice should be placed prominently in the references: © [year of original publication] IEEE. Reprinted, with permission, from [author names, paper title, IEEE publication title, and month/year of publication]

Only the **accepted** version of an IEEE copyrighted paper can be used when posting the paper or your thesis on-line. You may not use the **final published** version

In placing the thesis on the author's university website, please display the following message in a prominent place on the website: In reference to IEEE copyrighted material which is used with permission in this thesis, the IEEE does not endorse any of [university/educational entity's name goes here]'s products or services. Internal or personal use of this material is permitted. If interested in reprinting/republishing IEEE copyrighted material for advertising or promotional purposes or for creating new collective works for resale or redistribution, please go to

http://www.ieee.org/publications_standards/publications/rights/rights_link.html

to learn how to obtain a License from RightsLink.

If applicable, University Microfilms and/or ProQuest Library, or the Archives of Canada may supply single copies of the dissertation.

Kind regards,

[REDACTED]

REFERENCES

- Amini, S. B. (2012). Trajectory-aware mobile search. *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '12)*.
- Ashbrook, D. a. (2003). Using GPS To Learn Significant Locations and Predict Movement Across Multiple Users. *Personal and Ubiquitous Computing, 2003*.
- Ayani, R. (1988). *A Parallel Simulation Scheme Based on Distances Between Objects*. Royal Institute of Technology, Department of Telecommunication Systems-Computer Systems.
- Babak, P. A. (2009). The effect of wind on the propagation of an idealized forest fire. *SIAM Journal on Applied Mathematics*.
- Bahrani, S. R. (2013). Real-time simulations to support operational decision making in healthcare. *Proceedings of the 2013 Summer Computer Simulation Conference*.
- Barjis, J. (2011). Healthcare simulation and its potential areas and future trends. *SCS M&S Magazine*.
- Brotzge, J. K. (2006). Collaborative adaptive sensing of the atmosphere: New radar system for improving analysis and forecasting of surface weather conditions. *Transportation Research Record: Journal of the Transportation Research Board*.
- Cai, W. F. (1999). An auto-adaptive dead reckoning algorithm for distributed interactive simulation. *Proceedings of the thirteenth workshop on Parallel and distributed simulation*.

- Cai, W., Lee, F. B., & Chen, L. (1999). An auto-adaptive dead reckoning algorithm for distributed interactive simulation. *Parallel and Distributed Simulation, Proceedings. Thirteenth Workshop on. IEEE.*
- Celik, N. e. (2010). DDDAS-based multi-fidelity simulation framework for supply chain systems. *IIE Transactions.*
- Chandy, K., & Misra, J. (1979). Distributed simulation: A case study in design and verification of distributed programs. *IEEE Transactions on Software Engineering.*
- Chaturvedi, A. A. (2006). DDDAS for Fire and Agent Evacuation Modeling of the Rhode Island Nightclub Fire. *Computational Science – ICCS 2006.*
- Chen, D., Turner, S. J., Cai, W., Gan, B. P., & Low, M. Y. (2005). Algorithms for HLA-based distributed simulation cloning. *ACM Transactions on Modeling and Computer Simulation (TOMACS)*, 316-345.
- Chen, D., Turner, S. J., Cai, W., Theodoropoulos, G. K., Xiong, M., & Lees, M. (2010). Synchronization in federation community networks. *Journal of parallel and distributed Computing*, 144-159.
- Cheng, C. R. (2003). Location Prediction Algorithms for Mobile Wireless Systems. *Wireless Internet Handbook: Technologies, Standards, and Applications.*
- Community, N. (2014, 05 13). *NGSIM Community Home*. Retrieved from <http://ngsim-community.org>
- Conway, J. (1970). The game of life. *Scientific American* 223, no. 4.
- Daganzo, C. F. (2006). In traffic flow, cellular automata = kinematic waves. *Transportation Research Part B: Methodological.*

- Darema, F. (2004). Dynamic data driven applications systems: A new paradigm for application simulations and measurements. *Computational Science-ICCS 2004*.
- de Brébisson, A. e. (2015). Artificial neural networks applied to taxi destination prediction. *arXiv preprint*.
- Deguchi, Y. e. (2004). Hev Charge/Discharge Control System Based on Navigation Information. *SAE Convergence International Congress & Exposition on Transportation Electronics*.
- Eeckhout, J. (2004). Gibrat's Law for (All) Cities. *The American Economic Review, Vol. 94, No. 5*.
- EU Council. (2010). *DIRECTIVE 2010/40/EU OF THE EUROPEAN PARLIAMENT AND OF THE COUNCIL*. Official Journal of the European Union.
- Ferenci, S., Fujimoto, R., Ammar, M., Perumalla, K., & Riley, G. (2002). Updateable simulation of communication networks. *Proceedings of the sixteenth workshop on Parallel and distributed simulation*.
- Fujimoto, R. A. (2014). A Dynamic Data Driven Application System for Vehicle Tracking. *International Conference on Computational Science, Dynamic Data Driven Application Systems Workshop, June 2014*.
- Fujimoto, R. H.-K. (2007). Ad Hoc Distributed Simulations. *In Principles of Advanced and Distributed Simulation*.
- Fujimoto, R. M. (1989). The virtual time machine. *Proceedings of the first annual ACM symposium on Parallel algorithms and architectures*. . ACM.
- Fujimoto, R. M., Guin, A., Hunter, M., Park, H., Kannan, R., Kanitkar, G., . . . Pecher, P. (2014). A Dynamic Data Driven Application System for Vehicle Tracking. *2014 International Conference on Computational Science*.

- Fujimoto, R. M.-K. (2007). Ad hoc distributed simulation of surface transportation systems. *Computational Science–ICCS 2007*.
- Fujimoto, R. M.-K. (2007). Ad Hoc Distributed Simulations. *Principles of Advanced and Distributed Simulation*.
- Fujimoto, R. R.-K. (2006). Dynamic data driven application simulation of surface transportation systems. *Computational Science–ICCS 2006*.
- Fujimoto, R., Bock, C., Chen, W., Page, E., & Panchal, J. (2016). *Research Challenges in Modeling and Simulation for Engineering Complex Systems*. NSF Report.
- Gaba, D. (2004). The future vision of simulation in health care. *Quality and Safety in Health Care* 13.
- Gallistel, C. R. (2007). Dead reckoning, cognitive maps, animal navigation and the representation of space: An introduction. *Robotics and cognitive approaches to spatial mapping*.
- Gardner, M. (1970). Mathematical Games – The fantastic combinations of John Conway's new solitaire game "life". *Scientific American* 223.
- Gerlough, D. (1955). Simulation of freeway traffic on a general-purpose discrete variable computer. *University of California, Los Angeles, PhD Dissertation*.
- Gong, C. a. (2004). A methodology for automated trajectory prediction analysis. *AIAA Guidance, Navigation, and Control Conference and Exhibit*.
- Gui, F. A. (2009). A contextualized and personalized approach for mobile search. *Advanced Information Networking and Applications Workshops, 2009*.
- Henriksen, J. (1995). An introduction to SLX [simulation software]. *Simulation Conference Proceedings, 1995. Winter*. IEEE.

- Holmes, V. (1978). *Parallel algorithms for multiple processor architectures*. Ph.D. dissertation, Comp. Science Dept.. Univ. Texas at Austin.
- Hu, P. a. (2001). Summary of travel trends: 2001 national household transportation survey. *Oak Ridge National Laboratory Technical Report ORNL/TM*.
- Hunter, M., Biswas, A., Chilukuri, B., Guin, A., Fujimoto, R., Guensler, R., . . . Rodgers, M. (2016). *Energy-Aware Dynamic Data-Driven Distributed Traffic Simulation for Energy and Emissions Reduction*. DDDAS.
- Hybinette, M. a. (2001). Cloning parallel simulations. *ACM Transactions on Modeling and Computer Simulation (TOMACS) 11.4 (2001)*.
- Hybinette, M., & Fujimoto, R. M. (2001). Cloning parallel simulations. *ACM Transactions on Modeling and Computer Simulation*.
- Johannesson, L. A. (2007). Assessing the potential of predictive control for hybrid vehicle powertrains using stochastic dynamic programming. *IEEE Transactions on Intelligent Transportation Systems*.
- Kiesling, T. a. (2005). Towards Time-Parallel Road Traffic Simulation. *Proceedings of the 19th Workshop on Principles of Advanced and Distributed Simulation (PADS '05)*.
- Krumm, J. (2006). Real time destination prediction based on efficient routes. *Society of Automotive Engineers (SAE) 2006 World Congress*.
- Krumm, J. (2006). Real Time Destination Prediction Based On Efficient Routes. *SAE Technical Paper*.
- Krumm, J. (2009). Where will they turn: Predicting turn proportions at intersections. *Personal and Ubiquitous Computing, 2009*.

- Kunkel, K. E., Changnon, S. A., Lonquist, C. G., & Angel, J. R. (1990). A real-time climate information system for the midwestern United States. *Bulletin of the American Meteorological Society*, 1601-1609.
- Laasonen, K. (2005). Route Prediction from Cellular Data. *Workshop on Context-Awareness for Proactive Systems, 2005*.
- Lentz, K. P., Manolakos, E. S., Czeck, E., & Heller, J. (1997). Multiple experiment environments for testing. *Journal of Electronic Testing 11*, no. 3, 247-262.
- Li, X., Cai, W., & Turner, S. J. (2017). Cloning Agent-Based Simulation. *ACM Transactions on Modeling and Computer Simulation (TOMACS)*.
- Lin, K., & et al. (2016). Enhanced fingerprinting and trajectory prediction for IoT localization in smart buildings. *IEEE Transactions on Automation Science and Engineering 13.3*.
- Lu, Z., Noonan, D., Crittenden, J., Jeong, H., & Wang, D. (2013). Use of impact fees to incentivize low-impact development and promote compact growth. *Environmental science & technology*.
- Madey, G. R. (2007). Enhanced situational awareness: Application of DDDAS concepts to emergency and disaster management. *Computational Science-ICCS 2007*.
- Madey, G. R.-L. (2006). WIPER: The Integrated Wireless Phone Based Emergency Response System. *Proceedings of the 2006 International Conference on Computational Science*.
- Madey, G. R.-L. (2007). Enhanced Situational Awareness: Application of DDDAS Concepts to Emergency and Disaster Management. *Proceedings of the 2007 International Conference on Computational Science*.
- Mandel, J. e. (2008). A wildland fire model with data assimilation. *Mathematics and Computers in Simulation*.

- Mandel, J. J. (2012). Assimilation of perimeter data and coupling with fuel moisture in a wildland fire-atmosphere DDDAS. *Procedia Computer Science* 9.
- Marin, M. V.-C. (2013). Approximate Parallel Simulation of Web Search Engines. *PADS*.
- Marziale, N., Nobilia, F., Pellegrini, A., & Quaglia, F. (2016). Granular time warp objects. *Proceedings of the 2016 ACM SIGSIM Conference on Principles of Advanced and Discrete Simulation*.
- Mathew, T. (n.d.). *Department of Civil Engineering, Indian Institute of Technology, Bombay*. Retrieved 04 30, 2016, from Microscopic Traffic Simulation: https://www.civil.iitb.ac.in/tvm/1111_nptel/535_TrSim/plain/plain.html
- McGovern, A. D. (2011). Identifying predictive multi-dimensional time series motifs: an application to severe weather prediction. *Data Mining and Knowledge Discovery* 22.
- Miklušćák, T. G. (2012). Using Neural Networks for Route and Destination Prediction in Intelligent Transport Systems. *In Telematics in the Transport Environment*.
- Miklušćák, T., Gregor, M., & Janota, A. (2012). Using Neural Networks for Route and Destination Prediction in Intelligent Transport Systems. *Telematics in the Transport Environment*.
- Nagel, K., & Schreckenberg, M. (1992). A cellular automaton model for freeway traffic. *Journal de Physique*.
- Nagel, K., & Schreckenberg, M. (1992). A cellular automaton model for freeway traffic. *Journal de physique*.
- Neumann, J. v. (1956). *Probabilistic logics and the synthesis of reliable organism from unreliable components*. Princeton University Press.

- NGSIM Community Home. (n.d.). Retrieved from <http://ngsim-community.org>.
- Parashar, M. V. (2006). Towards dynamic data-driven management of the ruby gulch waste repository. *Computational Science–ICCS 2006*.
- Patrikalakis, N. M. (2004). Towards a dynamic data driven system for rapid adaptive interdisciplinary ocean forecasting. *Dynamic Data-Driven Application Systems*.
- Patterson, D. e. (2004). Opportunity Knocks: A System to Provide Cognitive Assistance with Transportation Services. *UbiComp 2004: Ubiquitous Computing*.
- Peacock, J., Wang, J., & Manning, E. (1978). Distributed simulation using a network of processors. *Proceedings of the 3rd Berkeley Workshop on Distributed Data Management and Computer Networks*.
- Pecher, P. H. (2014). Past and future trees: structures for predicting vehicle trajectories in real-time. *Winter Simulation Conference 2014*.
- Pecher, P. K., Hunter, M., & Fujimoto, R. M. (2014). Past and Future Trees: Structures for Predicting Vehicle Trajectories in Real-Time. *2014 Winter Simulation Conference*.
- Pecher, P., Crittenden, J., Lu, Z., & Fujimoto, R. (2018). Granular Cloning: Intra-Object Parallelism in Ensemble Studies. *Proceedings of the 2018 ACM SIGSIM Conference on Principles of Advanced Discrete Simulation*. ACM.
- Pecher, P., Hunter, M., & Fujimoto, R. (2016). Data-Driven Vehicle Trajectory Prediction. *SIGSIM-PADS 2016*, (pp. 13-22).
- Peng, L. D. (2014). Dynamic data driven application system for plume estimation using UAVs. *Journal of Intelligent & Robotic Systems 74*.

- Persad-Maharaj, N. e. (2008). Real-Time Travel Path Prediction Using GPS-Enabled Mobile Phones. *Proc. 15th World Congress on Intelligent Transportation Systems, ITS America, 2008*.
- Philip Pecher, M. H. (2015). Efficient Execution of Replicated Transportation Simulations with Uncertain Vehicle Trajectories. *Procedia Computer Science 51*, 2638-2647.
- Pournajaf, L., Xiong, L., & Sunderam, V. (2014). Dynamic Data Driven Crowd Sensing Task Assignment. *ICCS 2014. 14th International Conference on Computational Science*.
- Quaglia, F., & Baldoni, R. (1999). Exploiting intra-object dependencies in parallel simulation. *Information Processing Letters*, 119-125.
- Reynolds, R. W., & Marsico, D. C. (1993). An improved real-time global sea surface temperature analysis. *Journal of climate*, 114-119.
- Roughgarden, T. a. (2002). How bad is selfish routing? *Journal of the ACM (JACM) 49, no. 2 (2002)*.
- Schiff, J. L. (2008). *Cellular Automata: A Discrete View of the World*. Wiley-Interscience.
- Simmons, R. B. (2006). Learning to predict driver route and destination intent. *Proc. Intelligent Transportation Systems Conference*.
- Sinclair, A. J. (2008). Optimal and Feedback Path Planning for Cooperative Attack. *Journal of Guidance, Control, and Dynamics, Vol. 31, No. 6*.
- SLX FAQs. (2018, 01 04). Retrieved from <http://www.wolverinesoftware.com/SLXFAQs.htm>
- Song, L. a. (2012). Adaptive real-time tracking and simulation of heavy construction operations for look-ahead scheduling. *Automation in Construction*.

- Steenbruggen, J. E. (2015). Data from mobile phone operators: A tool for smarter cities? *Telecommunications Policy* 39.
- Stein, P. R. (1962). Non-linear transformation studies on electronic computers. *Los Alamos Scientific Lab.*
- Stoffers, M., Schemmel, D., Dustmann, O. S., & Wehrle, K. (2016). Automated Memoization for Parameter Studies Implemented in Impure Languages. *Proceedings of the 2016 annual ACM Conference on SIGSIM Principles of Advanced Discrete Simulation* (pp. 221-232). ACM.
- Suh, W., Hunter, M., & Fujimoto, R. (2014). Ad Hoc Distributed Simulation for Transportation System Monitoring and Near-Term Prediction. *Simulation Modeling Practice and Theory.*
- Technology, O. o. (n.d.). *United States Department of Transportation*. Retrieved 4 30, 2016, from ITS FAQ: <http://www.its.dot.gov/faqs.htm>
- Vahdatikhaki, F. a. (2014). Framework for near real-time simulation of earthmoving projects using location tracking technologies. *Automation in Construction.*
- Vakili, P. (1992). Massively parallel and distributed simulation of a class of discrete event systems: A different perspective. *ACM Transactions on Modeling and Computer Simulation (TOMACS)*, 214-238.
- Von Neumann, J. a. (1966). Theory of self-reproducing automata. *IEEE Transactions on Neural Networks* 5.
- Walsh, K., & Sirer, E. G. (2003). Simulation of large scale networks I: staged simulation for improving scale and performance of wireless network simulations. *Proceedings of the 35th conference on Winter simulation: driving innovation*, (pp. 667-675).

- Wolfram, S. (2002). *A New Kind of Science*. Wolfram Media.
- Xue, A. Y., Zhang, R., Zheng, Y., Xie, X., Huang, J., & Xu, Z. (2013). Destination prediction by sub-trajectory synthesis and privacy protection against such prediction. *Data Engineering (ICDE), 2013 IEEE 29th International Conference*.
- Xue, A. Z. (2013). Destination Prediction by Sub-Trajectory Synthesis and Privacy Protection Against Such Prediction. *IEEE International Conference on Data Engineering (2013)*.
- Xue, H. F. (2012). Data assimilation using sequential Monte Carlo methods in wildfire spread simulation. *ACM Transactions on Modeling and Computer Simulation (TOMACS)*.
- Ye, T. H. (2008). Large-scale Network Parameter Configuration using an On-line Simulation Framework. *IEEE/ACM Transactions on Networking*.
- Yuan, J. Z. (2010). T-drive: driving directions based on taxi trajectories. *Proceedings of the 18th SIGSPATIAL International Conference on Advances in Geographic Information Systems, GIS '10*.
- Yuan, J. Z. (2011). Driving with knowledge from the physical world. *The 17th ACM SIGKDD international conference on Knowledge Discovery and Data mining, KDD '11*.
- Ziebart, B. D. (2008). Navigate like a cabbie: Probabilistic reasoning from observed context-aware behavior. *Proceedings of the 10th international conference on Ubiquitous computing*.