SIMUtools 2023 : International Conference on Simulation Tools and Techniques

Comparing the Efficiency of Traffic Simulations using Cellular Automata

FERNANDO DIAZ-DEL-RIO, David Ragel-Diaz-Jara, M.J. Moron-Fernandez, D. Cagigas, D. Cascado-Caballero, Jose-L. Guisado-Lizar, and G. Jimenez-Moreno

fdiaz@us.es (corresponding author)

Dept. of Computer Architecture and Technology, Universidad de Sevilla Research Institute of Computer Engineering (I3US), Universidad de Sevilla, Avenida Reina Mercedes s/n, 41012 Sevilla, Spain

MINISTERIO DE CIENCIA E INNOVACIÓN

Financiado por la Unión Europea NextGenerationEU

Plan de Recuperación, Transformación y Resiliencia

AGENCIA ESTATAL DE INVESTIGACIÓN

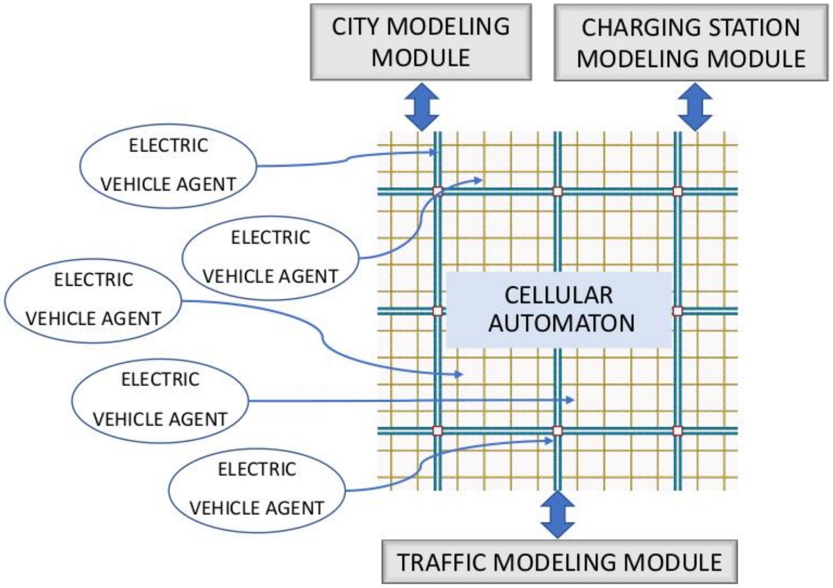
Grant TED2021-130825B-I00 funded by MCIN/AEI/10.13039/501100011033
and by the "European Union NextGenerationEU/PRTR"

SANEVEC Project

"A simulation approach to determine the deployment of an urban network of electric vehicle charging stations for environmental and social benefits"

https://grupo.us.es/sanevec/en/proyecto-sanevec-english/

- Objectives: SANEVEC aims to build a complete simulation tool. And to research, design and implement a computer simulation model to predict the effects of the layout of an urban network of electric vehicle charging stations on the following aspects : Traffic congestion, Air-quality, Carbon footprint, Electric grid usage.
- BUDGET: € 281,750. 12/01/2022 – > 11/30/2024
- University of Seville, Research Institute of Computer Engineering (I3US)



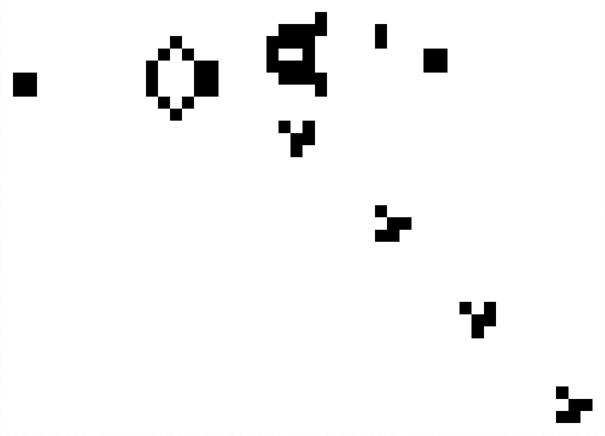
Our objectives

- Transportation sector is responsible for a substantial share of carbon dioxide emissions ➔ Shift toward electric vehicles.
- Electric vehicles requires the development of an extensive electric charging infrastructure.
- Simulating hundreds of city configurations for charging stations in short periods of time
  - to predict and prevent traffic congestion.
- Discrete vehicle movements: synchronous cellular automata (CA)
- Efficient Traffic Simulations: reducing execution time
  - ”make the common case fast”
  - reducing amount of memory
  - improving thread parallelism.

CONTENTS

Introduction/Related Works (I)

- Traffic simulation applications focus on microscopic models (simulating the movements of individual vehicles)
- Synchronous cellular automata (CA)
  - A cellular automaton consists of a regular grid of cells, each in one of a finite number of states
  - For each cell, a neighborhood is defined
  - Some fixed rules determines the new state of each cell in terms of its state and those of its neighborhood.



Conway's Game of Life

- A performance challenge that can be solved by
  - Using HPC (High Performance Computing) systems
  - ➔ Optimizing the part of the code where more than 90% of the execution time is spent.

Introduction/Related Works (II)

- Most well-known microscopic models is the Nagel-Schreckenberg (Na-Sch) model (easy to parallelize)

Nagel, K., Schreckenberg, M.: A cellular automaton model for freeway traffic. J. Phys. (1992).

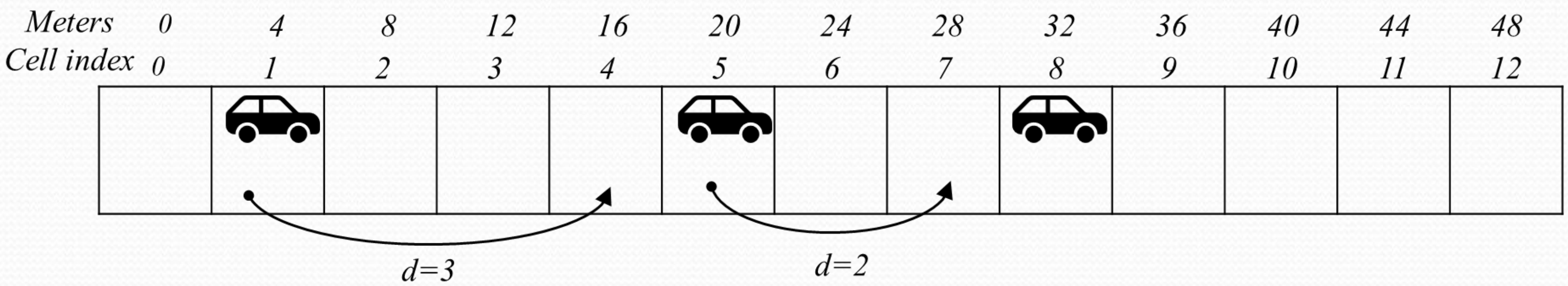
- Challenges: Cellular automaton models
  - Simulators in C, C++ (using OpenMP) or native protocols.
  - A few works using other languages such as Java, Erlang.
  - Traffic simulations may not be well suited to GPU or SIMD kernels, due to the inherent disperse memory accesses.
  - Good scalability wrt number of processors
    - Be careful with information exchanged after each time step.

CONTENTS

- Our objectives
- Introduction and Related Works
- **Representing vehicle movements**
  - **A novel model: reordering two iterations of the rules of the classical model**
- Efficient definition of data structures
- Results and Discussion
- Conclusion and future work

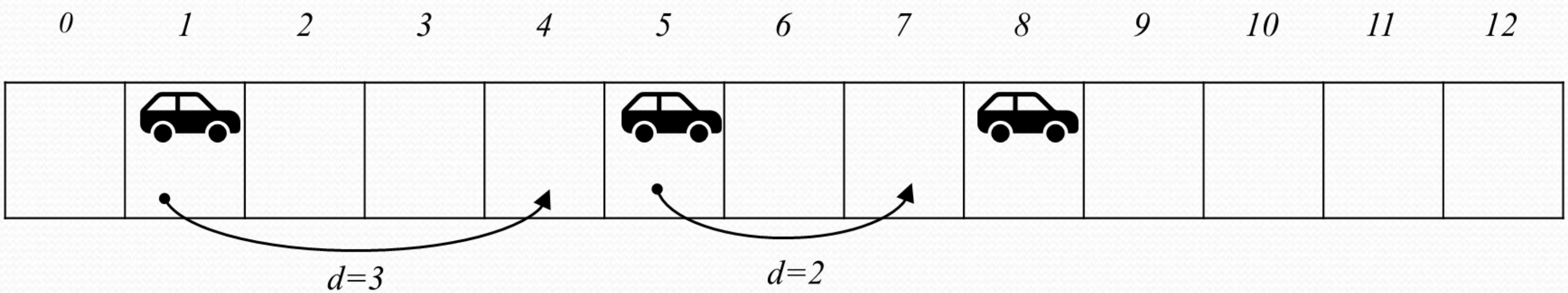
Representing vehicle movs. (I)

- Simplification: two-dimensional map is rendered and compacted it into a one-dimensional vector.
  - Each CA cell represents an area of a few meters
  - Special cells to represent crossroads and bifurcations
- **Most run-time consuming computing**: calculus of the ahead "free" distance (d_i)
  - Search for empty cells in front of each vehicle.



Representing vehicle movs. (II)

- Model Na-Sch discretizes space and time.
  - vehicles variables with two pairs of values for each cell *i*
  - current and next positions of vehicle *x(i, t)*, *x(i, t+1)*
  - current and next velocities *v(i, t)*, *v(i, t+1)*



x(i, t)	1				5			...				
x(i, t+1)				4			7	...				
v(i, t)	2				4			...				
v(i, t+1)				3			2	...				

Representing vehicle movs. (III)

- Na-Sch CA dynamic rules:

1. Acceleration: $v_i(t+1) = min(v_i(t)+1, v_{max})$;
2. Deceleration: $v_i(t+1) = min(d_i, v_i(t+1))$;
3. Randomization: $v_i(t+1) = max(v_i(t+1)-1, 0)$ (braking reduces velocity in one unit with probability P_b);
4. Movement: $x_i(t+1) = x_i(t) + v_i(t+1)$;

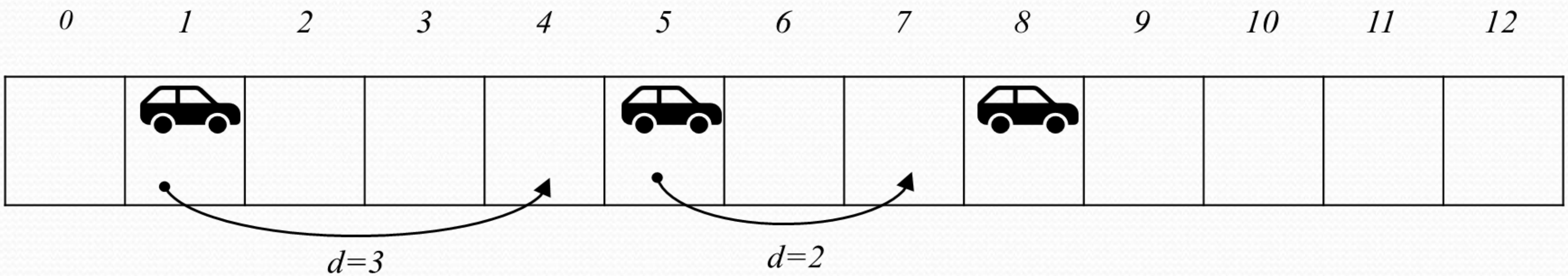
v_{max} : maximum speed that a vehicle can reach

d_i : number of empty cells in front of the vehicle.

Notation subscript i indicates the $i-$th vector element

A novel model (I)

- Condense 4 vectors into only 2

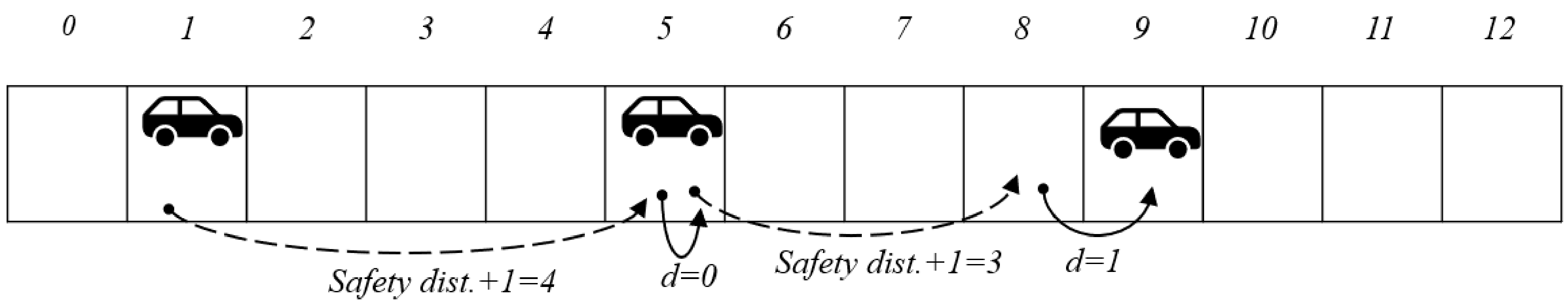


v(i, t)	2	-1	-1	-1	4	-1	-1	…				
v(i, t+1)	-1	-1	-1	3	-1	-1	2	…				

- Calling a random function is very time-consuming
  - High-quality uniform random numbers are not crucial here
    - coarse discretization, randomization of the third rule is artificial.
  - ➔ Previous generation of a random vector
    - Index to the random vector by a common number + cell index.

A novel model (II)

- Reduce the number of iterations to compute the ahead "free" distance (d_i) by storing future veh. velocities
- Re-ordering two iterations of the classical model



$v(i, t+1)$	3	-1	-1	-1	2	-1	-1	-1	4	-1	...	...
$v(i, t+2)$	-1	-1	-1	$d+v_5=$ $0+2=$ $\rightarrow v=2$	-1	-1	max. $v=$ $d+v_9=5$ $\rightarrow v=3$	-1	-1	-1	...	...

A novel model (III)

- Only two rules

1. Search (Deceleration) and Bounded Acceleration:

$$v_{i+v_i(t+1)}(t+2) = \min\left(d_i^* \quad + v_{i+v_i(t+1)+d_i^*}(t+1), v_i(t+1), v_{max}\right);$$

2. Randomization:

$$v_i(t+2) = \max(v_i(t+2) - 1, 0)$$

(braking reduces velocity in one unit with probability Pb);

d^*_i = number of empty cells ahead after the safety distance $v_i(t+1)$

CONTENTS

- Our objectives
- Introduction and Related Works
- Representing vehicle movements
  - A novel model: reordering two iterations of the rules of the classical model
- **Efficient definition of data structures**
- Results and Discussion
- Conclusion and future work

Efficient def. of data structures

- Prevent that complex structures play a significant role
- Prevent searching in disordered lists
- Prevent continuous updating of lists

v(i, t)	2	-1	-1	-1	4	-1	-1	-1	-1	0	...	...
v(i, t+1)	-1	-1	-1	3	-1	-1	2	-1	-1	1	...	...
p_veh_list	2				0					1		

Vehicle list (dynamic?)

index	*Cell index*	*Type*	*Routing preferences*	...
0	4	Car	Right	...
1	9	Car	Left	...
2	0	Truck	Left	...
3	...	...	...	...
...	...	...	...	...

Street list (static)

index	*begin*	*end*	*Next Street*	*Previous Street*	...
0	0	199	1	1244	...
1	200	200	2	0	...
2	201	260	4	1	...
3	261	261	5	7	...
4	262	361	7	2	...
...	...	...	...		...

CONTENTS

- Our objectives
- Introduction and Related Works
- Representing vehicle movements
  - A novel model: reordering two iterations of the rules of the classical model
- Efficient definition of data structures
- **Results and Discussion**
- Conclusion and future work

Results and Discussion (I)

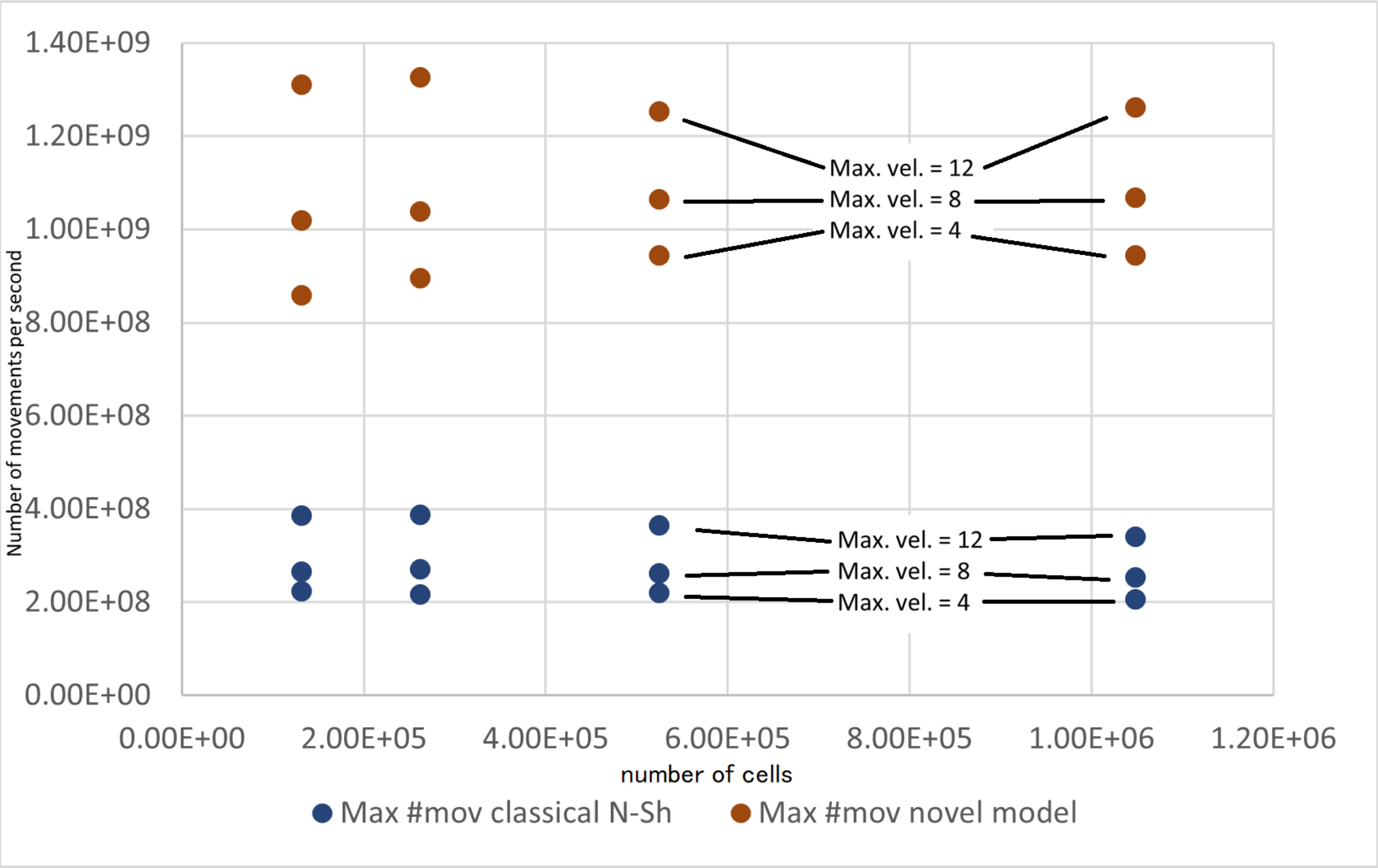
- A first comparison between languages
  - Algorithm written in Python : around 70x slower than C++
    - (even for small number of cells (64 Ki) and one core)
  - Speedup in C++ is easily achieved using OpenMP
    - (near to the number of physical cores if L3 is not saturated)
- Simulations on two computers
  - Laptop and modern desktop PC: machine has little influence on results.
  - Intel Core i7-10750H, 2.60GHz, 16.0 GB; 12th Intel Core i7-12700K, 3.60 GHz, 32.0 GB
- Parameter values: similar to those of current literature
  - 8% = ratio of the number of vehicles by the number of cells;
  - maximum velocity: 4, 8 and 12 cells per simulation step.
  - Random braking probability = 0.10;

Results and Discussion (II)

- Relation simulated vs. real time (a single 12-core PC)
- Mean values for a 1 million-inhabitants city
  - ≈ 5000 streets of 200m each
  - 4 m/cell ➔ 250 K cell/city
  - Dense traffic ratio : 8% cells are occupied (20 K veh.)
  - Real velocity = 8 m/s (28 Km/h) = 2.0 cells/mov.
  - ➔ 1 simulated mov. = Distance in 1 sec. of a real vehicle
  - ➔ $\frac{1\ G\ simulated\ veh.mov.}{1\ s_{simulation}} \approx \frac{1\ G\ real\ veh.mov.}{1\ s_{real\ (traffic)}}$

$$\frac{1\ G\ veh.mov.}{1\ s_{simulation}} \times \frac{1\ s_{real}(city)}{20\ K\ veh.mov.} = \frac{50\ K\ s_{real}(city)}{1\ s_{simulation}} \approx \frac{\mathbf{17\ h_{real}(city)}}{\mathbf{1\ s_{simulation}}}$$

Results (III): Na-Sch vs. novel



Results (IV)

	Number of vehicle movements per second			Speedup	
Max. Veloc	Classic	Optim	Novel	Classic vs Optim	Classic vs Novel
4	2.28177e+08	4.68137e+08	7.09059e+08	2.05	3.11
8	2.17847e+08	4.41291e+08	7.60969e+08	2.03	3.49
12	2.06842e+08	4.06585e+08	8.67613e+08	1.97	4.19

Using a vehicle list

Ratio veh/cells	Na-Sch classic using vehicle list (r.r. =0.1)	Na-Sch classic using vehicle list (r.r. =0.2)	Na-Sch classic using vehicle list (r.r. =0.8)
5	5.61475e+08	4.35744e+08	2.83068e+08
10	4.68071e+08	3.60048e+08	2.75381e+08
20	3.84744e+08	2.87258e+08	2.33214e+08

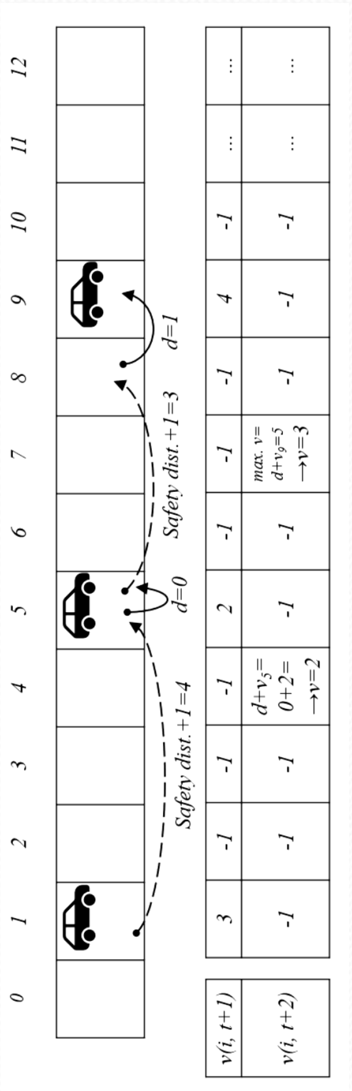
- Loop with one iteration for each vehicle.
  - *1)* Go through the vehicle list structure; *2)* read the cell index, which points to the corresponding cell where each vehicle is placed; *3)* where finally the movement is to be computed

CONTENTS

- Our objectives
- Introduction and Related Works
- Representing vehicle movements
  - A novel model: reordering two iterations of the rules of the classical model
- Efficient definition of data structures
- Results and Discussion
- **Conclusion**

Conclusion

- Compiled languages reduce run-time more than 70x
- Proposed a novel reordered vehicle movement model
  - reduces exec. time by more than 3x wrt classical Na-Sch
- Run-time in 12-core PC is close to supercomputers (thousands of cores, interpreted languages).
  - 17 h of a 10^6–inh. city in a second of simulation.
  - This also prevents energy wasting.
- A cell vector is usually faster than vehicle list
  - Important: reduce memory consumption.
  - Avoid disperse accesses to the computer memory



¡Gracias por su atención! … *¿Preguntas?*

Thank you for your attention! … *Questions?*

¡Gracias por su atención!
Thank you for your attention!

…

Questions?
¿Preguntas?