Preparation of Digital Maps for Traffic Simulation; Part 1: Approach and Algorithms

Daniel Krajzewicz, Georg Hertkorn, Julia Ringel, Peter Wagner
German Aerospace Centre, Institute of Transportation Research
Rutherfordstr. 2
12489 Berlin
Germany
E-mail: Daniel.Krajzewicz@dlr.de, Georg.Hertkorn@dlr.de, Julia.Ringel@dlr.de, Peter.Wagner@dlr.de

KEYWORDS

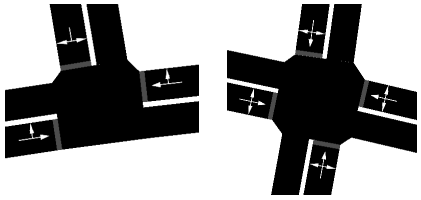
Microscopic traffic simulation, digital road maps, open source, traffic research

ABSTRACT

Traffic simulations are an accepted tool for investigations on road traffic and used widely within the traffic science community. Modern computer systems are fast enough to model and simulate traffic within large areas at a microscopic scale regarding each vehicle, replacing macroscopic simulations in most cases.
Although microscopic traffic simulations offer better quality than macroscopic ones, they also need additional data to describe the modelled road networks. A street's lanes are modelled explicitly within microscopic simulations and in most cases also the connections between their lanes over junctions.
If one wants to model large areas, the best source to get the description about their road network is the usage of digital maps. Unfortunately, most of these are used for routing purposes and do not contain the fine-grained information mentioned above that is needed by microscopic simulations. This document describes an algorithm for the computation of the needed information from simple road networks.

INTRODUCTION

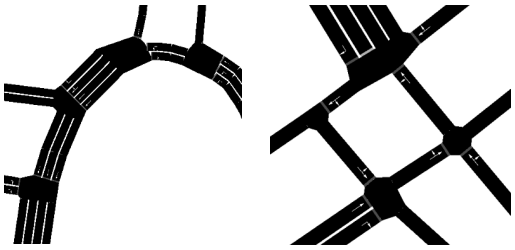
"SUMO" – an acronym for "Simulation of Urban MObility" – is an open source traffic simulation package developed at the Institute of Transportation Research at the German Aerospace Centre. This application is used within some of our institute's projects for simulating impacts of new optical sensors (see [1]) or traffic management strategies (see [2]). Descriptions of the SUMO package may be found in [3] or at the project's webpage [4].
The microscopic view on road networks contains information about the connections between consecutive lanes. When a junction has to be crossed, vehicles have to choose a proper lane in order to get to the desired one. Beside these lane-to-lane connections, one also has to model which of these connections are foes to which other connections in order to make vehicles coming from directions with a smaller priority wait.



Picture 1: Two rather uncomplicated junctions

We deal with large areas – cities or larger parts of these or highway networks – and we wanted to reduce the amount of work a user has to perform before a simulation can be started. Often, one has to edit all information mentioned above by hand. Our approach is an automatic generation of such data. For this purpose, beside the simulation program itself, the SUMO package also includes a program which allows the conversion of networks from other formats. These formats have their origin in other simulations, both macro- and microscopic ones, or in routing systems. While both microscopic simulations SUMO can import, Vissim by ptv (see [5]) and ARTEMIS by Prof. Dr. Peter Hidas ([6]) contain information about the described lane-to-lane connections, this information are not stored within the inputs to macroscopic simulations or routing systems. Macroscopic simulations do not use them as the flow is not modelled using vehicles but in an abstract way (see [7]) and so no lanes, but the streets' capacities are used (see. [8]). Routing systems do not consider the information about lane connections, too.
The needed automatic computation of lane-to-lane connections and of the information about foes was not as trivial as it sounds first. Road networks do hold many special cases to regard. Due to this, we decided to publish the algorithm as it may be interesting to other and also allow other to take a look at it in order to improve it.
Within this publication, we will describe the main paradigm used for modelling road networks and the information made available by the formats we regard. Then, we will describe which information is needed by SUMO and then, how we compute

this information from the available data. Some comparisons to the reality will follow, together with a conclusion and some thoughts about further work.
This publication does not contain the description about how the geometry of junctions or streets is computed, but only how to compute the logical connections.



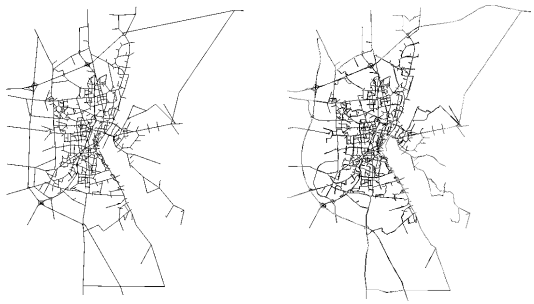
Picture 2: Two sets of rather complicated junctions as computed by the algorithm

ROAD NETWORK DESCRIPTIONS

Road networks are normally stored as a directed graph. Junctions are represented as nodes and streets as edges. Leaving out Vissim which has a different view on road networks, this paradigm is found throughout the formats we import, namely in Visum, ArcView, FastLane, ARTEMIS, or Navteq - networks.
The description of a junction is normally quite minimalist. Beside the position it is located at and an identifier, sometimes the information about its type is found which in most cases distincts between simple priority junctions and junctions controlled by a traffic light. The descriptions of the traffic light plans themselves are not that uniform and we will not discuss them herein. Still, one has to keep in mind that the information whether a link is controlled by a traffic light is necessary during the computation whether a link is foe to another.
An edge's description contains more parameter. At first, the number of lanes the edge consists of should be given. In the case of macroscopic simulation networks, this is sometimes not the case and the edge's capacity has to be used to compute the edges' lane number. Assuming the maximum capacity of a lane as 20000veh/h, the formula below computes the number of lanes if this information is not given. The only problem is that in some cases, edges used in macroscopic traffic simulation networks are using unreal values for the flow in order to fit the simulation to reality or to guarantee a high inflow. The second case can be caught and managed because it mostly occurs on feeding edges only which are mostly marked as such (at least in Visum). Furthermore, a maximum number of lanes can be applied. The first case can only be changed by hand.

$$lanes_{edge}=capacity_{edge}/max\_capacity_{lane} \quad (1)$$

Further information optionally stored within an edge's description is the edge's type. It is not found in all inputs and also no standard value sets exist. This is quite unfortunate as we will see later. Other road attributes are either stored directly within the edge or within the type and in the second case they must be retrieved indirectly. These attributes are: the maximum velocity allowed on the edge, possibly the length, information whether overtaking is allowed or not and geometrical information as a single edge may be not a straight connection between two junctions but may possess a curvature (see picture 3).



Picture 3: Two views at the city of Magdeburg; the left network uses straight connections for streets, the right one includes the streets' geometries

NEEDED INFORMATION

When looking at a junction in detail, one will see that some further information is needed. The vehicles using a street which has more than a single lane, have to choose the lane they use properly in order to get to the next street (see picture 2). The wish to model this fact is not only a try to make the simulation appear more realistic, but is also important for realistic results.
A further needed information is also mostly not available: whether a vehicle has to stop before entering a junction when another vehicle approaches from another direction. Of course, this behaviour depends on the junction‘s type. In the case of traffic light controlled junctions, the behaviour changes with the traffic lights‘ state. In the case of uncontrolled junctions, it depends on whether the roads participating in the junction have different priorities or the vehicles have to interrogate which of them may pass the junction first. In Germany, for example, many junctions use the right-before-left – rule. In all cases of uncontrolled junctions, vehicles have to wait only if their trajectories cross. Stop signs by now remain unregarded.

INFORMATION COMPUTATION

Some Remarks to the Import Procedure

To avoid duplicate methods, the import is divided into two parts. At first, the input is read using a reader capable to parse the format of the imported file and converted into the net converter's internal structures. Within the second step, the network converter uses the information stored within these internal structures to compute all values the simulation needs. If any of the needed information was already available within the input file, the computation is skipped and the read values are used. This method allows an easy implementation of import modules and guarantees the same results independent of the import format itself.

We will not go into depth about parsing of input files. Most of it is straightforward because the available information is the same for most cases and the algorithm described herein is executed within the second step.

Lane-to-Lane Relationship Computation

After loading, all loaded junctions (called 'nodes' from now on) and streets ('edges') are stored in two containers – one for the junctions and one for the edges. The order of those objects within these containers is not defined. All operations are executed on every junction and on every edge within the according container. The following operations are done in the given order:

1. for each edge: compute turnaround edges
2. for each node: sort each node's edges
3. for each node: compute each node's type
4. for each node: set edge priorities
5. for each edge: compute edge-to-edge connections
6. for each edge: compute lanes-to-edge connections
7. for each node: compute lane-to-lane connections
8. for each edge: recheck lanes
9. for each edge: append turnarounds

We will now describe what those functions do exactly do and why they are needed.

1. Compute Turnaround Edges

Among a street's successors, the turnaround direction is a special case as the number of lanes used to reach it is always one and because this direction is not regarded as an explicit direction: it is uncommon to have a lane which is only used to turn around as the wish to do so only seldom occurs. A further reason for computing the turnarounds is due to needing this information for the computation of the edges' clockwise order within step 2. Due to his peculiarities the backward edge is computed for each edge first and this information is stored to allow neglecting the backward direction on further processing (steps 2-8).

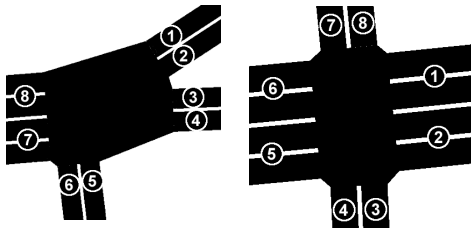
To compute the turning directions, we assume that an edge is the backward edge if the absolute value of the difference between the current edge's direction and the edge considered to be the backward direction is larger than 160°. The angle is measured at the current junction as each edge is a list of straights, and due to this a street's angle may differ along it.

This computation holds a trap: there may be more than a single edge that fits to the 160°-rule. In this case, the edge with the largest absolute direction difference is used. Still, to check for 160° is necessary as the usage of the edge with the highest rotation difference only would be false in cases where an edge has no backward direction at all.

2. Sort each Node's Edges

Within the second step, each of a junction's streets – both incoming and outgoing – are sorted clockwise by their direction. "Direction" means here the angle at the currently regarded junction. After this, the edges are sorted by their logical direction – whether they are incoming or outgoing. This is done by going through the sorted list of edges and checking whether the next edge (the one at the current position + 1) is the backward direction of the current one and is an incoming edge. In this case, the outgoing edge was sorted to lie left to the incoming one, but should be on the right side. In this case, the edges are swapped within the list.

The result of this operation is a clockwise sorted list of edges that participate in the junction and the incoming lanes lay before the outgoing ones as shown in picture 4.



Picture 4: The resulting order of edges after sorting

3. Compute Nodes' Types

We regard three different types of junctions: "no junction", "priority junction", and "right before left junction". The first case, "no junction" is a special case where no right of way rules are applied. A normal network should not contain such junctions, but if for example two highways cross, this type may be used. Within "priority junction" an incoming street and its consecution

has a higher priority than other incoming streets. Within "right before left junctions", all directions have the same priority and as common in Germany, vehicles which come from the right road may pass, the others will have to wait.
The resulting junction type is computed for each pair of edges which income to the junction. If one of the combinations yields in the type "priority junction", this type is used. Otherwise, the type "right before left" is used.
To determine the junction type for a single combination of two incoming edges, a two dimensional matrix is used. As indices, the velocities of the regarded edges are used; the values are the junction types. Table 1 shows the used matrix.

	-10	10-30	30-50	50-70	70-100	100-
-10	r	p	p	p	p	x
10-30		r	p	p	p	x
30-50			r	p	p	x
50-70				t	p	p
70-100					t	p
100-						x

Table 1: Junction types in dependence to two crossing streets; r: right-before-left, p: priority, x: no junction

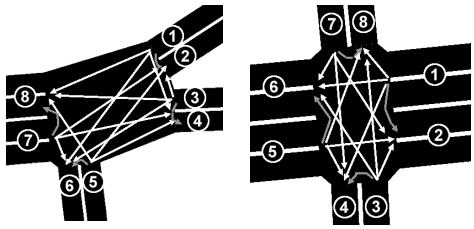
4. Set Edge Priorities

Now for each edge, we set his priority within a junction. This information will be needed later to determine how many lanes shall approach this edge – in the case of an outgoing edge – from an incoming edge. The priority of an edge within a junction may differ from the edge's overall priority because edges of different types may cross within the junction. The edge's overall priority is also not always available within the network description. If not, the edge's speed may be used.
The edge's priorities for a certain junction they participate within are computed as following:

- highest_incoming_edge2 = undefined
- highest_incoming_edge1 = get the incoming edge with the highest priority
- if further incoming edges exist:
  - highest_incoming_edge2 = get the incoming edge with the second highest priority
- if at least one outgoing edge exist:
  - highest_outgoing_edge1 = get the outgoing edge which is almost in the same direction as highest_incoming_edge1
- if further outgoing edges exist and highest_incoming_edge2 is not undefined:
  - highest_outgoing_edge2 = get the outgoing edge which is almost in the same direction as highest_incoming_edge2

5. Compute Edge-to-Edge Connections

Herein, the list of the edges that may be reached from the currently seen edge is computed. This is simply done by connecting all outgoing edges to the currently regarded incoming edge. The connected edges retrieved from the junction are already sorted as described in step 2. Picture 6 shows the generated edge-to-edge connections for our example.



Picture 6: The resulting connections between edges (grey: turnaround computed in step 1)

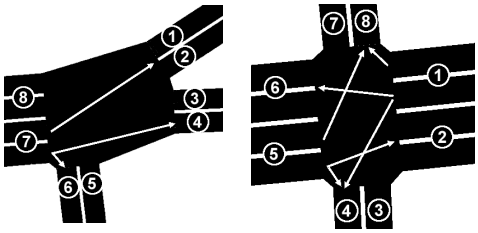
6. Compute Lane-to-Edge Connections

What we have to do herein is to fan out the lanes onto the outgoing edges, in order to compute which edges can be reached from a certain incoming lane of the currently regarded edge.
This computation is quite tricky and requires some heuristics. The problem is that the number of lanes used to reach a certain following edge is depending on the available number of edges and on the amount of traffic flow that will use this connection. For most cases, the flows can be determined using the number of lanes the following edge has. As the comparison at the end of this report show, this is not always true. The flows themselves are not available within the process of network conversion.
The following computation is done for each edge:

- get the list of connected edges beside the turnaround
- sort them by their angle
- for each edge in this list, compute its priority for the current edge:
  - priority = (connected edge's junction_priority + 1) * 2
- if one of the lower priorised outgoing roads goes to the right:
  - divide his importance by 2 as vehicles using it can leave the junction faster
- if there are no major roads at this junctions:
  - multiply the outgoing road that goes straight by 2, making it more important then the others
- compute the number of lanes that shall approach each of the connected edges:
  - sum up all priorities
  - for each outgoing (connected) edge:
    - number of lanes to use to reach this edge = this edge's priority for the current edge / priority sum
    - if number > number of current edge's lanes:
      - number = number of current edge's lanes

We now know how many lanes shall approach each of the connected edges. We still have to compute which lanes are used for which edge or

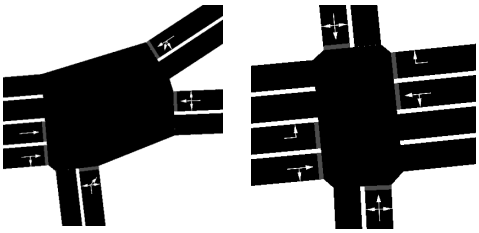
better to say from which lane the next edge may be approached.
This is done using the Bresenham algorithm for line computation assuming the list of lanes of the current edge to be one dimension and the (sorted) list of edges to approach the other one



Picture 7: Connections from lanes to edges; Shown are only connections from edges which have more than one lane – in the other case the single lane is connected to all connected edges

7. Compute Lane-to-Lane Connections

For each of a junction's outgoing edges, we now have to determine, which approaching lanes will yield on which of this outgoing edge's lanes. This is done using the Bresenham algorithm, too. The dimensions the algorithm goes by are the number of edges connected to the currently regarded outgoing edge and the number of lanes the currently regarded outgoing edge has.



Picture 8: After step7, the junctions are completely built for the most cases

8. Recheck Lanes

In step 6, we have computed from which lane which edge may be reached. In some cases, mainly if the connected edges have more lanes than the current edge, not all lanes were filled, yet. Within step 8, such mistakes are being searched and corrected. The algorithm simply checks whether a lane has no connected edge and if a neighbour lane has more than one connected. In this case the neighbour lane's connections to a directionally matching edge are moved to the side.

9. Append Turnarounds

Now, after all normal connections between all edge's lanes are set, the turnarounds may be added to the leftmost lane.

Foe Computation

The computation of which streams are foes to which requires only two steps. The wished output is the information about which connections from one incoming lane to an outgoing lane have to wait while other may drive. The problem is that not only one has to take into account right-before-left relationships, but also compute whether two streams are crossing at all.
Within the first step, for each connection the information is set whether it is controlled by a traffic light or not.
The second step is done for each junction. It assumes that the edges are sorted clockwise as described in "2. Sort each Node's Edges". The algorithm is as following:

- for each incoming edge i:
  - for each outgoing edge o:
    - compute right-hand link crossings(i, o)
    - compute left-hand link crossings(i, o)

where:

compute right-hand link crossings(i, o):
- ip = position of i in the sorted list of edges
- while ip!=position of o in the sorted list of edges:
  - move ip counter clockwise (decrement, wrapping)
  - i2 = edge at position ip
  - if i2 is an incoming edge:
    - op = position of o in the sorted list of edges
    - while op!=position of i in the sorted list of edges:
      - o2 = edge at position op
      - if o2 is an outgoing edge:
        - connection i to o crosses the connection i2 to o2
        - check priority(i, o, i2, o2)
      - move op counter clockwise (decrement, wrapping)

compute left-hand link crossings(i, o):
same as "compute right-hand link crossings(i, o)", but the pointers are moved clockwise (increment, wrapping).

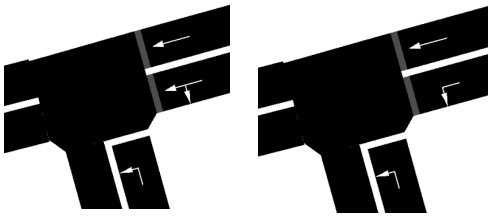
check priority(i, o, i2, o2):
- if already checked:
  - return
- mark as checked
- if connection i to o is a turnaround:
  - i to o is lesser priorised than i2 to o2
- same for i2 and o2
- if i has a higher junction priority than i2:
  - i2 to o2 is lesser priorised than i to o
- if i has a lesser junction priority than i2:
  - i to o is lesser priorised than i2 to o2
- if i and i2 have a high junction priority:
  - if o has a higher junction priority than o2:
    - i2 to o2 is lesser priorised than i to o
  - if o has a lesser junction priority than o2:
    - i to o is lesser priorised than i2 to o2
- if counter clockwise distance between i and i2 is smaller than between i2 and i:
  - i to o is lesser priorised than i2 to o2

- if counter clockwise distance between i and i2 is larger than between i2 and i:
  - i2 to o2 is lesser priorised than i to o
- if counter clockwise distance between i and o is larger than between i2 and o2:
  - i to o is lesser priorised than i2 to o2
- else:
  - i2 to o2 is lesser priorised than i to o

This algorithm has the benefit to compute all needed information without the need to know a junction's geometry. Only logical information is used.

COMPARISONS TO THE REALITY

Most of the evaluations we have made so far is based on evaluating the generated networks by hand and determining whether the generated junctions look like they should or not. The algorithm described in here is about 3 years old and worked well during such tests so far.
The only methodological analysis was done for a small area which real-life counterpart is located around our institute's site. For this case, the algorithm returns the correct result for all (177) but four junctions, being to almost 98% correct. One of the junctions which do not match is shown in picture 9. The reason for the mismatch is due to the fact, that in real life, most of the traffic is turning left – something the algorithm is not able to forecast. A further reason for false computation was the lack of information about additional left-turn lanes within the original network description.



Picture 9: False (left) and corrected (right) junction within the evaluated scenario

CONCLUSIONS

The comparisons to reality show that the error one should expect is quite small. When taking into account that if not using these methods one would have to edit every junction by hand, this approach seems to be very useful: generating the network for a city with about 10.000 junctions takes less than a minute using the SUMO net converting tool. Assuming you would do this by hand, you would have to touch every junction several times – if each of such actions would cost you only 1 second, you still would have to spend about three weeks working on it.

FUTURE WORK

The heuristics used within step 7 of lane-2-lane computation are something that should be revalidated. Although the results are fine for the networks we have used so far, such methods seem quite error-pruned and may cause problems in the future. As reported, only one methodological analysis was done so far. This disallows to predict the algorithm's performance for a broader set of situations than inner-city areas in Germany. Some tests should be done using comparisons of SUMO-generated networks and networks having the complete topology.
A special case for such world-covering evaluations would be networks with left-handed traffic, which so far are not capable to be generated at all.
There are also at least two further important topics which should be covered by additional heuristics. The first is the need to guess for each junction whether a traffic light is positioned on it or not. The second is determining whether a junction is located on a highway, because in some cases some of the connections must not be inserted.

REFERENCES

[1] IVF/DLR. OIS project pages at: http://www.dlr.de/vf/forschung/projekte/ois

[2] INVENT. Invent-Homepage at http://www.invent-online.de

[3] D. Krajzewicz, G. Hertkorn, C. Rössel, P. Wagner. 2002. "SUMO (Simulation of Urban MObility); An open-Source Traffic Simulation" Proceedings of the 4th Middle East Symposium on Simulation and Modelling (MESM2002), Edited by: A.~Al-Akaidi, pp. 183 - 187, SCS European Publishing House, ISBN 90-77039-09-0

[4] D. Krajzewicz, G. Hertkorn, C. Rössel, P. Wagner. 2002-2005. SUMO Homepage. http://sumo.sourceforge.net

[5] ptv. ptv Internet pages at: http://www.ptv.de/

[6] P. Hidas. 2004. "Modelling Vehicle Interactions in Microscopic Simulation of Merging and Weaving". In: Transportation Research Part C: Emerging Technologies, Elsevier Science Ltd. ISSN: 0968-090X

[7] D. Chowdhury, L. Santen, A. Schadschneider. 2000. "Statistical Physics of Vehicular Traffic and Some Related Systems"; Physics Reports 329, (4-6), S. 199-329, Elsevier, 2000, ISSN: 0370-1573; also available at: http://www.arxiv.org (cond-mat/0007053)

[8] C. Gawron. 1998. "Simulation-Based Traffic Assignment: Computing User Equilibria in Large Street Networks", Dissertation, Köln 1998