On Constructing a *Base Map* for Collaborative Map Generation and its Application in Urban Mobility Planning

Maik Drodzynski, Stefan Edelkamp, Andreas Gaubatz, Shahid Jabbar, Miguel Liebe

Computer Science Department,
Otto-Hahn-Str. 14
University of Dortmund,
D-44221 Dortmund
Germany

Abstract— **Urban mobility planning depends largely on the presence of good navigation data in the form of vectorized maps. Nonetheless, vector maps are not always available for many areas – especially for many of the third world countries. On the other hand, good paper maps collected by the city authorities are widely available.**

A solution is the *collaborative map generation* process that allows people to share the collected GPS traces. Nevertheless, the integration of these GPS traces is itself a challenge and requires a good *base map*.

This paper presents a method to extract calibrated road topology from raster maps to provide such a *base map* for collaborative map generation process. Our approach takes a bitmap and uses different graphics filters to infer the road geometry. We propose an aggregation algorithm that extracts the actual vectorized road fragments and construct a graph of road network.

To evaluate the proposed algorithms, we have implemented them and tested on real raster maps collected from the city authorities of Dortmund, Germany. We also report the integration of our approach into SUMO, a state-of-the-art traffic simulation tool for urban mobility.

I. Introduction

With the increase in traffic worldwide, especially in large urban areas, mobility planning and traffic simulation is a necessity for sustainable transportation. Both tasks rely largely on the availability of good vector maps.

The invention of collaborative map generation – *a product of Web 2.0* – provides a solution to the problem of urban mobility for those areas where their is an abundance of good vector maps. Projects like GPS-Tracks, Wikimapia, and Routeburner, let people add GPS traces collected by car, bicycle, or even during hiking to a common web portal. The advantage is a huge set of realistic GPS traces representative of different travel situations and scenarios.

Even though it solves the problem of having a good collection of GPS traces, it poses two new challenges to the community.

1) Combining these GPS traces to represent a single real road is still an open challenge. In [2], we have presented an algorithm that exploits AI clustering techniques to integrate different GPS traces and infer the actual road geometry. A recent extension proposed in [14] enhances the approach through learning based on genetic algorithms.
2) For an appropriate aggregation of GPS traces, a good *base map* is necessary. The absence of such a base map can distort the whole integration process and can result in a drift in the road topology.

It is the second problem that we address in this paper and propose an applicable and automated solution to the problem.

In practice, *base maps* are mostly borrowed from Google-Maps or Google-Earth. Nevertheless, in many areas of interest, detailed vector maps are scarcely available and in some regions of the world, vector maps are not available at all. On the other hand, low-cost raster maps are frequently accessible, e.g., in form of topographic or surveying images of high-quality. The images are often calibrated with respect to some form of global coordinate system to be translated to GPS.

Several efforts have been made in the direction of extracting the road geometry from images. In [13], we see a “context based extraction” of roads. The approach uses a set of aerial images of the same region but taken from different heights. Road extraction then proceeds iteratively by first extracting the most salient parts and the attributes of roads and then going for the finer ones. The whole process is guided by available context information. [3] discusses somewhat similar approach as ours for street graph extraction but does not integrate to any coordinate system or any traffic simulation. To the best of the authors’ knowledge, this is the first attempt to combine raster maps to traffic simulation for GPS trace generation.

The paper is structured as follows. We introduce the (set of) algorithms that transfer the bitmap to a graph. We show how roads are extracted, introduce the syntax and semantics of graphical filters, and present and analyze the algorithms to produce the road skeleton that is used to finalize the construction. The effects of graphical operations are illustrated with a running example.

Next, we turn to the automatic construction of the road network graph, its compression and simplification. Besides optional changes to the bitmaps during the conversion itself, the extended GUI allows a flexible post-processing of the

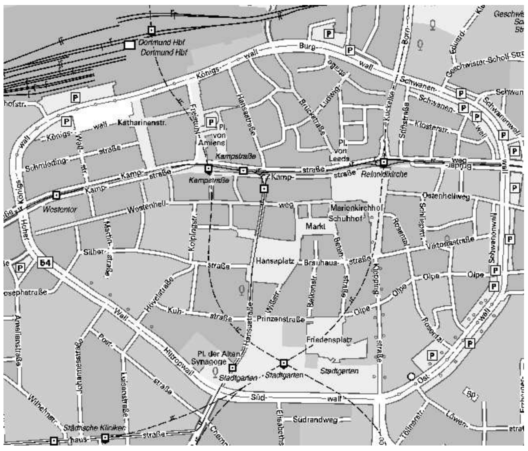


Fig. 1. A Fragment from a Topographic Map of Dortmund, Germany.

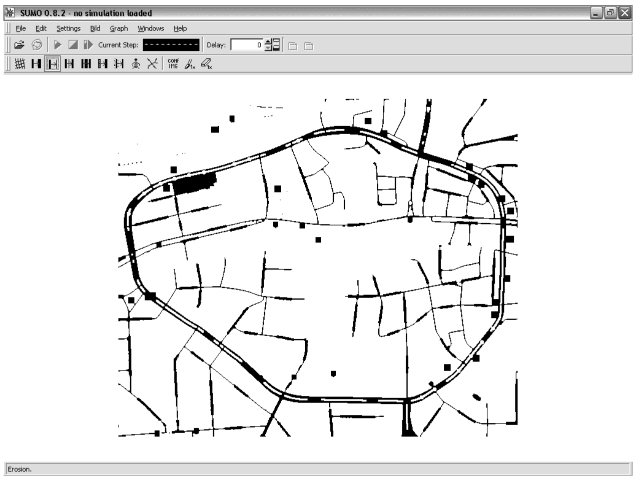

SUMO 0.8.2 - no simulation loaded
File Edit Settings Bild Graph Windows Help
Current Step:
Delay:
Erosion.

Fig. 2. Result of Erosion

traffic network.

We then present the integration of our approach to SUMO, a state-of-the-art traffic simulation tool and provides a feasible and applicable solution to apply traffic simulation directly on the raster maps. Last but not least, we summarize our work and address future research avenues.

II. Extraction of Road Surfaces

The raster map we started from, contains a set of pixels. Each pixel at position (i,j) carries its color information in form of a RGB triple. Fortunately, the set of topographic maps we considered do not contain much noise, such that the streets are easily discriminated by their color value (See Fig. 1). We allow the user to select the color value for road extraction. Next, we set all the information that belongs to the street network to black, and all other information to white. Using this simple filter yields some subtle problems: in our maps we have black letters for street and city names and further urban information, moreover, all railway tracks are drawn in black.

The solution to define black as the color of the road includes railway tracks. On the other hand, not using black would give rise to many white holes in the road infrastructure due to the street names. Moreover, as parking and play grounds are white in our input, selecting this discriminating color also has its drawbacks. Subsequently, the current extraction process is not perfect and needs some manual post-processing.

For automated post-processing we implemented six morphological filters: *Erosion*, *Dilatation*, *Morphological opening*, *Morphological closing*, *Gap closing*, and *Fragment elimination*. We illustrate the working of these graphical filters in the following using a running example. The mathematical basis for these filters is set theory and are termed morphological filters because they work on the *shape* of the image.

A. Erosion

In order to remove the city and street names from the map, we employ an erosion operation. An erosion operation is defined with respect to a *structuring element* called mask. Intuitively, it works like a net with holes in the shape of a mask. All the elements of the image that can pass through the holes disappear.

For two sets A and B in $\mathbb{Z}^2$, the *erosion* of A w.r.t B is defined as $A \ominus B = \{z \mid (B)_z \subseteq A\}$. Therefore, an erosion of A w.r.t the mask B consists of all points z, such that if B is shifted with its center point at z, it remains in A.

The form and size of the mask is crucial for the results of erosion. To avoid distortion of the road elements, a symmetric mask is required. As pixels are deleted by erosion (starting at the fringes of the road surfaces), the mask should not be selected to be too large, suggesting a 3×3 pixel map in our case. Using erosion a large part of the non-road elements such as railway tracks and names are eliminated. The result of erosion on our example is depicted in Fig. 2.

B. Dilatation

Due to the application of erosion operation, street lines might become distorted, especially at the places where they overlapped a street name. A dilatation operation is then used to smoothened these contours and to fill certain holes that might have appeared during erosion.

For two sets A and B in $\mathbb{Z}^2$, the *dilatation* of A w.r.t the mask B is defined as $A \oplus B = \{z \mid (B)_z \cap A \neq \emptyset\}$. Dilatation enlarges the number of black pixels and closes some gaps within the road surfaces. As in erosion, the choice of mask is crucial and it should be taken care of that no two road elements are merged into one.

C. Morphological Opening

The morphologic opening is a composite morphological operation based on both erosion and dilatation. For two sets A and B in $\mathbb{Z}^2$, the *morphological opening* of A w.r.t the mask B is defined as $A \bullet B = (A \ominus B) \oplus B$. This operation helps in removing small bridges between black surfaces, smoothening of the contours and elimination of small noises.

D. Morphologic Closing

The morphologic closing of a set A given the structuring element B in $\mathbb{Z}^2$ is defined as $A \circ B = (A \oplus B) \ominus B$. Using this operation will also smoothen contours, but in difference to a morphologic opening small bridges between black surfaces are strengthened and small gaps and indentations are filled.

E. Gap Closing

A problem that cannot be solved with the dilatation alone is the elimination of gaps within large road elements such as inter-state highways.

Several dilatations can close those gaps, but may lead to a merging of different road fragments. A solution to the problem is a specialized algorithm *gap closing*, which test for each white pixel if it has more than $n \in \{1, 2, ..., 8\}$ black neighbor pixel. The number n is to be provided by the user, with $n = 5$ as a feasible choice.

F. Fragment Elimination

Even after applying the above-mentioned filters, some of the railway tracks and other map symbols still remain on the map as isolated fragments. Such fragments are searched and removed in the *fragment elimination* phase.

We iterate through every pixel of the bitmap. If we find a black pixel x, the neighborhood of x is analyzed. We exploit the property that route fragments accumulate to continuous chains of black pixel. If all surrounding pixels of x are white, we assume x to be isolated from the other black pixels and color it white. In the method a odd integer $i > 3$ is taken as a parameter, which denotes the size of the square that is drawn around x. In a loop i is increased successively. If all the border pixels of the square are colored white, the iteration is stopped coloring all internal pixels also white. If this is not the case, the square is enlarged until the upper bound is met. Corner pixels of the bitmap have to be considered in a refined case study.

III. Road Skeleton Computation

After the road surfaces have been clearly extracted using the application of filters and some additional manual refinements, the skeleton has to be computed as a basis from which the street graph is generated. Roughly speaking, the skeleton of a pixel map is a set of thin curves, denoting the centerlines of the black surfaces. With respect to the road geometry, these are the centerline of the streets to look at. Using the mathematical notation of morphological arguments the skeleton of A is defined as $S(A) = \bigcup_{k=0}^{K} S_k(A)$ with $S_k(A) = (A \ominus kB) - (A \ominus kB) \circ B$, where B is the structuring element and $(A \ominus kB) = (\ldots(A \underbrace{\ominus B) \ominus B) \ominus B) \ominus \ldots) \ominus B)}_{k \text{ times}}$ Moreover, K is the last iteration before A is the empty set. i.e., $K = \max\{k \mid (A \ominus kB) \neq \emptyset\}$.

A. Skeleton Generation

A first implementation of the classical definition of the skeleton has yielded the aimed result. But, a skeleton generated by using the above mentioned definition is not guaranteed to be connected and may easily break down into pieces especially at crossings. This can make graph generation infeasible.

A heuristic solution to this problem is proposed by [1]. It is based on the medial axis transformation, MAT for short. The MAT of a region R with fringe G is defined as follows. For every pixel p in R, we search for a nearest neighbor in G. If there are more than one neighbor, then it belongs to the middle axis, the skeleton. An illustrative interpretation of the MAT is a fire in the Savannah, with dry grass that is set into flames at its fringe. The MAT corresponds to all positions that are reached by at least two fire frontiers. A direct conversion of this idea leads to inefficient algorithms, since the computation for distances from all interior points to all fringe pixels is involved. A comprehensive survey of skeletonizing techniques can be found in [12]. In [6], this heuristic solution has been successfully applied to find the boundaries of granulation cells.

The binary bitmap is processed in two stages that are successively altered until no more changes are to be observed. Let the considered pixel be arranged as follows.

p_9	p_2	p_3
p_8	p_1	p_4
p_7	p_6	p_5

In the first stage, all black pixels are marked that satisfy the following conditions: $2 \leq$ *Sum*$(p_1) \leq 6$, *Trans*$(p_1) = 1$, $p_2 \wedge p_4 \wedge p_6 = 0$, and $p_4 \wedge p_6 \wedge p_8 = 0$, where *Sum*$(p_1) = p_2 + p_3 + \ldots + p_9$ and *Trans*(p_1) denotes the number of 0/1 transitions in the sequence $\langle p_2, p_3, \ldots, p_9, p_2 \rangle$. In the following, two cases of pixel's positions are shown. On the left, we have *Sum*$(p_1) = 4$ and *Trans*$(p_1) = 3$, while on the right, we have *Sum*$(p_1) = 4$ and *Trans*$(p_1) = 1$ as required for the second condition.

●		
	p_1	●
●		●

●	●	●
	p_1	●

Condition 1 is violated, if p_1 has one, seven or eight black neighbors. In case of one neighbor p_1 cannot be deleted as it is at the end of a pixel chain. If black pixels with 7 or 8 neighbors are deleted, the region will quickly become sparse. Condition 2 takes care that the skeleton is not split, so that no pixel is deleted that lies on a chain of minimal width.

If all conditions are checked, the original is deleted. The manipulated bitmap is the input for the next step.

In the next stage, all black pixels are marked if the following conditions are satisfied. $2 \leq$ *Sum*$(p_1) \leq 6$, *Trans*$(p_1) = 1$, $p_6 \wedge p_8 \wedge p_2 = 0$, and $p_8 \wedge p_2 \wedge p_4 = 0$.

A pixel that satisfies the first two conditions and both first parity conditions is located east or south or is a north-west corner point. In this case the pixel is not in the skeleton and should be eliminated. Similarly, a pixel is eliminated that satisfies both second parity conditions is located north or wets or is a south-east corner point.

In Algorithm 1, we see the pseudo-code form of the skeletonizing algorithm that checks the entire bitmap for the pixels that satisfies the former conditions. Note that the pixels are not deleted in the first pass and only marked, so that they do not bring any change to the next iteration. In a second pass, all marked pixels are deleted.

Algorithm 1 Skeleton Generation

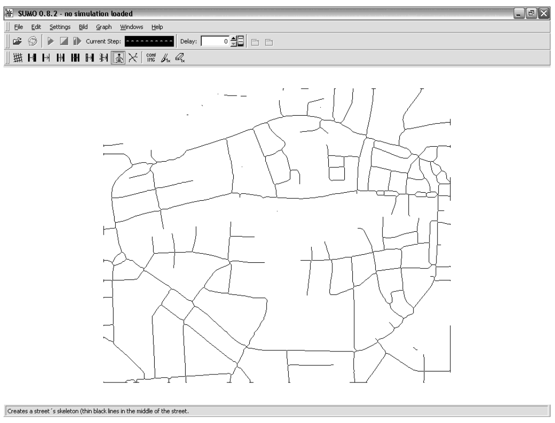
```
**Input:** 2DPixelArray $p$; *Height*; *Width*;
**repeat**
  *Marked* $\leftarrow$ **false**;
  **for** $i \leftarrow 1$ to *Height* **do**
    **for** $j \leftarrow 1$ to *Width* **do**
      **if** $p(i,j) = 1$ **then**
        **if** $2 \leq Sum(p(i,j)) \leq 6$ **AND**
          *Trans*$(p(i,j)) = 1$ **AND**
          $p(i,j-1) = p(i+1,j) = p(i,j+1) = 0$
        **AND**
          $p(i+1,j) = p(i,j+1) = p(i-1,j) = 0$
        **then**
          *Mark*$(p(i,j))$;
          *Marked* $\leftarrow$ **true**
        **end if**
      **end if**
    **end for**
  **end for**
  **for** $i \leftarrow 1$ to *Height* **do**
    **for** $j \leftarrow 1$ to *Width* **do**
      **if** *Mark*$(p(i,j))$ **then**
        $p(i,j) \leftarrow 0$;
        *Unmark*$(p(i,j))$
      **end if**
    **end for**
  **end for**
**until** *Marked* $=$ **false**
```

B. Skeleton Minimization

An immediate construction of the graph from the skeleton is problematic, since the skeleton is not necessarily *maximally sparse*. A skeleton is *maximally sparse*, if the elimination of a pixel having two neighbors can break down the skeleton. This condition can be used to minimize the skeleton and to identify pixels corresponding to crossings, dead-ends or to just an ordinary point. In total, we have to consider $2^8 = 256$ cases. The result of the skeleton generation and minimization phase is shown in Fig. 3.

IV. Graph Construction

Once the skeleton is generated, a *Tracking algorithm* constructs the underlying directed graph. This algorithm is based upon the Sweep-line paradigm: the pixels are processed in a column-wise fashion. As soon as a new pixel is found, which belongs to the skeleton, a sub-routine is invoked that traverse and process the connected component of the new pixel.


SUMO 0.8.2 - no simulation loaded

Creates a street´s skeleton (thin black lines in the middle of the street.

Fig. 3. The Generated Skeleton.

It generates nodes for all neighboring points. Furthermore, new pixels are added to the pixel chain at regular intervals (user-defined), and are connected with the previous pixels by new edges. As soon as a pixel representing a crossing is reached and converted to a node, a list of its neighbors is generated. This list is then iterated through, calling the same sub-routine recursively for every list element.

One thing that is to be especially taken care of is that the neighbors are not accessed twice. It can happen that during the traversing of the skeleton in this manner, a crossing is encountered that had been visited but not all of its neighbors had been processed. This problem can be resolved by using a flag that marks a pixel as visited and avoids re-processing when reached again.

Once a connected component has been processed completely, the sweep-line algorithm searches for another connected component in the picture. A node counter is set that counts the number of nodes that belong to a connected component. In this way we can get rid of those connected components that are too small to represent any street. These fragments may correspond to any noise that may have been left during the filtration process. The minimum size of a connected component should be provided by the user. Fig. 4 shows the street graph constructed for our example.

A. Graph Simplification

The graph constructed through the *Tracking algorithm* has a large number of nodes. In case of very large raster maps, the resulting graph could be enormous in size and would not even fit into the available memory. The graph can be simplified by the help of simple observations. Using some information on the road geometry, the number of nodes that are needed to represent a street can be reduced. In many cases, we have achieved a reduction of about half the nodes without losing any significant information about the road geometry. In the following, we discuss the two simplification techniques that are used in our approach.

1) Removal of Redundant Nodes: A node u is redundant, if it lies in the middle of a straight street. Typically, such nodes have an *in-degree* = *out-degree* = 1. Note that the

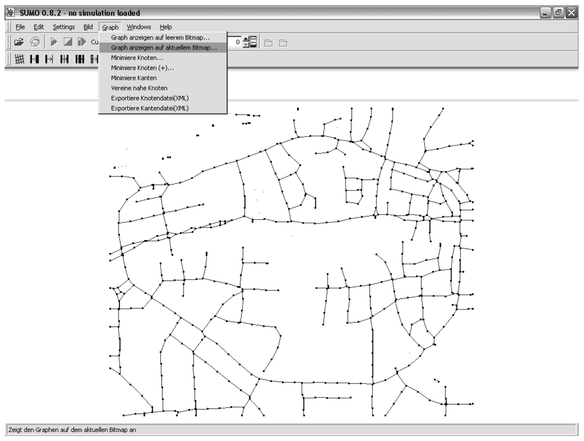

SUMO 0.8.2 - no simulation loaded
File Edit Settings Bild Graph Windows Help
Graph anzeigen auf leerem Bitmap...
Graph anzeigen auf aktuellem Bitmap...
Minimiere Knoten...
Minimiere Knoten (+)...
Minimiere Kanten
Vereine nahe Knoten
Exportiere Knotendatei(XML)
Exportiere Kantendatei(XML)
Zeigt den Graphen auf dem aktuellen Bitmap an

Fig. 4. The Street Graph.

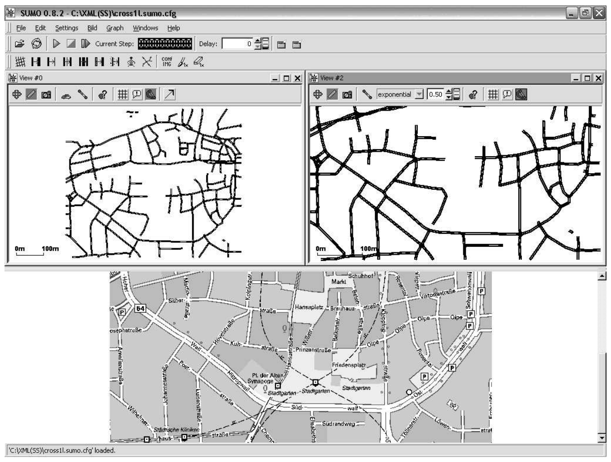

SUMO 0.8.2 - C:\XML(SS)\cross1l.sumo.cfg
Current Step:
Delay:
View #0
View #2
0m 100m
'C:\XML(SS)\cross1l.sumo.cfg' loaded.

Fig. 5. SUMO.

street must be straight, else we may lose the desired precision in road geometry. Let $u = (x_u, y_u)$, $v = (x_v, y_v)$, $w = (x_w, y_w)$ be the three consecutive nodes in the graph G. The simplification algorithm basically uses a collinearity test to find the redundant node. The node v is redundant if v lies on the line joining the nodes u and w. The collinearity test can be performed by testing if the value of the determinant

$$\begin{vmatrix} x_u & y_u & 1 \\ x_v & y_v & 1 \\ x_w & y_w & 1 \end{vmatrix}$$

is 0 or not.

The simplification algorithm described above can be adjusted to simplify a bit curved or wavy streets too. By introducing a tolerance parameter ϵ, only the nodes that lie within an ϵ-distance of the line joining the two neighboring nodes are removed. Increasing the value of ϵ can give lower number of nodes but a distortion in the road geometry as many curved streets would be approximated as straight ones.

The algorithm runs in $O(n^2)$ time and is actually a simplified form of Douglas-Peucker algorithm [4], which was developed to reduce the number of points to represent a digitized curve from maps and photographs. It considers a simple trace (certain forms of self-intersections can also be handled) of $n + 1$ points $\{p_0, \ldots, p_n\}$ in the plane that form a polygonal chain and asks for an approximating chain with fewer line segments. It is best described recursively: to approximate the chain from point p_i to p_j, the algorithm starts with segment $p_i p_j$. If the farthest vertex from this segment has a distance smaller than a given threshold ϵ, then the algorithm accepts this approximation. Otherwise, it splits the chain at this vertex and recursively approximate the two pieces. The $O(n \log n)$ algorithm [7] takes advantage of the fact that splitting vertices are to be located on the convex hull. It has latter been improved to $O(n \log^* n)$, where $\log^* n = \min\{k | \underbrace{\log \log \cdots \log}_{k \text{ times}} n = 1\}$.

2) Merging of Closely Situated Nodes: Another approach that can result in a simplified graph with much lesser nodes is to remove the nodes that lie very close to each other. Again, it can effect the smoothness of the streets, but when done with proper parameters the resulting graph can retain a good approximation of the road geometry and still consisting of lesser nodes. The simplification can be done by a linear time algorithm that runs in the time proportional to the number of edges. The algorithm iterates through the edge list. Let s and t denote the start and the end node of the edge e, respectively. For each edge $e = (s, t)$, we check if the Euclidean distance between s and t, $\|t - s\|_2 < \sigma$, where σ is the simplification parameter. If the distance is smaller, the node t is declared as a *close* node and is deleted.

V. Integration to a Traffic Simulator – SUMO

To test a utility of our approach in a different scenario, we decided to integrate our efforts with an existing traffic simulation system called SUMO [11]. SUMO stands for *S*imulation of *U*rban *M*obility. It is an open-source project developed and supported by DLR (German Aerospace Center). It has recently been used for traffic simulation and prognosis during the world soccer championship 2006. For real-time data collection during such an enormous event, a Zeppelin airship was also used. It provided live data for the traffic simulation in SUMO that made it possible to predict the traffic jams and to inform the city authorities to redirect the traffic in advance [8].

SUMO is a microscopic and multi-modal traffic simulation tool. The input to SUMO is a graph in XML format. It consists of three parts: the nodes, the edges, and a third data set that consists of the test routes in the street graph. These routes are to be used by the virtual vehicles during the traffic simulation.

For the parsing of XML data, an open source parser called XERCES is used. In the following, excerpts from the `Node.sumo.xml` and `Edge.sumo.xml` files are shown that contain the nodes and edges information, respectively:

```
<nodes>
  <node id="0" x="0.0" y="220.0 type="priority" \>
  <node id="1" x="61.0" y="234.0 type="priority" \>
  <node id="2" x="98.0" y="232.0" type="priority" \>
</nodes>
<edges>
 <edge id="0" fromnode="0" tonode="1" priority="78"
                          nolanes="1 speed=50.000 />
 <edge id="1" fromnode="0" tonode="2" priority="78"
                        nolanes="1 speed = 50.000 />
 <edge id="2" fromnode="0" tonode="2" priority="78"
                        nolanes="1 speed = 50.000 />
</edges>
```

A. *Post-processing of the SUMO Street Network*

After the vectorization and the generation of the street graph, the exported XML data can be post-processed for SUMO. Normally an edge means a street with only one lane and with allowed speed as 50 km per hour. In order to obtain a realistic trace, the street graph has to be extended to bring it more toward reality. But again, not all streets have more than one lane and different kinds of streets have different number of lanes. For this purpose, a user interface is developed and integrated in the SUMO environment that provides the facility to change the attributes of a street. To get this lane information, we have used satellite images of the city of Dortmund. In Europe, where most of the streets indeed consist of one lane for each direction, the process is fairly easier to be carried out by a human. Another post-processing step is to mark the road crossings that have a traffic signal installed.

B. *Traffic Routes*

As described above, the routes to be used by the virtual vehicles during the simulation are generated beforehand and saved as XML data. This routing information is not changeable during simulation time and must be provided along with the problem graph. A sub-program of SUMO helps in the generation of individual routes. The main input is the number of cars per time unit. A series of random routes are generated by picking the start and end nodes randomly and searching for the route between them. The generated routes are then used in SUMO for simulation.

VI. Conclusion

GPS based navigation and planning is becoming a *necessity* in the modern times, where urban mobility on large scale poses a genuine challenge for sustainable transportation. Navigation systems allow users to search for a *shortest* path, based on a given metric, let it be distance or travel time. However, their availability and utility is completely dependent on the underlying digital vector map.

Collaborative map generation process allows people to gather GPS traces and upload to a common web portal. The integration of these GPS traces to represent an actual road is an intensive process and poses difficult challenges to the community. Having a good base map can greatly help the integration process allowing a good filtration of GPS traces.

We have presented an approach to provide a vectorized and calibrated map extracted through raster maps. For the vectorization and graph generation process we generated a 0/1 pixel map of a topographic map by applying a variety of different digital image processing techniques. A skeletonizing algorithm then transform this processed bitmap into a road network skeleton. In order to construct the road graph, the skeleton is made *maximally sparse*. The street graph is generated by a tracking algorithm based on the *sweep-line* paradigm that processes the skeleton by iterating on the connected components.

The presented approach has been implemented in C++ and evaluated on raster maps collected by the city authorities of Dortmund, Germany. The implementation provides a flexible user interface to maintain and annotate the road network. The seamless integration of our approach to the microscopic traffic simulation tool SUMO results in a stand-alone tool for urban mobility planning in the areas where vector maps are scarce.

As future work, we plan to integrate the presented approach into our solution to the problem-1 namely the integration of different GPS traces presented in the beginning of this paper.

References

[1] H. Blum. *Models for the Perception of Speech and Visual Forms*. MIT Press, 1967.

[2] R. Bruntrup, S. Edelkamp, S. Jabbar, and B. Scholz. Incremental map generation with gps traces. In *International Conference on Intelligent Transportation Systems (ITSC)*, pages 574–579. IEEE Press, 2005.

[3] A. Dasen. AIS: A system for the Automatic Interpretation of Street maps. *in German)*. Master's thesis, Philosophische und Naturwissenschaftliche Universität zu Bern, 2001.

[4] D. H. Douglas and T. K. Peuker. Algorithms for the reduction of the number of points. *The Canadian Cartographer*, 10:112–122, 1973.

[5] S. Edelkamp, S. Jabbar, and T. Willhalm. Geometric travel planning. *IEEE Transactions on Intelligent Transportation Systems*, 6(1):5–16, 2005.

[6] A. Florio and F. Berrilli. A skeletonizing algorithm for granulation and supergranulation cell finding. In *Poster Proceedings of SOLE98 Workshop*, 1998.

[7] J. Hershberger and J. Snoeyink. An $O(n \log n)$ implementation of the Douglas-Peuker algorithm for line simplification. *ACM Computational Geometry*, pages 383–384, 1994.

[8] D. Krajzewicz, M. Bonert, and P. Wagner. The open source traffic simulation package SUMO. In *RoboCup 2006 Infrastructure Simulation Competition*, 2006.

[9] D. Krajzewicz, M. Hartinger, G. Hertkorn, P. Mieth, C. Rössel, J. Zimmer, and P. Wagner. Using the road traffic simulation (SUMO) for educational purposes. In *Traffic and Granular Flow (TGF)*, 2003.

[10] D. Krajzewicz, G. Hertkorn, C. Rössel, and P. Wagner. An example of microscopic car models validation using the open source traffic simulation SUMO. In *Proceedings of Simulation in Industry, 14th European Simulation Symposium*, pages 318–322. SCS European Publishing House, 2002.

[11] D. Krajzewicz, G. Hertkorn, C. Rössel, and P. Wagner. SUMO (Simulation of Urban Mobility); an open-source traffic simulation. In *Proceedings of the 4th Middle East Symposium on Simulation and Modelling (MESM2002)*, pages 183–187. SCS European Publishing House, 2002.

[12] L. Lam, S. Lee, and C. Y. Suen. Thinning methodologies-a comprehensive survey. *IEEE Trans. Pattern Anal. Mach. Intell.*, 14(9):869–885, 1992.

[13] H. Mayer, I. Laptev, A. Baumgartner, and C. Steger. Automatic road extraction based on multi-scale modeling, context, and snakes. In *International Archives of Photogrammetry and Remote Sensing*, volume XXXII, Part 3–2W3, pages 106–113, 1997.

[14] B. Schulz. Automatic inference of streetmaps based on gps traces. Master's thesis, Universität Dortmund, Germany, 2006. in German.