International Journal of Spatial Data Infrastructures Research, 2009. Vol. 4, 239-264.

A Note on Hierarchical Routing Algorithms based on Traverse-oriented Road Networks[*]

Clemens Strauss

Graz University of Technology, Institute of Geoinformation
Email: clemens.strauss@tugraz.at

Abstract
Finding the shortest route by means of electronic navigation aids has become almost an omnipresent task in our society. This task runs across the internet, for outdoor activities and of course within car navigation systems. Nowadays search for the shortest path gets more popular compared to the past because of rising fuel prices. That is why driving along an optimised route has financial benefits on the one hand, and on the other hand it helps to save energy resources of our environment. In this paper a hierarchical approach of route computation based on traverse elements is discussed. Furthermore an experimental setup is implemented based on an open source database management system (PostgreSQL/PostGIS) to verify the theoretical approach presented.

Keywords: Road Network Data Structure, Traverse Element, Dual Graph, Turn Restrictions, Shortest Path, Bidirectional Routing, Hierarchical Routing, Multi Level Hopping.

1. INTRODUCTION

A careful co-existence with our environment should be a timeless attitude of society and not only a media-driven temporary fashion. Recently the sensitivity for environmental topics has increased due to a rising occurrence of natural hazards and price increases of food and fuel.

That is why technologies and methods for a more efficient use of fossil energy sources and the development of new alternative and natural energy sources should get enhanced in any area of life. That applies to the transportation branch as well. Here, an efficient use of the available capacities of transportation is not

[*] This work is licensed under the Creative Commons Attribution-Non commercial Works 3.0 License. To view a copy of this license, visit http://creativecommons.org/licenses/by-nc-nd/3.0/ or send a letter to Creative Commons, 543 Howard Street, 5th Floor, San Francisco, California, 94105, USA.

DOI: 10.2902/1725-0463.2009.04.art13

only of economic but also of ecologic interest. Hence, the optimisation of routes has a positive impact on our environment.

This paper reviews existing concepts of route computation and presents an innovative extension for a hierarchically structured road network (*Multi Level Hopping*). The approach presented is based on the well known Dijkstra's label setting algorithm applied to a road graph composed of arc and traverse elements. A traverse element can be seen as an element that topologically connects two arc elements.

To evaluate the theoretical work we used. an open source database management system (PostgreSQL/PostGIS) for data storage including additional functions written in pl/pgSQL for route computation The interaction task of requesting a route computation and of receiving the routing result is realised by the script language PHP for a future web application. Finally, a UMN Mapserver is used for visualising the road network focused on the route computed.

The result of this work package provides the foundation for further investigation concerning the application of GIS methods to improve route calculation tasks. Among other things a digital elevation model is designed, including a digital slope model, and a selection of network elements based on their spatial characteristics, to improve the route computation capability.

2. NODE-BASED ROUTING VERSUS ARC-BASED ROUTING

In this section the specification of two different network description approaches including their topological connectivity are introduced. This theoretical part establishes the background necessary for a better understanding of the routing methods based on traverse-oriented road networks.

2.1. Common Basics

The principle of routing algorithms comparable to Dijkstra's label setting algorithm published in Dijkstra, 1959 is based on the investigation of the neighbourhood of a basic network element $\beta \in B\{\beta_1,\beta_2,...,\beta_n\}$. The relationship to a neighbouring basic network element is represented by an adjacent network element $\alpha = (\beta_i \beta_j) \in A\{\alpha_1,\alpha_2,...,\alpha_n\}$. Using these two types of elements a road graph $G = G(B,A)$ can be defined.

The search tree spreads from a source element β_S in search of an optimal route $route = \{\beta_S,\beta_{S+1},...,\beta_{D-1},\beta_D\}$ to a destination element β_D. During the search process basic network elements β_i are labelled according to three different states: unvisited, temporarily labelled and permanently labelled. The set of unvisited elements $U = \{\beta_{U_1}, \beta_{U_2}, ..., \beta_{U_n}\} \subseteq B$ contains all elements that have not been reached by the search tree so far. Elements in the set of temporarily labelled elements $T = \{\beta_{T_1}, \beta_{T_2}, ..., \beta_{T_n}\} \subseteq B$ have been visited at least once, but the optimal route to these elements has not been found yet. The last set $P = \{\beta_{P_1}, \beta_{P_2}, ..., \beta_{P_n}\} \subseteq B$ includes all permanently labelled elements. In this case the optimal route has already been found.

An optimisation process like finding the optimal route from a source to a destination requires the definition of a value to be optimized, the optimization criterion. In general there are two approaches: negative valuations – the higher the value the worse the route – and positive valuation – the higher the value the better the route (cp. Hofmann-Wellenhof et al.,2003).

Exemplary definitions of negative valuation are the geometrical length of an element or the cost that arises from using this element. These cost values can be associated with the basic network element c_β and with the adjacent network element c_α as well. The basic network graph definition can be upgraded to $G = G(B, A, c_B, c_A)$.

The optimisation criterion has to be designed depending on the valuation of the road graph. In the case of negative valuation the optimal route is characterised by the smallest sum of all cost values along the route: $min = \sum_{route}(c_\beta + c_\alpha)$.

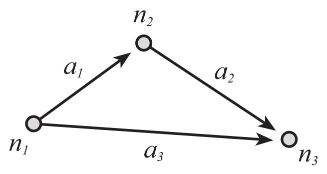
The spreading of the search tree proceeds until a predefined termination criterion is fulfilled. One practicable criterion is described by $\beta_D \in P$ wherein the destination element is an element of the set of permanently labelled basic network elements.

Another possible termination criterion states that all basic network elements are elements of the set of permanently labelled basic network elements: $B = P$. Hence, the search tree has investigated the whole road graph. Meeting this criterion in the case of large road graphs can be very time intense.

2.2. Characteristics of Node-based Algorithms

In the case of node-based algorithms, reviewed by Fu et al.,2006, the basic network element is represented by a node $n \in V\{n_1, n_2, ..., n_n\}$. The relationships between two nodes is established by means of an arc $a = (n_i n_j) \in E\{a_1, a_2, ..., a_n\}$, which is used as adjacent network element. From this follows that the road graph can be defined as $G = G(V, E)$. Compared to a road network these elements are equivalent to junctions (n) and roads (a). As far as the optimisation process is concerned an extension for road elements is necessary. To achieve this, costs of node elements c_n and costs of arc elements c_a have to be introduced. This leads to the road graph $G = G(V, E, c_V, c_E)$. The following example shows a small configuration of three nodes and three arcs used for a node-based approach. Figure 1 shows a sketch of the elements that are listed in tables below the figure.

Figure 1: Network elements of a node-based approach.



In table 1 the basic network elements are given. The record set was enlarged relating to the cost values. Additionally a column, containing spatial information for further visualisation tasks respectively route guidance applications, is added.

Table 1: Record set for basic network elements (nodes) related to figure 1.

Id	Cost	Geometry
n_1	c_{n_1}	$point_{n_1}$
n_2	c_{n_2}	$point_{n_2}$
n_3	c_{n_3}	$point_{n_3}$

The second table (table 2) related to the figure above contains adjacent network elements. The neighbourhood relationship is described by a predecessor element and a successor element for each arc element. As in table 1, cost information is given too.

Table 2: Record set for adjacent network elements (arcs) related to figure 1.

Id	Predecessor	Successor	Cost
a_1	n_1	n_2	c_{a_1}
a_2	n_2	n_3	c_{a_2}
a_3	n_1	n_3	c_{a_3}

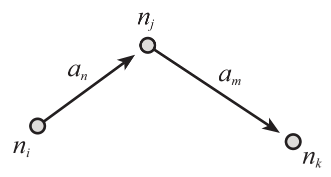
The optimisation criterion can be defined from the valuation selected as $min = \sum_{route} (c_n + c_a)$. For a lot of applications a valuation of arc elements by means of arc length or costs (e.g. fuel consumption or utilisation fee) is sufficient.

This approach of modelling a road network by using junctions as basic network element (n) and roads as adjacent network element (a) is sufficient for simple routing applications as long as no complex junction situations have to be modelled – for example turning restrictions or different turning costs (e.g. to turn left is more expensive than to turn right). To support such complexities using Dijkstra's label setting algorithm a topological adjustment of the road graph is necessary. Adding extra nodes or arcs can be summed as topological adjustment. An extension of a junction model from a simple node to a multi node complex, containing junction entry nodes and junction exit nodes, allows for restrictions and different costs of turns to be modelled. These approaches are described in detail in the works of Winter, 2002 and Gutiérrez et al., 2008.

2.3. The Traverse Element

The traverse element defines the topological relationship between three basic network elements applied to the node-based approach (cp. Caldwell, 1961). The traverse $t = (n_i n_j n_k) \in T\{t_1, t_2, ..., t_n\}$ consists of: a from-node n_i, an over-node n_j and a to-node n_k (figure 2). Among others Bartelme, 1991, Wieser, 1991 and Hofmann-Wellenhof et al., 2003 use this traverse element to model turning restrictions and different turning costs. For each permitted turn a traverse is defined. Hence, a forbidden turn is not listed in the traverse data set. Turning costs are implemented as an additional attribute of a traverse record. To consider this extra network element in the node-based Dijkstra's label setting algorithm is not trivial.

Figure 2: Configuration of a traverse element.



The definition of a traverse element based on arcs slightly differs from the node-based approach. An arc $a = (n_i n_j)$ represents the relationship between two nodes. Adopting this property to the definition of a traverse for node-based algorithm, the traverse can be redefined as $t = (a_n a_m) \in T\{t_1, t_2, ..., t_n\}$ where $a_n = (n_i n_j)$, $a_m = (n_j n_k)$. The consequences of this traverse definition are specified in subsection 2.4.

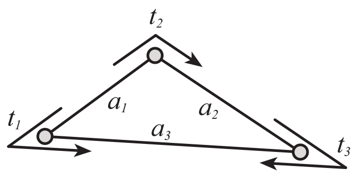
2.4. Characteristics of Arc-based Algorithms

Contrary to a node-based algorithm an arc-based method uses arcs as the basic network elements. Adjacencies are modelled by traverse elements that are defined during the process of road graph. This concept is identified as pseudo-dual graph approach by Winter 2002. The resulting road graph is given by $G = G(E,T)$.

There is no further need for node elements according to the computation of the search tree. This fact implies a routing from a source arc to a destination arc.

In figure 3 a situation corresponding to one shown in figure 1 is modelled using an arc-oriented approach.

Figure 3: Network elements of an arc-oriented approach.



The subsequent tables are filled based on the figure above. Table 3 contains information about the basic network elements. In case of an arc-based approach the geometry column of the basic network element table holds line elements instead of point elements used by the node-based approach. No other changes of the table are required.

Table 3: Record set for basic network elements (arcs) related to figure 3.

Id	Cost	Geometry
a_1	c_{a_1}	$linestring_{a_1}$
a_2	c_{a_2}	$linestring_{a_2}$
a_3	c_{a_3}	$linestring_{a_3}$

The table of adjacent network elements (see table 4) does not require any changes.

Table 4: Record set for adjacent network elements (traverses) related to figure 3.

Id	Predecessor	Successor	Cost
t_1	a_1	a_3	c_{t_1}
t_2	a_1	a_2	c_{t_2}
t_3	a_2	a_3	c_{t_3}

The validation of the road graph can be achieved by implementing costs for arc elements c_a and traverse elements c_t .Hence, the optimisation criterion for the valuated graph $G = G(E, T, c_E, c_T)$ can be formulated as $min = \sum_{route} (c_a + c_t)$. In this case an application of both cost values for a routing algorithm is advisable. The optimisation process is not limited to consider costs of arc (as with the node based approach) but is capable of including costs of turns in a junction environment as well.

This approach of handling restrictions and different costs of turns is advantageous for two reasons: Firstly, Dijkstra's label setting algorithm can be used unchanged. Secondly, the amount of manipulation processes and data organisation work to generate a graph of existing arc and traverse elements is less than in the case of a node-based road graph whose topological manipulation tasks have been explained in the subsection on the characteristics of node-based algorithms.

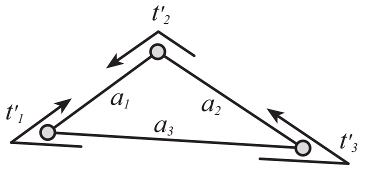
3. BIDIRECTIONAL ROUTING

The concept of a bidirectional approach is based on two search trees (cp. Fu et al.,2006 or Hofmann-Wellenhof et al.,2003). The first search tree originates from the basic network element β_S^S – source element of the entire route. It contains temporarily labelled elements $T^S = \{\beta^S_{T_1}, \beta^S_{T_2}, ..., \beta^S_{T_n}\}$ and permanently labelled elements $P^S = \{\beta^S_{P_1}, \beta^S_{P_2}, ..., \beta^S_{P_n}\}$. The second search tree starts to spread from the basic network element β^D_S which is the destination element of the entire route. This search tree again is defined by temporarily labelled elements $T^D = \{\beta^D_{T_1}, \beta^D_{T_2}, ..., \beta^D_{T_n}\}$ and permanently labelled elements $P^D = \{\beta^D_{P_1}, \beta^D_{P_2}, ..., \beta^D_{P_n}\}$.

The primary motivation for bidirectional routing algorithms is to minimise the number of calculation steps and thus save time in computing the route. This is explained by a smaller cardinal number of P^S and P^D contrary to the cardinal number of P arising from a monodirectional approach. Furthermore this ability of bidirectional routing is a prerequisite for hierarchical routing techniques.

When considering bidirectional approaches the need for directed adjacencies of basic network elements has to be kept in mind. For example there is only one permitted direction to pass one-way streets. Hence, the search tree with its root in β^S_S is allowed to go through this street in original direction. On the other hand, this move is forbidden for the search when routing starting with the target element β^D_S. From this follows that a new graph G' is needed in which the search tree can spread, with G' being the inverse graph of G. This graph G' is generated by switching the neighbourhood relation of adjacent network elements: Adapting the definition from $\alpha = (\beta_i \beta_j) \in A\{\alpha_1, \alpha_2, ..., \alpha_n\}$ to $\alpha' = (\beta_j \beta_i) \in A'\{\alpha'_1, \alpha'_2, ..., \alpha'_n\}$. This modification method can be used likewise for node-based and arc-based algorithms. The inverse graph approach for bidirectional routing applied to the arc-based road graph example mentioned above is demonstrated in figure 4. Compared with figure 3 the direction of the traverse elements t_i has switched.

Figure 4: Network elements of a bidirectional arc-oriented approach (inverse graph).



The same effect – switching directions of traverse elements – can be seen in table 5. The predecessor elements have changed their position with the successor elements.

Table 5: Record set for adjacent network elements (traverses) of the inverse graph related to figure 4.

Id	Predecessor	Successor	Cost
t'_1	a_3	a_1	c_{t_1}
t'_2	a_2	a_1	c_{t_2}
t'_3	a_3	a_2	c_{t_3}

Moreover the termination criterion for the search tree enlargement requires modification in comparison to the monodirectional case. In the case of a bidirectional approach, the algorithm terminates when both search trees overlap for the first time. This means that there is a basic network element β_i which belongs simultaneously to the search tree with its root in the source element P^S and to the search tree with its root in the destination element P^D. Hence, the termination criterion is given by $\beta_i \in P^S \wedge \beta_i \in P^D$, where the operating mode is equal to the termination criterion described in Nicholson, 1966.

Figure 5 shows an experimental setup using an arc-based road graph simulating a spatial limited Manhattan metric. All arcs and traverses were assigned equal weights. The source of the computed route is arc no. 230 and the destination arc no. 300. The result of the monodirectional routing algorithm contains a set of permanently labelled arcs P represented by black solid lines. Unvisited arcs and temporarily labelled arcs are visualised as thin grey lines. Additionally the calculated route is plotted as thick dashed black line. Because of the equal valuation of network elements the illustrated route is one of several possible solutions.

Figure 5: Monodirectional search tree from source arc no. 230 to destination arc 300; permanently labelled basic network elements only.

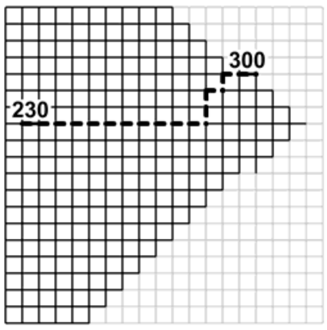
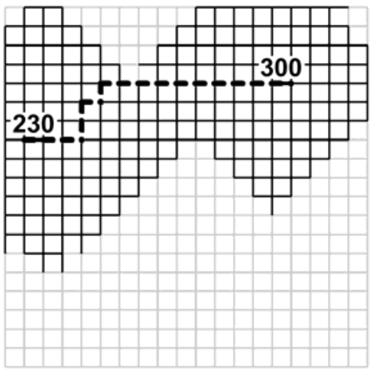


Figure 6 displays the result of a route computation with bidirectional algorithm. The underlying dataset conforms to the one used for the monodirectional route computation. In addition, an algorithmic enhancement had to be made for the handling of the adjacent inverse graph elements (traverses).

Contrary to the monodirectional solution two search trees are clearly visible: one with the centre in arc no. 230 (P^S) and the other one with its centre in arc no. 300 (P^D).

Comparing both solutions – monodirectional versus bidirectional – the cardinal numbers of the sets of permanently labelled elements differ: $|P| = 925$ respectively $|P^S| + |P^D| - 1 = 662$. The value -1 in the count compensates the twofold labelling of the arc that triggers the termination criterion.

Figure 6: Bidirectional search tree from source arc no. 230 to destination arc no. 300; permanently labelled basic network elements only.



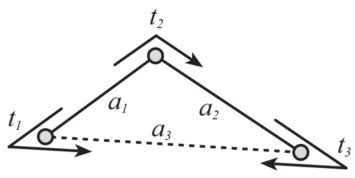
4. HIERARCHICAL ROUTING

Minimising the amount of calculation steps is the primary motivation for hierarchical routing. Furthermore the realisation of special traffic rules applied to different hierarchical levels (e.g. rural road, motorway, etcetera for different kinds of road users can be modelled by these approaches. Hierarchical structures are not limited to representing different levels for a single means of transport, but are capable of modelling complex multimodal traffic or transportation structures (e.g. switching from pedestrian mode to bus, train, aircraft, …). In this case hierarchical routing is used for multi modal transportation applications. Methods and approaches are discussed comprehensively in the literature amongst others by Car et al., 2001, Jagadeesh et al. 2002 and Fu et al. 2006.

4.1. Requirements

The appliance of a hierarchical routing algorithm implies a road graph including different hierarchical levels. A classification of basic network elements is sufficient – adjacent network elements need not be classified in the case of an arc-based approach. A visual example of a hierarchically structured road graph is shown in figure 7 where two basic network elements (a_1 and a_2) which are assigned to a level l_1 are drawn as solid black lines and one element (a_3), assigned to a level l_2, is drawn as dashed black line.

Figure 7: Hierarchically structured network elements of an arc-oriented approach.



The introduction of levels however requires the modification of all basic network elements. The information about the associated level of an element is stored in an additional attribute in this record set – cp. Table 6.

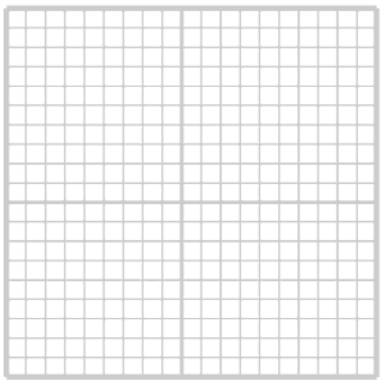
Table 6: Record set for hierarchically structured basic network elements (arcs) related to figure 7.

Id	Cost	Level	Geometry
a_1	c_{a_1}	l_1	$linestring_{a_1}$
a_2	c_{a_2}	l_1	$linestring_{a_2}$
a_3	c_{a_3}	l_2	$linestring_{a_3}$

A modification of the experimental setup was done to reflect the different hierarchical levels. Figure 8 demonstrates the result of this modification. Arcs which belong to the lowest level $l = 1$ are drawn as thin grey lines. The intermediate network level $l = 2$ is plotted as grey line of medium line width – its shape is similar to a cross. The highest level $l = 3$ is marked as thick grey line and surrounds all other arcs forming a window.

All successive computations and examples in this section are based on this specific experimental data set.

Figure 8: Experimental data set (hierarchically structured Manhattan metric).



4.2. Level Upward Part

The primary task of routing in a hierarchical structured network is to find a route using the advantages of higher levels, e.g. lower cost and/or faster movement. Hence, the route should include the highest reachable level if necessary. The initial point of the concept of the level upward part is similar to the Hierarchical

Wayfinding Algorithm (HWA) specified in Car et al., 2001, but it has to be seen against the background of traverse-oriented road network data. This fact is important for the level downward part, described in subsection 4.4.

Based on the method of bidirectional routing the process of search tree spreading is divided into two tasks:

1. Synchronous search tree spreading:
   While the search frontiers of both trees are located in the same level both search trees are spread: $max\left(l_{\beta \in P^S}\right) = max\left(l_{\beta \in P^D}\right)$. Spreading is limited to basic network elements that have the current level value or a higher one. This task continues until fulfilment of the termination criterion is fulfilled, terminating the whole process. If only one of the search frontiers has reached a higher level, the search continues with task 2.

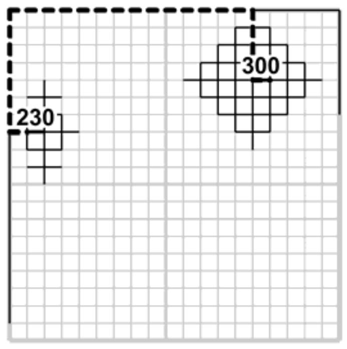
2. Asynchronous search tree spreading:

   As long as the levels of the search frontiers of both trees differ, an asynchronous search tree spreading is necessary: $max\left(l_{\beta \in P^S}\right) \neq max\left(l_{\beta \in P^D}\right)$. In this case only the search tree located in the lower level grows based on basic network elements that have the current level value or higher. Asynchronous search tree spreading continues while the termination criterion is not fulfilled. If the search frontier has reached the hierarchical level that is equal to the (higher) level of the opponent search tree level the routing proceeds with synchronous search tree spreading (task 1).

A beneficial side effect of this technique is the fact that if one search tree has reached a higher level, the computed route would not necessarily use this higher level.

An example of hierarchical routing is illustrated in figure 9 where the shortest route starting in arc no. 230 and ending up in arc no. 300 was calculated considering the hierarchical structure of the road graph. The influence of the arcs of different levels is clearly visible and is in accordance with human expectation.

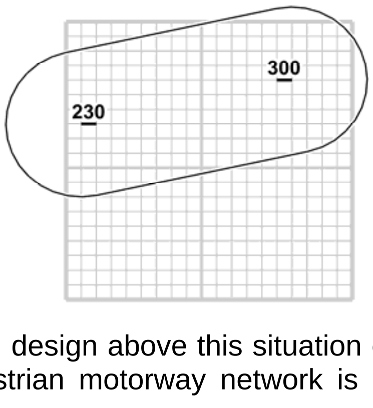
Figure 9: Search tree based on a bidirectional and hierarchical approach from source arc no. 230 to destination arc no. 300; permanently labelled basic network elements only.



4.3. Problems concerning unconnected Road Graphs in Higher Levels

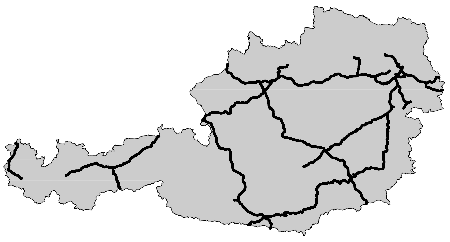
The fact that a road graph is used for routing processes disregarding connectivity of basic network elements of higher levels is a challenge to the capability of routing techniques (see Fu et al., 2006: p 3338). Figure 10 depicts such a situation. The experimental dataset is overlaid with a buffer – drawn as a thin black line with oval shape –which limits the road graph to its interior space. The shape of this buffer is determined by two parameters: a straight line connecting arc no. 230 and arc no. 300 used as centre line and a buffer size that prohibits a continuous connection of the highest level but at the same time maintains the connectivity in the medium level. Hence, a route connecting arc no. 230 with arc no. 300 cannot be computed using the highest possible level comparing to figure 9. From this it follows that an extension of the algorithm concerning a level downward move is indispensable.

Figure 10: Limitation of the road graph by a buffer; no connected graph within the highest level.



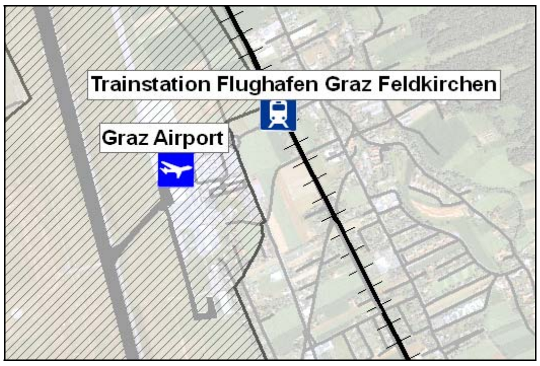
Irrespective of the artificial design above this situation can be found in real world too. For example the Austrian motorway network is separated by topographic barriers into a western part and an eastern part (. Figure 11). The connection of these two parts on the same hierarchical level is realised by leaving the national territory and using foreign road networks.

Figure 11: Austrian motorway network. (data source: Tele Atlas)



Furthermore the transfer in the course of using public means of transport can imply these difficulties as well. For example a short footpath connecting the train station with the airport (Figure 12).

Figure12: Footpath connecting railway station Graz-Feldkirchen with Graz airport. (data source: IKONOS, Tele Atlas)



4.4. Level Downward Part

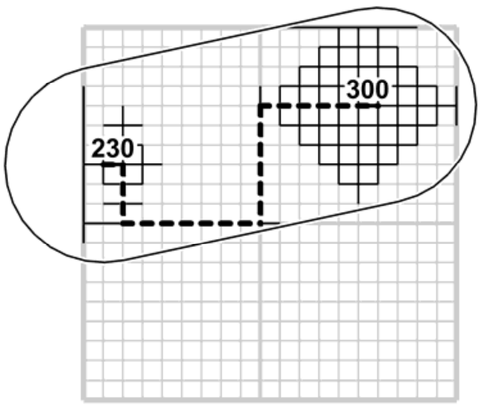
During the hierarchical search tree spreading process on a road graph described beforehand a situation occurs where all arc elements of the higher level hold a permanent label but the termination criterion is not fulfilled. The search process has to move down a level in the hierarchical structure in order to continue to spread the search tree.

Therefore the upward level part described previously has to be modified. The search tree spreading process has to take care of so called **level exit arcs**. These arcs belong to a lower level and receive temporary labels. While the search tree spreads they are excluded from the list of possible candidates for permanent labelling because only arcs of the same level or higher level of the current search tree are considered.

If the need for a downward move arises, the constraint of possible candidate arcs, which have to be of the same or higher level and temporarily labelled, is relaxed in order to include temporarily labelled arcs of the lower level. This is where the level exit arcs come into play. This key feature of upward and downward movement of the search tree can be termed as **multi level hopping**.

Figure 13 illustrates the result of a route computation based on a hierarchical data set with unconnected elements within the highest level. The route starting in arc no. 230 and ending in arc no. 300 uses network elements of lowest and intermediate level. It can be seen that the search tree has reached arc elements of highest level but these elements can not be included into the solution because they are not connected.

Figure 13: Search tree based on a multi level hopping approach applied to a limited road graph from source arc no. 230 to destination arc no. 300; permanently labelled basic network elements only.

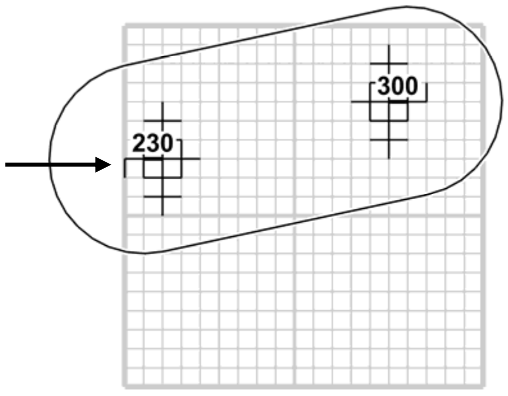


The search tree spreading required to obtain the result shown in figure 13 can be described as follows (in chronological order):

1. The initial situation is described by a source element and a destination element which are within the same hierarchical level – in this case the 1st level (compare figure 10). This causes a synchronous search tree spreading.

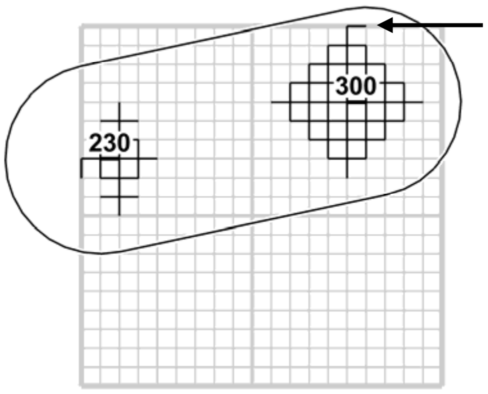
2. The frontier of the source tree has reached the 3rd level and causes the destination tree to spread in asynchronous fashion (figure 14).

Figure 14: event 2 – source tree has reached the 3rd level.



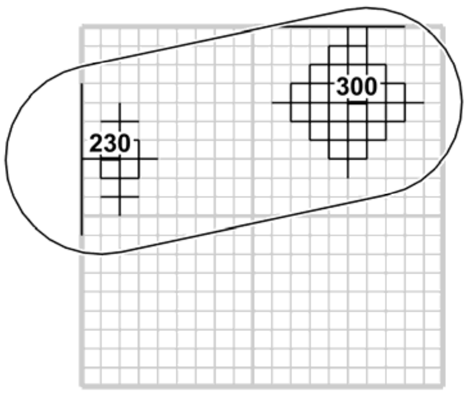
3. The search frontier of the destination tree has reached the 3rd (highest) level. Now both search trees are within the same level and a synchronous spreading is executed. (figure 15)

Figure 15: Event 3 – destination tree has reached the 3rd level.



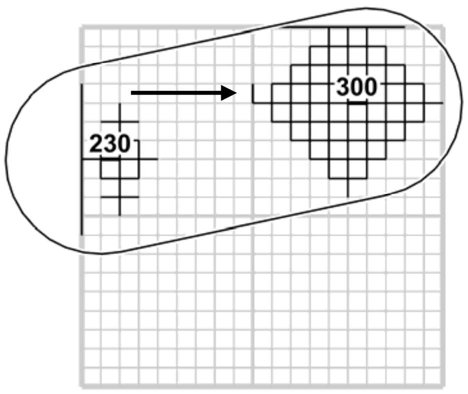
4. Once the destination tree has reached the limits described by the buffer it moves down to the 1st level. This entails an asynchronous spreading of the destination tree. (figure 16)

Figure 16: Event 4 – destination tree moves back to the 1st level.



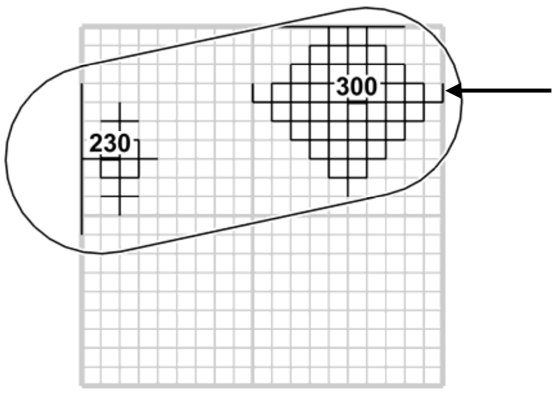
5. The destination tree has reached the 2nd level. Because it is still in a lower level than the source search tree, the asynchronous enlargement continues. (figure 17)

Figure 17: Event 5 – destination tree has reached the 2nd level.



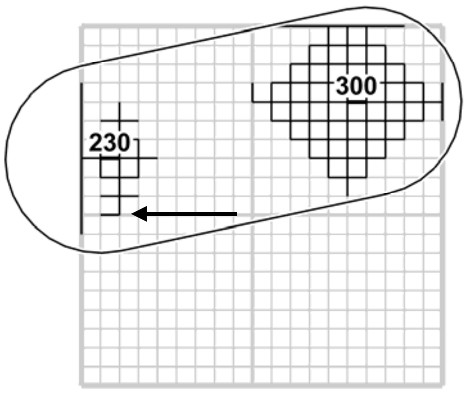
6. The search frontier of the destination tree has reached the 3rd level and is equal to the source tree now as far as its level of hierarchy is concerned. Therefore synchronous spreading can be performed. Explanatory note: The arc on the right side of the search tree (marked by an arrow) was labelled temporarily before. (figure 18)

Figure 18: Event 6 – destination tree has reached the 3rd level.



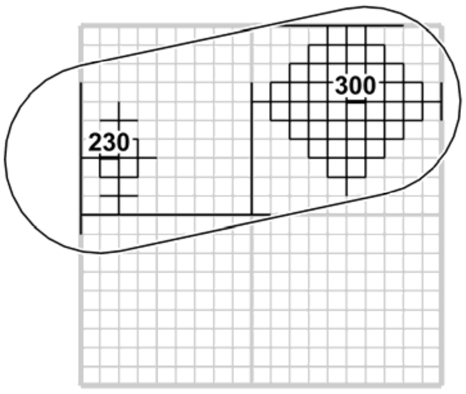
7. The source tree has reached its limit within level 3 and moves down to the 2nd level. An asynchronous spreading of the source tree is the consequence of this downward move. (figure 19)

Figure 19: Event 7 – source tree moves back to the 2nd level.



8. The search frontier of the source tree has reached one permanently labelled element of the destination search tree (figure 20). This event triggers the termination criterion and the computation of the optimal route has been finished. The final route computation result has been already displayed in figure 13.

Figure 20: Event 8 – termination criterion fulfilled.



5. IMPLEMENTATION AND EXPERIMENTAL SETUP

This section describes the practical implementation of the theoretical approach explained in the previous section. The aim was to create a fully functional library for a database management system concerning a hierarchical and bidirectional routing algorithm based on Dijkstra's label setting method.

5.1. Technologies Employed

The technologies employed are classified according to three main tasks: data storage with spatial capabilities including the route calculation, communication for requesting a route computation and the reception of the computation result, and finally a visualisation of this result.

5.1.1. Database Management System

An object-relational database was used for this specific application. More specifically an open source database management system (DBMS) – postgreSQL – was chosen for this task. The necessity to process tables containing basic network elements led to the requirement of upgrading this DBMS to PostGIS capability.

The great advantage of this DBMS can be seen in the diversity of spatial functions that are provided by default. This includes calculation of the length of a line element, generating buffers around geometries and topological analyses to name only a few.

This spatial DBMS stores the basic network elements (arcs) including cost values, level information and geometry data as well as adjacent network elements (traverses) with their cost values. Furthermore the algorithm itself was implemented on DBMS level using pl/pgSQL to create diverse functions. Some of these functions are provided by default while others are costume-designed. Both are callable by ordinary SQL statements.

5.1.2. PHP Script

Service oriented approach using PHP scripts facilitates interaction with the DBMS including execution call for route computation. Special postgreSQL functions called by PHP are employed to perform this task. One of these functions executes SQL statements on a chosen database, for example the routing application implemented in pl/pgSQL. As a result of this routing request the database returns a sequence of network elements that represents the computed route.

The return values are primarily formatted as text strings according to the Open Geospatial Consortium (OGC) Well Known Text (WKT) specifications. This WKT strings can be converted to a custom-designed XML structure for further purposes. A visualisation of this WKT can be identified as another task, described consecutively.

5.1.3. Mapserver

A mapserver generates a map-image based on various spatial data sources in consideration of pre-defined instructions. Spatial data sources are vector datasets (e.g. shape files), raster datasets (e.g. tiff images) or additional spatial information stored in geo databases (e.g. PostGIS). The instructions necessary to draw a map based on these datasets are stored in a mapfile, which sets the data sources, the order of appearance and the visual parameters

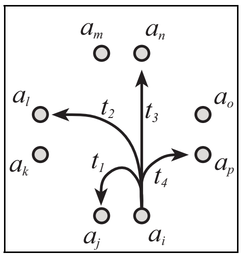
The figures with search trees presented earlier – e.g. subsection 4.4 – are created using a mapserver. For this special purpose an open source UMN mapserver was used (. Fischer, 2003).

5.2. Road Graph for Experimental Tests

An artificial, simplified road graph was created for testing the conceptual methods, the hierarchical node-based approach. The road graph forms a Manhattan metric road network.

Hence, a regular grid was generated modelling each edge with two arcs pointing in opposing directions. Uniform cost values were assigned to each arc. Additionally special arcs were added to represent higher levels (see section 4). This set of arcs represents the basic network elements. In figure 21 basic network elements are marked as $\{a_i,...,a_p\}$. The arc elements are displayed as points in a pseudo dual graph diagram to improve perceptibility.

Figure 21: Basic configuration of the experimental data set shown as pseudo dual graph diagram.



The neighbourhood relations of arc elements are modelled by traverses and valuated equally. Hence, a left turn is currently as expensive as a right turn. Only traverse elements $\{t_1,...,t_4\}$ which have the arc a_i as their predecessor are shown in figure 21. These four traverses represent an U-turn t_1, a left turn t_2, a move in straight direction t_3 and a right turn t_4. For each arc element within the road graph these four traverses where generated except the boundary elements on the left/right and upper/lower sides of the experimental data set.

From this follows that the experimental data set has 1520 arc elements and 5848 traverse elements.

5.3. Multi Level Hopping Algorithm

The characteristics of the multi level hopping algorithm are explained step by step below as presented in figure22 (the numbers in the figure refer to the step numbers below). The algorithm has been implemented using the language pl/pgSQL for a postgreSQL/PostGIS database environment.

Step 1 Initialisation:
Reception of input parameter values like source and destination elements for route computation and marking of these two elements as temporarily labelled elements in the arc record set. Locating the hierarchical level of source and destination elements. This part is required for a-/synchronous search tree enlargement.

Step 2 Source tree / asynchronism check:
If source tree and destination tree currently have the same level or if the source tree is operating on a lower level than the destination

tree only the source tree will be spread. Otherwise the spreading of the source tree will be skipped in this loop.

Step 3 Spreading of the source tree:
Only temporarily labelled basic network elements are used for the spreading process of the source tree. The element holding the lowest label value within the actual level or higher is transferred to the set of permanently labelled elements and its neighbourhood is investigated. When spreading the search tree an unvisited basic network element is transferred to the set of temporarily labelled elements. If the network element already belongs to the set of temporarily labelled elements the label is updated only. For these two steps the associated level is not considered –see. characteristics of level exit arcs.

Step 4 Detection of the maximum level value of the current search frontier:
This task is necessary for the level upward move. If a basic network element was added to the set of permanently labelled elements holding a higher level value than the actual one, the search tree would reach a higher level.

Step 5 Check for necessity to move down a level:
Contrary to step 4 dealing with upward moves, the need of a downward move is checked. If no temporarily labelled element of the current level or higher was found, the current level value would needs to be readjusted. The level value of temporarily labelled elements – that is the highest one within the set of the lower ones – is set as new current level value.

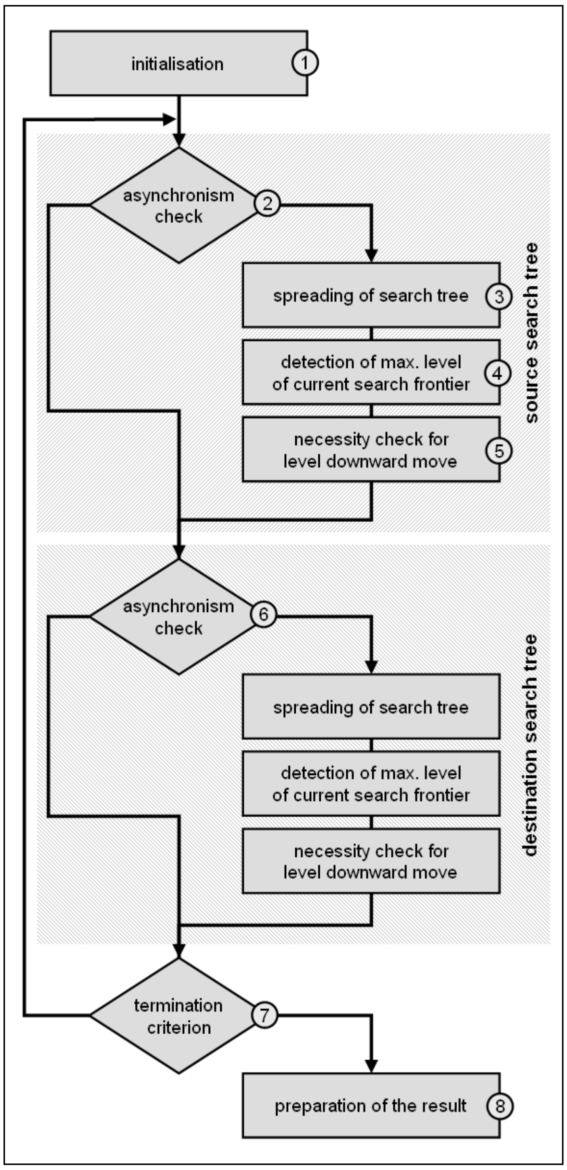
Step 6 Destination tree / asynchronism check:
This task is equivalent to step 2, in this case executed for the destination tree. Here the current level of the destination tree is compared to the source tree's level. If the current level is equal or lower than the one of the source tree, the same tasks (step 3, step 4 and step 5) are applied to the destination tree.

Step 7 Termination criterion:
The process is terminated if two different situations apply. Firstly, if the two search trees overlap the route computation can be considered as successfully terminated. Secondly, both search trees have spread to their maximum, but an overlapping could not be achieved. In this case the execution of the algorithm is aborted with the additional information that this event was caused by non-

connectivity of the network. If neither the termination criterion is met the algorithm will go back to step 2.

Step 8 Preparation of the result:
The computed route is saved as WKT. The WKT can easily be formatted depending on the user's output requirements.

Figure 22: Flowchart of multi hopping algorithm


initialisation
1
asynchronism check
2
spreading of search tree
3
detection of max. level of current search frontier
4
necessity check for level downward move
5
source search tree
asynchronism check
6
spreading of search tree
detection of max. level of current search frontier
necessity check for level downward move
destination search tree
termination criterion
7
preparation of the result
8

6. CONCLUSION AND FUTURE PERSPECTIVES

This paper is based on a simple method of route computation which has been upgraded in order to be able to model restrictions and different costs of turns within junction environment by means of traverse network elements in the sense of a pseudo dual graph approach. The advantages of such a method primarily are twofold: Firstly, Dijkstra’s label setting algorithm can be used unchanged. Secondly, an arc-based approach allows a simple modelling of complex junction situations.

Furthermore an extension for route computation based on hierarchical structured road networks – the so called multi level hopping – solves the problem posed by unconnected graphs in higher levels of hierarchical road network. The examples of the Austrian motorway network and additionally by multi modal transportation applications underline the need for a multi level hopping capability.

A combination of traditional route computation with technologies and methods derived from GIS is a promising road for a multi modal and multi level routing solution. Future research will focus on the consideration of digital elevation models, respectively digital slope models in order to optimise fuel consumption.

Moreover an application dependent validation of basic and adjacent network elements regarding different vehicle groups as well as different good categories (e.g. dangerous goods) will augment the routing capability.

Finally an intense investigation of the technologies used so far should be done as well. Here the emphasis should be put on optimisation of computation time, data storage infrastructure and multi user capability. An additional computation result export function to common geo-browsers, like Google Maps or Virtual Earth, is also part for future work.

REFERENCES

Bartelme, N. (1991). Datenmodelle für Netzwerk-Applikationen in GIS, *Wiener Schriften zur Geographie und Kartographie – GIS und Kartographie*, 6: 17-22.

Caldwell, T. (1961). On Finding Minimum Routes in a Network With Turn Penalties, *Communications of the ACM*, 4 (2): 107-108.

Car, A, Taylor, G., Brunsdon, C. (2001). An analysis of the performance of a hierarchical wayfinding computational model using synthetic graphs, *Computers, Environment and Urban Systems*, 25: 69-88.

Dijkstra, E. W. (1959). A Note on Two Problems in Connexion with Graphs. *Numerische Mathematik,* 1: 269-271.

Fischer, T. (2003). *UMN Mapserver 4.0 – Handbuch und Referenz*, MapMedia Berlin.

Fu, L., Sun, D., Rilett, L. R. (2006). Heuristic shortest path algorithms for transportation applications: State of the art, *Computers & Operations Research*, 33: 3324-3343.

Gutiérrez, E., Medaglia, A. L. (2008). Labeling algorithm fort he shortest path problem with turn prohibitions with application to large-scale road networks, *Annals of Operations Research*, 157: 169-182.

Hofmann-Wellenhof, B., Legat, K., Wieser, M. (2003). *Navigation – Principles of Positioning and Guidance*, Springer Wien New York.

Jagadeesh, G. R., Srikanthan, T., Quek, K., H. (2002). Heuristic Techniques for Accelerating Hierarchical Routing on Road Networks, *IEEE Transactions on Intelligent Transportation Systems*, 3 (4): 301-309.

Nicholson, J. A. T. (1966). Finding the shortest route between two points in a network, *Computer Journal*, 6: 275-280.

Wieser, M. (1991). Wesen und Nutzen von Optimierungsstrategien am Beispiel des AIS, Mitteilungen *der geodätischen Institute der Technischen Universität Graz*, (70): 57-68.

Winter, S. (2002). Modeling Costs of Turns in route Planning, *GeoInformatica*, 6 (4): 345-361.