Wayfinding by multi-level heuristic node promotion in real road networks

Joris Maervoet*, Pascal Brackman‡, Patrick De Causmaecker†§, Katja Verbeeck* and Greet Vanden Berghe*§
*KU Leuven, Department of Computer Science, CODeS, Gebr. De Smetstraat 1, 9000 Ghent, Belgium
†KU Leuven-Kulak, Department of Computer Science, CODeS, Etienne Sabbelaan 53, 8500 Kortrijk, Belgium
‡RouteYou.com, Kerkstraat 108, 9050 Ghent, Belgium
§iMinds - ITEC - KU Leuven

***Abstract*—The present article introduces the application of the multi-level heuristic node promotion algorithm to real road networks for vehicle navigation. In contrast with many classical shortest path algorithms, this hierarchical shortest-path approximation algorithm integrates in a multi-tier web architecture in such a way that routing queries as well as a minor data updates are processed in a short amount of time. The multi-level heuristic node promotion algorithm was first proposed by Jagadeesh et al. [1] for two levels, although it is only effective when an irreversible graph transformation has been applied on the road network during preprocessing. This irreversible transformation, which consists of slip road removal and dual carriage way reduction, is problematic in contemporary route planning applications. Several heuristic adaptations to both the data preprocessing and the algorithm are introduced and motivated. These adaptations bypass the irreversible graph transformation and restore the effectiveness of the heuristic node promotion algorithm.**

A computational experiment shows the application of the hierarchical algorithm to the 5-level time-weighted road network graph of Belgium, in combination with node pruning using a rectangular area. It analyses the effects of each of the adaptations on the routing performance. This experiment is conducted in the context of a routing web application for tourism and leisure purposes, but the suggested approach is effective for hierarchical shortest-path applications in general.

***Index Terms*—Shortest path problem, heuristic algorithms, data preprocessing.**

I. Introduction

ROUTEYOU.COM manages a web 2.0 environment enabling users to interactively create, share and use tourist routes. Besides, it offers a routing platform for various application developers and digital content providers. One of its basic components is a routing engine that computes a route of interest, mainly intended for vehicle navigation, over the road network between two points selected by the user.
This calculation entails finding the path with the lowest cost in a directed weighted graph. The engine supports several routing modi, each of them referring to another type of edge weights. For the modus **'shortest'**, the weights represent the edge length. A time-weighted graph enables finding the **'fastest'** route. These time weights are estimates of the average time it is necessary for a vehicle to traverse the edge. Moreover, the engine offers a gamut of modi, further referred to as **'nicest'**, tailored to subdomains of leisure and tourism. For these modi, the weights correspond to the multiplications of the edge length and an unsuitability factor ≥ 1, which is inversely proportional to the suitability of the edge to the subdomain. For cycling, for example, unsuitability is determined by both physical and scenic road characteristics and traffic regulations.

The research has been carried out as part of the industrial PhD project 'Structural heuristics for personalised routes' funded by the IWT (090726) and the company RouteYou.

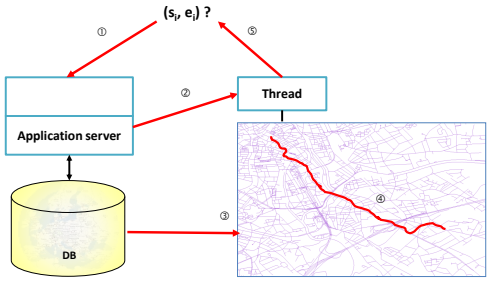


Fig. 1. Routing process diagram. Upon each routing request from starting point s_i to end point e_i (step 1), the application server starts a thread (step 2). Once or repeatedly, this thread loads the portion of the road network nodes and edges that spatially intersect a bounding box (step 3). When the path finding process has terminated (step 4), the thread returns the result to the client and dies, releasing the memory occupied by the thread (step 5).

The path calculation infrastructure is a multi-tier architecture consisting of the following properties.

- All road network graph information is stored exclusively in a relational database with a spatial extension.
- The actual path calculation is executed on an application server.
- The road network graph is subject to minor updates such as new road segments or changing costs w.r.t. the traffic situation.

This architecture supports a multi-purpose GIS system with high scalability w.r.t. the number of parallel users of services offered by the system. It gives rise to the following processes supporting the routing application:

1) **preprocessing** the road network. This step involves the transformation of geographical data into an indexed and annotated road network graph. The word *indexed* refers to structures that optimise data consultation in process 2a, e.g. static clusters or a spatial index, such as an R-tree or QuadTree. The word *annotated* refers to the

enrichment of the data by heuristic information (e.g. node coordinates, obstacle segmentation lines, hierarchical labels) in order to accelerate the search in process 2b.

2) the routing process, triggered by end user requests. It is shown in Figure 1 and consists of:
   a) **consulting** and reindexing relevant data. Reindexing in its turn involves building structures in the annotated graph, which facilitate the search process in process 2b.
   b) the shortest path finding process itself.
3) **updating** the indexed and annotated road network. An atomic update involves the update of an edge weight or the insertion or deletion of an edge or a node.

Peculiarity of the infrastructure. Process 2a does not apply to traditional infrastructures. In memory-based infrastructures, all data is loaded in memory only once, after preprocessing. In disk-based infrastructures, the indexed and annotated data is written on a disk in an adapted format during preprocessing, and disk access is integrated in the shortest path finding process. When the shortest path calculation is written as an extension of the RDBMS, elementary graph operators and indices can be integrated in order to improve performance and avoid reindexing [2].

Requirements. Given the online nature of the application, the twofold routing process is evaluated based on the 3 following **performance criteria**: *(1) computational time*: end users need the answer in less than 5 seconds; *(2) memory footprint*: a low amount of memory used by the routing process, increases the number of parallel requests the web application allows handling; and *(3) quality of the route*. Quality is expressed in terms of relative error with regard to the cost of the exact solution. It is required that *(4) the computational time* of one atomic update is less than 1 second. Other performance criteria are: *(5) the computational time* of preprocessing, and *(6) the size* of the indexed and annotated data generated during preprocessing.

Process 2a in the multi-tier architecture is time consuming (criterion (1)). Its time complexity is at least linear to the size of the consulted data and each query introduces a constant latency time. Moreover, the size of the consulted data is proportional to the memory footprint (criterion (2)). This imposes that the routing algorithm requires only a concise subset of the data, transferred from the database in a low number of efficient queries, irregardless of the routing distance.

In the overview of heuristic approaches to one-to-one wayfinding in Section II-A of the present article, we show that classical hierarchical shortest path algorithms suit the requirements imposed by the application best. Section II-B discusses the applicability of these approaches to several routing modi and Section II-C further outlines this type of approaches in the field. Section III of this article introduces a new hierarchical shortest path algorithm, which is called multi-level heuristic node promotion (MLHNP).

Similar to other hierarchical path finding approaches, this algorithm can only be effective on real road network data if the road network graph is transformed during preprocessing. The application of this transformation is problematic in contemporary routing applications. Section IV introduces a set of heuristic adaptations to the MLHNP approach such that it is effective in untransformed network graphs. Section V evaluates the performance of hierarchically finding the 'fastest' routes for a test set of 1000 routing queries in Belgium. Section VI is the conclusion of this article.

II. Related work

A. Heuristics for one-to-one shortest path finding

During the last two decades, one-to-one shortest path algorithms for realistic road networks have received extensive attention. Zhan and Noon [3] compared 15 optimal shortest path algorithms on two test sets covering the U.S. The most effective one-to-one algorithms are variants of the algorithm proposed by Dijkstra [4], which differ with regard to the data structure for finding the node with the smallest label for examination during search. Heuristic shortest path algorithms use extra knowledge about the road network in order to make the search process less computationally expensive.

The A* algorithm, first proposed by Hart et al. [5], limits the search area to an ellipse by including a heuristic estimation of the cost to the end node in the node traversal priority function. The extra knowledge consists of the node's geographical coordinates. The algorithm is guaranteed to find the optimal solution when the heuristic is admissible i.e. always underestimates the cost. In distance-weighted networks, Euclidean distance is an admissible heuristic. Jacob et al. [6] showed that the introduction of an overdo parameter, multiplying the Euclidean distance to the end node, drastically decreases computational time while preserving the solution quality to a large extent. The idea behind this 'overdo heuristic' is a better estimate of the conservative lower bound of the A* algorithm. Other A* variants aim at improving the cost estimate itself by: using a learning algorithm [7], looking ahead one node further [8], precomputing the shortest paths between a selection of landmarks [9] or modelling obstacles through segmentation lines [10].

Branch pruning is another form of limiting the search area, which involves ignoring the nodes during the search process that have low probability to being on the shortest path from starting to end node. This could be realised by the use of a stochastic model [11] or just by restraining the search space to a rectangular area containing start and end node, in the direction of their connecting line [12], or parallel to the axes of the coordinate system [13]. More recent work focuses on heavily precomputed data structures defining a set of prunable nodes for each of the individual edges. Wagner and Willhalm [14] approximate the set of nodes containing a shortest path by a geometry (so-called consistent container). Lauther [15] saves for each of the edges e and each of the predefined regions r whether there exists a shortest path through e to r. Gutman [16] introduces the reach of a node v i.e. for all shortest paths P through v, the maximum of the cost of the prefix and the suffix of P. Next, the nodes are pruned, for which the reach is higher than the underestimation of both the prefix and suffix of the shortest path under construction. The same

author also presents a faster preprocessing method, which computes the upper bounds of the maxima. A bi-directional version of this algorithm has been improved by Goldberg et al. [17]. Moreover, the introduction of shortcut arcs drastically increased the preprocessing efficiency.

Hierarchical shortest path algorithms use a road network divided into multiple levels (e.g. the road classes) in order to limit the search link space drastically: only close to the start and end node the lower level links are taken into account. This divide-and-conquer strategy was first introduced and related to the field of cognitive psychology by Car and Frank [18]. Initially, the preprocessing phase of this type of approaches started from a hierarchically labelled network, and most algorithms were approximative. In the context of these *classical hierarchical approaches*, Fu et al. [19] identified the hierarchical shortest path algorithm as the most efficient heuristic in real transportation networks. They refer to the experiments by Liu [20], which show that the hierarchical algorithm reduces the average computational time of Dijkstra to one tenth, whereas A* only reduces it to half and bi-directional search to 80% (as first introduced by Dantzig [21]).
A second type of approaches involves hierarchy computation during preprocessing, and does guarantee optimal results. Sanders and Schultes [22], [23] obtain a highway hierarchy by iteratively selecting all edges that appear in shortest paths between any nodes u and v, but outside the local neighbourhood of u and v. It integrates highway node contraction similar to shortcut arcs. A contraction hierarchy [24] arises from iteratively contracting any node in a heuristic order of importance. For the latter approach, path calculation times of 0.2-0.3 ms are reported. The transit-node approach [25] starts from a map divided into a set of disjoint regions. It involves identification - for each of the regions - of a small set of nodes where any shortest path leaving or entering the region passes. Next, all shortest paths between any two transit nodes of different regions and between any node of a region r and any transit node of r are precalculated. This approach further reduces the path calculation time to the order of 5-20 μs. Goldberg et al. [17] found that the performance of their algorithm based on reach pruning is very similar to the performance of a hierarchical algorithm. They introduced the concept of *cardinality reach*, referring to the traversal order during search, in order to emulate hierarchical behaviour in terms of a reach algorithm.

Multi-tier and requirement suitability of the heuristics. Both the variants of the Dijkstra and A* algorithm can easily be integrated in the multi-tier architecture, in combination with pruning the nodes by a rectangular area. This is illustrated in Figure 1. Efficient bounding box selection is supported by almost every spatial DBMS. Nevertheless, these algorithms do not scale for routing queries over higher distances d, since the size of the data consulted by process 2a has a worst-case complexity of $\mathcal{O}(d^2)$, assuming a uniform spatial distribution of the road network graph.
Branch pruning heuristics have typically been designed for pruning during the shortest path finding process. Only when the pruning technique can be adopted by process 2a, i.e. integrated in a few efficient database queries, it might scale for routing queries over higher distances. This integration is possible for the algorithms by Wagner and Willhalm [14] and Lauther [15], but does not realise a significant scalability improvement. In the case of reach pruning, contemporary spatial DBMS technology does not enable to efficiently query the spatial points of which one of the attributes (reach number) is higher than the euclidean distance to two given points. Moreover, a dynamic version of this algorithm has only been considered for edge weight changes, for the version without shortcuts and without experimental evaluation [26].
The multi-tier integration of both the highway and contraction hierarchy approaches is problematic since there is no sound mechanism to reduce the number of consulted edges in advance in a scalable way. Data updates can be executed in short computational time [26], but are constrained to changing the edge weights. The transit-node approach is compatible with the multi-tier architecture, but burdens the precalculated data size, and is unsuitable for efficient data updates.
Classical hierarchical approaches integrate generally well in the multi-tier architecture because the algorithms operate on strongly reduced parts of the road network graph, which can be indexed by location and level. This enables a low memory footprint and computational time for process 2a. During preprocessing, the road network annotation is derived from the source data by simple operators such as intermediate node reduction and spatial containment. As a consequence, preprocessing times are rather short, the generated data size is small and minor data updates can be processed in less than 1 second. The limitations of this type of approaches are the approximative character of the resulting paths, and the necessity of a predetermined network hierarchy, which is discussed below.

B. Applicability of classical hierarchical approaches

Classical hierarchical approaches are mostly applied in transportation network graphs, in which edge weights represent travel time. The road classes then simply determine the levels of the network. This road class heuristic is a good indicator for the actual speed over the edges and the subnetwork of the n highest classes is fully connected by nature. The road classes can be easily obtained from geographical content providers.

Furthermore, classical hierarchical approaches are applicable to a wider domain of *routing modi*. For some countries, the road class is also a good heuristic to find the shortest path in a distance-weighted road network graph. Jagadeesh et al. [1] showed for Singapore that it was due to the road class reflecting the inherent hierarchy of the graph topology. Alternatively, Chou et al. [27] suggest to promote edges with a high length to the higher level for this type of weights. The same classical hierarchical algorithms can be used in this context of finding the 'nicest' routes, but the generation of hierarchy for these modi is beyond the scope of this article. For hierarchical networks in general, we assume that

- each subnetwork of the n highest classes is fully connected.
- the number of edges in the higher classes is restricted.

- the edges of a higher class are more probable to be included in the shortest path from any node a to any node b of the graph. This results typically in a selection of edges of lower unsuitability (higher speed) enriched by edges that realise topological connectivity (e.g. slip roads, 'unsuitable' ('low speed') path that is an essential shortcut).

C. Classical hierarchical shortest path finding

Classical hierarchical wayfinding requires a network graph to be divided in two or more (connected) graphs. **Leave points** denote the nodes that enable traversal from a given level network to a higher level network. The term **entry points** is used for the ones that give entry to a lower level network. The term **transition points** refers to both entry and leave points. Several algorithms have been proposed that optimally conduct the search process over different levels.

Car and Frank [18] proposed a bottom-up approach to multi-level hierarchical routing. It uses a hierarchical tree of meshes i.e. a set of edges bounded by higher-level edges, and starts from the lowest level. It involves that first the shortest path from the start node to the closest leave point and from the closest entry point to the lower level are found, using a classical algorithm, on the lowest level network. Next, the infix path is recursively searched for on the higher level network. The recursion ends when start and end node are situated in the same or in adjacent meshes, or when no higher level is available. In this case, the shortest path from start to end node is found using a classical algorithm. In the end, the resulting subpaths are concatenated. Later on, Cho and Lan [28] adopted the bottom-up approach and studied the memory-error trade-off for different combinations of Dijkstra and A* with 'overdo heuristic' on the different levels of operation.

One major drawback of the bottom-up approach is that the choice of the closest transition point is not necessarily the best choice to produce the path with the lowest cost. Therefore, Liu [20] introduced the stitching approach, i.e. the meshes in which the start and end node are located are added to the higher-level network, and a classical algorithm is used to find the shortest path from the start to the end node in the stitched network graph. Chou et al. [27] suggested to calculate the shortest paths from the start node to all leave points and from all entry points to the end node and to select the combination with the lowest cost afterwards. Jagadeesh et al. [1] introduced the two-level hierarchical algorithm with heuristic node promotion. It is a top-down approach that starts routing at the higher level if start and end node are situated in the same or in adjacent meshes. This higher level routing stage involves finding the shortest path from start to end node on the higher level network, enriched by a set of *virtual links* from the start node to each of the leave points and from each of the entry points to the end nodes, using a classical algorithm. The virtual link weights are cost estimates based on Euclidean distance. Next, the two virtual links selected in the resulting path are replaced by the shortest path from start to end node of the virtual links, obtained by routing on the lower level network. The authors showed that their method outperforms (computation time vs. result quality) the two-level version of the bottom-up approach.

The algorithms described above typically integrate a pruning technique in order to speed up the search during each of the subroutings. Jagadeesh et al. [1] constrain the search space to the subgraph formed by the N closest nodes to the start and end node. Liu [20] uses cell meshes to prune the search graph. In the first case, it may be difficult to find a general measure for N or an accurate distance function for scenic or time-weighted routing modi. The latter case will certainly work for planar graphs, but may not be effective in non-planar graphs without intensive preprocessing of overlapping cells.

Strauss [29] raised some issues concerning unconnected higher level network graphs that apply to the hierarchical approaches above. He identified the case of (1) a disconnected higher-level graph due to the data selection, (2) a disconnected higher-level graph because of the topographic restrictions of the dataset, and (3) the low-level shortcut which connects two high-level subnetworks. Examples of the latter case are a footpath connecting the railway system with the airport or a shortcut between two highways which often appears in paths with the lowest travel time. In this context, Liu [20] suggests to add 1-edge shortcuts within the higher network during preprocessing. Strauss [29] introduces the multi-level hopping algorithm to cope with all these issues. This method alternates between synchronous and asynchronous search tree spreading in a bidirectional manner.
We argue that each of the issues raised can be avoided by the following measures: (1) expand the data selection in case the routing algorithm detects a disconnection, (2) organise the global higher network in such a way that it is a connected graph (this could be imposed without any loss of route quality)[1], and (3) classify the edges according to the probability of being included in any shortest path. The last measure has already been suggested by Jagadeesh et al. [1] for slip roads between motorways.

Concluding remark. Classical hierarchical algorithms can be categorized into four main algorithmic strategies: top-down, bottom-up, stitching and level hopping. In combination with a spatial or mesh-based pruning, the first three strategies integrate well in the multi-tier architecture. Given the upper bound number u of edges contained in a cell or spatial container, the worst-case size complexity of the data consulted by process 2a is limited to $\mathcal{O}(2 \cdot l \cdot u)$, with l the number of levels. Section III introduces a variant of the top-down approach, which was shown to yield the optimal trade-off of computational time and result quality. The multi-level hopping algorithm requires process 2a to consult as much data as a Dijkstra or A* algorithm and therefore it does not fulfil the requirements.

[1]Note that a disconnected highway graph due to dataset restriction often coincides with the inability to produce the *real-world* fastest path with any algorithm, because this path runs through one or more neighbouring datasets. This is not a valid excuse for the algorithm to fail, but it indicates that this type of case applies to rather experimental than operational planners.

TABLE I
MLHNP CONCEPT DEFINITIONS

	Concept	Definition
Graph	$G = (V, E)$	a directed road network graph in which each of the edges $e \in E$ is assigned a weight $w(e)$ and a level $l(e) \in [0, maxlevel]$. Any directed edge e starts in node $from(e) \in V$ and ends in node $to(e) \in V$. We will use the symbols $\dot{\in}$ and $\widetilde{\in}$ to denote set membership of V and E.
	G_L	the subgraph of G, consisting of edges $e \,\widetilde{\in}\, G : l(e) \geq L$. G_L should be connected for each $L \in [0, maxlevel]$.
	$edge(n_1, n_2, w)$	an edge from vertex n_1 to n_2 of weight w
	$reduced(G, X)$	the transformed version of G in which all intermediate nodes (i.e. nodes that only connect to two physical ways) have been removed. Intermediate node reduction is achieved by iteratively replacing each pair of edges $edge(a, b, w_1)$ and $edge(b, c, w_2)$ with intermediate node b, by edge $edge(a, c, w_1 + w_2)$. The exception list X is an optional parameter. Any node of X that is a node of G must not be removed.
Paths	$\mathcal{P}(a, b, G)$	the set of paths from vertex a to b in graph G. It consists of any sequence P of subsequent edges from graph G, for which the first edge starts in a and the last edge ends in b
	P_i	the edge with position i in path P
	$P_{i...j}$	the subpath of P starting by P_i and ending by P_j
	$\|P\|$	the cost of path P is defined as the sum of the weights of all its edges
	$dist(a, b, G)$	the shortest path distance from vertex a to b in G is defined as $min_{P \in \mathcal{P}(a,b,G)} \|P\|$. The set of shortest paths $shortest(a, b, G)$ contains any path P for which $\|P\| = dist(a, b, G)$
	$error_{rel}(Q)$	the relative error of an approximate shortest path $Q \in \mathcal{P}(a, b, G)$ is defined as $\frac{\|Q\| - dist(a,b,G)}{dist(a,b,G)}$
Cells	cell c	a spatial region containing nodes of a graph G.
	$cells(L)$	the set of cells of level L in graph G. Each polygon that is enclosed by edges in G_L and eventually the dataset boundary, and does not overlap any other edges in G_L, is an element of $cells(L)$.
	$l(c)$	$L \| c \in cells(L)$
	$cells(n, L)$	the subset of $cells(L)$ of cells that spatially overlap node $n \,\dot{\in}\, G$. $\forall n \,\dot{\in}\, G, L \in [1, maxlevel] : cells(n, L) \neq \emptyset$
	$contiguous(c_1, c_2)$	is true when the cells c_1 and c_2 are contiguous i.e. their spatial intersection is at least a line.
Node promotion	$leavepoints(c)$	the set of candidate transition points between any route starting in cell c planned in any G_m with $m < l(c)$ and $G_{l(c)}$ $leavepoints(c) = \{n \in c \| \exists e \,\widetilde{\in}\, reduced(G_{l(c)}) : from(e) = n \wedge to(e) \notin c\}$
	$entrypoints(c)$	the set of candidate transition points between $G_{l(c)}$ and any route ending in cell c planned in any G_m with $m < l(c)$ $entrypoints(c) = \{n \in c \| \exists e \,\widetilde{\in}\, reduced(G_{l(c)}) : to(e) = n \wedge from(e) \notin c\}$
	$estimate(n_1, n_2)$	a cost estimate of the shortest path from n_1 to n_2
	$promote_{st}(n, G')$	$\Leftrightarrow \neg(\exists e \,\widetilde{\in}\, G' : from(e) = n)$
	$promote_{end}(n, G')$	$\Leftrightarrow \neg(\exists e \,\widetilde{\in}\, G' : to(e) = n)$

III. MULTI-LEVEL HEURISTIC NODE PROMOTION IN TRANSFORMED NETWORK GRAPHS

The present section introduces the multi-level heuristic node promotion algorithm (MLHNP). It is a recursive version of the two-level algorithm by Jagadeesh et al. [1]. Whereas slip roads and dual carriageways typically occur in networks for vehicle navigation, the algorithm is only effective on road networks that have been subjected to the following graph transformations: (1) slip road removal, and, (2) dual carriage way and multi-lane reduction into single-track ways. These processes reduce complex intersections to a single node connecting the main directions.

A. Algorithm

MLHNP assumes the availability of the concepts listed in Table I, applied to a network divided in $maxlevel + 1$ levels. This algorithm approximates the path of lowest cost from vertex s_{global} to e_{global} in graph G, returning an approximate path Q with a minimal $error_{rel}(Q)$. It is initiated by the call $mlhnp(s_{global}, e_{global}, maxlevel, G)$ to the algorithm listed in Figure 2. Figure 3 shows an example top recursive run of the algorithm. First, it determines its level of execution l_{exec} as the highest level for which n_1 and n_2 are not located in contiguous or same cells. Next, the search graph is pruned and constrained to the edges of level $\geq L$. When $l_{exec} > 0$ and $promote_{st}(n_1, G')$, the search graph is extended by a set of virtual edges, from n_1 to n_1's cell leave points. These virtualisations are part of the so-called *node promotion* because n_1 is promoted to $G_{l_{exec}}$. This promotion is $heuristic$ because

$mlhnp(n_1, n_2, l_{restr}, G)$:
  $L_{cand} \leftarrow \{l | \forall c_1 \in cells(n_1, l), c_2 \in cells(n_2, l) : \neg(c_1 = c_2 \vee contiguous(c_1, c_2))\}$
  $l_{exec} \leftarrow min(l_{restr}, max(L_{cand} \cup \{0\}))$
  $G' \leftarrow reduced(G_{l_{exec}}, \{s_{global}, e_{global}\})$
  if $l_{exec} > 0$ **then**
    if $promote_{st}(n_1, G')$ **then**
      $G' \leftarrow G' \cup \{edge(n_1, n, estimate(n_1, n)) | n \in leavepoints(cells(n_1, l_{exec}))\}$
    end if
    if $promote_{end}(n_2, G')$ **then**
      $G' \leftarrow G' \cup \{edge(n, n_2, estimate(n, n_2)) | n \in entrypoints(cells(n_2, l_{exec}))\}$
    end if
  end if
  $R \leftarrow P \in shortest(n_1, n_2, G')$
  if $l(R_0) =$ 'v' **then**
    $R \leftarrow concatenation(mlhnp(from(R_0), to(R_0), l_{restr} - 1, G), R_{1...size(R)-1})$
  end if
  if $l(R_{size(R)-1}) =$ 'v' **then**
    $R \leftarrow concatenation(R_{0...size(R)-2}, mlhnp(from(R_0), to(R_0), l_{restr} - 1, G))$
  end if
  return R

Fig. 2. Multi-level heuristic node promotion.

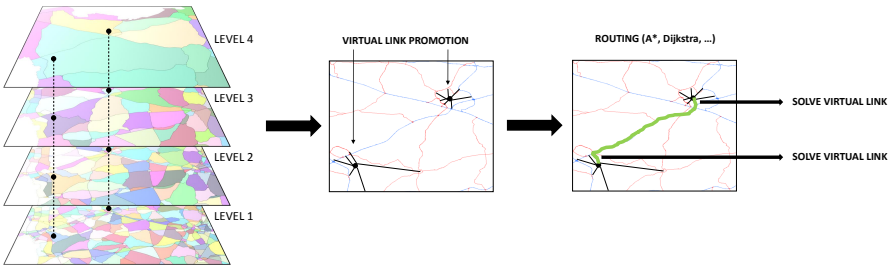


Fig. 3. Recursive run of the multi-level heuristic node promotion algorithm.

each of the virtual edge weights is a heuristic cost estimate of the shortest path from n_1 to the leave point. Analogously, when $l_{exec} > 0$ and $promote_{end}(n_2, G')$, the search graph is extended by a set of virtual edges, from n_2's cell entry points to n_2. Next, a classical routing algorithm is applied from n_1 to n_2. This step is further referred to as **the basic shortest path calculation**. In the resulting path, each selected virtual link is replaced by the path resulting from the recursive call to this function. The recursive decrement of the level of execution guarantees that the algorithm is finite.

B. The role of transition points and the heuristic estimate

Each cell corner connecting to other edges of the higher level network, is registered as a transition point. The transition point definitions in Table I realize this pattern likewise. A recursive run of MLHNP typically generates a route of the type
$(startnode) - virtuallink - (transitionpoint) - fixedroute - (transitionpoint) - virtuallink - (endnode)$,
in which the virtual links are solved in deeper recursions. So the solution of the virtual link typically contains higher level cell border edges and the transition point in the resulting route is the node where the route switches between the cell border and rest of the network. This means that the shortest path search on the border is preferably executed downlevel. This is a scalable approach that complies with hierarchical road network data containing more than two levels.[2].

The ultimate algorithmic choice of transition point is influenced by the virtual link cost estimate function $estimate(n_1, n_2)$. The better the approximation, the higher the chance that the transition point generating the shortest path is selected. Finding the shortest path in a graph with systematic overestimation of the virtual link cost, yields preference of transition points close to the starting node/end node. Systematic underestimation tends to minimise the cost of the global path apart from the virtual link. Jagadeesh et al. [1] approximate the real cost by the Euclidean distance in case of distance-weighted edges, and by the Euclidean distance divided by the average speed of the weights of the cell's edges in case of time-weighted edges. The latter approach assumes a homogeneous speed distribution over the cell. This is certainly not the case for cells of levels greater than 1. Individual variance between the real virtual link cost and its approximation leads more easily to a wrong transition point choice (i.e. resulting in a path with higher cost) than systematic deviation. Moreover, it requires extra preprocessing time and storage costs. Therefore we just use an arbitrary average resistance r_{avg} in order to estimate the virtual link cost for each of the routing modi as follows:
$estimate(n_1, n_2) = dist_{eucl}(n_1, n_2) \cdot r_{avg}$
This value is a single estimate of the ratio of the real path cost from any n_1 to any n_2 to the euclidean distance between n_1 and n_2.

[2]We have considered alternative transition point definitions. (1) Each transition from the cell border to the lower level network. This approach involves that the transition points are candidate points where the shortest path switches between the cell border and the network inside of the cell. This approach is promising w.r.t. any of the routing performance criteria, but is not scalable because at high levels we could easily obtain cells of 10000 transition points. (2) Each transition from the cell border to edges of one level lower. This approach scales but imposes some additional requirements. Each cell of level n should overlap edges of exactly level $n-1$. Moreover, it only supports paths that have a level increase/decrease by one, which deviate greatly from shortest paths in common hierarchies.

C. Preprocessing overview

Starting from a road network graph cleared from slip roads and dual carriage ways, the following transformations and annotations are necessary during the preprocessing phase.

1) Intermediate node reduction of each graph $G_L : L \in [0, maxlevel]$. This transformation is necessary to speed up the search process of the shortest path algorithm and it supports higher level scalability of MLHNP. It is a reversible transformation because each transformed edge can be unambiguously translated into the original sequence of edges.
2) Appropriate indexing of each of the resulting graphs $reduced(G_L)$.
3) Cell generation and cell contiguity registration
4) Appropriate indexing of the cells and contiguities by node.
5) Generation and indexing of the transition points by cell.

Step 3 involves building polygons formed by the minimal cycles of the planar version of the road network graph G_L, for each $L \in [1, maxlevel]$. Planarisation involves that edge crossings are replaced by dummy vertices, for each subnetwork G_L, and that duplicate line parts are merged into a single line belonging to the highest level of both sources. Moreover, all directed edges have been made undirected in the planar graph. The planar graph has been extended by the border of the routable area, its level labelled by $maxlevel + 1$. Non-border edges may end at the border, but must not exceed it.

As a result, this planar graph has a spatial representation in which none of the edges intersect, and any edge (not part of the border) encloses two minimal cycles of the graph.
This cell generation can be implemented as an iterative top-down process, which uses G_N to split the cells of level $N+1$ into cells of level N. At the top level it starts from one or more cells representing the routable area. In order to split one cell of level $N+1$, all topology of G_N overlapping the cell is loaded, and minimal cycle navigation starts at a random edge of exactly level N (if not available, the cell can be copied down level). During navigation, the algorithm chooses the leftmost edge at each node. Navigation ends when a visited node is encountered, closing the polygon. The next navigation starts in a random edge of exactly level N that has only been visited once. The split-down finishes when no more start edges are available. Each time an edge is visited twice, a cell contiguity is registered. Note that choosing the same direction (leftmost or rightmost) for all navigations will guarantee that all minimal cycles are visited when any edge is visited in forward and backward direction.

D. Updating overview

Processing an atomic data update is quite straightforward. In case of a weight update of an edge of level m, it is only required to update the accumulated weight in any graph $reduced(G_L)$ with $L \leq m$. In case of insertion or removal of an edge e of level m, it can be required (1) to break up one or two edges and (2) to reduce a novel intermediate node in any $reduced(G_L)$ with $L \leq m$. When an edge of level $m > 0$ that encloses both two cells in $cells(L)$ with $0 < L \leq m$ is removed, the cells (and their contiguities) should be merged. The remaining subset of the union of transition points of both cells should be determined. When the insertion of an edge of level $m > 0$ realizes a full separation of a cell in $cells(L)$ with $0 < L \leq m$ into two new cells, the cells (and their contiguities) should be split. The new cells inherit a subset of the original transition point set, and new transition points are amongst the nodes of the separation boundary.

IV. Multi-level heuristic node promotion in real road networks

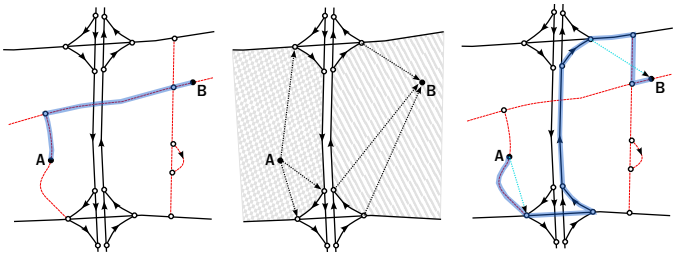
Geographic content providers commonly model physically separated lanes as well as slip roads as separate edges. It is unclear how slip road removal and dual carriageway reduction, as suggested in the previous section, can be dealt with in a routing application for vehicle navigation.

- The transformation processes may be ambiguous, or only realisable in combination with turn restrictions, for instance when a complex intersection does not interconnect all directions.
- Contemporary routing applications enable visually selecting a starting or end point located on any of the edges originally provided. So the user may select a specific lane or slip road. The transformation projects many of such edges to one edge or node. So the routing in the transformed graph does not comply with the *route-by-click* philosophy of the application, unless the start and end neighbourhood in the graph is enriched by original data, or the route is corrected after the routing process.
- Contemporary routing services return a line string, which is the exact route's geometry including the appropriate lanes and slip roads to be taken. It is not clear how this geometry can be reconstructed after the routing process.

This section introduces several heuristic adaptations to the MLHNP approach in Section III such that it is effective in network graphs that have not been subjected to slip road removal and dual carriageway reduction.

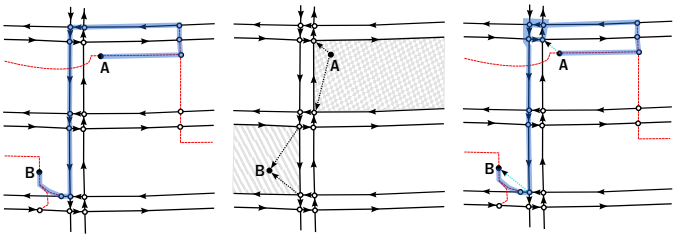
TABLE II
LEGEND.

○	node
———	a high level edge (an arrow restrains the navigation direction)
- - -	a low level edge (an arrow restrains the navigation direction)
A●	starting node
●B	end node
▬	path
▒	cell of starting node
▒	cell of end node
·····→	virtual link involved
·····→	selected virtual link that is replaced after recursion
·····→	virtual link that causes problems
– – →	useful virtual link



(a) Shortest path (b) Cells and virtual links (c) Resulting path

Fig. 4. Scenario 1. The corresponding legend is in Table II.

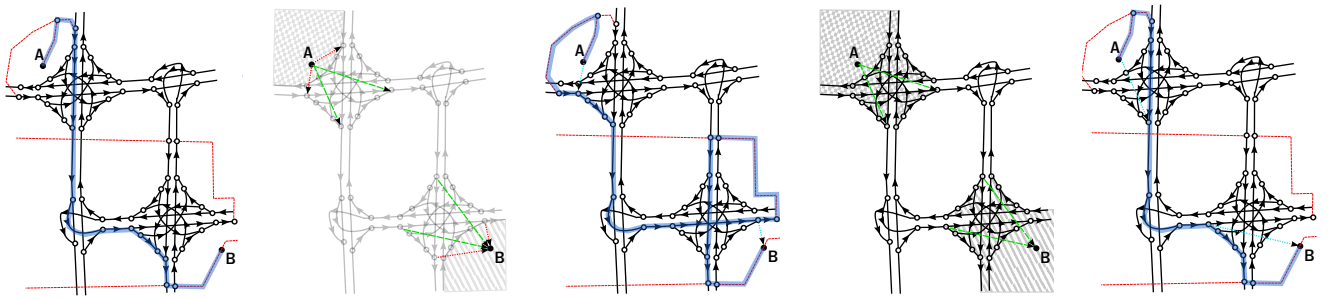


(a) Shortest path (b) Cells and virtual links (c) Resulting path

Fig. 5. Scenario 2. The corresponding legend is in Table II.

A. Adaptation 1: cell classification and merge

1) Problem: First, when applying MLHNP to a transformed graph, the cell contiguity function in Table I is a good heuristic to determine the level of execution l_{exec}. This is not the case in a graph with dual carriageways and complex intersections. The routing scenario in Figure 4 is a typical routing example that should be executed at the lowest level because the cells in which start and end node are located would be contiguous



(a) Shortest path (b) Cells and virtual links (c) Resulting path (d) Corrected cells and virtual links (e) Resulting path after correction

Fig. 6. Scenario 3. The corresponding legend is in Table II.

TABLE III
ADAPTATION 1

	Concept	New definition
Cells	$cells(L)$	the set of cells of level L in graph G. Each polygon that is enclosed by edges in G_L or edges of the dataset boundary, but does not overlap any other edges in G_L, is an element of $cells^{old}(L)$. Each merge of each cell in $cells^{old}(L)$ that is not part of a DCW or CI, with all of the contiguous DCWs and CIs, is an element of $cells(L)$.

after transformation.

Second, as described in Section III-B, transition points determine where MLHNP switches between high and low level routing. In graphs with complex intersections, the cell and transition point definitions do not always generate optimal transition point locations. Figure 5, 6a, 6b and 6c illustrate that the transition points are located between the (candidate) virtual links and the complex intersection they are part of. This often restricts the cell border trajectory, which is planned at the lower level, to one direction.

2) Solution: Both issues have been sorted out by a heuristic cell definition adaptation, illustrated in Table III. This adaptation reshapes the cell polygons, emulating the form they would have after road network graph transformation, using the classical cell definition. A first effect is that the cells in $cells(L)$ overlap near dual carriage ways (DCWs) and complex intersections (CIs). The contiguity of these overlapping cells can be used as an approximation of the contiguity of the routing graph which has been subject to the removal of DCWs and CIs. The other effect is that the transition points are located at the right side of the CIs, such that high level routing only concerns with routing between CIs and low level routing with routing over the CI, as illustrated in Figure 6d and 6e. In other words, the cell form is used as a heuristic to find unique transition points from/to the outside world of the cell.

3) Realisation: The cell classification and merge procedure need to be executed for achieving this adaptation, after the iterative top-down cell generation (Section III-C). Figure 7 shows the cell processing phases that produce the adapted cells and transition points in the neighbourhood of a CI. First, all minimal cells are generated from the minimal cycles of

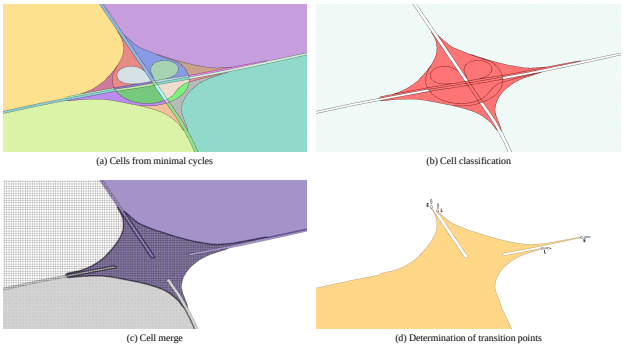


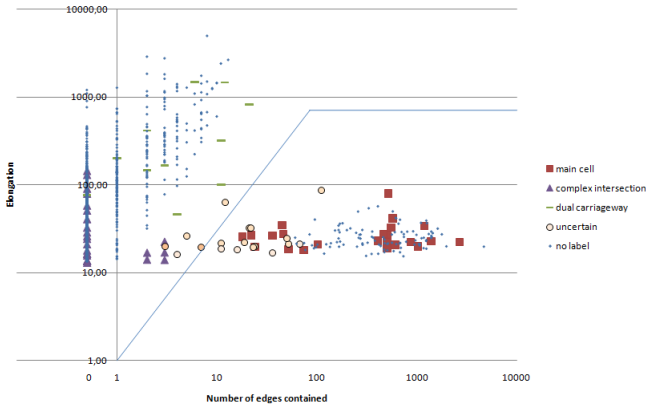
Fig. 7. Cell builder and merger states in the neighbourhood of a complex intersection.

the planar graph. These cells are depicted in Figure 7a, in which cells have a different colour. Next, 2-step classification is applied to each of these cells. Figure 7b shows regular cells in hatched blue, CI parts in red, and DCW parts in white. This classification is used by the cell merge algorithm. It results in only four cells, represented in Figure 7c, each containing as much as possible of the complex intersection. Figure 7d shows how entry points (E) and leave points (L) are assigned to the bottom left cell.

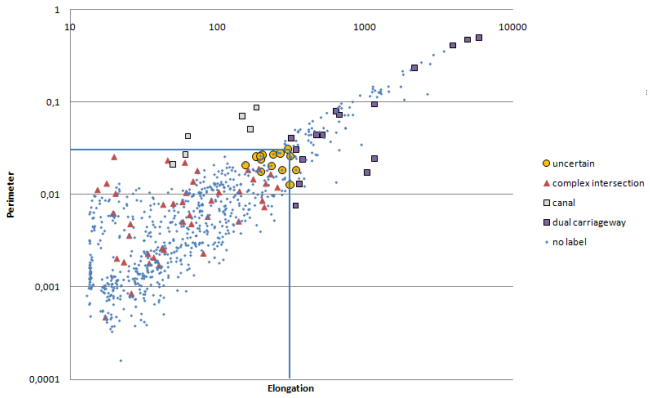
Cell classification. This process discriminates between parts of CIs, parts of DCWs and regular cells. The classification is twofold. For each of the two stages, a selection of level 1 cells of a sample area, has been tagged manually in order to construct a classifier.

A first classifier has been constructed in order to discriminate regular cells from CI and DCW parts. The graph in Figure 8a shows 78 manually tagged cells as a function of 1) the number of level 0 edges contained in the cell (X-axis) and 2) the cell's elongation $= perimeter^2/area$ (Y-axis), both on a logarithmic scale. These parameters are similar to the area and elongation parameters identified by Delafontaine et al. [30] in order to detect sliver polygons in map overlaying.

Figure 8a contains labels for regular cells, CI parts, DCW parts and cells that have been described as uncertain. Examples of uncertain cells are those that are cut off by the global border and cells that are located between the two edges of a dual carriageway at one side but contain a considerable portion of



(a) Classification of regular cells versus CI or DCW parts. The elongation is depicted as a function of the number of edges contained, for 78 manually labelled cells.



(b) Classification of CI versus DCW parts. The perimeter of 78 manually labelled cells is depicted as a function of the elongation.

Fig. 8. Classification of regular cells, CI parts and DCW parts.

urban area at the other side. Note that all unlabelled cells of the sample area are represented as well.

DCW parts are characterised by high elongation and a relatively small number of edges. They form a cluster with CI parts, characterised by a small number of edges. This cluster is disjoint from the cluster of regular cells, which have high numbers of edges and low elongation. Figure 8a depicts a linear classifier, generated in the logarithmic plane. It identifies a cell as being regular when $elongation < \#edges^{1.3424}$. In practice, the number of edges contained within a cell is counted by an SQL query. In order to speed up this query, the count is limited to 1000, before it is passed to the classifier. This is indicated by the plateau in the classifier.

The classifier scales well to cells of higher levels. Higher level DCW parts are even longer, containing more crossing edges. Thus, this category tends to move up right in the graph. CI parts are quasi static. Regular cells will move to the right because they become larger. So none of these categories tends to interfere with the classifier.

Next, non-regular cells are categorised into CI and DCW parts. The graph in Figure 8b shows 78 manually tagged cells as a function of the cell's elongation $= perimeter^2/area$ (X-axis), and, the cell's perimeter, here in digital degree (Y-axis), both on a logarithmic scale.

Figure 8b shows the manually assigned labels for CI parts, DCW parts and cells that have been described as uncertain. A special type of uncertain cells, i.e. DCW parts in which a canal is situated between the traffic flows, have been assigned a separate label. Note that all unlabelled non-regular cells of the sample area are represented as well.

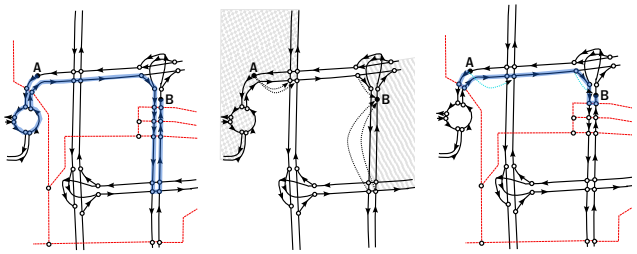
DCW parts are characterised by high elongation and a long perimeter, whereas these parameters have low numbers for CI parts. Canals in DCWs cause lower elongation. Figure 8b depicts the classifier $elongation < 300 \wedge perimeter < 0.3$, which is true in case of a CI part.

The classifier scales well to cells of higher levels. DCW parts of higher levels tend to have longer perimeters and higher elongation. Thus, this category tends to move up right in the graph, away from the classification boundaries. CI parts are static.

$merge\_cells(ALL, CIS, DCWS)$:
  $Candidates \leftarrow CIS$
  while $Candidates \neq \emptyset$ **do**
    for each $complexInt \in Candidates$ **do**
      if $\exists n : contiguous(complexInt, n) \wedge n \in Candidates$ **then**
        for each $n : contiguous(complexInt, n) \wedge n \in Candidates$ **do**
          $complexInt \leftarrow merge(complexInt, n)$
          $CIS \leftarrow CIS \setminus \{n\}$
          $Candidates \leftarrow Candidates \setminus \{n\}$
          $ALL \leftarrow ALL \setminus \{n\}$
        end for
      else
        $Candidates \leftarrow Candidates \setminus \{complexInt\}$
      end if
    end for
  end while
  $Candidates \leftarrow CIS \bigcup DCWS$
  while $Candidates \neq \emptyset$ **do**
    $planning \leftarrow new\ array$
    for each $cell \in Candidates$ **do**
      $planning[cell] \leftarrow \{n : contiguous(cell, n) \wedge n \notin Candidates\}$
    end for
    for each $cell \in keys(planning)$ **do**
      for each $colonist \in planning[cell]$ **do**
        $colonist \leftarrow merge(colonist, cell)$
      end for
      $Candidates \leftarrow Candidates \setminus \{cell\}$
      $ALL \leftarrow ALL \setminus \{cell\}$
    end for
  end while
  return ALL

Fig. 9. Cell merge algorithm.

Cell merge. For each of the levels, the cell merge algorithm is applied to the set of classified cells. The algorithm in Figure 9 takes the complete set of cells ALL and its subsets CIS and $DCWS$ that have been identified as CI / DCW parts, as its input. The algorithm first clusters neighbouring CI parts in singular cells. Next, all CI clusters and DCW parts are merged iteratively by each of their neighbouring regular cells. The outcome is illustrated in Figure 7c. This algorithm tries to maximise the cell overlap at CIs but prevents that cells absorb continuing DCWs along neighbouring regular cells. Note that a



(a) Path without virtualisation (b) Cells and virtual links in case of virtualisation (c) Shortest path with virtualisation

Fig. 10. Scenario 4. The corresponding legend is in Table II.

TABLE IV
ADAPTATION 2

	Concept	New definition
	$promote_{st}(n, G')$	$\Leftrightarrow n = s_{global}$
	$promote_{end}(n, G')$	$\Leftrightarrow n = e_{global}$

cell merge of A by B involves that B inherits A's neighbours.

Transition point assignment. Once all cells have been merged, the leave points and entry points of each cell can be easily determined, according to the definitions listed in Table I. This is shown in Figure 7d.

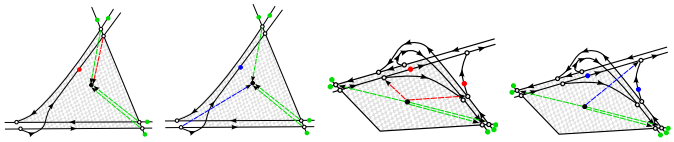
B. Adaptation 2: easing the node promotion condition

1) Problem: In a transformed road network graph, MLHNP node promotion takes place only if starting node n_1 or end node n_2 are missing in graph G'. In a graph with dual carriageways, this can result in paths with extra costs which can easily be avoided. This is shown in Figure 10. The shortest path from A to B starts and ends at higher level edges, while it consists of lower level links in the neighbourhoods of A and B.

2) Solution: This issue is compensated by a change of the node promotion conditions, as proposed in Table IV. It involves node promotion for each recursion with $l_{exec} > 0$, producing virtual links from the global start and to the global end. This intervention enables MLHNP to take into account the lower level neighbourhood around the global start and end node, as illustrated in Figure 10b and 10c. From now on, the MLHNP algorithm in Figure 2 no longer requires $\{s_{global}, e_{global}\}$ as the exception list for the intermediate node reduction of $G_{l_{exec}}$.

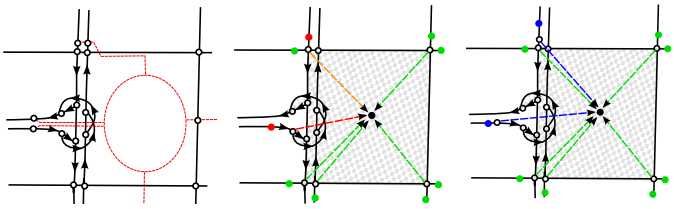
C. Adaptation 3: transition point corrections

1) Problem: Despite Adaptation 1, some of the transition points are still not on locations that support the effectiveness of the MLHNP algorithm. This can be caused by (a) incorrect classification of certain regular, DCW part or CI part cells, (b) the inability of the cell merge algorithm to merge a cell with the complete CI or DCW along its boundaries, or (c) the fact that a very commonly used connection with the inside of a cell is located outside the cell, as illustrated in Figure 12b. Note that issue (b) is not unlikely to occur because the merge algorithm does not support self-tangent



(a) Incorrect (b) Correct (c) Incorrect (d) Correct

Fig. 11. Transition point corrections by length and by couple. The basic legend is in Table II. Arrows represent the virtual links created from/to any of the entry/leave points to/from a point within the cell. Green arrows refer to correct, red to incorrect and blue to corrected transition points. The dots mark the witness-edges of the transition points.



(a) Network graph (b) Incorrect (c) Correct

Fig. 12. Transition point correction in case of early access/late exit. The basic legend is in Table II. Arrows represent the virtual links created from any of the entry points to a point within the cell. Green arrows refer to correct, orange to partially correct, red to incorrect and blue to corrected transition points. The dots mark the witness-edges of the transition points.

or multiple polygons as cell representation (Figure 11a), or because the mechanism that prevents the merge of DCW parts located along contiguous cells is too restrictive (Figure 11c). We identified three anomalous transition point patterns of a cell c.

1) *A cell c does not have any entry (leave) point located on a complex intersection along the cell border.* Figure 11a is an example of this pattern. MLHNP becomes less effective with $l_{exec} = l(c)$ in this case because it does not allow that a path enters (leaves) the lower level graphs of c via the missing intersection.
2) *The choice of an entry (leave) point constrains the subsequent (preceding) path direction on the cell border.* This has been exemplified in Figure 6a, 6b and 6c, and occasionally still occurs, as illustrated in Figure 11c. This makes MLHNP less effective because the choice of transition point at the higher level brings along decisions that should be made at the lower level.
3) *A transition point is located between its cell and a very commonly used connection with the cell.* The network in Figure 12a contains examples of this pattern for both the late exit and the early access case. It is shown for the latter case in Figure 12b. In both cases, MLHNP will never allow routes from/to this cell using this connection, whereas a significant number of shortest paths starting (ending) in this cell and ending (starting) in a non-contiguous cell pass through this connection.

2) Solution and realisation: Both pattern 1 and 2 can be countered by moving entry points one step backwards and leave points one step further, out of the cell area, under certain conditions. We cannot simply apply this correction to any transition point. This would increase the chance of generating

TABLE V
ADAPTATION 3A

	Concept	New definition
Node promotion	$leavepoints(c)$	$witness_{lp}(c) = \{\forall e \widetilde{\in} reduced(G_{l(c)}) \vert \exists n \in c : from(e) = n \wedge to(e) \notin c\}$ $outer_{lp}(c) = \{\forall e \in witness_{lp}(c) \vert (length(difference(e,c)) < length(e)/2) \vee$ $(\exists e_2 \in witness_{lp}(c) : e_2 \neq e \wedge to(e_2) = to(e) \wedge length(difference(e,c) < \sigma_1)\}$ $leavepoints(c) = \{\forall n \vert \exists e \in outer_{lp}(c) : to(e) = n\} \bigcup \{\forall n \vert \exists e \in witness_{lp}(c) \backslash outer_{lp}(c) : from(e) = n\}$
	$entrypoints(c)$	$witness_{ep}(c) = \{\forall e \widetilde{\in} reduced(G_{l(c)}) \vert \exists n \in c : to(e) = n \wedge from(e) \notin c\}$ $outer_{ep}(c) = \{\forall e \in witness_{ep}(c) \vert (length(difference(e,c)) < length(e)/2) \vee$ $(\exists e_2 \in witness_{ep}(c) : e_2 \neq e \wedge from(e_2) = from(e) \wedge length(difference(e,c) < \sigma_1)\}$ $entrypoints(c) = \{\forall n \vert \exists e \in outer_{ep}(c) : from(e) = n\} \bigcup \{\forall n \vert \exists e \in witness_{ep}(c) \backslash outer_{ep}(c) : to(e) = n\}$

TABLE VI
ADAPTATION 3B

	New concept	Definition
Cell	$parent(c_2)$	the singleton of the cell c_1 that was used to generate c_2 during the iterative top-down cell generation process i.e. the only cell of level $l(c_2)+1$ that contained c_2 before the cell classification and merge processes[a].
	$has\_siblings(c)$	$\Leftrightarrow \exists c_s : parent(c_s) = parent(c) \wedge c_s \neq c$
	$ancestors(c)$	$= parent(c) \cup ancestors(parent(c)); ancestors(\emptyset) = \emptyset$
Node promotion	$early\_accesses(c_a)$	$on\_backlink(n_1,n_2,l,e) \Leftrightarrow e \widetilde{\in} reduced(G_l) : to(e) = n_2 \wedge intersects(n_1,e)$ $has\_forwardlink(n_1,l) \Leftrightarrow \exists e \widetilde{\in} G : from(e) = n_1 \wedge l(e) = l$ $early\_accesses(c_a) = \{(n_c,n_a,e) \vert \exists c_c : c_a \in ancestors(c_c) \wedge n_c \in entrypoints(c_c) \backslash entrypoints(c_a) \wedge$ $n_a \in entrypoints(c_a) \wedge distance(n_c,n_a) < \sigma_2 \wedge \exists e : on\_backlink(n_c,n_a,l(c_a),e)$ $\wedge has\_forwardlink(n_c,l(c_c)) \wedge has\_siblings(c_c)\}$
	$late\_exits(c_a)$	$on\_forwardlink(n_1,n_2,l,e) \Leftrightarrow e \widetilde{\in} reduced(G_l) : from(e) = n_2 \wedge intersects(n_1,e)$ $has\_backlink(n_1,l) \Leftrightarrow \exists e \widetilde{\in} G : to(e) = n_1 \wedge l(e) = l$ $late\_exits(c_a) = \{(n_c,n_a,e) \vert \exists c_c : c_a \in ancestors(c_c) \wedge n_c \in leavepoints(c_c) \backslash leavepoints(c_a) \wedge$ $n_a \in leavepoints(c_a) \wedge distance(n_c,n_a) < \sigma_2 \wedge \exists e : on\_forwardlink(n_c,n_a,l(c_a,e))$ $\wedge has\_backlink(n_c,l(c_c)) \wedge has\_siblings(c_c)\}$
	$entrypoints'(c)$	$= \{n_c \vert \exists (n_c,n_a,e) \in early\_accesses(c)\} \cup \{n_a \vert na \in entrypoints'(c) \wedge \neg(\exists (n_c,n_a,e_1) \in$ $early\_accesses(c) \wedge \neg(\exists e_2 : e_2 \widetilde{\in} reduced(G_{l(c)}) \wedge to(e_2) = n_a \wedge e_1 \neq e_2))\}$
	$leavepoints'(c)$	$= \{n_c \vert \exists (n_c,n_a,e) \in late\_exits(c)\} \cup \{n_a \vert na \in leavepoints'(c) \wedge \neg(\exists (n_c,n_a,e_1) \in late\_exits(c) \wedge \neg(\exists e_2 :$ $e_2 \widetilde{\in} reduced(G_{l(c)}) \wedge from(e_2) = n_a \wedge e_1 \neq e_2))\}$
	$reduced'(G_L)$	$= reduced(G_L, \{(n_c,n_a,e) \vert \exists c : l(c) = L \wedge \exists (n_c,n_a,e) \in early\_accesses(c) \cup late\_exits(c)\})$

[a]As CI and DCW part cells have been merged with regular cells, this function is applied exclusively to regular cells. It is possible that c_1 does not completely contain c_2 anymore or that it has been classified as non-regular and therefore does not exists anymore.

paths of higher cost. It makes the predictions returned by the cost estimate function more inconsistent, and causes transition point conflicts in case the algorithm is called between nodes of cells that have a common neighbouring cell. Consider a set of witness-edges of a leave point n: $\{\forall e \widetilde{\in} reduced(G_{l(c)}) : from(e) = n \wedge to(e) \notin c\}$ and of an entry point n: $\{\forall e \widetilde{\in} reduced(G_{l(c)}) : to(e) = n \wedge from(e) \notin c\}$. A transition point t of cell c is moved when

- *one of its witness-edges has a longer line length outside c than inside c.* In practice, this heuristic measure intercepts any instance of pattern 1. Figure 11b is an example of this correction.
- *there exist witness-edges of other (same type) transition points of c that have the replacement point in common with one of its own witness-edges which has a line length outside c smaller than* σ_1. Figure 11d is an example of this correction. In practice, this measure intercepts a reasonable number of instances of pattern 2. However, there exist cases of pattern 2 that cannot be corrected without generating other anomalous patterns or violating other constraints. Figure 13a shows an example of pattern 2 that requires moving the entry points towards the neighbours of the neighbours of the cell.

This first adaptation can be achieved by introducing a correction step to the preprocessing procedure, or by changing the transition point generation step, as suggested by the new concept definitions in Table V.

Concerning pattern 3, both early accesses and late exits (EA/LE) can be intercepted by moving the transition point to the EA/LE point. An EA/LE node of a cell of level L usually is an intermediate node in G_L and thus not contained in $reduced(G_L)$. Therefore the new transition point can only be used when the intermediate node is reintroduced in $reduced(G_L)$.

An EA/LE pattern is detected by the following heuristic. An EA/LE point typically connects with a cell c_a of level L by a way of a level M, with $0 < M < L$. Moreover, this node of G_M belongs to a CI, which has been merged by a cell c_c during the cell merge procedure, and this point typically is a transition point of c_c. Pattern 3 can efficiently be detected by

matching the transition points n_a of a cell c_a and the transition points n_c of the cells c_c that originally were generated by splitting c_a. These points form an EA(LE) pattern when:

- the distance between n_c and n_a is lower than σ_2,
- n_c is located on an edge in $reduced(G_L)$ ending/starting in n_a,
- there is an edge in $reduced(G_M)$ of exactly level M starting/ending in n_c, and,
- c_c has sibling cells i.e. there must exist a way of level M that crosses c_a.

This pattern covers all instances of pattern 3 identified so far. Figure 12 shows two examples of an EA pattern and their corrections. Table VI shows the extension of preprocessing by an EA/LE correction step. It assumes that from now on the MLHNP algorithm uses the concepts $entrypoints'(c)$, $leavepoints'(c)$ and $reduced'(G_L)$ instead of the concepts without apostrophe.

D. Adaptation 4: improving the transition point selection

1) Problem: Given any shortest path starting and ending in two non-contiguous cells of level L, MLHNP is able to find the shortest path if (a) all of its edges outside the cells belong to G_L, and (b) it chooses the transition points that generate this path. Currently, the transition point choice is based on the minimum of the sum of the cost of the candidate higher level path and the cost estimation(s) of the lower level path (Section III-B). In certain cases, this choice may not be the optimal one, because the cost estimation is inconsistent with the real lower level cost. The inconsistency risk is notably higher when anomalous transition points of pattern 2 are involved. Figure 13e and 13f illustrate the wrong choice of transition point given the transition point situation in Figure 13a. The current adaptation addresses the algorithmic improvement of the transition point choice.

2) Solution and realisation: **Higher level correction** applies when the non-virtual part of any resulting path R in the algorithm in Figure 2 passes any of the promoted transition points. This symptom indicates that another transition point was used to enter or leave the cell border. In this case R is replaced by the path that would result from the choice of the transition point in which the original non-virtual path passes. This intervention has few risks because the lower level routing is able to generate the removed subpath on the cell border.
The applicability of the **lower level correction** depends on the ability of the algorithm behind the basic shortest path calculation to generate shortest paths from or to the other transition points after calculation. If they were all known, MLHNP would not be the preferred hierarchical strategy in general. This correction only applies if the algorithm knows some of them. The Dijkstra algorithm, for example, is able to produce all shortest paths from/to the other transition points that have a lower cost than the path from a to b. Suppose that MLHNP has solved a virtual link at execution level m, that was created at execution level $l > m$. If it detects a shortest path from or to a nearby (distance lower than σ_3) transition point generated at level l with a significantly lower cost than the current solution of the virtual link, the algorithm tracks back to level m, replaces all virtual links by the calculated paths so far and recalculates the shortest path. If the calculated paths still contain virtual links, these are solved at a level lower than m. This backtracking mechanism is illustrated in Figure 13.

V. Experiment

A. Objectives and methodology

The goal of this experiment is to validate the MLHNP approach for a database containing a real road network of a considerable size. In addition, the experiments have been set up to measure the performance for each of the adaptations presented in the previous section on a representative test set. A test set consists of n routing queries represented by a start and end node, which are part of the road network graph G. Performance on a test set is expressed in terms of averages or percentiles of the following criteria for the routing operations generated by all queries in the test set. Note that a single MLHNP run may consist of several recursive runs.

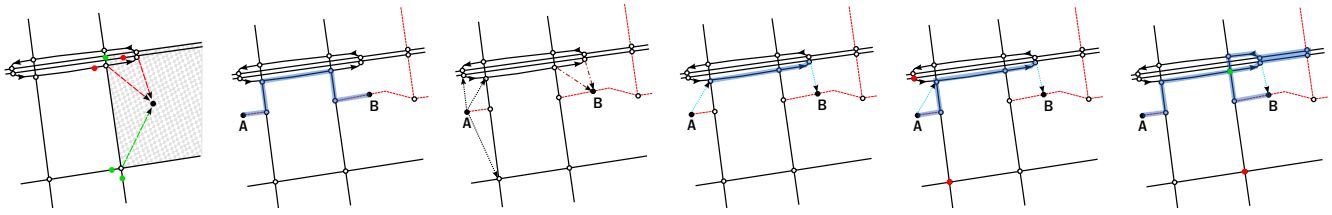
- The **relative error**, as defined in Table I. It reflects the quality of the resulting route. The number of optimal solutions found is equal to the number of 0.00% deviations in the test set.
- The **number of recursive runs.**
- The **number of loaded nodes** is the sum of the number of graph nodes that are loaded from the database over all recursive runs.
- The **number of visited nodes** is the sum of the number of graph nodes for which the basic routing algorithm has calculated a cost, over all recursive runs. If the same node has been visited in n runs, it is counted n times. It indicates the search space reduction of our approach.
- Two criteria depend on the hardware and software infrastructure[3] on which the routing is executed. The **calculation time** is the sum of the basic routing algorithm's computation time over all recursive runs. It is strongly related to the number of visited nodes. The **memory footprint** is the maximal amount of memory consumed by the routing process. Since most of the loaded data is flushed before new data is loaded in a new run, this criterion only depends on the most memory-intensive run.

A test set of routing queries was generated as follows. First, an orthonormal grid was created over the routable area. The grid distribution is assumed representative for routing for tourism and leisure purposes. Then, the generator creates start and end nodes by selecting random grid points. Once a node has been picked, it cannot occur in any other query. Before routing, the grid nodes are replaced by the closest node in the road network graph.

B. Geographical dataset

The network studied here is the time-weighted road network of Belgium, divided in 5 hierarchical levels. The edge weights

[3]All tests have been executed using PHP (Command Line Interface) 5.3.3 and a PostGis database on a Debian 6.0 environment with an Intel Xeon E5520 @ 2.27GHz processor.



(a) Anomalous pattern (b) Shortest path and resulting path after correction (c) Virtual links (d) Path on high level (e) Low level routing at front with negative backtracking analysis (f) Low level routing at back with positive backtracking analysis

Fig. 13. Example of an uncovered anomalous transition point pattern (leftmost subfigure) and the backtracking mechanism. The basic legend is in Table II. In the leftmost subfigure, arrows represent the virtual links created from any of the entry points to a point within the cell. Green arrows refer to correct and red to incorrect transition points. The dots mark the witness-edges of the transition points. In the two rightmost subfigures, red/green dots indicate transition points subject to a positive/negative backtracking analysis.

were set as the edge distance multiplied by a discrete speed class value. The test set used in this experiment contains 1000 routing queries, generated from an orthonormal grid (approx. 4 km) superimposed on the routable area. The road network graph consists of 1190220 directed edges and 526253 nodes. Preprocessing results in 1357/515/211/37 cells of level 1/2/3/4 respectively, after a cell merge with a reduction rate of ca. 91%. The total numbers of transition points per level are 22693/8493/3296/572 (22564/8390/3175/466 after Adaptation 3).

C. Basic wayfinding and MLHNP settings

MLHNP applied Dijkstra as the basic shortest path algorithm in this experiment. In order to speed up the search process, we use a rectangular pruning area parallel to the axes of the coordinate system (Karimi et al. [13]) supported by the indexing structure of the spatial DBMS. As the infrastructure diagram in Figure 1 may suggest, each basic shortest path calculation is preceded by a fetch of all links of G' that start within the rectangular pruning area from the database. The size and position of this area is determined by the following rules.

- Construct the rectangle formed by start node a and end node b.
- Extend the rectangle such that it contains any virtual edge in G'.
- Extend the rectangular by $e\%$ equivalent meridian degree in both directions.
- If the rectangle is too elongated for the current level i.e. $min(breadth, height)/max(breadth, height) < min(r_{level} \cdot (l_{exec} \cdot 2 + 1), r_{max})$, elongate the smallest sides such that this condition does not hold.
- When the Dijkstra algorithm returns an infinite cost, it means that the pruned area does not contain any path from a to b. In this case, the rectangular selection is extended by $e\%$ digital equivalent meridian degree again. After a cost reparation process, the Dijkstra algorithm is continued. This extension is repeated until the algorithm finds a solution.
  In this experiment, the road network graph G (and also any directed subgraph G_L) is not a strongly connected graph. Therefore, the number of repetitions should be limited. When this number is reached in any of the recursive runs, MLHNP is halted and does not generate a solution, except in Adaptation 4. In this version, a subcall to the basic shortest path calculation is able to return an infinite cost path which is taken into account for a lower level correction.

This heuristic approach tends to minimise the probability that the path with the lowest cost is not located in the rectangle. It is tailored to the hierarchical approach. Two rectangular pruning settings are tested during the experiment: the *small box* setting with $e = 6$, $r_{level} = 0.1$ and $r_{max} = 0.6$ and the *large box* setting with $e = 12$, $r_{level} = 0.2$ and $r_{max} = 1$.
MLHNP's average resistance r_{avg} has been set to 4. The settings of Adaptation 3 and 4 are $\sigma_1 = 3km$, $\sigma_2 = 1km$ and $\sigma_3 = 2.5km$.

D. Results

Table VII shows the routing performance after the stepwise introduction of each of the MLHNP adaptations over the testset presented in Section V-B, for both rectangular pruning settings. The optimal costs (on which the error rate is based) have been obtained by the application of a Dijkstra algorithm without pruning involved. Three out of 1000 queries in the test set have start and end nodes that are not connected in the road network graph. These have not been taken into account in the statistics, neither have the queries that failed because one of the MLHNP recursive runs could not find a finite path. Their numbers are given in the first row. The large box pruning setting yield lower error rates than the small box setting, at the expense of the average number of visited nodes. As expected, larger boxes extend the search space but small boxes may cut away solutions of lower cost. Below, only the large box pruning numbers are given.
Three out of 997 queries failed in the case of **Adaptation 1**. Each of these three queries fails at the top recursion with execution level L_{top}. The queries have a start/end node located on a motorway, which, in $reduced(G_{L_{top}})$, only connects to a long edge ending/starting at the data set border, whereas paths from the start node to the rest of the graph exist in G. This issue is fixed by the next adaptation. Adaptation 1 yields a trade-off between an average number of visited nodes of 5239.80 and an error rate of 9.44%.

TABLE VII
EXPERIMENT RESULTS. ANY AVERAGE/PERCENTILE NUMBER IS ONLY BASED ON THE FOUND SOLUTIONS.

	Small box pruning				Large box pruning			
	Adapt. 1	**Adapt. 2**	**Adapt. 3**	**Adapt. 4**	**Adapt. 1**	**Adapt. 2**	**Adapt. 3**	**Adapt. 4**
# solutions not found	3	0	1	0	3	0	1	0
1% pctl calculation time (s)	0.027	0.036	0.037	0.039	0.033	0.040	0.042	0.050
avg calculation time (s)	0.161	0.169	0.158	0.173	0.259	0.279	0.263	0.295
99% pctl calculation time (s)	0.732	0.727	0.563	0.566	1.051	1.081	0.900	1.057
avg memory footprint (MB)	19.286	19.496	19.081	18.342	28.830	29.347	29.296	28.970
avg # routing runs	4.243	4.408	4.514	5.848	4.2263	4.394	4.503	5.821
avg # loaded nodes	6966.03	7337.83	7461.49	8621.27	16019.19	16831.94	17305.26	20526.69
1% pctl # visited nodes	628.30	852.84	884.50	924.88	864.00	1050.56	1065.70	1173.72
avg # visited nodes	3591.65	3773.92	3660.50	4123.67	5239.80	5498.72	5401.31	6103.70
99% pctl # visited nodes	10902.66	11018.52	9607.15	10192.68	15256.91	15719.72	14617.90	15990.08
# exact solutions	137	145	192	240	167	175	234	300
avg error rate	11.85%	11.08%	8.81%	7.43%	9.44%	8.73%	6.47%	4.91%
99% pctl error rate	77.76%	66.89%	58.09%	52.74%	63.58%	53.05%	50.11%	36.82%
# error decrease / increase	-	72 / 25	300 / 117	274 / 23	-	63 / 23	301 / 114	291 / 17

The application of **Adaptation 2** improves the solution quality by 8.73%. The node promotion condition relaxation introduces an extra number of recursive runs, reflected in the average number, leaving the other runs roughly as they are. This causes a higher number of visited and loaded nodes. Some individual solutions show a higher error rate because a higher number of node promotions increases the risk of choosing a wrong transition point.

Adaptation 3 is the most effective adaptation. It lowers the number of visited nodes, whereas the error rate is reduced to 6.47%. Most of the transition point corrections move the points away from their cells. This could explain the higher average number of runs[4] and loaded nodes. Before the introduction of Adaptation 3, malicious transition point locations often caused detours in the lower level paths. The elimination of these detours probably decreases the number of visited nodes. For 1 query, an entry point was selected that only connects to a motorway dead end in G, making the query fail.

Adaptation 4 further reduces the error rate to 4.91%. Higher level correction generates a higher average number of runs (for the same reason as Adaptation 3). Lower level correction introduces an extra number of reparation runs. Therefore, both corrections increase the average number of loaded and visited nodes. Some queries generate worse solutions for this adaptation than for the previous one. This occurs in case of higher level correction because the risk of choosing a wrong transition point at the lower level increases. It also occurs in case of lower level correction when the replacing path contains again a virtual link, which has an estimate-based cost.

VI. CONCLUSION

We have introduced the MLHNP algorithm, enabling finding the shortest path in hierarchically organised road network graphs, while supporting several types of edge weights. The authors of the two-level version of this algorithm have suggested a graph transformation in order to cope with dual carriageways and slip roads in real road networks. This transformation does not fully comply with the requirements of contemporary routing applications.

Therefore, four adaptations of the MLHNP approach have been proposed that bypass the graph transformation. The first adaptation introduces a cell classification and merge during preprocessing. The classification method can be more generally used for functional road class prediction in GIS. The hierarchical routing algorithm uses the resulting cell set to emulate contiguity and transition points as if the network graph were subject to the original graph transformation. In the here presented application domain, an effective cell merge is to be preferred over cell grouping because it reduces storage costs and optimises the routing data indexation. Adaptation 2 makes the algorithm take into account the local context of dual carriageways and slip roads when the start or end node is located on these types of way. The third adaptation fine-tunes the transition point locations of the first one. A last adaptation enables the algorithm to revise its transition point choices throughout the recursion tree. It applies to hierarchical node promotion algorithms in general.

The adapted MLHNP approach has been designed for a web application that uses a spatial database and that enables hierarchical routing for several routing modi. Here, database operations have a high impact on routing performance. Both storage costs and calculation time of preprocessing are restricted.

The MLHNP algorithm has been successfully applied to a medium-sized hierarchical road network with time-weights, using Dijkstra and rectangular pruning for individual routings. The experiment showed the effectiveness of each of the adaptations. The combined MLHNP and pruning approach achieves a low average number of visited nodes, maintaining an average relative error of 4.91%.

[4]For instance, suppose a front-side virtual link from s_{global} to $lp \in leavepoints'(c)$, which is selected in path R. When lp is moved out of c, the level of execution in the recursive run $mlhnp(s_{global}, lp, l_{restr}, G)$ is on average higher, because on level l_{restr}, the (previously same) cells of s_{global} and lp become neighbours.

REFERENCES

[1] G. R. Jagadeesh, T. Srikanthan, and K. H. Quek, "Heuristic techniques for accelerating hierarchical routing on road networks." *IEEE Transac-*

tions on Intelligent Transportation Systems, vol. 3, no. 4, pp. 301–309, 2002.

[2] J. Gao, R. Jin, J. Zhou, J. X. Yu, X. Jiang, and T. Wang, "Relational approach for shortest path discovery over large graphs," *PVLDB*, vol. 5, no. 4, pp. 358–369, 2011.

[3] B. F. Zhan and C. E. Noon, "Shortest path algorithms: An evaluation using real road networks," *Transportation Science*, vol. 32, no. 1, pp. 65–73, February 1998.

[4] E. W. Dijkstra, "A note on two problems in connexion with graphs." *Numerische Mathematik*, vol. 1, pp. 269–271, 1959.

[5] P. E. Hart, N. J. Nilsson, and B. Raphael, "A formal basis for the heuristic determination of minimum cost paths," *IEEE Transactions On Systems Science And Cybernetics*, vol. 4, no. 2, pp. 100–107, 1968.

[6] R. Jacob, M. Marathe, and K. Nagel, "A computational study of routing algorithms for realistic transportation networks," *J. Exp. Algorithmics*, vol. 4, p. 6, 1999.

[7] J. L. Bander and C. C. White III, "A heuristic search algorithm for path determination with learning," *IEEE Transactions on Systems, Man, and Cybernetics - Part A: Systems and Humans*, vol. 28, no. 1, pp. 131–134, 1998.

[8] J. L. Adler, "A best neighbor heuristic search for finding minimum paths in transportation networks," *Transportation research record*, vol. 1651, pp. 49–53, 1998.

[9] A. V. Goldberg and C. Harrelson, "Computing the shortest path: A search meets graph theory," in *Proceedings of the sixteenth annual ACM-SIAM symposium on Discrete algorithms*, ser. SODA '05. Philadelphia, PA, USA: Society for Industrial and Applied Mathematics, 2005, pp. 156–165.

[10] F. Hahne, C. Nowak, and K. Ambrosi, "Acceleration of the a*-algorithm for the shortest path problem in digital road maps," in *OR*, J. Kalcsics and S. Nickel, Eds. Springer, 2007, pp. 455–460.

[11] L. Fu, "Real-time vehicle routing and scheduling in dynamic and stochastic traffic networks," Ph.D. dissertation, University of Alberta, Edmonton, Alberta, 1996.

[12] H. A. Karimi, "Real-time optimal route computation: a heuristic approach," *ITS Journal*, vol. 3, no. 2, pp. 111–27, 1996.

[13] H. A. Karimi, P. Sutovsky, and M. Durcik, "Accuracy and performance assessment of a window-based heuristic algorithm for real-time routing in map-based mobile applications," in *Map-based Mobile Services*, ser. Lecture Notes in Geoinformation and Cartography, L. Meng, A. Zipf, and S. Winter, Eds. Springer Berlin Heidelberg, 2008, pp. 248–266.

[14] D. Wagner and T. Willhalm, "Geometric speed-up techniques for finding shortest paths in large sparse graphs," in *Proceedings of Algorithms - ESA 2003, 11th Annual European Symposium, Budapest, Hungary, September 16-19, 2003, Lecture Notes in Computer Science*, G. D. Battista and U. Zwick, Eds., vol. 2832. Springer, 2003, pp. 776–787.

[15] U. Lauther, "An extremely fast, exact algorithm for finding shortest paths in static networks with geographical background," *IfGI prints, Institut für Geoinformatik, Universität Münster*, vol. 22, pp. 219–230, 2004.

[16] R. J. Gutman, "Reach-based routing: A new approach to shortest path algorithms optimized for road networks," in *Proceedings of the Sixth Workshop on Algorithm Engineering and Experiments and the First Workshop on Analytic Algorithmics and Combinatorics, New Orleans, LA, USA, January 10, 2004*, L. Arge, G. F. Italiano, and R. Sedgewick, Eds. SIAM, 2004, pp. 100–111.

[17] A. V. Goldberg, H. Kaplan, and R. F. Werneck, "Reach for A*: Efficient point-to-point shortest path algorithms," in *Proceedings of the eighth Workshop on Algorithms Engineering and Experiments*, vol. MSR-TR-200. Society for Industrial and Applied Mathematics, 2006, pp. 129–143.

[18] A. Car and A. Frank, "General principles of hierarchical spatial reasoning: The case of wayfinding," in *Proceedings of the 6th International Symposium on Spatial Data Handling*. Taylor and Francis, September 1994.

[19] L. Fu, D. Sun, and L. R. Rilett, "Heuristic shortest path algorithms for transportation applications: state of the art," *Computers and Operations Research*, vol. 33, no. 11, pp. 3324–3343, 2006.

[20] B. Liu, "Route finding by using knowledge about the road network," *IEEE Transactions on Systems, Man, and Cybernetics–Part A: Systems and Humans*, vol. 27(4), pp. 436–448, 1997.

[21] G. Dantzig, "On the shortest route through a network," *Management Science*, vol. 6, pp. 187–190, 1960.

[22] P. Sanders and D. Schultes, "Highway hierarchies hasten exact shortest path queries," in *Algorithms ESA 2005*, ser. Lecture Notes in Computer Science, G. S. Brodal and S. Leonardi, Eds. Springer Berlin Heidelberg, 2005, vol. 3669, pp. 568–579.

[23] ——, "Engineering highway hierarchies," in *Proceedings of the 14th conference on Annual European Symposium - Volume 14*, ser. ESA'06. London, UK, UK: Springer-Verlag, 2006, pp. 804–816.

[24] R. Geisberger, P. Sanders, D. Schultes, and D. Delling, "Contraction hierarchies: faster and simpler hierarchical routing in road networks," in *Proceedings of the 7th international conference on Experimental algorithms*, ser. WEA'08. Berlin, Heidelberg: Springer-Verlag, 2008, pp. 319–333.

[25] H. Bast, S. Funke, P. Sanders, and D. Schultes, "Fast routing in road networks with transit nodes," *Science*, vol. 316, no. 5824, p. 566, 2007.

[26] D. Schultes and P. Sanders, "Dynamic highway-node routing," in *Experimental Algorithms*, ser. Lecture Notes in Computer Science, C. Demetrescu, Ed. Springer Berlin Heidelberg, 2007, vol. 4525, pp. 66–79.

[27] Y.-L. Chou, H. E. Romeijn, and R. L. Smith, "Approximating shortest paths in large-scale networks with an application to intelligent transportation systems," *INFORMS Journal on Computing*, vol. 10, no. 2, pp. 163–179, 1998.

[28] H.-J. Cho and C.-L. Lan, "Hybrid shortest path algorithm for vehicle navigation," *J. Supercomput.*, vol. 49, no. 2, pp. 234–247, 2009.

[29] C. Strauss, "A note on hierarchical routing algorithms based on traverse-oriented road networks," *International Journal of Spatial Data Infrastructures Research*, vol. 4, pp. 239–264, 2009.

[30] M. Delafontaine, G. Nolf, N. van de Weghe, M. Antrop, and P. de Maeyer, "Assessment of sliver polygons in geographical vector data," *Int. J. Geogr. Inf. Sci.*, vol. 23, no. 6, pp. 719–735, 2009.