Automatic Generation of Meaningful Virtual Direction Markers in Road Networks

Jörg Roth

Department of Computer Science, Nuremberg Institute of Technology,
Kesslerplatz 12, 90489 Nuremberg, Germany
Joerg.Roth@th-nuernberg.de

Abstract. Direction markers are widely used in real road networks. They show directions to important destinations such as cities, industrial areas, airports or touristic sights. In this paper, we introduce *virtual direction markers* that only exist digitally in, e.g., a navigation application. We present an approach to automatically compute and place such markers without additional manual editing. Our assumption: a direction marker resides on an optimal path from an arbitrary start to the respective target. However, a naïve approach would place confusing or misleading markers. Thus, we add the concept of *cropping* that removes unwanted markers. Finally, we introduce the notion of *relevant crossings* and *default links* that additionally improve direction markers.

Keywords: Road network, Direction marker, Navigation

1 Introduction

Direction markers in road networks are widely known (Fig. 1). They indicate highway exits or suggest turning at a crossing for a certain destination. Real direction markers are placed by, e.g., road traffic departments. These have local knowledge about interesting destinations (e.g. cities, important sights) and know useful locations to place these signs [9]. In addition to standardized traffic signs, we also observe traditional and historic signs, very often for pedestrians [8].

In this paper, we introduce *virtual direction markers* that only appear as data in, e.g., a navigation application. Our virtual direction markers are automatically computed from an existing road network, without additional manual editing. This enables to compute meaningful direction markers even on large road networks with many millions of links and crossings.

Virtual direction markers are useful for several applications:

- For a trip that is guided by a navigation application, direction markers placed on a map may indicate a crossing situation more clearly.
- For non-guided, explorative driving, meaningful directions markers may show interesting new destination options for the driver.

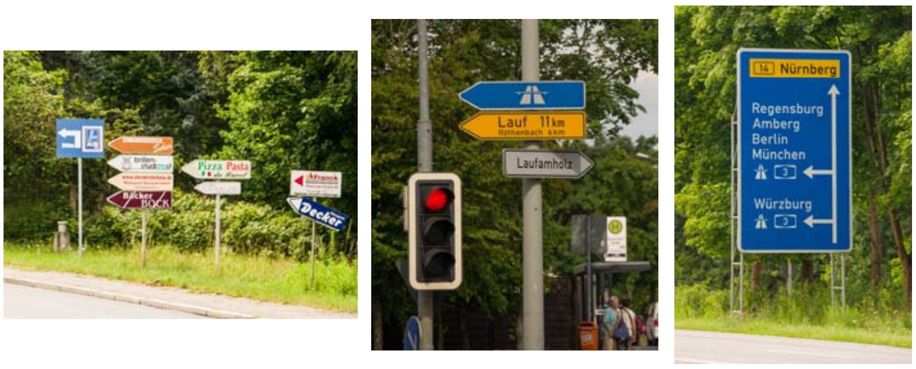

Fig. 1. Examples of real direction markers

- An automatically generated driver's logbook may contain textual trip entries that are sequences of turns and straight driving. The turns may be expressed by direction markers, e.g. "*10 km straight, turned left to 'Nuremberg', 5 km straight, turned right to 'New Museum of Arts'*...".

In addition, such markers can be placed on automatically generated maps that e.g. are printed.

We distinguish *confirmative* und *distinctive* direction markers. The difference is, whether the driver is confirmed to stay on the current route or whether the crossing distinguishes between different targets and tells a driver to turn or exit. Typical *confirmative* markers are signs placed on highways that indicate, the distance to certain city '*straight ahead*' is, e.g., 100 km. In contrast, *distinctive* direction markers require a driver to make a certain decision, e.g. to turn left for *city A*, or to turn right for *city B*. For a typical application, distinctive markers are more interesting.

Our goal is to compute a set of *confirmative* and *distinctive* direction markers for *every* crossing in the road network and for *any* interesting destination. In addition to the algorithmic problem, we have to face some questions to perform this task:

- What are suitable destinations?
- At which crossing are direction markers useful for a certain destination?
- How do we formally separate distinctive from confirmative markers?

As the computation may be part of a mastering step of geo data (e.g. for a navigation application) it is not time-critical. However, an approach must be suitable for many millions of crossings and destinations.

The following approach is based on a long research in the area of untraditional navigation problems. Corresponding solutions are integrated into the *HomeRun* platform [2], more specifically into the *donavio* component for navigational functions [3]. Besides the traditional point-to-point route planning, donavio covers a wide range of further solutions:

- computation of all alternative routes that hold a certain cost limit [7],
- computation of suitable stop-over-points of a certain type (e.g. fuel station) that downgrade the optimal route not too much [7],
- computation of $m \times n$ optimal routes between m starts and n targets without the need to compute all $m \times n$ individual paths [5],
- computation of traveling salesman routes on real road networks (i.e. the distance between stop-over-points are not approximated, but real costs) [5],
- computation of isochrones, i.e. the area of targets that can be reached within a given cost limit; also multi-isochrones (i.e. from multiple starts),
- prediction of a target from a partially observed route [4],
- prediction of a driven route from imprecise position sensors based on the optimal route assumption [6].

As a new donavio function, we want to enrich the road network to carry direction information.

2 Computing Virtual Direction Markers

2.1 A First Approach

We start with some definitions. Let V denote the set of crossings and E the set of (directed) links between crossings. Further let $out(v)$ denote all outgoing links from and $in(v)$ all incoming links to a crossing v.

For an $e \in E$, $v_{begin}(e)$ denotes the start crossing, $v_{end}(e)$ the end crossing. We simply write $e=(v_{begin}(e), v_{end}(e))$. In addition we call $\overline{e}=(v_{end}(e), v_{begin}(e))$ the *reverse link* of e.

Links have costs, denoted by $c(e)$ or $c(v_1, v_2)$. Costs are the measure that a certain direction wants to minimize. In other words: following a certain direction marker, the total costs to the direction is less than going via another direction. Typical costs are driving time or driving distance.

Finally, we need a set of interesting direction targets, we call *landmarks*. Let M denote the set of all landmarks. We assume M neither is a subset of E nor V, but there is an obvious relation $lm(m) \subset V$ for $m \in M$ that maps a landmark m to a subset of crossings. For area-like landmarks (e.g. cities), a reasonable lm identifies all crossings within the surface area of m.

We are now looking for all direction markers for all links. For a certain link e we call $dir(e) \subseteq M$ the set of these markers. We can express the basic idea of the first approach to compute $dir(e)$ as follows:

All direction markers to a landmark reside on an optimal route from a virtual starting point to this landmark.

Here, '*virtual starting point*' means: we can choose any crossing that has no actual relation to the landmark or to the respective link. We also do not actually drive from this starting point. However, all links of the corresponding route are directed to the landmark, thus can create our direction markers.

A naïve approach would compute optimal routes from all crossings to all landmarks. This however, would be too time-consuming. A first simplification is thus: we reverse the direction of route computation and start from the landmarks. Important: we do not reverse the driving direction, but only the ordering of visiting the links. This in particular is important, as links in road networks are directed and have to respect one-ways or different speed limits for different directions. To compute all links to the landmark we use a Dijkstra-like approach [1].

A second simplification: we limit the range for directions. Only within a range from a landmark, we assign direction markers. We, e.g., would not expect a direction marker 'Rome in 2000 km', but only within 500 km. We call the maximum range *maxC*(*m*) – it depends on the respective landmark.

An algorithm to compute *dir*(*e*) for each $e \in E$ can be sketched as follows:

```
Algorithm compute_dir(V, E, M, c, lm, maxC) → dir

for each e∈E  {  dir[e]={};  }
for each m∈M  {
    create empty arrays g, state, backLink, d;
    for each v∈V  {  backLink[v]←undef;  }
    for each v∈lm(m)  {  g[v]←0;  state[v]←open;  }
    for each v∉lm(m)  {  g[v]← -1;  state[v]←not_visited;  }
    for each e∈E  {  d[e]←undef;  }
    openList←lm(m);
    while not openList.isEmpty()  {
        v←openList.poll();          // get crossing with state open and minimal g
        if g[v]>maxC(m) then break;  // limit of interest reached
        state[v]←closed;
        for all e=(v, v_n)∈out(v) with state[v_n]≠closed {
            g_new←g[v]+c(v, v_n);
            if state[v_n]=not_visited or g_new<g[v_n]  {
                g[v_n]←g_new;  backLink[v_n]←v;
                state[v_n]←open;  openList.add(v);
                d[e]←m;
            }
        }
    }
    for each e∈E with d[e]≠undef  {  dir[e].add(d[e]);  }
}
```

The *openList* is always sorted by *g*, to quickly get the crossing with minimal *g*. Possible data structures for these are Fibonacci Heap or Priority Queue.

The algorithm above computes only the directions. If we additionally request the distance to the landmark, we could extend the data structure of *d* (*dir* respectively) to also store distance information. E.g., we could change *d*[*e*]=*m* to *d*[*e*]=(*m*, g_{new}).

2.2 Improving this Approach, Cropping

Even though the majority of directions are reasonable, some assignments are misleading, unexpected or at least not helpful. We can distinguish the following problems as illustrated in Fig. 2.

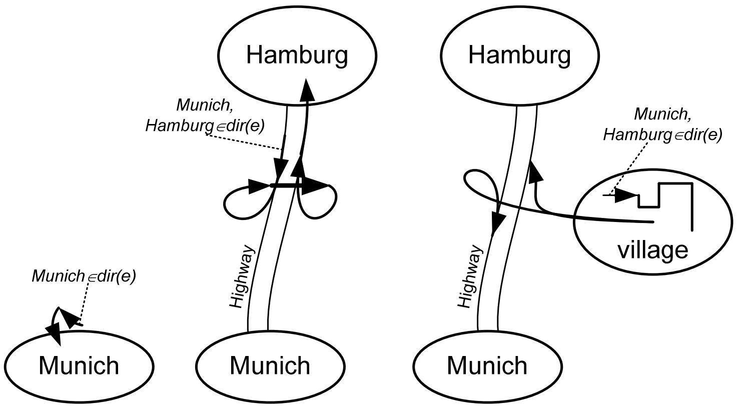

Fig. 2. Problems with the naïve approach

- Fig. 2 left: If we are very close to the landmark ('Munich'), directions are confusing. The problem is even worse, if we are close and the respective link points away from the landmark – even if this was the optimal path.
- Fig. 2 middle: A starting link could point away from the landmark ('Hamburg'), but it could still be on the optimal path, as some further links could perform a 180° turn. This is very common on highways. Consider the highway to 'Munich' shortly before an exit. This link also is the shortest path to the opposite direction 'Hamburg' as an optimal path would exit, cross the highway and enter the highway in the other direction.
- Fig. 2 right: A starting link could be in countryside region ('village'). However, such links tend to be a starting point to nearly any landmark.

The approach to solve these problems is to cut all virtual routes to the landmark both from the virtual start and from the target. We call this mechanism *cropping*. This means from each virtual route, we remove a certain amount of length from the start and landmark and only place direction markers, if they are sufficiently far away from these (Fig. 3).

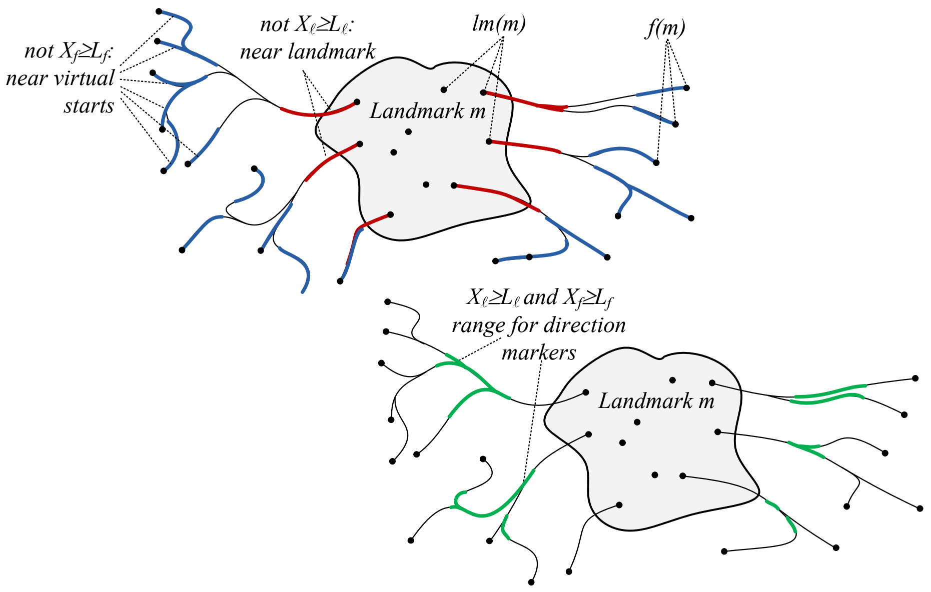

Fig. 3. The idea of cropping

It turned out that a similar mechanism such as the *maxC* to express distances is not suitable, as we have to express much more complex notions of *distance*. We e.g. want to express *'we have to drive at least 1 km on country roads from the start'*. For this, we introduce two application-dependent metric vectors $X_\ell=(x_{\ell 1},\dots x_{\ell n})$, $X_f=(x_{f1},\dots x_{fn})$ that formalize the desired distance properties. Here X_ℓ describes the metric value going from the landmark, X_f from the (virtual) starting point. Later, each crossing on a virtual route carries an individual pair of (X_ℓ, X_f).

To increase the metric value if we pass a link, we need a function

$$hop: E \times X \to X;\ e, (x_1,\dots x_n) \to (y_1,\dots y_n) \qquad (1)$$

We require *hop* to be monotone, i.e.

$$y_i \geq x_i \text{ for all } i \qquad (2)$$

Finally, we need two limits L_ℓ, L_f. Later, we consider a crossing to be *'not near the landmark'*, if $X_\ell \geq L_\ell$, and *'not near the virtual start'*, if $X_f \geq L_f$. Here ≥ denotes pair wise comparison, i.e.

$$\begin{pmatrix} x_{a1} \\ x_{a2} \\ \dots \\ x_{an} \end{pmatrix} \geq \begin{pmatrix} x_{b1} \\ x_{b2} \\ \dots \\ x_{bn} \end{pmatrix} \text{ iff } \forall i : x_{ai} \geq x_{bi} \qquad (3)$$

To give an example: let be X_ℓ, X_f vectors of three components (x_1, x_2, x_3) with

- x_1 is the total amount of meters of the route from landmark or start respectively,
- x_2 is the amount of meters going over country roads or higher classified roads,
- x_3 is the amount of meters going over municipal road or lower classified roads.

We consider crossing '*not near the landmark*', if it is more than 5 km away and the route to the landmarks goes at least over one country road (or higher class). We consider a crossing '*not near the virtual start*', if it is more than 1 km from the start and we had to drive at least 100 m over municipal roads (or lower class). We express this by

$$L_\ell = \begin{pmatrix} 5000 \\ \varepsilon \\ 0 \end{pmatrix} \quad L_f = \begin{pmatrix} 1000 \\ 0 \\ 100 \end{pmatrix} \tag{4}$$

whereas ε is the smallest distance greater than zero. Note that these definitions for X_ℓ, X_f, L_ℓ and L_f are our suggestion for landmarks such as cities or villages.

To effectively compute X_f, we later require the set of virtual start crossings denoted by f. These are crossings that have a *backLink* entry, but to do appear themselves as *backLink* endpoint, i.e.

$$f(m) = \{\, v \mid backLink[v] \neq undef \,\} \setminus \{\, backLink[v] \mid backLink[v] \neq undef \,\} \tag{5}$$

Note that a *backLink* always points to the landmark.

We now have all tools to compute the X_ℓ and X_f. The computation of X_ℓ can be performed '*on the fly*' when visiting all crossings up to costs *maxC*. For X_f we have to face a problem: a certain crossing may be on an optimal route from *multiple* virtual starts. We thus have to aggregate a single X_f from multiple routes. Our approach is, to choose the maximum value of the two routes, whereas max is defined

$$\max\left(\begin{pmatrix} x_{a1} \\ x_{a2} \\ \dots \\ x_{an} \end{pmatrix}, \begin{pmatrix} x_{b1} \\ x_{b2} \\ \dots \\ x_{bn} \end{pmatrix} \right) = \begin{pmatrix} \max(x_{a1}, x_{b1}) \\ \max(x_{a1}, x_{b1}) \\ \dots \\ \max(x_{an}, x_{bn}) \end{pmatrix}. \tag{6}$$

Fig. 4 illustrates the mechanism.

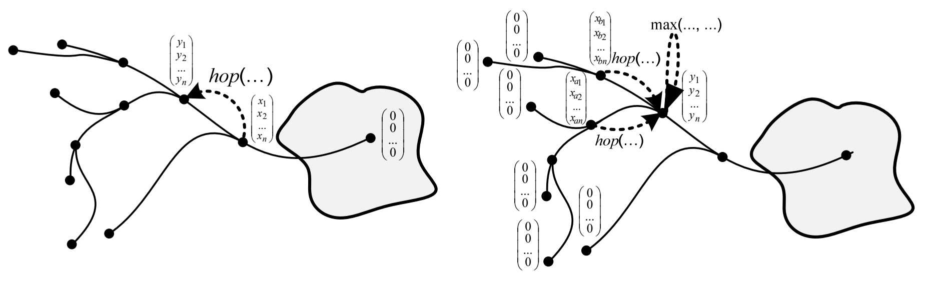

Fig. 4. Updating X_ℓ (left) and X_f (right)

Now we are able to present the improved algorithm that computes virtual direction markers with *cropping* (modifications to the naïve algorithm are printed **bold**):

```
Algorithm $compute\_dir_{\ell f}(V, E, M, c, maxC, \mathbf{hop}, \mathbf{L_\ell}, \mathbf{L_f}) \rightarrow dir$

for each $e \in E$ { $dir[e]=\{\}$; }
for each $m \in M$ {
    create empty arrays $g$, $state$, $backLink$, $d$, $\mathbf{x_\ell}$, $\mathbf{x_f}$;
    for each $v \in V$ { $backLink[v] \leftarrow undef$; $\mathbf{x_f[v] \leftarrow undef}$; }
    for each $v \in lm(m)$ { $g[v] \leftarrow 0$; $state[v] \leftarrow open$;    $\mathbf{x_\ell[v] \leftarrow (0,\ldots,0)^T}$; }
    for each $v \notin lm(m)$ { $g[v] \leftarrow -1$; $state[v] \leftarrow not\_visited$; $\mathbf{x_\ell[v] \leftarrow undef}$; }
    for each $e \in E$ { $d[e] \leftarrow undef$; }
    $openList \leftarrow lm(m)$;
    while not $openList.isEmpty()$ {
        $v \leftarrow openList.poll()$;        // get crossing with state open and minimal $g$
        if $g[v] > maxC$ then break;  // limit of interest reached
        $state[v] \leftarrow closed$;
        for all $e=(v, v_n) \in out(v)$ with $state[v_n] \neq closed$ {
            $g_{new} \leftarrow g[v]+c(v, v_n)$;
            if $state[v_n]=not\_visited$ or $g_{new}<g[v_n]$ {
                $g[v_n] \leftarrow g_{new}$; $backLink[v_n] \leftarrow v$;
                $state[v_n] \leftarrow open$; $openList.add(v)$;
                $\mathbf{x_\ell[v_n] \leftarrow hop(e, x_\ell[v])}$;
                $d[e] \leftarrow m$;
            }
        }
    }

    compute $\mathbf{f(m)}$;
    for each $\mathbf{v \in f(m)}$ {
        $\mathbf{x_f[v] \leftarrow (0,\ldots,0)^T}$;
        loop {
            $\mathbf{v_n \leftarrow backLink[v]}$;
            $\mathbf{x_{fnew} \leftarrow hop((v,vn), x_f[v])}$;
            if $\mathbf{x_f[vn]=undef}$ then $\mathbf{x_f[v_n] \leftarrow x_{fnew}}$;
                                 else $\mathbf{x_f[v_n] \leftarrow \max(x_f[v_n], x_{fnew})}$;
            if value of $\mathbf{x_f[v_n]}$ not changed then break loop;
            $\mathbf{v \leftarrow v_n}$;
        }
    }
    for each $e=\mathbf{(v, v_n)} \in E$ with $d[e] \neq undef$ and $\mathbf{x_\ell[v_n] \geq L_\ell}$ and $\mathbf{x_f[v] \geq L_f}$
    { $dir[e].add(d[e])$; }
}
```

Whereas the computation of X_ℓ is integrated into the main loop, X_f has to be computed after each landmark iteration. However, the computation still is efficient, as it linearly depends on the visited crossings and only loops through the crossings within *maxC*.

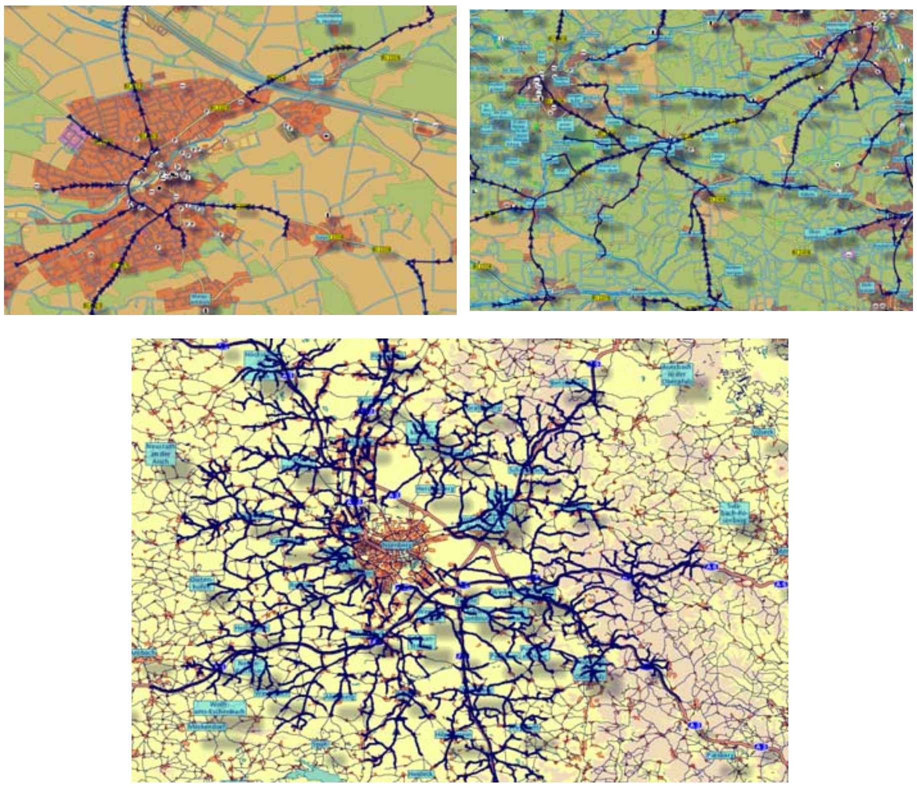

Fig. 5. Range of links considered for direction markers for the landmark 'Nuremberg'

Fig. 5 illustrates the range of links considered as direction markers for a certain landmark, i.e. links that hold $x_\ell[v_n] \geq L_\ell$ and $x_f[v] \geq L_f$ in the algorithm above.

2.3 Distinctive Direction Markers, Relevant Crossings

Up to now, we compute *all* directions for a link, even if they are confirmative, i.e. only encourage the driver to stay on the current route to find the landmark. We now want to switch to distinctive direction markers that push the driver to make a decision. For this, we introduce two concepts:

- *relevant crossings*: these are crossings that are considered to be appropriate for a certain landmark;
- *default links*: these are links that are considered to be walking or driving *straight ahead*, in particular not *turn* or *exit*.

Relevant crossings are represented by a relation $rel \subset V \times M$, where $rel(v, m)$ means: the crossing v is relevant to be a position for a direction marker to landmark m. The relation *rel* is application-dependent. As an example: a crossing in an industrial area is probably not suitable for direction markers to touristic sights. We can easily set up a relation *rel* based on the road network and further digital map data.

As a second concept, we need *default links*. Let $def(e)$ denote the default link for a link $e \in E$. The default link is an outgoing link to the incoming link e, but not $\bar{e}$, i.e.

$$def(e) \in out(v_{end}(e)) \setminus \{ \bar{e} \} \tag{7}$$

Similar to *rel*, *def* is application-dependent. This means, we cannot compute *def* solely from the road topology, but we have to introduce certain rules to identify the one outgoing link that is considered to drive '*straight ahead*'. These rules may take into account

- the road geometry, in particular the incoming angle to and outgoing angle from the crossing;
- road types, e.g. highway, exits, road classifications;
- priority rules, rights of way;
- road identities, road names.

As an example: if the difference of incoming to outgoing angle is less than 30°, the road name remains the same and the driver still has right of way, then the corresponding outgoing link is the default link.

The rule set may be very large and cover different scenarios. Sometimes it is simpler to decide, which link is not the default link, e.g. a highway exit never is the default link. Some incoming connections do not have any default link at all, e.g. think of three-way-junctions. In this case, we set $def(e)=undef$. Fig. 6 illustrates some examples for default links.

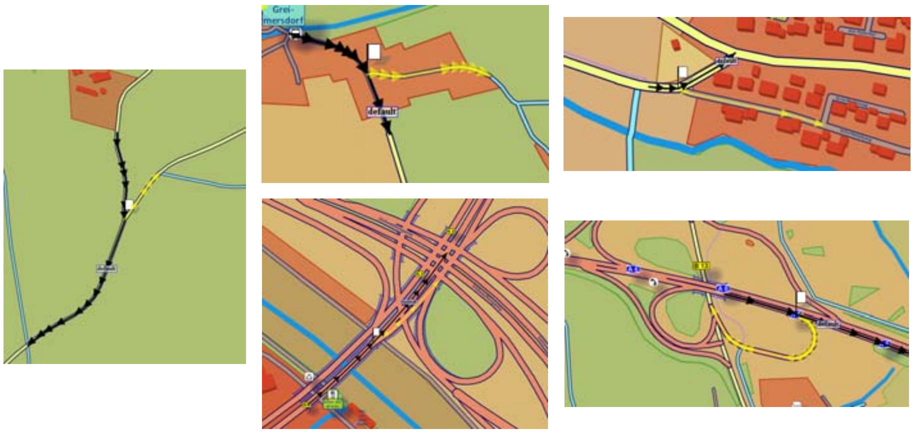

Fig. 6. Default link examples (incoming and default link: black, non-default-link: yellow)

We now are able to introduce distinctive directions *dist*(e): they are relevant according to *rel* and *do not* reside on the default link according to *def*. We are able to compute *dist* from a formerly computed *dir* with the following algorithm:

```
Algorithm $compute\_distinctives(V, E, M, dir, rel, def) \rightarrow dist$

for each $e=(v, v_n) \in E$ {
    $dist(e) \leftarrow \{\}$;
    for each $m \in dir(e)$ {
        if $rel(v, m)$ and $(\exists e_{in} \in in(v) : def(e_{in}) \neq e$ and $m \in dir(e_{in}))$
            then add $m$ to $dist(e)$
    }
}
```

Fig. 7 shows some examples of computed distinctive direction markers.

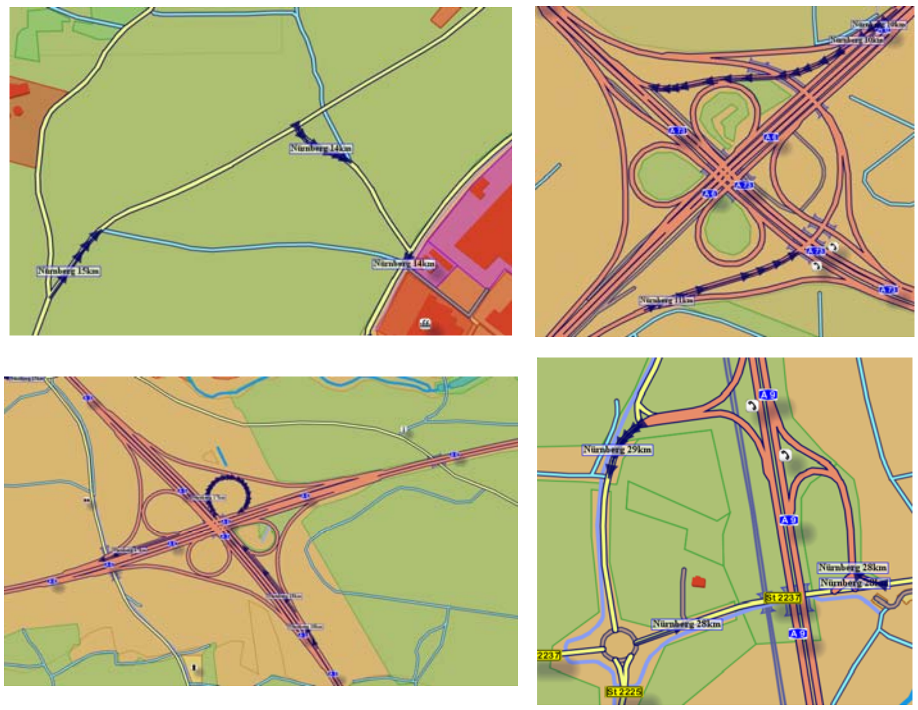

Fig. 7. Examples of distinctive direction markers

Some words about the performance: as a first observation, we consider performance not as a critical issue. The computation of direction markers is not performed at runtime but in the context of mastering and bundling the geo data for an application. Thus, it may take some hours without any drawback. Second: as the landmarks are processed independently, the processing can easily be distributed to different CPUs or even computing hosts to parallelize the computation.

3 Conclusions

In this contribution, we presented an approach to automatically compute direction markers that point to landmarks in road networks. We distinguished confirmative and distinctive markers. The main idea: every direction marker resides on an optimal path from a virtual starting point to the landmark. As nearby the virtual starting point and nearby the landmark such markers were misleading, we introduced the mechanism of cropping that cuts the virtual paths with the help of application-defined metrics. We presented an efficient approach to propagate these metrics among the network. To only get distinctive direction markers, we additionally introduced relevant crossings and default directions. The approach is successfully integrated into the donavio platform.

References

1. Dijkstra E.W: A note on two problems in connexion with graphs, Numerische Mathematik 1, 269–271 (1959)
2. Roth, J.: Die HomeRun-Plattform für ortsbezogene Dienste außerhalb des Massenmarktes, in A. Zipf, S. Lanig, M. Bauer (eds.) 6. GI/ITG KuVS Workshop Location based services and applications, Heidelberger Geographische Bausteine, Vol. 18 (in German) (2010)
3. Roth, J.: Modularisierte Routenplanung mit der donavio-Umgebung, in Werner M., Haustein M. (eds.): 9. GI/ITG KuVS Workshop Location based services and applications, Sept. 13-14, 2012, TU Chemnitz, Universitätsverlag Chemnitz, 119-131 (in German) (2013)
4. Roth, J.: Predicting Route Targets Based on Optimality Considerations, International Conference on Innovations for Community Services (I4CS), Reims (France) June 4-6, 2014, IEEE xplore, 61-68 (2014)
5. Roth, J.: Efficient many-to-many path planning and the Traveling Salesman Problem on road networks, International Journal of Knowledge-based and Intelligent Engineering Systems Vol. 20 (2016), IOS Press, 135–148 (2016)
6. Roth, J.: The Offline Map Matching Problem and its Efficient Solution, 16th International Conference on Innovations for Community Services (I4CS), Vienna, Austria, June 27-29, 2016, Springer CCIS 648, 23-38 (2016)
7. Roth, J.: Efficient Computation of Bypass Areas, Gartner, Huang (eds): Progress in Location-Based Services 2016, Proc. of the 13th International Conference on Location-Based Services, Vienna, Austria, Nov. 14–16, 2016, Springer LNG&C, 193-210 (2016)
8. Traditional Direction Signs. Traffic Advisory Leaflet 6/05, Department for Transport, London (2005)
9. Vienna Convention on Road Signs and Signals. United Nations Economic and Social Council, March, 28th (2006)