Algorithms for Reliable Navigation and Wayfinding

Shazia Haque[1], Lars Kulik[1], Alexander Klippel[2]

[1]Department of Computer Science and Software Engineering
University of Melbourne, Victoria, 3010, Australia
s.haque2@pgrad.unimelb.edu.au
lars@csse.unimelb.edu.au

[2]Cooperative Research Centre for Spatial Information, Department of Geomatics
University of Melbourne, Victoria, 3010, Australia
aklippel@unimelb.edu.au

Abstract. Wayfinding research has inspired several algorithms that compute the shortest, fastest, or even simplest paths between two locations. Current navigation systems, however, do not take into account the navigational complexity of certain intersections. A short route might involve a number of intersections that are difficult to navigate, because they offer more than one alternative to turn left or right. The navigational complexity of such an intersection may require modified instructions such as *veer right*. This paper, therefore, presents a *reliable path algorithm* that minimizes the number of complex intersections with turn ambiguities between two locations along a route. Our algorithm computes the (shortest) most reliable path, i.e., the one with the least turn ambiguities. Furthermore, we develop a variation of this algorithm that balances travel distance and navigational complexity. Simulation results show that traversing a reliable path leads to less navigational errors, which in turn reduces the average travel distance. A further advantage is that reliable paths require simpler instructions.

1 Introduction

Turn right at the next intersection. Most speakers of the English language are not only able to understand this simple instruction, but will also picture a prototypical intersection at which the above mentioned right turn has to be performed [8, 25, 12]. This prototypical instantiation of a situation model [27] will work for many intersections, especially in grid-like street networks typical for cities in North America. For historic city centers, as they are often found in European cities such as Paris or Rome, this instruction can easily be ambiguous and we might run into a spatial decision problem.

Knowing of the difficulties of natural language descriptions of complex spatial situations, an up-to-date navigation assistance system would not rely on a simple linguistic description alone but provide additional information about how to proceed. Real world examples can easily be found at Internet route planners providing instructions such as: *After 273 m turn right into Weinberg Strasse*.

The ambiguity of this instruction, however, becomes apparent if a traveler encounters an intersection where two streets equally qualify as 'right'. If a linguistic label such

as ‘right’ applies to more than one turn, we call such a turn *instruction equivalent*. We summarize some important issues why the ambiguity of instruction equivalent turns is hard to resolve even if an instruction supplies a street name:

- Not every street bears a street sign indicating its name.
- Traveling by car leaves the unfamiliar driver often with too little time to find and read the respective street sign.
- Driving by night makes the identification of street signs a futile endeavor.
- The street signs of large intersections with many branches are often too far away making them impossible to read.

One solution is to refine the set of instructions. Instead of using only the set {*left*, *right*, *straight*} in combination with a {*street name*}, a refinement could, for example, include the modifiers *bear* and *sharp* leading to the instruction set {*sharp left*, *left*, *bear left*, *straight*, *bear right*, *right*, *sharp right*}.Yet, we have found three further indications in the communicative behavior of route direction givers that show the scrutinies that still persist at intersections with instruction equivalent branches.

1. Direction concepts between the main axes (left, right, straight) are not easily associated with a single linguistic term. While direction changes close to 90 degree left and right are referred to as *left* and *right* with little variation, a plethora of composite linguistic expressions is used in between (i.e. around the diagonals) with often subtle differences in the meaning of the hedge terms [26, 2, 11, 16]. We find expressions such as: *turn slightly right*, *go right 45 degrees*, *veer right*, *sharp right bend* and so forth[1].

2. The conceptualization of directions different from the main axes is not straight forward and the sizes and boundaries of sectors, or more generally, the semantics of corresponding spatial prepositions, are ongoing research questions [9, 4].

3. People show conceptual and linguistic difficulties with intersection that deviate from prototypical ones. A comparison of intersections with different navigational complexities revealed the following associated linguistic behaviors (see Figure 1, [14]). The more (structurally and navigationally) complex an intersection is:

- the more verbose are verbal route directions,
- the more varied are the verbalizations,
- the more references are made to the structure,
- the more alternative instructions are offered (redundant information),
- the more modifications are applied to reduce ambiguity.

The approach we therefore present in this article to solve this problem is to calculate the route differently: instead of relying on shortest or fastest algorithm, the navigational complexity of an intersection is taken into account. This complexity is based on how many alternatives are available within the same sector in which the branch to take is located. In other words, the complexity depends on how many *instruction equivalent choices* are available at an intersection. The *unreliability* of a turn is a measure of the probability of a navigator making a mistake while taking that turn which is directly

[1] The verbalizations are taken from a data set collected at the University of Santa Barbara, California. They are all instructions for the same turning angle of 140.625 degrees.

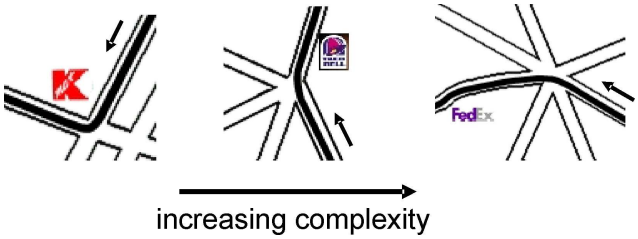

Fig. 1. Increasing navigational complexity of intersections. While the leftmost navigational situation is described as *go all the way down to k-mart and make another RIGHT*, the rightmost navigational situation yields descriptions such as *aand..h.. continue down straight until you co..come to a ..eh .. SIX-intersection.. ROAD.. and youll take thee.. you won.. you will NOT go STRAIGHT, you will go..you will go LEFT on the THIRD (ping).. the third ehm.. intersection. and travel down.. THAT and reach... the destination*.

related to the navigational complexity and ambiguity of the decision point/intersection. This means that if, for example, the navigator is told to *turn left* and more than one turn is present at that intersection that could qualify as a left turn, then the unreliability of that turn increases (depending on the number of similar options available). Accordingly, the unreliability of a route (origin to destination) depends on the unreliability of each turn that the navigator encounters. It has been shown in [23] that drivers are prepared to take suboptimal routes in terms of travel time, if these routes are potentially easier to describe and to follow. Additionally, it is confirmed that successful vehicle navigation systems rely as much on clarity of route instructions as on the length of routes [19].

The remainder of this paper is structured as follows. In Section 2 we introduce our basic algorithm called the *shortest most reliable path algorithm*. Section 3 gives an overview of the simulation environment we used to test the algorithms and to explore the error behavior of a simulated agent. Section 4 details experimental results comparing the classic shortest path algorithm and the shortest most reliable path algorithm. Cognizant of the trade-off between distance and reliability, we introduce a modification of our algorithm to allow for the best balance between travel distance and navigational complexity in Section 5. The new algorithm is compared to the other two in section 6.

2 Shortest Most Reliable Path Algorithm

Our goal is to compute the most reliable route between two locations. Graphs are a standard data structure for representing road and transportation networks. A graph G consists of a set of vertices V and edges $E \subset V \times V$ connecting the vertices. In case of a road network each vertex v represents an intersection and each edge e represents a road. Since the number of roads entering or leaving an intersection is bounded by a small number, we can consider G as a sparse graph which implies that $|E| = \mathcal{O}(|V|)$, where $|E|$ is the number of edges and $|V|$ is the number of vertices in the graph. In a weighted graph a function $w : E \rightarrow \mathbb{R}^+$ assigns a weight to each edge $e \in E$. Finding the shortest paths, the paths of least cost between vertices in a weighted graph, is a fundamental network analysis function and a classic problem in computation [6, 3].

The following algorithm is based on the simplest path algorithm [7]. The simplest path algorithm aims to minimize the instruction complexity of a route description, i.e., it favors intersections that are easy to describe. It is based on a classification of intersections that assigns a weight to each intersection type [18]. The weight used in the simplest path algorithm represents the instruction complexity to negotiate an intersection. The landmark spider algorithm given in [1] uses the same algorithm as in [7] with the only difference in the weighting function for an intersection, which depends on the distance, saliency, and orientation of a traveler with respect to any landmark present near the intersection. The aim is to generate a *clearest* [1] path in terms of spatial references and landmarks used to describe the route.

The key difference between the shortest and the (most) reliable path algorithm is that a weight w is not only assigned to each edge but also to *each pair of connected edges* in the graph. This approach is inspired by the simplest path algorithm in [7]. The difference, however, is that the weight in [7] reflects the instruction complexity of a decision point, whereas in our approach the weight represents the ambiguity of each turn at an intersection point. The weight r measures how many alternative edges can be considered as instruction equivalent. Since a turn involves two edges, the weight is a function for a pair of connected edges: $r : \varepsilon \rightarrow \mathbb{R}^+$, where $\varepsilon = \{((v_i, v_j), (v_j, v_k)) \in E \times E\}$. The weight function r used in the reliable path algorithm characterizes the unreliability measure for a turn represented by an edge pair, i.e., turning from the edge (v_i, v_j) into the edge (v_j, v_k). In addition, the reliable path algorithm takes into account the weight assigned to each edge to compute the shortest most reliable path between the origin and the destination vertex, because there could be many equally reliable paths between the two vertices.

2.1 Computing the Most Reliable Path

The actual shortest most reliable path algorithm, as presented in Algorithm 1, is an adapted version of Dijkstra's shortest path algorithm [6, 3]. The input of the algorithm is a graph G that is connected (there is a path from any vertex to any other vertex), simple (there are no edges from a vertex to itself and at most one edge between two different vertices) and directed (each edge is an ordered pair of vertices). The algorithm first initializes all edges connected to the source vertex with an unreliability of zero and distance equal to their respective distance from the source vertex. It also sets the previous connected edge of these edges as NIL. Then, the algorithm iterates through each edge minimizing the cumulative unreliability measure, as well as the cumulative distance from the source (as a second priority). This is achieved in line 8 of Algorithm 1: *competing edges* are all edges that have the same minimum unreliability from the source vertex. Amongst the competing edges the edge which has the minimum distance from the source vertex is selected. The cumulative unreliability as well as cumulative distance from the selected edge to all connected edges is recalculated. The algorithm assigns at each iteration to each edge its previous edge as a predecessor, which is the edge that provides the minimum unreliability and in case of ties the edge that has the shortest distance from the source (see line 15 of Algorithm 1). The predecessor assignment facilitates the reconstruction of the final path. The algorithm iterates until an edge is selected which has the destination vertex d as its terminating vertex. Reconstructing

Algorithm 1: Shortest Most Reliable Path Algorithm

Input : Graph $G = (V, E)$ is a connected, simple and directed graph;
$s \in V$ is the origin vertex; $d \in V$ is the destination vertex;
ε is the set of pairs of (directed) edges that share their "middle" vertex;
$w : E \to \mathbb{R}^+$ is the graph edge weighting function;
$r : \varepsilon \to \mathbb{R}^+$ is the graph turn weighting function;
$w_s : E \to \mathbb{R}^+$ stores the edge weight of the reliable path from s;
$r_s : E \to \mathbb{R}^+$ stores the turn weight of the reliable path from s;
$S = \{\}$ is a set of visited edges; $P = \{\}$ is an ordered set of vertices

Output: The shortest most reliable path $P = \langle s, v_1, \ldots, v_N, d \rangle$

1 $w_s(e) \leftarrow \infty; r_s(e) \leftarrow \infty;$ for all $e \in E$
2 **for** $(s, v_j) \in E$ **do**
3 $\quad r_s(s, v_j) \leftarrow 0$
4 $\quad w_s(s, v_j) \leftarrow w(s, v_j)$
5 $\quad$ previousEdge$(s, v_j) \leftarrow$ NIL
6 **end**
7 **while** $|E \backslash S| > 0$ **do**
8 $\quad$ Find $e \in E \backslash S$ so that amongst competing edges where r_s is minimum, $w_s(e)$ is minimum
9 $\quad$ **if** $e = (v_j, d)$ **then**
10 $\quad\quad$ terminate the loop
11 $\quad$ **end**
12 $\quad$ Add e to S
13 $\quad$ **for** $e' \in E$ **do**
14 $\quad\quad$ **if** $(e, e') \in \varepsilon$ **then**
15 $\quad\quad\quad$ **if** $(r_s(e') > r_s(e) + r(e, e')) \vee \Big((r_s(e') = r_s(e) + r(e, e'))$ $\wedge\, (w_s(e') > (w_s(e) + w(e'))) \Big)$ **then**
16 $\quad\quad\quad\quad$ $r_s(e') \leftarrow r_s(e) + r(e, e')$
17 $\quad\quad\quad\quad$ $w_s(e') \leftarrow w_s(e) + w(e')$
18 $\quad\quad\quad\quad$ previousEdge$(e') \leftarrow e$
19 $\quad\quad\quad$ **end**
20 $\quad\quad$ **end**
21 $\quad$ **end**
22 **end**
23 Add d to P
24 **while** $e \neq$ NIL **do**
25 $\quad$ Prepend starting vertex of e to P
26 $\quad$ $e \leftarrow$ previousEdge(e)
27 **end**

the shortest most reliable path is then a matter of iterating through the previous edges of d until the previous edge is undefined (NIL), see lines 23–27 of Algorithm 1. As represented in Algorithm 1, the shortest most reliable path algorithm is a *single pair* algorithm. If we exclude lines 9–11 from the algorithm, it iterates until all edges are

visited and returns the shortest most reliable paths from the source to all other vertices, i.e., it operates like a *single source* algorithm.

2.2 Computational Complexity

In this section we determine the computational time complexity of the shortest most reliable path algorithm (Algorithm 1). Consider line 8 of Algorithm 1 (which corresponds to the extract minimum operation of Dijkstra's algorithm [6, 3]): this operation takes $2|E|$ steps as it has to check all the edges of the graph once, to get the value of minimum unreliability, and again to select the edge with minimum distance amongst those having the same least unreliability. Similarly, in the worst case of a fully connected graph where every pair of vertices is connected by an edge, the FOR loop starting at line 13 would have to compute the unreliability of every edge connected to the selected edge (this operation corresponds to the relax operation in Dijkstra's algorithm [6, 3]). In order to compute the unreliability of an edge the algorithm needs to know the orientation of all other edges connected to the selected edge which leads to $2|E|$ steps. Both operations happen $|E|$ times, which leads to a total number of $|E|(2|E| + 2|E|)$ steps. However, since geographical networks are sparse graphs with a small limited number of roads at an intersection, the number of steps reduces to approximately $|E|(2|E|)$, which leads to a time complexity of $\mathcal{O}(|E|^2)$.

The shortest path algorithm has a complexity of $\mathcal{O}(|V|^2)$, whereas as the shortest most reliable path algorithm has a complexity of $\mathcal{O}(|V|^4)$ in the case of a fully connected graph because of $|E| = |V|(|V| - 1)$. Since most of geographical networks can be considered to be planar graphs, which have a maximum number of $|E| = 3(|V| - 2)$ edges, the complexity of the shortest most reliable path algorithm is $\mathcal{O}(|V|^2)$.

The operation of the most reliable simplest path algorithm can be seen as a mapping from the original graph G to a graph $G' = (\widetilde{E}, \varepsilon)$, where $\widetilde{E}$ is the set of edges E ignoring their direction (for details see [7]). The graph G' might not be planar leading to a slower performance by Algorithm 1 as compared to Dijkstra's shortest path algorithm. As compared to the simplest path algorithm, for which the time complexity has been shown to be $\mathcal{O}(|V|^2)$ in [7], the shortest most reliable path algorithm can be slightly slower as it needs to consider the distance as well, whereas the algorithms in [7, 1] return the first simplest or clearest paths, respectively.

One way of improving the performance of shortest most reliable path algorithm is to use a binary heap for the operation in line 8, which would lead to $\mathcal{O}(\log |E|)$ steps instead of $\mathcal{O}(|E|)$ leading to an overall time complexity of $\mathcal{O}(|E| \log |E|)$.

3 Simulation and Evaluation of Wayfinding Algorithms

As mentioned above, wayfinding research has lead to a variety of algorithms, each focusing on providing improved navigation assistance for a traveler in its own way. In order to analyze and evaluate these algorithms simulations are necessary. In our case, we were interested in carrying out two classes of simulations: (1) after loading a street network (via a coordinate and adjacency list), execute a particular algorithm for every

path present (between two vertices not directly connected) and then analyze the resulting navigational instructions on the basis of distance and unreliability; (2) simulate a human navigator traversing a route (between the same origin and destination), on the basis of instructions provided by different algorithms.

To achieve the above objectives, we developed a simulation environment using Java 2D API. As shown in Figure 2, using this simulation environment we could load a street network graph and select different algorithms to calculate routes. The type of a turn a potential navigator would perceive was based on angles[2] (in degrees) set in the simulation environment. The request for a path could be entered both interactively as well as in batch mode from a file.

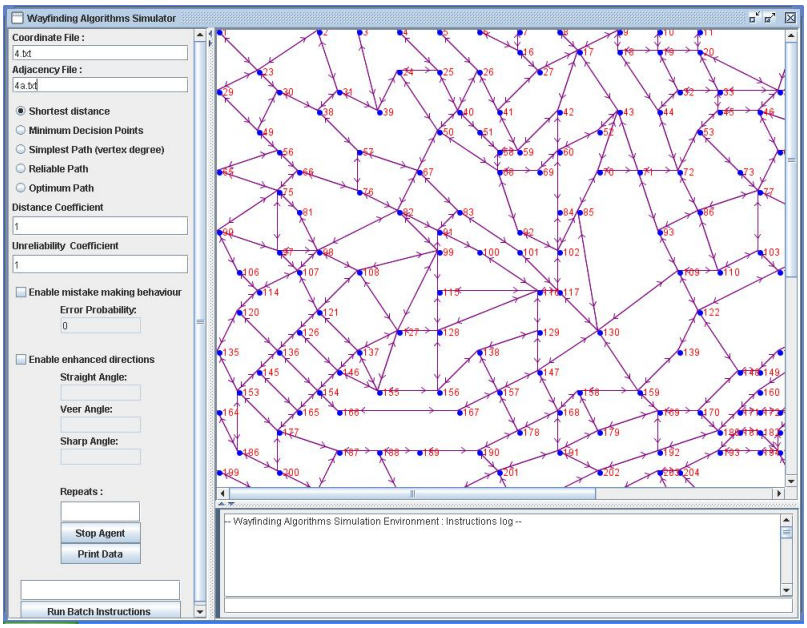

Fig. 2. Wayfinding Algorithms Simulation Environment

The simulated navigator was based on the following assumptions:

- It was error free, in the sense that if it was told to go left it would only choose one of the left turns available (and would not go straight or turn right).
- Depending on the simulation environment user settings it should be able to perceive a turn type based on the angle of the direction change with respect to the current direction of travel.

[2] We allowed three types of turns: veer, left/right and sharp depending on angles set for these turns. All turns between 0° and the *straight angle* were regarded as straight, between the *straight angle* and the *veer angle* as veer, between the *veer angle* and the *sharp angle* as left/right and between the *sharp angle* and 180° as sharp turns by the navigator.

– The simulated navigator does not make a mistake while executing the first instruction which does not involve a turn.
– In case the simulated navigator made a wrong choice when executing an instruction (correctly) and subsequently was given an instruction which it could not execute, it would realize that it has made a mistake and requery a path to its destination. Similarly, after choosing a wrong turn (while correctly following an instruction) if it ended up at a wrong destination and ran out of instructions, it requeries the path to its destination. A count of requeries reflects the number of times the navigator got 'consciously' lost during the travel.

3.1 Turn Unreliability Weighting Function

The unreliability measure of a turn from edge e into edge e' (both connected by a middle vertex) is given by

$$r(e, e') = \text{count of turns that are instruction equivalent to } e' - 1$$

The type of a turn refers to the linguistic variable used to identify the turn. The unreliability of the entire path R from origin s to destination d is the sum of unreliability measures encountered along the path, given by

$$R(s, d) = \sum_{i=0}^{N-1} r(e_i, e_{i+1})$$

where N is the total number of turns present on the path, $e_0 = (s, v_1)$ and $e_N = (v_N, d)$.

3.2 Comparison of Shortest and Shortest Most Reliable Path Algorithms

Since most of the current automated navigation systems are based on computing the shortest path given a source and destination vertex, we chose to compare the performance of the shortest most reliable path algorithm with that of the shortest path algorithm. This performance evaluation was carried out using the following metrics:

– **Distance traveled**: The distance traveled by the simulated navigator following the path provided by the algorithm.
– **Number of requeries**: This is the number of times the simulated navigator recognized that it is on the wrong path and did a requery from its current location to the destination. This is a measure of the cost that might be incurred by a human navigator sending messages via a mobile device. For our experiment we allowed the navigator up to five requeries per run after which it would finally stop.
– **Stopping distance from destination**: The shortest distance to the destination where the simulated navigator finally stops (after 5 requeries). If the destination was reached successfully this distance would be zero.
– **Total distance**: This is the sum of the distance traveled plus, in case the agent did not reach its destination, the stopping distance from the destination.

– **Missed target travels**: For multiple runs of the simulated navigator, this gives the number of times it failed to reach its destination (after 5 requeries).
– **Actual Unreliability**: This gives the average actual unreliability faced by the simulated navigator while traversing a path (for multiple runs it would give the average value per run per path).

Note that the requery limit and shortest stopping distance to destination metrics actually favor the shortest path algorithm, otherwise considering the unreliability involved on long routes, the simulated navigator could have traveled much more if there were no limit.

4 Experimental Results

Based on the method of path unreliability calculation (see Section 3.1) and using a simple instruction set comprising just 'straight', 'left', and 'right', we evaluated the two algorithms: *shortest path* and *shortest most reliable path*. We tested the algorithms on the street network dataset for the part of the city of Paris comprising more than 120,000 paths (making use of the simulation environment mentioned in Section 3). The results are presented in Figure 3. The bar graphs show for the shortest most reliable path algorithm that most paths had an unreliability of 0 and that the worst unreliability of a path was 4 (for 9 cases). For the shortest path algorithm, most paths had an unreliability between 1 and 10 and the worst unreliability was 15 (10 cases that are barely visible).

Figure 4 details for the same simulation, the ratios of the distances that a navigator would face for individual routes comparing the two algorithms directly. The results show that for more than half of the number of paths the shortest most reliable path algorithm returns a distance $1\sim1.4$ times (represented as a category distance ratio of 1 in the figure) the distance given by the shortest path algorithm. For around 40000 paths this ratio was between $1.5\sim2.4$ (represented by a rounded distance ratio of 2 in the figure) and for some rare cases this ratio was significantly higher reaching up to 22.

If the instruction set is more refined, i.e., has more than just 'left', 'right' and 'straight' turns such as 'bear left' or 'turn sharp left', then the calculation of unreliability measure would reflect this finer granularity, i.e., now fewer turns would be ambiguous. This is supported in the current implementation as it only implies specifying more turn types. However, this would come at the cost of higher instruction complexity.

In order to carry out an in depth comparative analysis, using the above mentioned simulation environment, navigator characteristics, and performance metrics, we chose 50 randomly selected paths (origin and destination separated by at least 3 vertices) from the dataset of the part of Paris. We made use of the simple instruction set comprising 'straight', 'left' and 'right' instructions. We ran the simulated navigator 50 times per path with 2 additional conditions for the instruction 'straight', as there is debate whether this direction concept is an axis of a sector [16, 12]. It can be assumed, however, that in the absence of competing objects a linguistic expression can be applied more flexible [24] and that in general it can be distinguished between giving an instruction and understanding an instruction. We therefore looked at a second condition for the instruction 'straight' allowing for 12 degrees per side resulting in a 24 degrees sector.

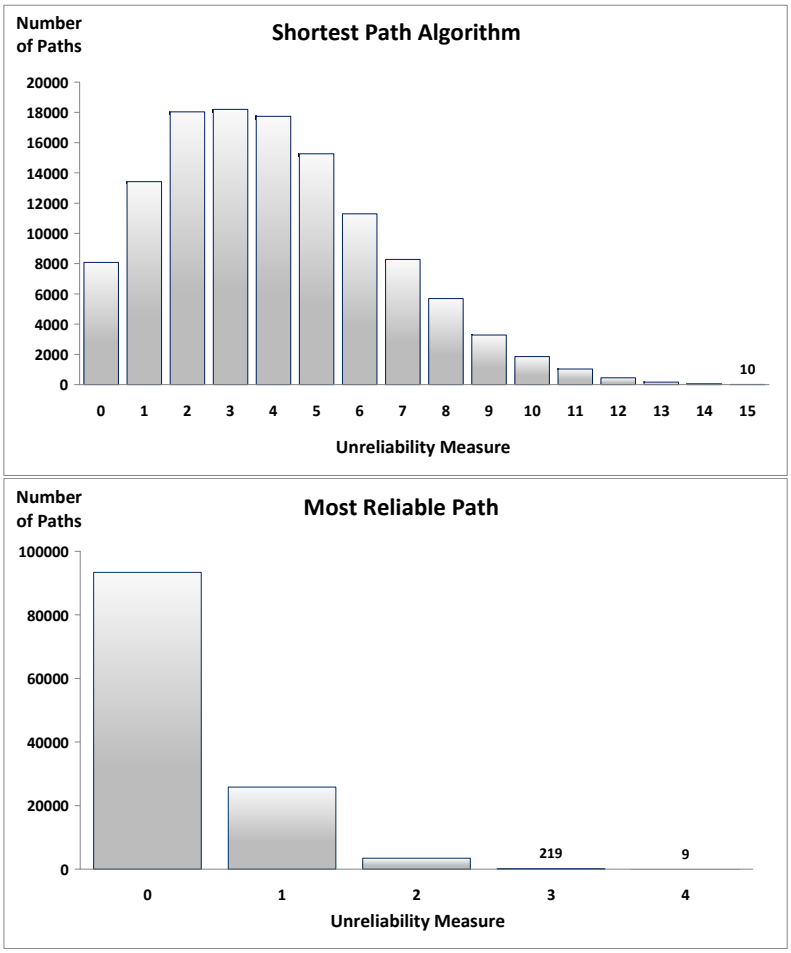

Fig. 3. Comparison of unreliability measures rendered by the two algorithms for the same street network.

The aggregated results for the total 2500 runs are detailed in Table 1. They show the behavior of the simulated agent when it follows the instructions.

As seen from these results, there is a clear advantage for the shortest most reliable path algorithm in terms of missed target travels. Likewise, the stopping distance was closer to the actual destination than in case of the shortest path algorithm. Additionally, the number of requeries for the shortest path algorithm was greater by a factor of more than 5. In terms of distance, we see that the total distances traversed by the simulated navigator for the two algorithms are comparable (the distance traversed following the shortest most reliable when the straight angle was set to $0\,^\circ$ was only 4.3% greater than the distance of the shortest path;and in case of straight angle set to $12\,^\circ$ it was only 6% lesser than the distance of the shortest path). The analysis of individual path runs

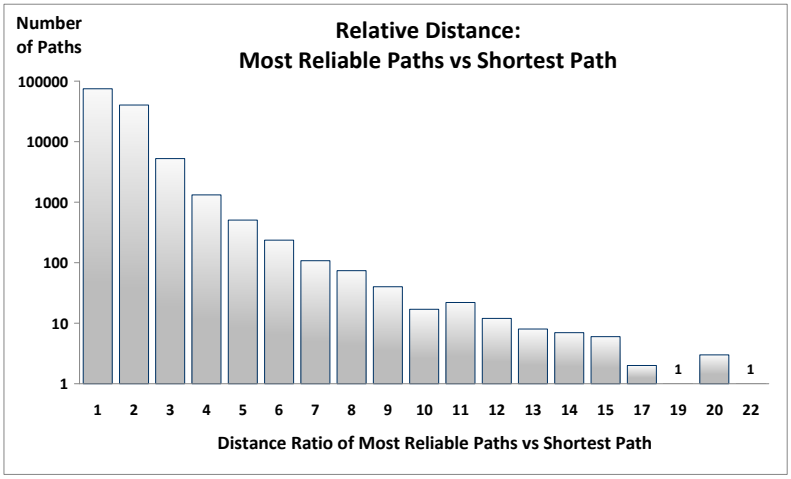

Fig. 4. Number of paths vs rounded ratio of distances.

Table 1. Simulation based results for various performance metrics comparing shortest and shortest most reliable path algorithms

Algorithm	Shortest Path	Shortest Most Reliable Path	Shortest Path	Shortest Most Reliable Path
Straight Angle	$0\,^\circ$		$12\,^\circ$	
Distance Travelled	125719.5	137118.5	106959.5	103114.5
Stopping Distance	6134.1	374.6	2888.5	104.9
Total Distance	131853.6	137493.1	109848	103219.4
Requeries	6773	1291	5209	406
Missed target travel	566	36	319	24
Actual Unreliability	8.3	1.3	5.8	0.5

revealed that for around half the paths (27 for straight angle $= 0\,^\circ$ and 20 for straight angle $= 12\,^\circ$), the distance traversed by the simulated navigator on the reliable path was greater than that traveled while following the shortest route. Amongst these we find for 17 paths in case of the straight angle $= 0\,^\circ$ and for 13 paths in case of the straight angle $= 12\,^\circ$ that the reliable distance was more than 10% longer than the shortest distance. A few sample cases are listed in Table 2.

5 Path Optimization

As seen from the charts in Figures 3 and 4 as well as the simulation results described in Section 4, there are trade-offs meeting the objectives of providing the shortest route and providing a most reliable route to the navigator. The most reliable route has a lower chance of making a mistake due to less ambiguities and reduces the cost associated with requerying the navigation system but it also leads to a greater distance between two

Table 2. Sample simulation results for four paths comparing Shortest and Shortest Most Reliable Path Algorithm

Algorithm	Unreliability	Requeries	Total distance	Relative distance increase
Shortest	10	226	4246.8	14.7%
Shortest Most Reliable	0	0	4870.5	
Shortest	6	143	2265.78	46.6%
Shortest Most Reliable	1	25	3321	
Shortest	6	134	2237.9	59.0%
Shortest Most Reliable	0	0	3558.5	
Shortest	1	66	1480.6	83.1%
Shortest Most Reliable	0	0	2710.6	

locations; computing the shortest route can increase the chance of making mistakes significantly. A ranking of the criteria for human route selection is given in [10]. Amongst those criteria, the overall distance and least time of travel are ranked most highly. Therefore, it is desirable to find a route which optimizes both objectives of shortest distance as well as least unreliability, which in case of an error can result in considerably longer paths.

A naïve approach to find a more optimal route would be to consider all paths connecting an origin and a destination and amongst them choose the one which is the most optimal for both criteria. However, considering the worst case of a fully connected graph shows that this strategy is too costly: using the Binomial coefficient $\binom{k}{i}$, which gives the number of generating i unordered outcomes from k possibilities, shows that the maximum number of paths that can exist between an origin and a destination vertex in a connected graph is

$$k!(1 + \sum_{i=1}^{k} 1/i!),$$

where $k = |V| - 2$; $|V|$ is number of vertices present in the graph. The computation of $\mathcal{O}(|V|^k)$ paths is infeasible for practical purposes.

A significantly less costly way of finding an optimal trade-off between the shortest and the most reliable path is to assign an optimum weight to each turn, which depends on both the distance and unreliability involved in taking that turn. This optimum turn weight is given by

$$op(e, e') = \lambda_d w(e') + \lambda_u r(e, e'),$$

where λ_d determines the weight of the impact for the distance of the edge e' and λ_u determines the weight of the unreliability measure when turning from edge e to edge e'. The optimum weight of the entire optimal path OP from the origin s to the destination d is the sum of the optimum weights of the turns encountered on the entire path, i.e.,

$$OP(s, d) = \sum_{i=0}^{N-1} op(e_i, e_{i+1}),$$

where N is the total number of turns present on the path, $e_0 = (s, v_1)$ and $e_N = (v_N, d)$. A path P is a ordered series of vertices given as $P = \langle v_0, v_1, \ldots, v_N, v_{N+1} \rangle$ with origin $s = v_0$ and destination $d = v_{N+1}$. The optimal values of λ_d and λ_u depend on the characteristics of the street network and have to be calibrated for each network.

5.1 Optimum Reliable Path Algorithm

The optimum reliable path algorithm, as presented in Algorithm 2, is again a modified form of Dijkstra's shortest path algorithm [6, 3]. Note that the initial conditions at the onset of the algorithm are the same as those for the shortest most reliable path algorithm mentioned in Algorithm 1. The only addition is the graph optimum turn weighting function $op : \varepsilon \rightarrow \mathbb{R}^+$ and $op_s : E \rightarrow \mathbb{R}^+$, which stores the optimum weight of the optimal path from s.

The algorithm first initializes all edges connected to the source vertex with zero unreliability and distance equal to their respective distance from the source vertex. Based on these values, it calculates the optimum weight of these edges. It also sets the connected edge of these edges as NIL. Next, the algorithm iterates through each edge minimizing the cumulative optimum weight from the source and selecting the edge with minimum value of op_s. The cumulative optimum weight from the selected edge to all connected edges is recalculated. The algorithm assigns to each edge its predecessor edge at each iteration to facilitate reconstruction of the final path. The algorithm iterates until an edge is selected, which has the destination vertex d as its terminating vertex. Reconstructing the optimum reliable path is then a matter of iterating through the previous edges of d until the previous edge is undefined (NIL).

5.2 Computational Complexity

The computational complexity of the optimum reliable path algorithm is the same as that for the shortest most reliable path algorithm as discussed in Section 2.2. However, it is slightly faster than Algorithm 1, since it does not have to check all edges twice for extracting the edge with the minimum optimum weight from the source s.

6 Performance Evaluation of the Optimum Reliable Path Algorithm

6.1 Calibration of λ_d and λ_u

In order to calibrate the values of λ_d and λ_u for the street dataset of this specific part of Paris, we ran the optimum path algorithm for various values for λ_d and λ_u, each time for the entire dataset with over 122,000 paths. We compared the results for the different λ_d and λ_u values with those obtained from the shortest path and shortest most reliable path algorithm. This comparison was done on the basis of the ratio of the distances for the optimum path and the shortest path (given by shortest path algorithm) and the difference of unreliability given by the optimum path algorithm and the shortest most reliable path algorithm. The goal of this procedure was to find values that best approximate

Algorithm 2: Optimum Reliable Path Algorithm

Input : Graph $G = (V, E)$ is a connected, simple and directed graph;
$s \in V$ is the origin vertex; $d \in V$ is the destination vertex;
ε is the set of pairs of (directed) edges that share their "middle" vertex;
$w : E \rightarrow \mathbb{R}^+$ is the graph edge weighting function;
$r : \varepsilon \rightarrow \mathbb{R}^+$ is the graph turn weighting function;
$w_s : E \rightarrow \mathbb{R}^+$ stores the edge weight of the reliable path from s;
$r_s : E \rightarrow \mathbb{R}^+$ stores the turn weight of the reliable path from s;
$op : \varepsilon \rightarrow \mathbb{R}^+$ is the graph optimum turn weighting function;
$op_s : E \rightarrow \mathbb{R}^+$ stores the optimum weight of the optimal path from s;
$S = \{\}$ is a set of visited edges; $P = \{\}$ is an ordered set of vertices

Output: The optimum reliable path $P = \langle s, v_1, \ldots, v_N, d \rangle$

```
1  w_s(e) ← ∞; r_s(e) ← ∞; op_s(e) ← ∞; for all e ∈ E
2  for (s, v_j) ∈ E do
3  |   r_s(s, v_j) ← 0
4  |   w_s(s, v_j) ← w(s, v_j)
5  |   op_s(s, v_j) ← λ_d × w(s, v_j)
6  |   previousEdge(s, v_j) ← NIL
7  end
8  while |E\S| > 0 do
9  |   Find e ∈ E\S so that op_s(e) is minimized
10 |   if e = (v_j, d) then
11 |   |   terminate the loop
12 |   end
13 |   Add e to S
14 |   for e' ∈ E do
15 |   |   if (e, e') ∈ ε then
16 |   |   |   if op_s(e') > op_s(e) + op(e, e') then
17 |   |   |   |   r_s(e') ← r_s(e) + r(e, e')
18 |   |   |   |   w_s(e') ← w_s(e) + w(e')
19 |   |   |   |   op_s(e') ← op_s(e) + op(e, e')
20 |   |   |   |   previousEdge(e') ← e
21 |   |   |   end
22 |   |   end
23 |   end
24 end
25 Add d to P
26 while e ≠ NIL do
27 |   Prepend starting vertex of e to P
28 |   e ← previousEdge(e)
29 end
```

both, reliability and distance. While favoring distance by high λ_d values (i.e. up to 6) resulted in unreliability measures of up to 14 with the mean value around 4, favoring reliability by high λ_u values created outliers with respect to the distance to travel. For our set of simulations we settled on values of $\lambda_d = 1$ and $\lambda_u = 6$ (see Figure 5).

This combination offered advantages for both aspects: most paths have a length close to the shortest distance; additionally, most paths do not encounter unreliable intersections. The maximum difference of unreliability compared to the most reliable path was 4 for very few cases.

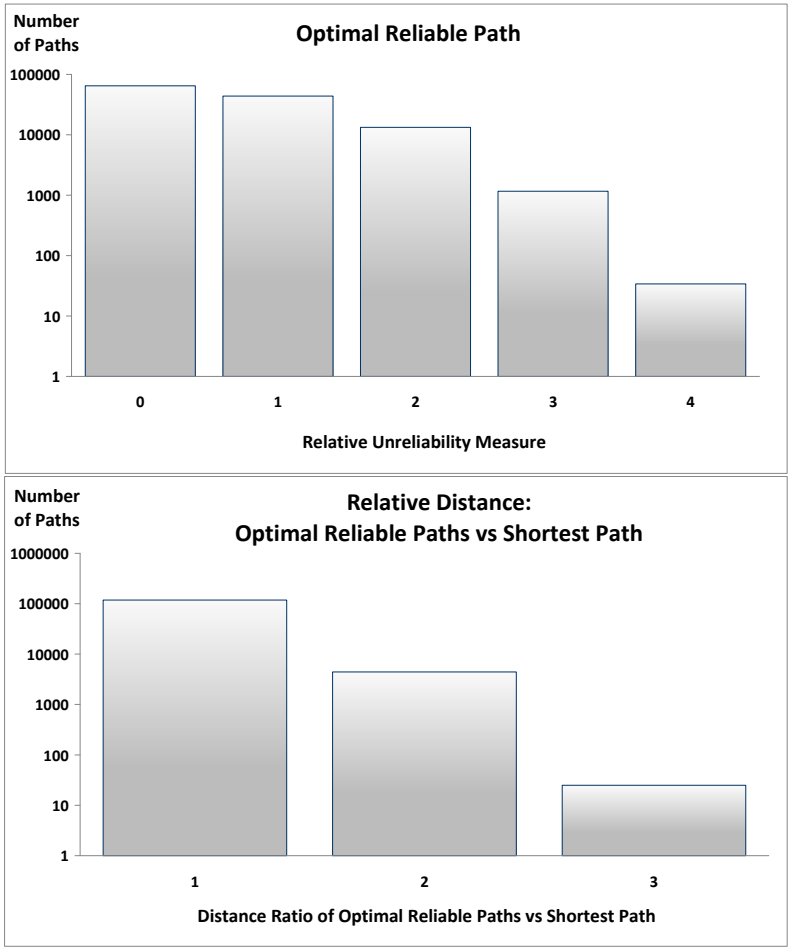

Fig. 5. (top) Number of paths vs difference between optimum path unreliability and most reliable path unreliability. (bottom) Number of paths vs rounded ratio of optimum path distance and shortest path distance. Displayed are the values for $\lambda_d = 1$ and $\lambda_u = 6$.

6.2 Experimental Results

For a comparative in depth analysis of the optimum reliable path algorithm with both, shortest path and shortest most reliable path algorithms, we used the same simulation

environment, navigator characteristics, and performance metrics as discussed in Section 3. As parameters for the optimum path we chose the values determined in the study discussed in the previous section, $\lambda_d = 1$ and $\lambda_u = 6$. The aggregated results for 50 runs per route on the same set of 50 routes are presented in Tables 3 and 4.

The data shows that the distance traveled by the simulated navigator when traversing the optimum reliable path was 13.6% and 14.6% shorter than the distance it traversed following the shortest path, for straight angle 0° and straight angle 12°, respectively. The analysis of the individual path runs revealed the following: for only 10 paths in the case that the straight angle was set to 0° and 12 paths in case of the straight angle was set to 12° was the distance traveled by the simulated navigator on the optimum reliable path greater compared to following the shortest path. Amongst these for only 4 paths in case the straight angle was set to 0° and 3 paths in the case the straight angle was set to 12° the distance was greater than 10% (the worst one being 20% greater in case of the straight angle being set to 12°). This, overall, can be regarded as a considerable saving of travel distance. However, the number of requeries done on optimum reliable path was greater than the number of requeries done while traversing the most reliable path (but still far less than those done on shortest path). The missed target travels, stopping distance from destination and actual unreliability were only slightly greater than those for most reliable path, indicating an overall better optimal performance.

Table 3. Aggregate results for various performance metrics (with no angle specified for straight)

Algorithm	Shortest Path	Shortest Most Reliable Path	Optimum Path $\lambda_d = 1, \lambda_u = 6$
Distance Travelled	125719.5	137118.5	113158.9
Stopping Distance	6134.1	374.6	696.1
Total Distance	131853.6	137493.1	113855
Requeries	6773	1291	2799
Missed target travel	566	36	63
Actual Unreliability	8.3	1.3	3.4

Table 4. Aggregate results for various performance metrics (with a straight angle of 12°)

Algorithm	Shortest Path	Shortest Most Reliable Path	Optimum Path $\lambda_d = 1, \lambda_u = 6$
Distance Travelled	106959.5	103114.5	93641.7
Stopping Distance	2888.5	104.9	123
Total Distance	109848	103219.4	93764.7
Requeries	5209	406	1076
Missed target travel	319	24	26
Actual Unreliability	5.8	0.5	1.1

7 Conclusions and Future Work

In this work, we developed wayfinding algorithms that favor reliable paths with lesser navigational complexity. Our expectation was that it would lead to fewer mistakes a wayfinder potentially could make during route following. The simulation results support our expectation. When traversing a reliable path, the simulated navigator was found to be getting at (or closer to) the actual destination more often than when traversing the shortest path. The simulation also showed an advantage in terms of the number of times the simulated navigator got lost and needed to re-contact the automated navigation system, as well as the overall distance traveled. The data set used for the simulations was a complex street network. More simulation based analyses need to be done, making use of a variety of street networks. Cognitive studies with human navigators would provide additional proof of our hypothesis.

Our method of unreliability measure calculation took into account the instruction equivalent choices encountered at complex intersections. This approach can be further refined by the following aspects:

Including landmarks. Calculating the unreliability of a route is based on the ambiguity at intersections introduced by instruction equivalent branches. Salient features (landmarks) in the environment reduce this ambiguity [22]. The specific placement of a landmark at an intersection can be integrated into its measure of salience [15] and there are approaches which base—in more general terms—the calculation of routes on the availability of landmarks [1].

Incorporating user preferences. Next generation route directions will be adapted to several characteristics of a user. Hotly debated is the familiarity of a user with her environment. Tracking user movements by GPS devices will allow for establishing patterns of frequently visited places which can be assumed to be known [17, 20]. Knowledge about one's environment influences the amount of information provided in route directions. Further factors are the modality of travel and specific user characteristics.

Navigational complexity. An additional refinement of the navigational complexity could take into account whether a turn is made to the left or to the right. Depending on the general side one is traveling on it would be taken into account if a street has to be crossed. Likewise, the error-free criterion (see Section 3) could be changed to allow errors right from the start.

Cognitive load. A different perspective on route directions is offered by taking the spatial context of a route into account to reduce the number of instructions that has to be given for a specific route [21, 13, 5]. While these approaches aim to reduce the number of instructions by, for example, using complex intersection as landmarks a combination with the presented optimal algorithm would be feasible to calibrate the trade-off between the overall amount of information required to communicate a route and the unreliability associated with navigationally complex intersections.

A last point concerns the computational time complexity of the reliable path algorithms, which is worse than the one for the shortest path (quadratic in the worst case). This could be improved by using a binary heap as mentioned in Section 2.2. The optimum reliable path algorithm provides a configurable way of tuning the system towards either shortest or most reliable paths. However, calibrating the values of λ_d and λ_d for an entire street network as done in Section 6.1 might not always produce the best re-

sults for each and every path and sometimes these values might need to be calibrated for individual paths.

The very last point would take into consideration the modality that is chosen to communicate the action that has to be performed at an intersection depending on the navigational complexity of an intersection.

References

1. D. Caduff and S. Timpf. The landmark spider: Representing landmark knowledge for wayfinding tasks. In *Reasoning with Mental and External Diagrams: Computational Modeling and Spatial Assistance*, pages 30–35, 2005.
2. L. A. Carlson-Radvansky and G.D. Logan. The influence of reference frame selection on spatial template construction. *Journal of Memory and Language*, 37:411–437, 1997.
3. T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms*. MIT Press and McGraw-Hill, Cambridge, MA, 2nd edition edition, 2001.
4. K. R. Coventry and S.C. Garrod. *Saying, Seeing, and Acting: The Psychological Semantics of Spatial Prepositions*. Psychology Press, Hove, 2004.
5. R. Dale, S. Geldof, and J.-P. Prost. Using natural language generation in automatic route description. *Journal of Research and Practice in Information Technology*, 37:89–105, 2005.
6. E. W. Dijkstra. A note on two problems in connexion with graphs. *Numerische Mathematik.*, 1:269–271, 1959.
7. M. Duckham and L. Kulik. "Simplest" paths: Automated route selection for navigation. In W. Kuhn, M.F. Worboys, and S. Timpf, editors, *Spatial Information Theory*, volume 2825 of *Lecture Notes in Computer Science*, pages 169–185. Springer, 2003.
8. G.W. Evans. Environmental cognition. *Psychological Bulletin*, 88:259–287, 1980.
9. N. Franklin, L.A. Henkel, and T. Zangas. Parsing surrounding space into regions. *Memory and Cognition*, 23:397–407, 1995.
10. R.G. Golledge. Path selection and route preference in humn navigation: A progress report. In A.U. Frank and W. Kuhn, editors, *Spatial information theory: A theoretical basis for gis (COSIT '95). Number 988 in Lecture Notes in Computer Science.*, Berlin, 1995. Springer.
11. A. Herskovits. *Language and Spatial Cognition: An Interdisciplinary Study of the Representation of the Prepositions in English*. Cambridge University Press, Cambrigde, England, 1986.
12. A. Klippel. Wayfinding choremes. In W. Kuhn, M. Worboys, and S. Timpf, editors, *Spatial Information Theory. Foundations of Geographic Information Science, International Conference, COSIT 2003, Ittingen, Switzerland, September 24-28, 2003, Proceedings*, LNCS 2825, pages 320–334, Berlin, 2003. Springer.
13. A. Klippel, H. Tappe, L. Kulik, and P. U. Lee. Wayfinding choremes - a language for modeling conceptual route knowledge. *Journal of Visual Languages and Computing*, 16:311–329, 2005.
14. A. Klippel, T. Tenbrink, and D.R. Montello. The role of structure and function in the conceptualization of directions. in revision.
15. A. Klippel and S. Winter. Structural salience of landmarks for route directions. In A.G. Mark D.M. Cohn, editor, *Spatial Information Theory (COSIT'05)*, Lecture Notes in Computer Science. 3693, pages 347–362. Springer, Berlin, 2005.
16. B. Landau. Axes and direction in spatial language and spatial cognition. In E. van der Zee and Slack J., editors, *Axes and direction in spatial language and spatial cognition*, pages 18–38, 2003.

17. C. Li. User preferences, information transactions and location-based services: A study of urban pedestrian wayfinding. *Computer, Environment and Urban Systems*, forthcoming.
18. D. M. Mark. Finding simple routes: 'Ease of description' as an objective function in automated route selection. In *Proceedings, Second Symposium on Artificial Intelligence Applications (IEEE)*, pages 577–581, Miami Beach, 1985.
19. A.J. May, T. Ross, and S.H. Bayer. Drivers' informational requirements when navigating in an urban environment. *Journal of Navigation*, 56:89–100, 2003.
20. K. Patel, M. Y. Chen, I. Smith, and J. A. Landay. Personalizing routes. In *The 19th Annual ACM Symposium on User Interface Software and Technology, October 15 - 18 in Montreux, Switzerland*. ACM Press, 2006.
21. K.-F. Richter and A. Klippel. A model for context-specific route directions. In C. Freksa, M. Knauff, B. Krieg-Brckner, B. Nebel, and T. Barkowsky, editors, *Spatial Cognition IV: Reasoning, Action, Interaction, International Conference*, LNAI 3343, pages 58–78, Berlin, 2005. Springer.
22. M. Sorrows and S. C. Hirtle. The nature of landmarks for real and electronic spaces. In *Spatial Information Theory: Cognitive and Computational Foundations of Geographic Information Science, International Conference COSIT '99*, pages 37–50. Springer, 1999.
23. L. A. Streeter, D. Vitello, and S. A. Wonsiewicz. How to tell people where to go: Comparing navigational aids. *International Journal of Man/Machine Studies*, 22:549–562, 1985.
24. T. Tenbrink. Identifying objects on the basis of spatial contrast: An empirical study. In C. Freksa, editor, *Spatial Cognition IV: Reasoning, Action, Interaction, International Conference*, volume Lecture Notes in Computer Science, Volume 3343, pages 124–146, Berlin, 2005. Springer.
25. B. Tversky and P. U. Lee. How space structures language. In *Spatial Cognition, An Interdisciplinary Approach to Representing and Processing Spatial Knowledge*, pages 157–175, 1998.
26. C. Vorwerg and G. Rickheit. Typicality effects in the categorization of spatial relations. In C. Freksa, C. Habel, and K. F. Wender, editors, *Spatial cognition: An interdisciplinary approach to representing and processing spatial knowledge*, pages 203–222, Berlin, 1998. Springer.
27. R. A. Zwaan and G. A. Radvansky. Situation models in language comprehension and memory. *Psychological Bulletin*, 123:162–185, 1998.