# Simplest Instructions: Finding Easy-to-Describe Routes for Navigation

Kai-Florian Richter[1] and Matt Duckham[2]

[1] Transregional Collaborative Research Center SFB/TR 8 Spatial Cognition
Universität Bremen, Germany
richter@sfbtr8.uni-bremen.de
[2] Department of Geomatics, The University of Melbourne, Australia
mduckham@unimelb.edu.au

**Abstract.** Current applications for wayfinding and navigation assistance usually calculate the route to a destination based on the shortest or fastest path from the origin. However, numerous findings in cognitive science show that the ease of use and communication of route instructions depends on factors other than just the length of a route, such as the number and complexity of decision points. Building on previous work to improve the automatic generation of route instructions, this paper presents an algorithm for finding routes associated with the "simplest" instructions, taking into account fundamental principles of human direction giving, namely decision point complexity, references to landmarks, and spatial chunking. The algorithm presented can be computed in the same order of time complexity as Dijkstra's shortest path algorithm, $O(n^2)$. Empirical evaluation demonstrates that the algorithm's performance is comparable to previous work on "simplest paths," with an average increase of path length of about 10% compared to the shortest path. However, the instructions generated are on average 50% shorter than those for shortest or simplest paths. The conclusions argue that the compactness of the descriptions, in combination with the incorporation of the basic cognitive principles of chunking and landmarks, provides evidence that these instructions are easier to understand.

## 1 Introduction

Automated wayfinding assistance is an increasingly popular and economically important application area for geographic information science. Systems for automated wayfinding assistance employ computationally efficient algorithms for calculating the shortest or fastest route to a destination, but typically do not account for human principles of direction giving, and ignore how humans conceptualize their environment. The instructions generated, while generally usable, seem artificial and often make it hard to form survey knowledge about imminent wayfinding decisions, i.e., to prepare for what is coming up.

In recent years several approaches have emerged that cover (at least part of) the generation of route instructions that respect for human principles of wayfinding and direction giving [1–4]. However, these approaches usually aim at improving the presentation of route instructions for a previously calculated route. This paper presents a new algorithm that addresses the problem of *finding* the best route with respect to the simplicity of *route instructions*.

Given a geographic network, our algorithm finds the route between a source and destination that is the "simplest" to describe, in terms of the complexity of its associated routing instructions. Building on fundamental principles of human direction giving, the route instructions generated are expected to be easier for a human wayfinder to remember, communicate, and use. The underlying algorithm is based on the widely-known Dijkstra's shortest path algorithm [5]. As will be demonstrated in the paper, the extensions made to Dijkstra's algorithm do not increase computational complexity. Thus, the primary contribution of this paper is an efficient algorithm for generating cognitively ergonomic route instructions.

The next section presents related work on automatic generation of cognitively ergonomic route instructions, in particular *simplest paths* and *context-specific route instructions*. Section 3 introduces the algorithm that allows for finding the best route instructions to guide a wayfinder from origin to destination, including a discussion of the computational and cognitive properties of the algorithm. Section 4 then presents the results of an empirical evaluation of the algorithm, looking at the length of the paths and the instruction sequences generated. Section 5 concludes the paper with a discussion of the paper's contribution and an outlook on future work.

## 2 Generating Route Directions

This paper deals with determining a route between two points in a network space. Efficient algorithms exist for this task, primarily Dijkstra's shortest path algorithm [5]. Shortest path algorithms apply a cost function that is somehow related to the network's structure in its embedding geographical reference frame (e.g., distance between vertices, speed of movement, or direction of travel). However, shortest path algorithms neither account for human conceptualization of space nor for principles of human direction giving.

Human route instructions reflect the instruction-giver's knowledge about an environment. When asked to provide instructions, humans activate the spatial knowledge of the route to be described, identify the relevant information, structure this information, and communicate it to the requester [6, 7]. The literature specifically includes two important principles that are employed when providing instructions: 1) references to landmarks and 2) combining multiple consecutive decision points into a single instruction, termed *spatial chunking* by Klippel et al. [8]. Landmarks are important for acquiring and organizing knowledge about our surrounding space [9, 10]. In route instructions, they are frequently referred to; landmarks may signal crucial actions, locate other landmarks in relation to the referenced landmark, or confirm that the right track is still being followed [7, 11]. Often, humans subsume instructions for several decision points into a single "chunked" one. For example, "turn right at the third intersection" corresponds to going straight at the next two decision points and then turning right at the third one.

A range of existing research has addressed the automatic generation of route instructions that account for human principles of direction giving. Some of this work only covers parts of the generation process, such as the identification [12, 13] or integration [14, 15] of landmarks. Others focus on generating instructions that mimic the way humans present such information [1, 3], or that adapt to human conceptualization of wayfinding situations [4].

In the following, two approaches are presented in more detail that form the basis for the new algorithm proposed in this paper. The first approach is that of *simplest paths* [16], which aims to find a route that is easy to follow, by minimizing the number and complexity of decision points. The second approach is that of *context-specific route instructions* [17], which aims to generate route instructions for a *given* route that are easy to conceptualize and to remember.

## 2.1 Simplest Paths

Duckham and Kulik [16] extend standard shortest path search by a heuristic that associates a cost with each pair of connected edges (rather than each edge as in classic shortest path approaches). This cost reflects the complexity of negotiating the "decision point" represented by the two adjacent edges (e.g., turning from one edge onto another — see Figure 1). The specific weighting used is based on an adaptation of earlier work by Mark [18], who classifies different types of intersections according to the complexity of describing the action to be performed there. Duckham and Kulik, accordingly, term their algorithm *simplest path* algorithm. In a simulation experiment, they show that their algorithm generally results in paths that are only slightly longer than the shortest path.

While the costs employed account for structural differences of intersections, they do not account for functional aspects, for example, (possible) ambiguity in the direction to take at an intersection, nor landmarks or other environmental characteristics that might be exploited in instructions. In summary, like shortest paths, the simplest path finds the cheapest route according to a cost function. Unlike shortest paths, the cost function used applies to the complexity of navigation decisions rather than travel distance or time.

## 2.2 Context-Specific Route Directions

Context-specific route instructions account for environmental characteristics and a route's properties; they adapt the action to be taken to the current surrounding environment. Such instructions are termed "context-specific" because of the explicit adaptation to the structure and function in wayfinding [17]. A computational process, called Guard (Generation of Unambiguous, Adapted Route Directions), has been developed by Richter [19] for generating context-specific route instructions. Guard unambiguously describes a specific route to the destination, with instructions adapted to environmental characteristics. Figure 2 provides an overview of the generation process.

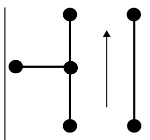
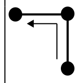
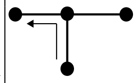
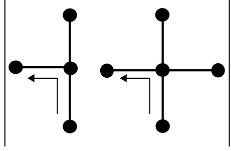
| | | |
|---|---|---|
| straight on |  | 1 slot |
| turn (not at intersection) |  | 4 slots |
| turn left or right at T-intersection |  | 6 slots |
| turn left or right at intersection |  | $5 + deg(v)$ slots |

**Fig. 1.** Weighting (slot values) of different intersection types; from [16] (modified). $deg(v)$ denotes the degree of an intersection, i.e. the number of branches meeting at this intersection.

GUARD works on a geographic network. This graph is annotated with information on landmarks, for example, their location and shape. The generation of context-specific route instructions is a three-step process. In the first step, for every decision point of the route, all instructions that unambiguously describe the route segment to be taken are generated, resulting in a set of possible instructions for each decision point. GUARD accounts for different types of landmarks in generating instructions whose role in the route instructions depends on their location relative to the route [15, 19].

Next, GUARD performs *spatial chunking*. GUARD is flexible with respect to the principles used in these steps. For example, it allows integrating the chunking principles presented by Klippel et al. [8] or Dale et al. [3]. Finally, in the third step of GUARD, the actual context-specific route instructions are generated. Here, from all possible instructions, those that best describe the route are selected. As this is realized as an *optimization* process, "best" depends on the chosen optimization criterion. Just as with the chunking principles, GUARD is flexible with respect to the criterion used. Optimization results in a sequence of chunks that cover the complete route from origin to destination. Due to the aggregation of instructions performed in chunking, instructions for some decision points will be represented implicitly, thus reducing the communicated information.

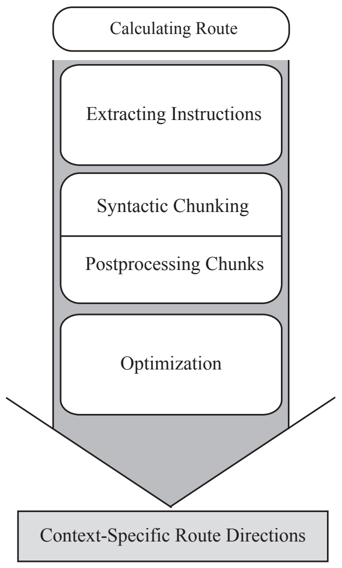


**Fig. 2.** Overview of GUARD, the generation process for context-specific route instructions.

In summary, the approach to context-specific route instructions finds the best instruction sequence, according to the optimization criterion, but for a given route.

## 3 Simplest Instructions Algorithm

In this section, the "simplest instructions" (SI) algorithm will be introduced. The algorithm combines the reasoning behind both simplest paths and context-specific route instructions. Like GUARD, the SI algorithm generates the best instructions for a route according to optimization criteria related to human direction giving. Further, like simplest paths, the algorithm *finds* the best route, i.e., the route associated with the lowest cost in terms of instruction complexity.

### 3.1 The Algorithm

The SI algorithm is based on Dijkstra's shortest path algorithm. Like Dijkstra's algorithm, the SI algorithm operates on a network represented as a graph $G = (V, E)$ comprising a set of vertices $V$ and edges $E$ connecting vertices, $E \subseteq V \times V$. Dijkstra's algorithm determines for each vertex in a graph the shortest path from a given start vertex (origin). It uses a cost function that determines the cost of traversing an edge. These costs are represented as the edges' labels. Starting from the origin, at each step the edge with the lowest costs is selected, which is then marked as visited. The costs for reaching all unvisited edges adjacent to the current vertex are then updated, i.e., it is checked whether the newly found path from origin to these edges is cheaper than the previously known one.

The SI algorithm, given in Algorithm 1 and explained in more detail below, differs from Dijkstra's algorithm in three key respects, considered in more detail in the following subsections:

1. the algorithm models the *instructions* required to describe a route, including the possibility of using landmarks in those instructions (section 3.2);
2. the cost function is associated with pairs of edges, rather than individual edges, to represent the cognitive cost of negotiating a decision point (section 3.2); and
3. the algorithm accounts for chunking of instructions, combining multiple instructions into one low cost (i.e., cognitively efficient) instruction (section 3.3).

### 3.2 Instructions and costs

The *complete line graph* (or *evaluation mapping*) is the graph $G' = (E', \mathcal{E})$. $E'$ is the set of edges in $G$, where the direction of edges is ignored (i.e., $(v_i, v_j) = (v_j, v_i)$ in $E'$). $\mathcal{E}$ is the set of pairs of vertices in $E$ that share their "middle" vertex, i.e., $\mathcal{E} = \{((v_i, v_j), (v_j, v_k)) \in E \times E\}$ [20, 16]. We refer to the elements of $\mathcal{E}$ as *decisions*. In other words, a pair of adjacent edges in $E$ represents a decision an agent can take to move from one edge to the next.

The algorithm models the instructions required to describe a route as a set $I$ of (arbitrary) labels. Instructions are associated with pairs of adjacent edges ("decisions"), describing a decision to move from one edge to another (e.g., "turn left at the intersection"). Instructions may include references to landmarks (e.g., "turn left at the post office"). Each pair of adjacent edges may have zero or more instructions associated with it (e.g., the instructions "turn left at the intersection" and "turn left at the post office" might both encode the same decision at a particular decision point). Each instruction may be associated with zero or more pairs of adjacent edges (e.g., "turn left at the intersection" might be a valid instruction for describing decisions at several different decision points). However, we assume no ambiguity in instructions, and disallow the possibility that the same instruction might be used to encode different decisions from the *same* edge (e.g., where "turn left at the intersection" can be used to describe more than one decision at a particular edge).

Formally, for the set of instructions $I$, the *labeling* function $l$ is defined to be $l : \mathcal{E} \rightarrow I^2$. Thus, for a given pair of adjacent edges $(e, e')$, $l(e, e') = \{i_1, .., i_n\}$ gives the (possibly empty) set of instructions that describe that decision. Conversely, the *decision* function is defined to be $d : E \times\ I\ \rightarrow E \cup\ \{\varnothing\}$. For a given edge $e$ and instruction $i$, $d(e, i) = e'$ gives the edge $e'$ that results from executing the decision $i$ at edge $e$. Note that $e'$ may be the empty set (indicating that it is not possible to execute the decision $i$ at edge $e$). Further note that since $d$ is functional, we disallow ambiguity: there will be at most one edge $e' \in E$ that results from applying an instruction at edge $e$.

Each instruction has a cost associated with it using the function $w : I \rightarrow \mathbb{R}^+$ that models the cognitive effort associated with executing an instruction. Thus for a given instruction $i$, $w(i)$ yields the *cognitive cost* of executing instruction $i$. Potentially any cost function may be used, but in this empirical evaluation of the algorithm that follows we adopt the same cost function as previous work [17, 19]. The SI algorithm minimizes the costs associated with traversing *pairs of edges* rather than individual edges (cf. [16]). In this way, the SI algorithm aims to minimize the *cognitive cost* of negotiating the decision points in a route, instead of minimizing the costs associated with travel (like distance or travel time).

## 3.3 Chunking

Allowing for multiple instructions for an edge corresponds to having a set of possible instructions to describe how to reach the next decision point from the current one (similar to step 1 of GUARD). It is important to note that these multiple instructions do not correspond to having multiple edges between two vertices. Each instruction may have different costs associated with it and selection of an edge is determined by the instruction with the lowest costs. In the SI algorithm a vertex that has once been selected as the one with the lowest costs is never visited again in the search for the optimal path. If each instruction corresponded to an edge, then after selecting the one with the lowest costs all other edges connecting the two vertices would be unreachable. However, as discussed below, this would render transferring spatial chunking to the path-search algorithm impossible. That is, since a global selection criterion—spatial chunking—is introduced, the local selection criterion—choosing the next edge based on a instruction's costs—needs to account for more than one possibility to traverse this edge.

Spatial chunking is realized in the SI algorithm by spreading instructions forward through the graph from the edge currently being processed. This corresponds to step 2 of GUARD. Humans do not generate instructions with arbitrarily long chunks, so spreading instructions also needs to account for the cognitive and structural characteristics of the emerging chunks. This is implemented in a way that the approach is flexible with respect to the superordinate chunking principles used (again, this is as in GUARD). Considering, for instance, the principles discussed in [4] and [17], chunks cannot be arbitrarily long unless a structural feature, such as a landmark, unambiguously marks its end.

In terms of spreading instructions through the graph, a distinction needs to be drawn between an edge being *reachable* and being *chunkable*. An edge $e_t$ is *reachable* from edge $e_s$ with an instruction $i$ if there exists a path from $e_s$ to $e_t$ that can be encoded as sequence of executions of instruction $i$. An edge $e_t$ is *chunkable* from $e_s$ with an instruction $i$ if the sequence of $i$ instructions required to reach $e_t$ from $e_s$ is also valid according to the employed superordinate chunking rules. As chunkable edges can be covered with a single instruction, the costs for all edges in a chunk are the same, namely those for reaching the first edge of a chunk using the chunked instruction.

An instruction needs to be spread forward as long as there are edges reachable with it. However, a cost update is only performed for those edges that are chunkable. This way of realizing spatial chunking can be viewed as dynamically introducing new edges to the graph that connect first and last vertex of a chunk. Figure 3 illustrates how instructions spread across neighboring edges and the distinction between reachable and chunkable edges. This cost updating is the reason why all instructions need to be considered when selecting the edge with the lowest costs. Globally, it might be less expensive to select an instruction that is locally more expensive if this instruction allows to cover more edges in the final path.

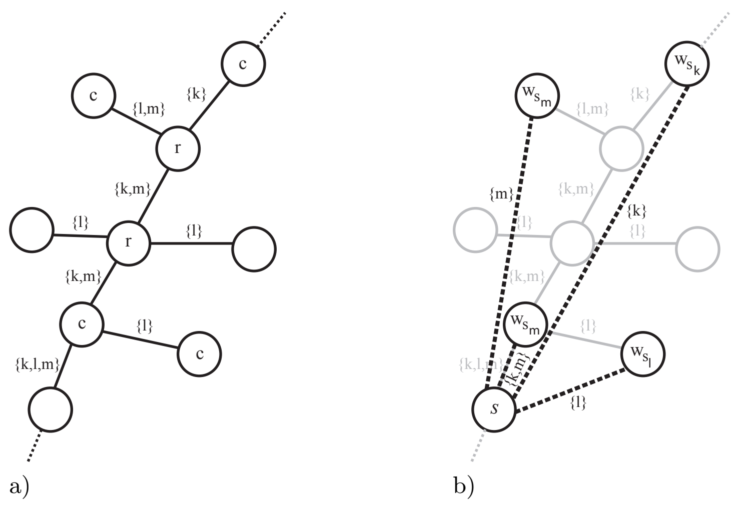


**Fig. 3.** Spreading instructions: a) If the instructions for reaching consecutive edges match, a vertex is *reachable*, denoted by $r$. If this combination also adheres to the superordinate chunking principles, a vertex is *chunkable*, denoted by $c$; b) Chunkable vertices can be reached in a single step. This corresponds to dynamically introducing new edges between the first vertex of the chunk (denoted by $s$) and the chunkable one; reaching these vertices has the same costs as reaching the chunk's first edge (denoted by the different $w_s$).

Formally, to implement this behavior in our algorithm we assume a *chunk validity* function $v : E \times E \times I \rightarrow \{true, false\}$. For a given start edge $e_s \in E$, terminator edge $e_t \in E$, and instruction $i \in I$, $v(e_s, e_t, i) = true$ only if $e_t$ is chunkable from $e_s$ using instruction $i$. We do not provide any further details of the actual procedure used to check chunk validity in this paper, since this issue is already covered in great detail in [4] and [17].

## 3.4 Algorithm description

The SI algorithm is presented in Algorithm 1. In addition to the structures introduced above (the graph $G$, the complete line graph of $G$, the instruction set $I$, the labeling function $l$, the decision function $d$, the chunk validity function $v$) the algorithm requires an origin (starting) edge $o$ as input.

The algorithm generates a predecessor function, $p : E \rightarrow E \times I$, that stores for each edge the preceding edge in the least cost path and the instruction used to reach the edge from its predecessor. If several edges are chunkable by an instruction, this predecessor is the first edge of the chunk (see Figure 3). Accordingly, when the algorithm visits an edge $e$, it needs to be checked for every instruction holding for $(e, e')$ whether it lowers the costs associated with $e'$, and whether it can be used to reach other edges from $e'$ as well (Algorithm 1, lines 12–13). Chunking (i.e., the updating of weights to edges other than those directly connected to the current edge) can be postponed until after the edge weights have been updated (Algorithm 1, lines 14–30). In order to keep track of which instructions need to be further considered at an edge, all instructions the edge has been reached by so far need to be stored, along with the edge the current edge has been reached on (for correctly setting predecessors). Formally, for each edge, the algorithm also stores a set $U_e$ of instruction/edge pairs $(i, e)$, initialized in line 4.

Next, the algorithm updates the weights associated with the unvisited edges that are incident with the current edge $e$ (Algorithm 1, lines 14–18). This step is as used in the simplest path algorithm [16], and is essentially the same core condition used in Dijkstra's algorithm operated upon the complete line graph.

Finally, Algorithm 1, lines 14–30 performs chunking by propagating instructions forward from the current edge as far as is possible. As already discussed, all edges that are reachable store information about the instruction being forward propagated (line 26), but only those edges for which the resulting instruction would be a valid chunk have their weights updated (lines 27–29).

## 3.5 Computational time complexity analysis

The computational time complexity of finding the lowest cost path through the graph is $O(|E|^2)$, because in the worst case for each edge visited the algorithm must update the costs of every other edge. Similarly, the computational cost of spreading the selected instruction through the graph is $O(|E|^2)$, because it requires visiting each edge in turn and in the worst case, spreading instructions to every other edge in the graph. Thus the overall time complexity of the algorithm is $O(|E|^2 + |E|^2) = O(|E|^2)$. Compared to Dijkstra's shortest path algorithm, which is $O(|V|^2)$, in a totally connected graph with $|E| = |V|^2$ edges, this results in an overall time complexity of $O(|V|^4)$.

However, as argued in [16], geographic routing problems never deal with totally connected graphs. If instead a planar graph is assumed, then by Euler's formula (simple connected planar graph has $n$ vertices - $m$ edges + $f$ faces = 2) the number of edges is linear in the number of nodes, $|E| \leq 3(|V| - 2)$. Thus, the

**Algorithm 1**: Simplest instruction algorithm with multiple instructions and postponed chunking

```
Data: $G = (V, E)$ is a connected, simple, directed graph; $G' = (E', \mathcal{E})$ is the
      complete line graph of $G$; $o \in E$ is the origin (starting) edge; $I$ is a set of
      instructions; $w : I \to \mathbb{R}^+$ is the instruction weighting function; $l : \mathcal{E} \to I^2$
      is the labeling function; $d : E \times I \to E \cup \{\varnothing\}$ is the decision function;
      $v : E \times E \times I \to \{true, false\}$ is the chunk validity function.
Result: Function $p : E \to E \times I$ that stores for each edge the preceding edge
        and the instruction used in the least cost path.
1  // Initialize values;
2  forall $e \in E$ do
3      Initialize $c : E \to \mathbb{R}^+$ such that $c(e) \leftarrow \infty$;
4      Initialize $U_e \leftarrow \varnothing$;
5  Set $S \leftarrow \{\}$, a set of visited edges;
6  Set $p(o) \leftarrow (o, i)$ for some arbitrary $i \in I$;
7  Set $c(o) = 0$;
   // Process lowest cost edge until all edges are visited
8  while $|E \backslash S| > 0$ do
9      Find $e \in E \backslash S$ such that $c(e)$ is minimized;
10     Add $e$ to $S$;
11     forall $e' \in E \backslash S$ such that $(e, e') \in \mathcal{E}$ do
           // Update instruction/edge pairs from e to e'
12         forall $i \in l(e, e')$ do
13             $U_{e'} \leftarrow U_{e'} \cup \{(i, e)\}$;
14     forall $e' \in E \backslash S$ such that $(e, e') \in \mathcal{E}$ do
           // Update costs to e' based on instruction weights
15         forall $i \in I$ such that $d(i, e) = e'$ do
16             if $c(e') > c(e) + w(i)$ then
17                 Set $c(e') \leftarrow c(e) + w(i)$;
18                 Set $p(e') \leftarrow (e, i)$;
       // Perform chunking by propagating instructions forward
19     forall $(i, e_p) \in U_e$ do
20         Set $e_x \leftarrow e'$;
21         Set $X \leftarrow S \cup \{\varnothing\}$;
22         while $e_x \notin X$ do
23             $X \leftarrow X \cup \{e_x\}$;
24             Set $e_n \leftarrow d(e_x, i)$;
25             if $e_n \neq \varnothing$ then
26                 $U_{e_n} \leftarrow U_{e_n} \cup \{(i, e_p)\}$;
27                 if $c(e_n) > c(e')$ and $v(e_p, e_n, i) = true$ then
28                     Set $c(e_n) \leftarrow c(e')$;
29                     Set $p(e_n) \leftarrow (e_p, i)$;
30             $e_x \leftarrow e_n$;
```

overall time complexity for the SI algorithm applied to a planar graph is $O(|V|^2)$. This is the same as the complexity of Dijkstra's algorithm, so we conclude the SI algorithm does not increase the computational time complexity compared with the simplest path or shortest path algorithms, at least for planar graphs.

### 3.6 Reconstructing routes

Algorithm 1 generates a predecessor function $p : E \rightarrow E \times I$ that stores for each edge the preceding edge in the least cost path and the instruction used to reach the edge from its predecessor. Reconstructing a path to a particular destination edge $t$ is then simply a matter of backtracking from $t$ using $p$, at each step storing the predecessor edge and instruction in a list. Algorithm 2 gives an example of reconstructing routes using $p$ (using an algebraic language notation, where $E$ and $I$ form alphabets, and the lists of edges and instructions in the route are stored as sequences $P$ and $L$ of letters—words—from those alphabets, constructed by iteratively prepending letters to the initially empty word with the concatenate $+$ operator). In Algorithm 2, retrieval of the instruction word $L$ simply requires direct backtracking through the predecessor list (line 7). Construction of the edge list word $P$ requires an additional loop to retrieve the edges between chunked instructions (lines 8–15).

**Algorithm 2**: Algorithm for reconstructing the simplest instruction path

**Data**: $G = (V, E)$ is a connected, simple, directed graph; $o \in E$ is the origin (starting) edge; $t \in E$ is the target (destination) edge; $d : E \times I \rightarrow E \cup \{\varnothing\}$ is the decision function; $p : E \rightarrow E \times I$ is the predecessor function (generated by algorithm 1).

**Result**: A sequence (word) $P$ of edges corresponding to the optimal path and a sequence (word) $L$ of labels corresponding to the best sequence of instructions.

```
1  Set $P$ to be the empty word $\lambda$;
2  Set $L$ to be the empty word $\lambda$;
3  Set $T$ to be the empty word $\lambda$;
4  Set $e \leftarrow t$;
5  while $e \neq o$ do
6      Let $p(e) = (e_p, i)$;
7      Set $L \leftarrow i + L$;
8      Set $e' \leftarrow e_p$;
9      Set $T \leftarrow e'$;
10     while $(e', e) \notin E$ do
11         $e'' \leftarrow d(e', i)$;
12         Set $T \leftarrow T + e''$;
13         Set $e' \leftarrow e''$;
14     Set $P \leftarrow T + P$;
15     Set $e \leftarrow e_p$;
```

## 4 Comparisons

As we have demonstrated in the last section, the SI algorithm is able to incorporate fundamental principles of human direction giving without increasing the overall computational complexity of the route generation algorithm. In addition to computational complexity, other measures of the performance of the SI algorithm are the length of paths the algorithm generates, and the number of instructions required to describe the route.

The length of the path described by the simplest instructions is necessarily equal to or longer than the shortest path. Thus, the length of the simplest instruction path, when compared with the shortest path, provides a measure of the detour a wayfinder would need to take when using the simplest instruction path. Conversely, the length of instructions generated by the SI algorithm is expected to be shorter that those generated by the shortest path (assuming one instruction per decision point).

To analyze these aspects, the lengths of paths and instruction sequences produced by the SI algorithm are compared with those produced by Dijkstra's shortest path as well as the simplest path algorithm. Thus, these results aim to provide an indication of the balance struck by the different algorithms between the desire for direct (short) routes and simple (short) route instructions.

### 4.1 Data

The algorithm's performance was tested using several different geographic data sets. The results in this section were derived from a transportation network data set representing part of the inner city (CBD) and surrounding districts of Melbourne, Australia (see Figure 4). The transportation network was augmented with objects representing landmarks. The landmark objects in this case were derived from the railway infrastructure in the area (essentially they are train stations). Clearly, railway infrastructre will not always be appropriate for human wayfinding, but provides an adequate simplification in the context of the following experiments.

### 4.2 Results of experiments on path length

All three algorithms—shortest path, simplest path, simplest instructions—were used to calculate paths between randomly chosen origins and destinations in the network. For each origin/destination pair, each algorithm calculated a path. For each of 53,000 different origin/destination pairs tested, the resulting paths from each algorithm were compared in terms of path length and the number of associated instructions required to describe the path (in simplest and shortest paths, the number of decision points; in simplest instructions the number of chunked instructions). Figure 5 depicts typical differences in shortest, simplest, and simplest instruction paths between two points.

On average, paths determined with the SI algorithm were 13.31% longer than the shortest path between origin and destination. Using simplest paths,

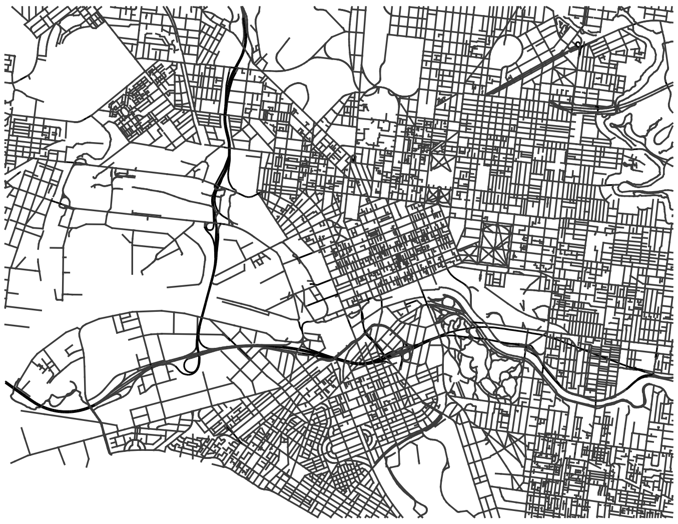


**Fig. 4.** The test area: part of Melbourne, Australia.

there was an increase in length of 12.52% (a result comparable to [16], which found lengths of simplest paths that on average were 15.8% longer than the corresponding shortest path). The average path length of simplest paths and simplest instructions were almost equal (3,645.82m to 3,671.49m, standard deviation of 1,821.49m and 1,840.99m, respectively).

A *t*-test was conducted to test the hypothesis that the simplest instruction paths are longer than the simplest paths. The test showed that the differences in length were significant at the 5% level, so we conclude that the SI algorithm does generate paths that are longer than the simplest path. However, while the differences were significant, a test for effect size results in very small differences, 0.06. This value indicates that the actual differences in length between simplest paths and simplest instruction paths are very small; the average increase in length for paths generated by the SI algorithm is 25.67m.

With respect to the shortest paths, the simplest instruction paths were again significantly longer than the corresponding shortest path, as expected. In more detail, from all 53,000 paths, 40,232 SI paths were less than 15% longer than the corresponding shortest (with 6829 paths being equally long to the shortest path); 3481 were more than 25% longer than the shortest path; only for 54 paths was this increase more than 50%. In summary, for only 6.7% of all cases were the simplest instruction paths more than 25% longer than the shortest path; in 75.9% of all cases the simplest instructions paths were less than 15% longer than the shortest path.

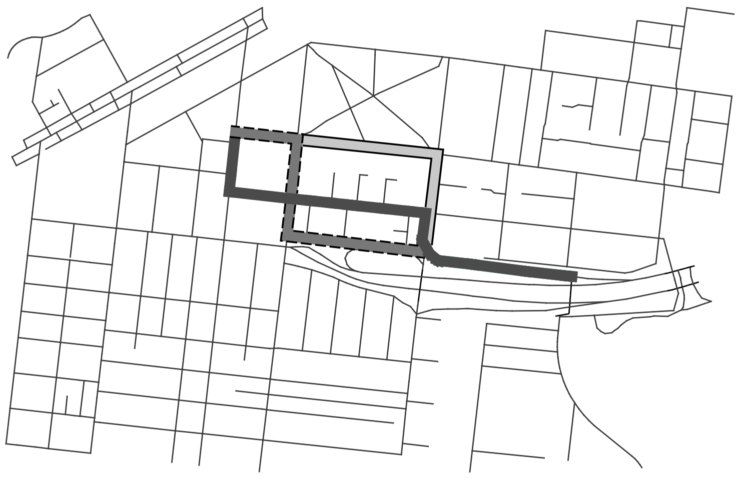


**Fig. 5.** Shortest path (the thick black line), simplest path (the gray line), and SI path (the dark gray line with dashed border) for a sample origin / destination pair.

### 4.3 Results of experiments on instruction length

The algorithm for simplest instructions significantly reduces the number of instructions needed to descibe the routes generated. Assuming an instruction is required for every decision point in shortest and simplest paths, the number of instructions required to describe the simplest instruction paths was on average average 57.93% less than required for shortest paths, and 55.37% less than required for simplest paths. A Wilcoxon signed-rank test confirms that this difference is statistically significant at the 5% level. This result is comparable to previous evaluations done with GUARD and illustrates the strengths of spatial chunking even when employed on data sparsely annotated with landmarks. The result also highlights that while simplest paths can help to minimize the cognitive complexity of individual instructions used in the route, they do not necessarily provide savings in terms of the global number of instructions required to describe the route. By contrast, the paths generated by the SI algorithm do dramatically reduce the overall length of route instructions.

One final test was to compare the length of the instructions generated by the SI algorithm with the length of instructions generated using GUARD applied to the corresponding shortest path. As expected, in all cases the length of the instructions generated by the SI algorithm was equal to or less than the corresponding chunked instructions generated by GUARD applied to the shortest path. We used the same Wilcoxon-test as before to test the difference for stastitical significance. It shows that there is a significant advantage in using the SI algorithm over simply applying GUARD to the shortest paths.

## 5 Conclusions

We have presented the simplest instructions (SI) algorithm, which is based on Dijkstra's shortest path algorithm. The SI algorithm integrates fundamental principles of human direction giving, namely references to landmarks and spatial chunking, in finding a route between an origin and a destination. Multiple labels attached to an edge capture several options to describe which action to perform for reaching the next decision point. Generation of these labels is based on GUARD, a process for producing cognitively ergonomic route instructions. Spatial chunking, i.e., the subsumption of several consecutive instructions to a single one, is realized by spreading labels forward through the graph and dynamically introducing new edges. The SI algorithm computes paths in the same order of time complexity as the generalized Dijkstra algorithm, $O(n^2)$.

An empirical evaluation, comparing the path lengths produced by the SI algorithm with the corresponding shortest and simplest paths, has shown promising results. On average, the SI paths are about 13% longer than the shortest path, which is in the same range as the increase in length introduced by simplest paths. The length of instructions, however, is decreased by 57%, i.e., on average slightly more than two consecutive instructions can be chunked into a single one, reducing the amount of information that needs to be communicated by more than half. Thus, with only a slight increase in path length, the SI algorithm produces instructions that can be expected to be significantly easier to follow.

For a small number of paths, however, there is a considerable increase in path length. To counter these cases, the algorithm may be adapted to balance the increase in path length and the ease of instructions. Here, human subject studies are called for to elicit sensible parameters. Furthermore, as spatial chunking relies to a good part on the presence of landmarks, a more detailed analysis of the relationship between the density of landmarks on one hand, and path length and chunking ratio on the other hand will reveal a more detailed picture of the algorithm's performance and may point out refined methods in choosing labels and applying chunking. Also, an analysis of how an environment's structure influences the resulting paths may allow identifying strategies on how instructions may be automatically adapted to different environmental situations.

## Acknowledgments

Kai-Florian Richter's research is supported by the Transregional Collaborative Research Center SFB/TR 8 Spatial Cognition, which is funded by the Deutsche Forschungsgemeinschaft (DFG). Matt Duckham's research is partially supported under the Australian Research Council's Discovery Project funding scheme (project number DP0662906).

## References

1. Tversky, B., Lee, P.U.: Pictorial and verbal tools for conveying routes. In Freksa, C., Mark, D.M., eds.: Spatial Information Theory, Berlin, Springer (1999) 51–64 LNCS 1661.
2. Habel, C.: Incremental generation of multimodal route instructions. In: Natural Language Generation in Spoken and Written Dialogue, Palo Alto, CA, AAAI Spring Symposium 2003 (2003)
3. Dale, R., Geldof, S., Prost, J.P.: Using natural language generation in automatic route description. Journal of Research and Practice in Information Technology **37**(1) (2005) 89–105
4. Klippel, A., Tappe, H., Kulik, L., Lee, P.U.: Wayfinding choremes — a language for modeling conceptual route knowledge. Journal of Visual Languages and Computing **16**(4) (2005) 311–329
5. Dijkstra, E.: A note on two problems in connexion with graphs. Numerische Mathematik **1** (1959) 269–271
6. Denis, M.: The description of routes: A cognitive approach to the production of spatial discourse. Cahiers Psychologie Cognitive **16**(4) (1997) 409–458
7. Lovelace, K.L., Hegarty, M., Montello, D.R.: Elements of good route directions in familiar and unfamiliar environments. In Freksa, C., Mark, D.M., eds.: Spatial Information Theory, Berlin, Springer (1999) 65–82 LNCS 1661.
8. Klippel, A., Tappe, H., Habel, C.: Pictorial representations of routes: Chunking route segments during comprehension. In Freksa, C., Brauer, W., Habel, C., Wender, K.F., eds.: Spatial Cognition III, Berlin, Springer (2003) 11–33 LNAI 2685.
9. Hirtle, S.C., Jonides, J.: Evidence of hierarchies in cognitive maps. Memory & Cognition **13**(3) (1985) 208–217
10. Couclelis, H., Golledge, R.G., Gale, N., Tobler, W.: Exploring the anchor-point hypothesis of spatial cognition. Journal of Environmental Psychology **7** (1987) 99–122
11. Michon, P.E., Denis, M.: When and why are visual landmarks used in giving directions? In Montello, D.R., ed.: Spatial Information Theory, Berlin, Springer (2001) 400–414 LNCS 2205.
12. Raubal, M., Winter, S.: Enriching wayfinding instructions with local landmarks. In Egenhofer, M., Mark, D., eds.: Geographic Information Science, Berlin, Springer (2002) 243–259 LNCS 2478.
13. Elias, B.: Extracting landmarks with data mining methods. In Kuhn, W., Worboys, M., Timpf, S., eds.: Spatial Information Theory, Berlin, Springer (2003) 375–389 LNCS 2825.
14. Caduff, D., Timpf, S.: The landmark spider: Representing landmark knowledge for wayfinding tasks. In Barkowsky, T., Freksa, C., Hegarty, M., Lowe, R., eds.: Reasoning with mental and external diagrams: computational modeling and spatial assistance - Papers from the 2005 AAAI Spring Symposium, Menlo Park, CA (2005) 30–35
15. Hansen, S., Richter, K.F., Klippel, A.: Landmarks in OpenLS - a data structure for cognitive ergonomic route directions. In Raubal, M., Miller, H., Frank, A.U., Goodchild, M.F., eds.: Geographic Information Science, Berlin, Springer (2006) 128–144 LNCS 4197.
16. Duckham, M., Kulik, L.: ”Simplest” paths: Automated route selection for navigation. In Kuhn, W., Worboys, M., Timpf, S., eds.: Spatial Information Theory, Berlin, Springer (2003) 169–185 LNCS 2825.

17. Richter, K.F., Klippel, A.: A model for context-specific route directions. In Freksa, C., Knauff, M., Krieg-Brückner, B., Nebel, B., Barkowsky, T., eds.: Spatial Cognition IV, Berlin, Springer (2005) 58–78 LNAI 3343.
18. Mark, D.: Automated route selection for navigation. IEEE Aerospace and Electronic Systems Magazine **1** (1986) 2–5
19. Richter, K.F.: A uniform handling of different landmark types in route directions. In Winter, S., Duckham, M., Kulik, L., Kuipers, B., eds.: Spatial Information Theory, Berlin, Springer (2007) 373–389 LNCS 4736.
20. Winter, S.: Weighting the path continuation in route planning. In: GIS '01: Proceedings of the 9th ACM international symposium on Advances in geographic information systems, New York, NY, USA, ACM (2001) 173–176