“Simplest” Paths: Automated Route Selection for Navigation

Matt Duckham and Lars Kulik

National Center for Geographic Information and Analysis
University of Maine, Orono, ME 04469, USA
{duckham,kulik}@spatial.maine.edu

Abstract. Numerous cognitive studies have indicated that the form and complexity of route instructions may be as important to human navigators as the overall length of route. Most automated navigation systems rely on computing the solution to the shortest path problem, and not the problem of finding the “simplest” path. This paper addresses the issue of finding the “simplest” paths through a network, in terms of the instruction complexity. We propose a “simplest” paths algorithm that has quadratic computation time for a planar graph. An empirical study of the algorithm’s performance, based on an established cognitive model of navigation instruction complexity, revealed that the length of a simplest path was on average only 16% longer than the length of the corresponding shortest path. In return for marginally longer routes, the simplest path algorithm seems to offer considerable advantages over shortest paths in terms of their ease of description and execution. The conclusions indicate several areas for future research: in particular cognitive studies are needed to verify these initial computational results. Potentially, the simplest paths algorithm could be used to replace shortest paths algorithms in any automated system for generating human navigation instructions, including in-car navigation systems, Internet driving direction servers, and other location-based services.

Keywords. Navigation, wayfinding, route selection, shortest path, instruction complexity.

1 Introduction

Most people will have had the experience of giving or receiving directions for navigating through an unfamiliar geographic environment. For example, visiting a foreign city for the first time, a tourist might ask a passer-by for directions to a hotel or visitor attraction. In such situations, often what is required is not the *shortest* route to a destination, but the *simplest* route, in terms of how easy it is to explain, understand, memorize, or execute the navigation instructions for the route. Most automated navigation systems rely on computing the solution to the shortest path problem, and not the problem of finding the “simplest” path.

As a motivational example, consider the network in Fig. 1. This network might represent, for instance, the block structure of a road network in a city.

Assuming the network is embedded in the Euclidean plane, there exist six equivalent optimal shortest routes from intersection i_1 (upper left) to intersection i_9 (lower right)[1]. Without yet defining precisely what is meant by "simplest," intuitively only two of these six routes seem to be optimal simplest routes. Only $(i_1, i_2, i_3, i_6, i_9)$ and $(i_1, i_4, i_7, i_8, i_9)$ avoid the more complex 4-way intersection i_5, and only these two routes require just one "turn." We might describe the route $(i_1, i_2, i_3, i_6, i_9)$ with the instruction sequence "orient yourself, go straight ahead, and turn right at the end of the road." In contrast, the route $(i_1, i_2, i_5, i_8, i_9)$ might require a longer instruction sequence to describe, such as "orient yourself, go straight ahead, turn right at the first intersection, then turn left at the second intersection."

In this paper, we propose an algorithm that can be used to select routes that minimize the complexity of instructions, rather than the distance traveled. The goal is to simplify navigation in an unknown environment when following a route instruction. Following the literature review in Section 2, the algorithm is introduced in Section 3, and its computational properties are reviewed. In Section 4, an empirical comparison of the simplest paths algorithm with the shortest paths algorithm is conducted, using an example road network data set. A discussion of the results, conclusions, and suggestions for further work is contained in Section 5.

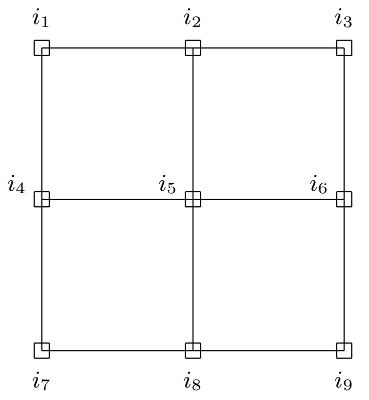


Fig. 1. Example network with six shortest and two "simplest" paths

[1] i.e. $(i_1, i_2, i_3, i_6, i_9)$, $(i_1, i_2, i_5, i_6, i_9)$, $(i_1, i_2, i_5, i_8, i_9)$, $(i_1, i_4, i_5, i_6, i_9)$, $(i_1, i_4, i_5, i_8, i_9)$, $(i_1, i_4, i_7, i_8, i_9)$

2 Background

Several cognitive studies have indicated that the form and complexity of route instructions may be as important in human navigation as the overall length of a route. Streeter and co-authors [1, 2] looked at the use of verbal instructions in human navigation. They found that human navigators were prepared to select suboptimal routes, in terms of the total length of a route, in favor of routes that were potentially easier to describe or follow. Golledge ranked ten different criteria used in human route selection, based on experiments using human subjects [3]. While the related criteria of shortest distance and least time were ranked most highly, many other criteria, including number of turns, also ranked highly. Michel Denis and coauthors studied the "informational units" contained within human route descriptions [4]. Amongst other findings, Denis et al. showed that those route descriptions that were considered more likely to prevent users from making errors were preferred by study subjects. Following on from Denis, Tversky and Lee have examined the relationship between pictorial and verbal descriptions of routes [5, 6]. Tversky and Lee found both pictorial and verbal route descriptions exhibited considerable redundancy in information, a feature they attribute to the importance of reassurance and error prevention for human navigators. Research by Richter and Klippel [7], into optimal locations for "You-are-here" maps, emphasizes the importance of the overall number of decision points and the number of branches at an intersection as route selection criteria, in addition to the length of a route. Research from the vehicle navigation literature has also confirmed that successful vehicle navigation systems rely as much on clarity of route instructions as length of route [8, 9]. Shortest path algorithms only minimize route efficiency, in terms of distance or travel cost, and not route description complexity.

Algorithms for finding an optimal route that is not the shortest path have been proposed by Shapiro et al. [10] as well as Liu [11, 12]. The approach of Shapiro et al. can be used to take advantage of the type of roads (major roads versus minor roads), generating a route that prefers major routes. The time to find a route in the graph is considerably reduced, since the algorithm essentially restricts the search to major roads. Liu [11, 12] also developed an efficient approach that incorporates road network knowledge to narrow the search for routes, using the fact that major roads partion a road network. However, none of these approaches have focused explicitly on the simplicity of a route description.

The aim of the simplest path algorithm presented in this paper is to minimize the complexity of a route description, based on the amount of information required to negotiate each decision point. Determining how much information is communicated by some route description may itself be a difficult question (see [13–15] for a discussion of the information content of geographic information in general and route descriptions in particular). However, the simplest path algorithm presented in this paper does not rely on any particular model of geographic information content, so long as some measure of instruction complexity can be derived for decision points along a route.

2.1 Automated Route Selection for Navigation

In 1986, David Mark [16] published a paper entitled "Automated route selection for navigation" (hence the subtitle for this paper). Mark proposed a modification of the A* shortest path algorithm, which took into account both the total length and the "ease of description" of the route. Based on work of Streeter and co-authors [1, 2], Mark [17] classified different intersections according to the complexity of the instructions needed to successfully negotiate that intersection. Using this classification, Mark's algorithm adjusted the weights used in the shortest path computation to preferentially select routes through intersections that could be described using less complex instructions.

The simplest paths algorithm below was motivated in part by the work presented by Mark. However, the algorithm below differs in at least three ways from that presented in [16]. First, the simplest path algorithm does not use distance or any other metric information in its operation. The algorithm computes the simplest paths using only a measure of instruction complexity. Even without any metric information, the results in Section 4 indicate that simplest paths are still comparable in length to shortest paths. Second, the lack of distance information means the simplest path algorithm does not depend on an arbitrary assignment of weights to the relative importance of distance and instruction complexity. Given a weighting function for instruction complexity, the simplest path algorithm proposed here yields a uniquely determined simplest path from start to destination. Third, the algorithm presented below is a single source algorithm, able to efficiently compute the distance from a single source vertex to every other vertex. Recent work by Duckham et al. [18] has indicated that single source route finding algorithms are particularly important in the context of navigation under imprecision, where a user's precise location or destination may be unknown. Other uses of single source route finding might include applications where multiple route options need to be presented to the user, for example finding the routes from a user's current location to a variety of tourist attractions. The computational properties of the algorithm are discussed in more detail in Section 3.1.

3 Formal Model of Simplest Paths

Graphs are a common mechanism for representing networks, such as road networks. A graph G comprises a set of vertices V and edges E connecting those vertices. A weighted graph additionally has a function $w : E \rightarrow \mathbb{R}^+$ associating a weight with 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. There have been estimated to be more than 2000 articles published on the topic since the 1950s [19]. As a result, we make no attempt here to summarize the different approaches to computing shortest paths; any introductory textbook on algorithms (e.g. [20]) or artificial intelligence (e.g. [21]) will contain such a summary.

The essential difference between shortest and simplest paths algorithms is that the latter uses a weighting function that associates a weight with each *pair of connected edges* (rather than each edge) in the graph, $w : \mathcal{E} \to \mathbb{R}^+$ where $\mathcal{E} = \{\big((v_i, v_j), (v_j, v_k)\big) \in E \times E\}$. The intuition behind these weights is that they should reflect the amount or complexity of information required to negotiate the "decision point" represented by the edge pair (i.e. negotiating the path from v_i to v_k through intersection v_j). The more information needed, the higher the weight for an edge pair. The later discussion in Section 4.1 contains an example of such a function.

The actual simplest path algorithm is presented in Algorithm 1 below. The graph used in the simplest path algorithm is assumed to be directed (i.e. the direction of the edges is significant), connected (i.e. there exists a path from any vertex to any other vertex), and simple (i.e. there are no edges from a vertex to itself and at most one edge between two different vertices). Algorithm 1 operates by first initializing all edges connected to the starting vertex with zero weight. Thus, the algorithm applies no cost to the initial orientation stage of routing. A more sophisticated algorithm might include weights for initial orientation, perhaps preferentially selecting initial orientations that are easier to explain (for example, orientation towards an easily visible nearby landmark). Next, the algorithm iterates through each edge, minimizing the cumulative instruction complexity. At each iteration, the edge associated with the minimum instruction complexity is selected, and the cumulative instruction complexity from that edge to all connected edges is recalculated. The algorithm interates until all edges have been visited. At no point in the algorithm is any distance information involved in the calculation, only the instruction complexity.

Algorithm 1: Simplest path algorithm

```
**Initial conditions**: $G = (V, E)$ is a connected, simple, directed graph; $s \in V$ is the starting vertex; $\mathcal{E}$ is the set of pairs of (directed) edges that share their "middle" vertex, $\mathcal{E} = \{((v_i, v_j), (v_j, v_k)) \in E \times E\}$; $w : \mathcal{E} \to \mathbb{R}^+$ is the graph weighting function; $c_s : E \to \mathbb{R}^+$ stores the weights of the simplest path from $s$; $S = \{\}$ is a set of visited edges.

Initialize $c_s(e) = \infty$ for all $e \in E$
for all $(s, v_i) \in E$ do
    set $c_s(s, v_i) = 0$
while $|E \backslash S| > 0$ do
    Find $e \in E \backslash S$ such that $c_s(e)$ is minimized
    Add $e$ to $S$
    for all $e' \in E \backslash S$ do
        if $(e, e') \in \mathcal{E}$ then
            set $c_s(e') = \min\big(c_s(e'), c_s(e) + w(e, e')\big)$
```

The function c_s in Algorithm 1 stores the weights of the simplest paths from s to every destination vertex. To recover the simplest path to a particular destination vertex $d \in V$, we must first find the edge $(v_i, d) \in E$ where $c_s(v_i, d)$ is minimum. Reconstructing the simplest path is then a matter of iterating backwards through the edges, at each iteration choosing the least costly edge (Algorithm 2).

Algorithm 2: Retrieving the simplest path using c_s (see also Algorithm 1)

```
**Initial conditions**: Weights $c_s$ from simplest path algorithm (see Algorithm
1), starting vertex $s \in V$ and destination vertex $d \in V$.

Initialize $t = d$, path $p = (t)$
while $t \neq s$ do
    Find $(v_1, t) \in E$ such that $c_s(v_1, t)$ is minimized
    Prepend vertex $v_1$ to path $p$
    Set $t = v_1$
```

3.1 Computational Issues

One way of explaining how the simplest path algorithm operates, is by considering the mapping of graph $G = (V, E)$ onto $G' = (E', \mathcal{E})$, which we term the *evaluation mapping* μ. In the evaluation mapping, the set of vertices E' in G' is mapped from the set of edges E in G, where the direction of the edge is ignored (i.e. $(v_i, v_j) = (v_j, v_i)$ in E'). The set of (directed) edges in G' is mapped from the set of pairs of connected (directed) edges in G, introduced above and denoted using $\mathcal{E}$. Note that the evaluation mapping does not form the dual of a graph. Furthermore, the evaluation mapping is not injective and does not necessarily preserve planarity. Fig. 2 provides an example of an evaluation mapping μ, which maps (undirected) edges in G onto vertices in G', and pairs of connected (directed) edges in G onto (directed) edges in G'. Finding the simplest paths from v in G means finding the shortest paths from any vertex in G' which v maps onto (i.e. any edge in E which contains v). This can be accomplished with any single source shortest paths algorithm, like the Dijkstra algorithm (see [20]). Therefore, the simplest paths algorithm in Algorithm 1 can be considered essentially equivalent to first mapping the graph, then finding the shortest path through the mapped graph using a conventional shortest path algorithm.

It follows that the time complexity of the simplest path algorithm is closely related to the time complexity of the shortest path algorithm. Dijkstra's algorithm has a time complexity of $O(|V|^2)$, where $|V|$ is the number of vertices in the graph G. Therefore, the simplest path algorithm has a time complexity of $O(n^2)$, where n is the number of vertices in G', or equivalently the number of edges in the graph G. In the worst case, the graph G' could have as many as $n = |V|(|V| - 1)$ vertices, leading to a complexity of $O(|V|^4)$ for the simplest path algorithm. However, such a worst case would require a totally connected graph (every pair of vertices is connected by an edge). Most geographic networks can be considered to be planar graphs. Planar graphs have a maximum

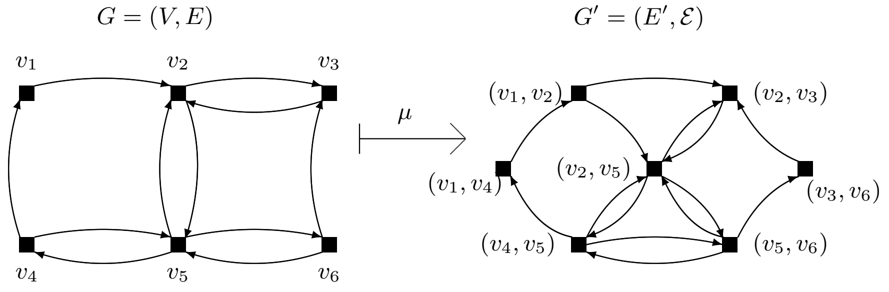


Fig. 2. Evaluation mapping μ between graphs in the simplest path algorithm

number of edges $m = 3(n - 2)$. This follows from Euler's Polyhedron Formula: a simple connected planar graph with n vertices, m edges, and f faces satisfies $n - m + f = 2$. As a result, for a simple planar graph the time complexity of the simplest path algorithm is $O(|V|^2)$. For planar graphs with nonnegative edges Henzinger and co-workers [22] recently presented a linear-time algorithm for single-source shortest paths. Since the graph G' might not be planar, the simplest path algorithm can be slower than the shortest path algorithm.

As indicated in Section 2.1, the simplest path algorithm is a single source algorithm. Like the Dijkstra algorithm, it computes the shortest path to every destination in the graph. Mark's original work uses the A* algorithm to compute the distance from a single source to a single destination only [16]. The relationship between A* and Dijkstra is well documented in the literature (e.g. [21]). The key difference between the two algorithms is that the A* algorithm uses a heuristic to preferentially explore more promising paths. For geographic information, the Euclidean distance to the destination forms an ideal heuristic, as it is a consistent underestimate of the network distance to the destination. Unfortunately, there is no natural heuristic for the mapped graph. There is no obvious choice for a non-trivial heuristic function that provides a consistent underestimate of the instruction complexity between any two vertices in the mapped graph. Unless such a heuristic function could be found, A* cannot be used for simplest paths, although this has not yet been fully investigated. In the worst case the computational complexity of the A* algorithm will be $O(n^2)$, the same as for the Dijkstra algorithm. However, the heurisitic means that A* is on average more efficient than Dijkstra where only one route from a single source to a single destination is needed.

4 Comparison of Simplest and Shortest Paths Algorithms

In order to explore the performance and properties of the simplest path algorithm, the algorithm was implemented in Java and tested using an example weighting function and an example road network data set. Section 4.1 below provides details of the weighting function used, with the results following in Section 4.2.

4.1 Choosing a Weighting Function

Any weighting function $w : \mathcal{E} \rightarrow \mathbb{R}^+$ can be used in the simplest path algorithm in Algorithm 1. However, as stated in Section 3, the weights should be chosen to reflect the amount or complexity of information required to negotiate the "decision point" represented by a pair of edges $(e_i, e_j) \in \mathcal{E}$. The weights must be ordered so that more complex decision points are associated with higher weights. There are several studies that contain classifications of route instructions that might be used as a basis for a weighting function. For example, Denis et al. provide a model of idealized route instructions [23, 4] that might be used. Several other models, including Kuipers' TOUR model [24, 25] and the PLAN model of Chown et al. [26], might yield different weighting functions. In implementing the simplest path algorithm in this study, the weighting function described below follows Mark's original work [16], in which the weights were derived from the work of Streeter and coauthors [1, 2].

In Mark [16], instructions are classified into *frames*, each frame having several *slots* for different elements of an instruction. A generic turn instruction is modeled as a frame containing a total of 9 slots. Each slot covers information about whether to turn left or right (3 slots), how to recognize when to turn (2 slots), how to recognize if the navigator has gone too far (1 slot), and summary information providing an overview of the turn (3 slots). A turn at a T-junction contains only 6 slots, since it is easy to recognize and not possible to overshoot a T-junction (so information about how to recognize the turn and what to do it the navigator goes too far are unnecessary). A turn in the road that is not at an intersection contains just 4 slots, and so on. The number of slots can be considered as a measure of the information content of the instruction needed to negotiate a decision point, and so was used as the weighting function in this experiment.

Three small deviations from the weights used in Mark [16] seem sensible. First, Mark's weighting contains no cost for going straight on. A navigation system should at least be able to reassure a user that they are on the right track, even if just going straight on, so in our weighting function the cost of straight on was one slot. Second, Mark [16] included a 3-slot weighting for an instruction where the name of a road changed while continuing straight on. The algorithm as implemented does not include this weighting, as this initial work focuses solely on the geometry and topology of the road network, not the attributes. However, such weights could easily be implemented (see discussion in Section 5). Third, Mark's original model does not distinguish between general intersections of different degree. Intuitively, the instruction for turning left at a 3-way intersection requires less information than at a 4-way intersection, which in turn requires less information than for a 5-way intersection. To reflect this wrinkle, the weight for turning left or right at a general intersection vertex v was set to $5 + deg(v)$. This means for a 4-way intersection, the weighting is as in Mark [16], 9 slots, but for a 3-way intersection the weighting is 8 slots, and for a 5-way 10 slots.

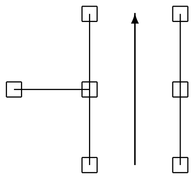
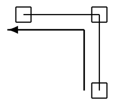
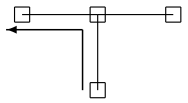
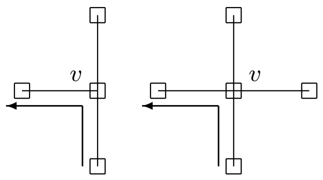
Straight on		1 slot
Turn (not at intersection)		4 slots
Turn left or right at T-junction		6 slots
Turn left or right at other junction		$5 + deg(v)$ slots

Fig. 3. Weighting of different intersection types, based on [16]

The instruction classification and associated weights are illustrated in Fig. 3. Note that this weighting function means that unlike shortest paths, simplest paths are not *symmetric*: the simplest route from A to B may not be the same as the simplest route from B to A. For example, coming from one direction, turning right at a T-junction is weighted 6 slots, while coming from the other direction, turning left at a 3-way intersection is weighted 8 slots. The weighting function also means that, unlike shortest paths, simplest paths do not satisfy the *triangle inequality*: the simplest path from A to C may be longer than the length of the simplest path from A to B plus the length of the simplest path from B to C. The triangle inequality might be violated because adding the lengths of the simplest path from A to B and from B to C ignores the information required to negotiate the intersection at B, information that is included in the calculation of the simplest path from A to C. Since simplest paths are neither symmetric nor fulfil the triangle inequality, simplest paths cannot be determined by any metric.

4.2 Algorithm Performance

The results of using of the shortest path algorithm on several example road network data sets were encouraging. This paper reports the results of an exhaustive analysis on road network data set for the city of Bloomington, Indiana, USA, which exhibits a wide range of different network configurations, including a dense downtown grid network and sparser suburban networks. Fig. 4 shows a comparison of simplest and shortest paths between two vertices from within a downtown area of Bloomington.

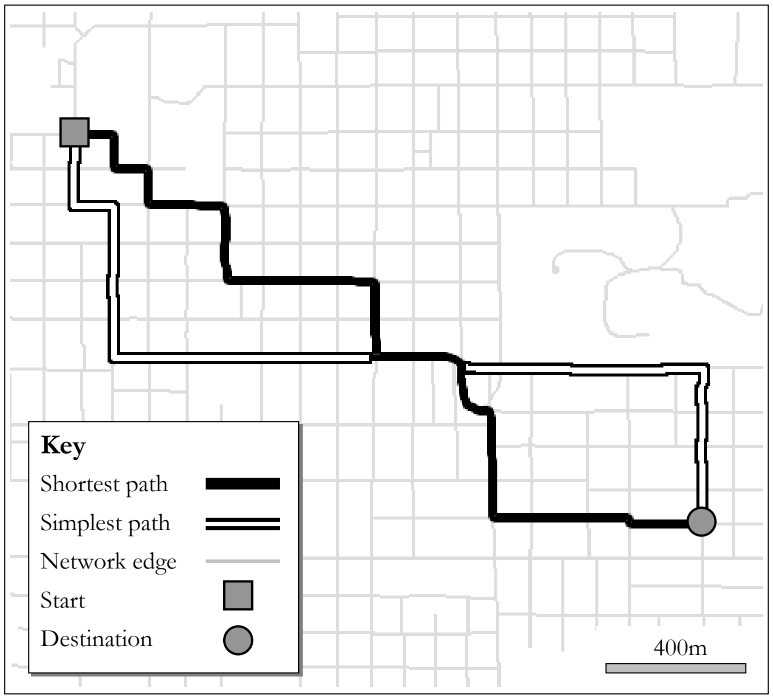


Fig. 4. Example comparison of a shortest and simplest path (approximate scale)

The thick black line in Fig. 4 shows the shortest path. The double black line shows where the simplest path deviates from the shortest path. The simplest path contains only four turns, all made at three-way intersections. The shortest path contains many more turns: a total of 12 turns, made at a range of two-, three- and four-way intersections. The lengths of the two paths are very similar: the simplest path is 2% longer than the shortest path.

The performance of the simplest path algorithm was exhaustively tested for the entire data set. With more than 3000 vertices in the data set, this involved computing the shortest and simplest paths for a total of over 10 million pairs of vertices. A comparison of the lengths of the simplest and shortest paths for one set of 3200 shortest paths from a single source to every other vertex in the data set is shown in Fig. 5. The figure provides a scatter plot of the normalized simplest path length (the ratio of simplest to shortest path lengths), plotted against shortest path length.

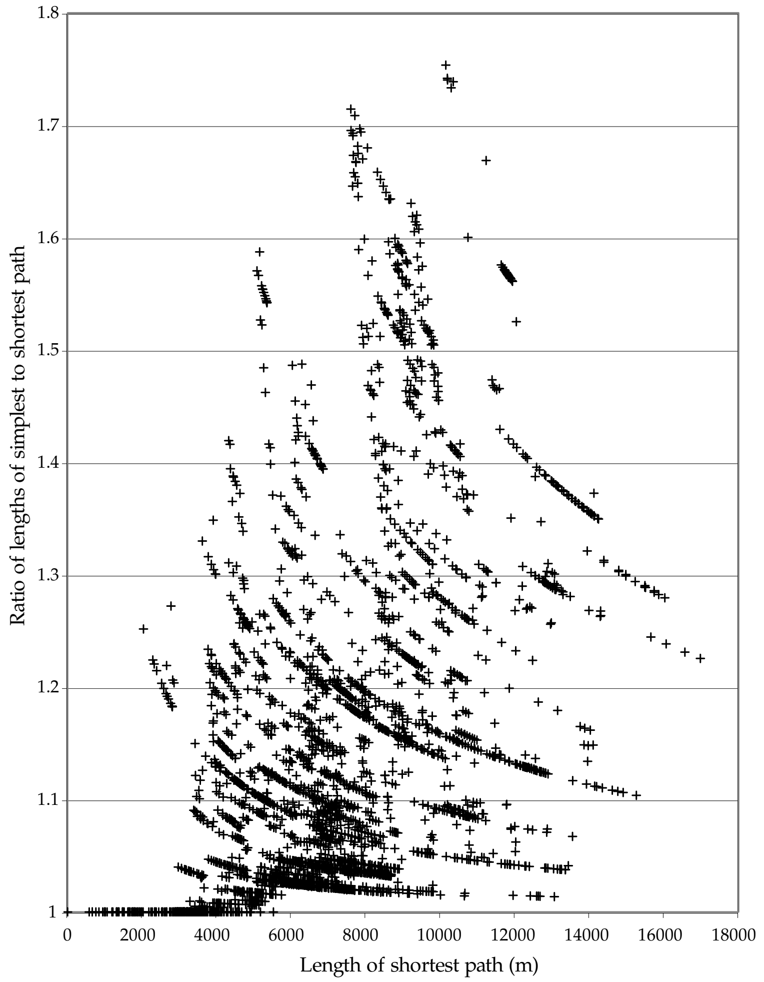


Fig. 5. Scatter plot comparing the simplest and shortest path lengths for a single source

In this example, more than 90% of the simplest paths between two vertices are less than 50% longer than the corresponding simplest path. This example is typical: for the entire data set of more than 10 million simplest paths, 93.2% of the shortest paths are less than 50% longer than the corresponding shortest path. Over the entire data set, on average a simplest path is 15.8% longer than the corresponding shortest path. A noticeable but incidental feature of the scatter plot is that it exhibits some strongly correlated "stripes" running from the top left to the bottom right. These occur because the set of all shortest paths from a single starting vertex is strongly correlated, with many of the paths containing similar sequences of edges.

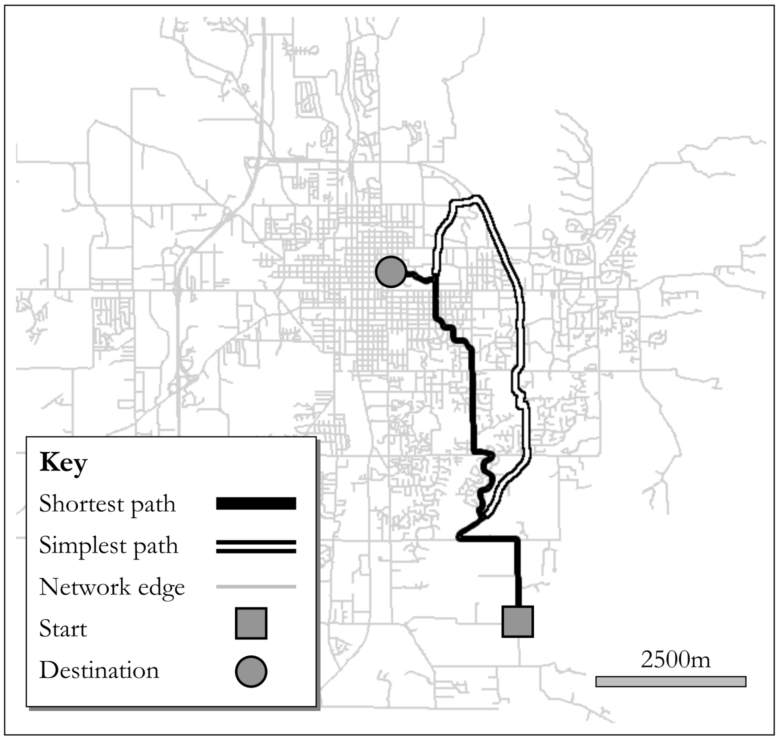


Fig. 6. Comparison of a typical shortest and simplest path (approximate scale)

Fig. 6 shows an example of one path where the starting point is some way out of the dense downtown road network. The example in Fig. 6 is drawn from the data set plotted in Fig. 5. The simplest path (shown as double black line) has total length of approximately 10km, 30% longer than the shortest path (thick black line), placing it somewhere near the middle of the scatter plot in Fig. 5. Fig. 7 shows the "worst case" outlier simplest path, also drawn from the data set plotted in Fig. 5. This simplest path has a total length of approximately 18km, 75% longer than it's corresponding shortest path, placing it right at the top of Fig. 5. The most noticeable features of the simplest paths in both Fig. 6 and 7 is that they tend to skirt the city center areas, in favor of longer routes through less dense road networks. In the "worst case" example in Fig. 7, the simplest path actually goes south, in the opposite direction from the destination, in order to join a long straight road with relatively few intersections (actually an Interstate highway), and travels north past the destination before switching back south to the destination. Despite being the “worst case,” this does not seem an unreasonable route to choose. The route avoids the numerous complicated intersections traversed by the shortest path, at the cost of increased total distance.

Note that Interstate highway has fewer intersections than it would appear from Fig. 7 alone: several minor roads cross the Interstate, but do not intersect it (underpasses or bridges). This is the reason why the simplest path must travel south away from the destination to join the Interstate and north past the destination to leave it. The example highlights that the assumption of a planar graph (see Section 3.1) is an oversimplification for road networks. Note also that the algorithm has no access to information about what sort of road it is selecting. A more sophisticated algorithm could preferentially select routes that use more important or faster roads. That is not what is happening in Fig. 7: it is simply that the relatively low frequency of intersections and straightness of the highway means it is a low cost route for the simplest path algorithm.

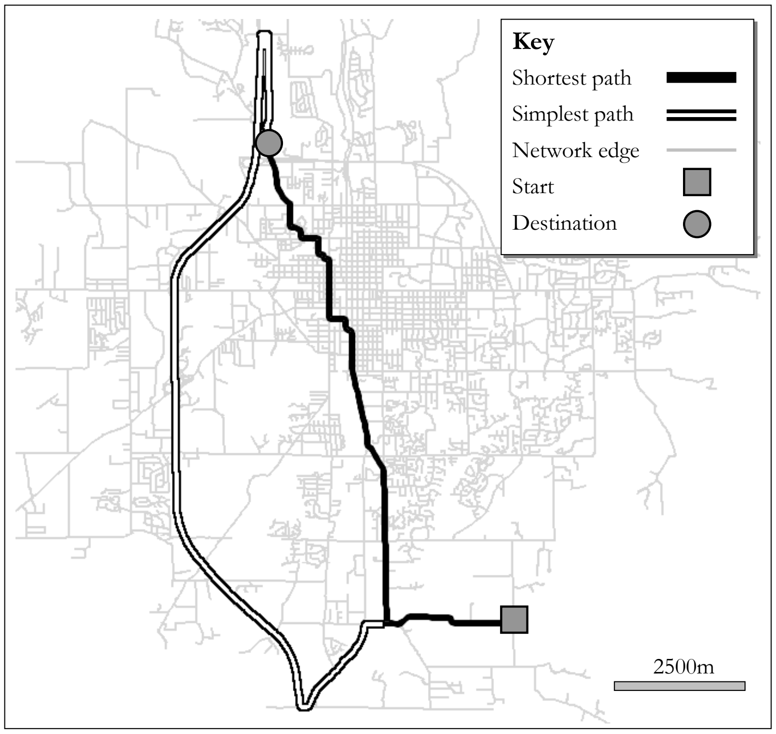


Fig. 7. Comparison of a “worst case” shortest and simplest path (approximate scale)

Finally, Fig. 8 summarizes the entire data set of 10 million shortest paths. Fig. 8 shows the spatial distribution of the standard deviation of the normalized lengths of the simplest paths. Each point in Fig. 8 represents the standard deviation of normalized length for all simplest paths starting from that point. The standard deviations have been classified in five quantiles (five classes with equal cardinality).

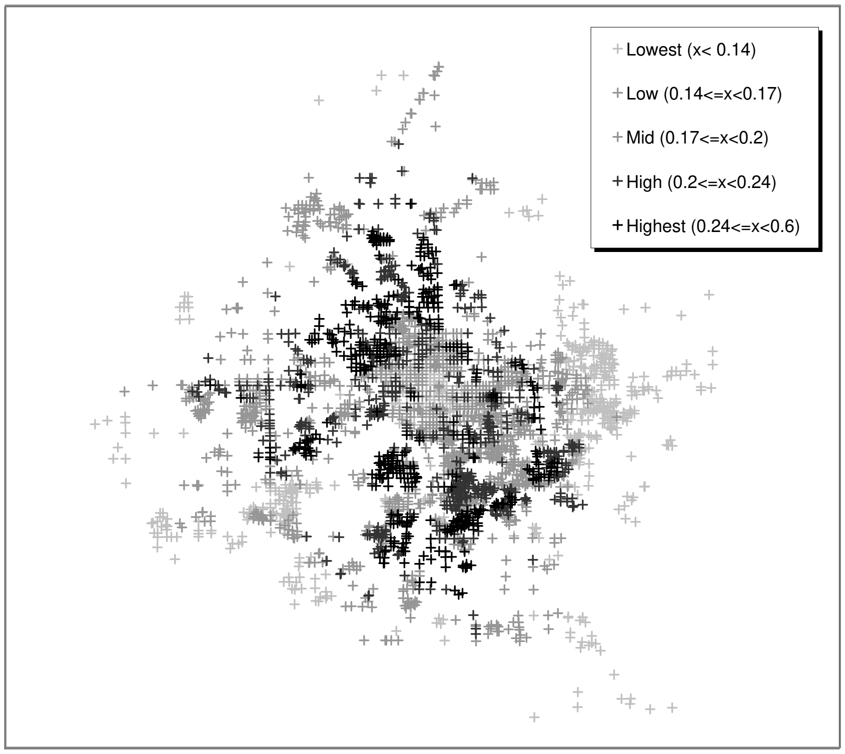


Fig. 8. Spatial distribution of the standard deviation of normalized simplest path length

The figure shows generally low standard deviations (lighter gray data points) in the denser downtown road network regions and in the sparse suburban road network regions (c.f. the road network in Fig. 7). The higher standard deviations (darker gray points) generally occur at the periphery of the dense downtown road networks, at the transition to sparse suburban road networks. This greater variability can be interpreted as a result of the deviations of the simplest path from the shortest path being more pronounced at the periphery of the dense downtown road networks. Starting locations within the downtown road network have no option but to traverse this denser network. Starting locations well away from the downtown road network require relatively smaller deviations to avoid the denser network if necessary. Those locations at the periphery of the denser downtown road network are more likely to require larger deviations from the shortest path, such as that shown in Fig. 7. A similar diagram showing the spatial distribution of mean normalized simplest path length revealed no appreciable spatial pattern.

5 Discussion and Conclusions

The results of this work are generally encouraging. The simplest paths are on average only 16% longer than the shortest paths, and more than 90% of simplest paths are less than 50% longer than the corresponding shortest paths. In return for paths of slightly longer length, the simplest paths algorithm produces routes through a network that are *cognitively plausible*, in the sense that they are based on a cognitive model of the complexity of instructions needed to complete a route. Further, the simplest paths algorithm requires no distance or other metric information to operate. Diagrams like Fig. 4 provide compelling (if subjective) evidence that the simplest paths are indeed more cognitively plausible that simplest paths. Even the "worst case" examples, like Fig. 7 do not appear to be unreasonable routes. A major component of future work, therefore, should be cognitive studies with human subjects to test the hypothesis that the route instructions based on simplest paths are preferable, less error prone, or easier to explain and use for human navigators, when compared with route instructions based on shortest paths.

The analysis presented in this paper has empirically examined the relationship between the lengths of simplest and shortest paths. While the data set used in the analysis does exhibit a range of different network configurations and densities, further empirical work studying different types of road network might reveal some different properties (for example, the road networks in many European cities, which have developed over longer periods of time). Moreover, the work presented in this paper lacks a theoretical basis for the relationship between the length of simplest and shortest paths. As a result, further theoretical work is needed to explore this relationship, including the investigation, identification, and classification of different types of graph with respect to the properties of simplest and shortest paths through those graphs.

The simplest path algorithm can have a longer computation time than shortest path algorithms. Assuming the graph is planar, the algorithm has at most quadratic computation time. In some cases, graphs may be locally non-planar, as in road networks where roads cross but do not intersect, (e.g. overpasses or bridges). However, road networks still retain relatively low connectivity, so it seems unlikely that these locally non-planar graphs would significantly increase the computational complexity of the simplest paths algorithm. The simplest paths algorithm can efficiently compute the simplest paths from a single source vertex to every other vertex in the graph. This property can be vital for dealing with imprecision in navigation, for example, where the location or destination of a navigation agent may not be precisely known [18].

The simplest path algorithm might easily be adapted to provide more sophisticated behavior. As seen in Fig. 7, the simplest path algorithm sometimes selects major roads simply by virtue of their straighter geometry and less connected topology (i.e. fewer intersections). A modification of the weighting function could easily be used to explicitly prefer certain types of road [10–12]. For example, major roads could be preferred in the route selection by making the weights for turning onto a major road smaller, and turning off a major road larger. Conversely, minor roads could be avoided by making the weights for turns onto a minor road larger, and turning off a minor road smaller. Many other possible adaptations of the algorithm could result from other modifications of the weighting function. Another important component of future work, therefore, should be cognitive studies with human subjects to determine what types of weighting functions are most appropriate for human users in a particular contexts.

Acknowledgements

This work was conducted under the SmartMaps project, supported by the National Imagery and Mapping Agency, grant no NMA201-01-1-2003. The authors would like to acknowledge David Mark, for an early conversation which helped initiate many of the ideas for this work, as well as later comments on a first draft which helped shape the paper. In addition, comments and suggestions from Max Egenhofer and Mike Worboys were valuable. This research has also benefited from collaboration with Jörg-Rudiger Sack at Carleton University, Ottawa (NSERC supported Collaborative Research and Development grant). The authors are also grateful to the three anonymous reviewers for their constructive comments.

References

1. Streeter, L., Vitello, D., Wonsiewicz, S.: How to tell people where to go: Comparing navigational aids. International Journal of Man Machine Interaction **22** (1985) 549–562
2. Streeter, L., Vitello, D.: A profile of driver's map-reading abilities. Human Factors **28** (1986) 223–239
3. Golledge, R.: Path selection and route preference in human navigation: A progress report. In Frank, A., Kuhn, W., eds.: Spatial Information Theory: A Theoretical Basis for GIS (COSIT '95). Number 988 in Lecture Notes in Computer Science, Berlin, Springer (1995) 207–222
4. Denis, M., Pazzaglia, F., Cornoldi, C., Bertolo, L.: Spatial discourse and navigation: An analysis of route directions in the city of Venice. Applied Cognitive Psychology **13** (1999) 145–174
5. Tversky, B., Lee, P.: How space structures language. In Freksa, C., Habel, C., Wender, K., eds.: Spatial Cognition. Number 1404 in Lecture Notes in Computer Science, Berlin, Springer (1998) 157–176

6. Tversky, B., Lee, P.: Pictorial and verbal tools for conveying routes. In Freksa, C., Mark, D., eds.: Spatial Information Theory. Cognitive and Computational Foundations of Geographic Information Science (COSIT'99). Number 1661 in Lecture Notes in Computer Science, Berlin, Springer (1999) 51–64
7. Richter, K.F., Klippel, A.: "You-are-here maps": Wayfinding support as location based service. In Moltgen, J., Wytzisk, A., eds.: GI-Technologien für Verkehr und Logistik. IfGI Prints 13, Münster (2002)
8. Burnett, G.: 'Turn right at the traffic lights': The requirement for landmarks in vehicle navigation systems. Journal of Navigation **53** (2000) 499–510
9. May, A., Ross, T., Bayer, S.: Drivers' informational requirements when navigating in an urban environment. Journal of Navigation **56** (2003) 89–100
10. Shapiro, J., J., W., Nir, D.: Level graphs and approximate shortest paths algorithms. Networks **22** (1992) 691–717
11. Liu, B.: Using knowledge to isolate search in route finding. In: Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence, IJCAI 95. Volume 1., Montréal, Québec, Canada, Morgan Kaufmann (1995) 119–125
12. Liu, B.: Intelligent route finding: combining knowledge, cases and an efficient search algorithm. In: 12th European Conference on Artificial Intelligence (ECAI-96), Budapest, Hungary, John Wiley and Sons (1996) 380–384
13. Frank, A.U.: Pragmatic information content—how to measure the information in a route description. In Duckham, M., Goodchild, M.F., Worboys, M.F., eds.: Foundations in Geographic Information Science. Taylor & Francis, London (2003) 47–68
14. Goodchild, M.F.: The nature and value of geographic information. In Duckham, M., Goodchild, M.F., Worboys, M.F., eds.: Foundations in Geographic Information Science. Taylor & Francis, London (2003) 19–31
15. Worboys, M.F.: Communicating geographic information in context. In Duckham, M., Goodchild, M.F., Worboys, M.F., eds.: Foundations in Geographic Information Science. Taylor & Francis, London (2003) 33–45
16. Mark, D.M.: Automated route selection for navigation. IEEE Aerospace and Electronic Systems Magazine **1** (1986) 2–5
17. Mark, D.M.: Finding simple routes: 'ease of description' as an objective function in automated route selection. In: Proceedings, Second Symposium on Artificial Intelligence Applications (IEEE), Miami Beach (1985) 577–581
18. Duckham, M., Kulik, L., Worboys, M.F.: Imprecise navigation. GeoInformatica **7** (2003) 79–94
19. Pallottino, S., Scutellà, M.: Shortest path algorithms in transportation models: Classical and innovative aspects. In Marcotte, P., Nguyen, S., eds.: Equilibrium and Advanced Transportation Modelling. Kluwer, Amsterdam (1998) 245–281
20. Cormen, T., Leiserson, C., Rivest, R., Stein, C.: Introduction to Algorithms. Second edn. McGraw-Hill (2001)
21. Luger, G., Stubblefield, W.: Artificial Intelligence: Structures and strategies for complex problem solving. Third edn. Addison-Wesley, Reading, MA (1998)
22. Henzinger, M.R., Klein, P., Rao, S., Subramanian, S.: Faster shortest-path algorithms for planar graphs. Journal of Computer and System Sciences **55** (1997) 3–23
23. Denis, M.: The description of routes: A cognitive approach to the production of spatial discourse. Cahiers de Psychologie Cognitive **16** (1997) 409–458
24. Kuipers, B.: Representing Knowledge of Large-Scale Space. PhD thesis, Mathematics Department, Massachusetts Institute of Technology (1977) Technical Report 418, M.I.T. Artificial Intelligence Laboratory.

25. Kuipers, B.: Modelling spatial knowledge. Cognitive Science **2** (1978) 129–153
26. Chown, E., Kaplan, S., Kortenkamp, D.: Prototypes, location and associative networks (plan): Towards a unified theory of cognitive mapping. Journal of Cognitive Science **19** (1995)