

# Adaptive GPS Algorithms

Chad Seibert  
Division of Science and Mathematics  
University of Minnesota, Morris  
Morris, MN 56567  
seib0060@morris.umn.edu

March 16, 2011

## **Abstract**

Global positioning systems (GPS) are very popular as a means of finding a route from one destination to another. However, some GPS algorithms for finding routes are not intelligent with respect to current weather conditions, traffic levels, and construction zones. We design algorithms that allow us to incorporate these factors and find a path with expected shortest path.

# 1 Introduction

Finding the shortest distance between two places is a common occurrence when travelling. Often times, this is done with a global positioning system (GPS) that is capable of providing directions. These devices often present only one possible option for the route to the destination. What if the user wanted a path subject to different constraints? That is, not necessarily the shortest path, but the path that has the tightest bounds on expected travel time. Also, what if such a system could give better directions by incorporating the current road conditions? If the roads are covered with glare ice, then we do in fact want to travel the minimum distance between two points. If some roads are heavy in traffic, then we might prefer to seek longer routes that will be quicker due to lighter traffic. We design a hybrid algorithm allowing us to incorporate such information as traffic levels, weather conditions, and construction zones to find an optimal shortest distance between two points.

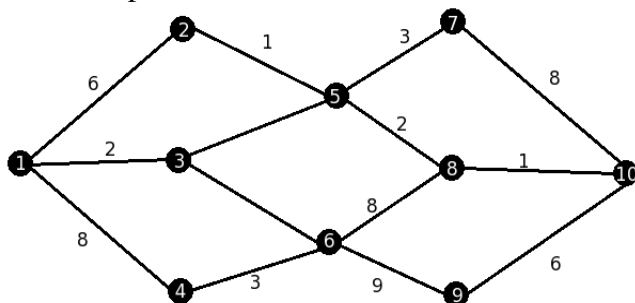
## 2 Preliminary Concepts

### 2.1 Deterministic Shortest Path Problem

The deterministic shortest path problem lies at the heart of our system. Given a weighted graph  $G = (V, E)$  and two vertices  $s, t \in V$ , determine a path of least weight from  $s$  to  $t$ . Many algorithms have been designed to solve this problem efficiently; one of the most popular is Dijkstra's Algorithm. Dijkstra's algorithm works on directed and undirected graphs with positive edge weights. We give an overview of the algorithm; the interested reader should consult [2] for more information.

We begin by marking all nodes as unvisited and setting the start vertex to the current vertex. By marking, we mean setting a flag associated with each vertex. We then compute the tentative distance for each unvisited neighbor, that is, the distance to the current node plus the distance to the neighbor. If this distance is less than the previously recorded distance, replace the old distance with this instance. After computing all the tentative distances for each unvisited neighbor, mark the current vertex to visited and set the node with minimal tentative distance as the current vertex. Continue until the current vertex is the target vertex. Algorithm 1 describes a pseudo-code of Dijkstra's algorithm, and we give an illustration of Dijkstra's Algorithm in Example 1 below.

**Example 1:** Given the following instance of a Shortest Path problem, we would like to find a shortest path from vertex 1 to vertex 10.



We first begin by considering the neighbors of 1. We see that the shortest paths to 2, 3,

---

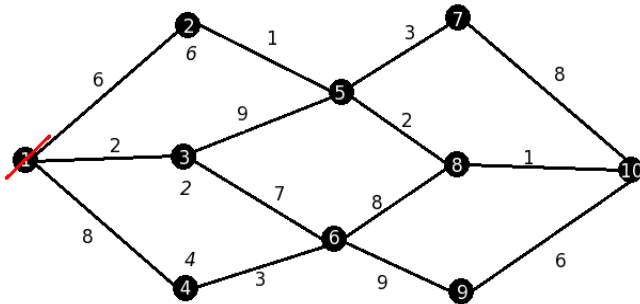
**Algorithm 1** Dijkstra's Algorithm.  $G = (V, E)$  is the graph,  $s$  is the source vertex and  $t$  is the destination vertex.

---

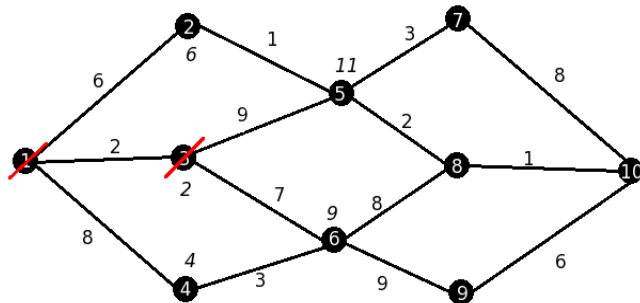
```
1: for  $v \in V$  do
2:    $dist[v] \leftarrow \infty$ 
3:    $prev[v] \leftarrow undefined$ 
4: end for
5:  $dist[t] \leftarrow 0$ 
6:  $Q \leftarrow V$ 
7: while  $Q \neq \emptyset$  do
8:    $curmin \leftarrow \infty$ 
9:   for each  $u \in Q$  do
10:    if  $dist[u] < curmin$  then
11:       $curmin \leftarrow dist[u]$ 
12:       $minnode \leftarrow u$ 
13:    end if
14:  end for
15:  if  $curmin = \infty$  then
16:    return  $\emptyset$ 
17:  end if
18:   $Q \leftarrow Q - \{u\}$ 
19:  for each neighbor  $v$  of  $u$  do
20:     $tentativedist \leftarrow dist[u] + weight[u, v]$ 
21:    if  $tentativedist < dist[v]$  then
22:       $dist[v] \leftarrow tentativedist$ 
23:       $prev[v] \leftarrow u$ 
24:    end if
25:  end for
26: end while
27:  $path \leftarrow \emptyset$ 
28:  $u \leftarrow t$ 
29: while  $prev[u]$  is defined do
30:    $path \leftarrow path \cup \{u\}$ 
31:    $u \leftarrow prev[u]$ 
32: end while
33: return  $path$ 
```

---

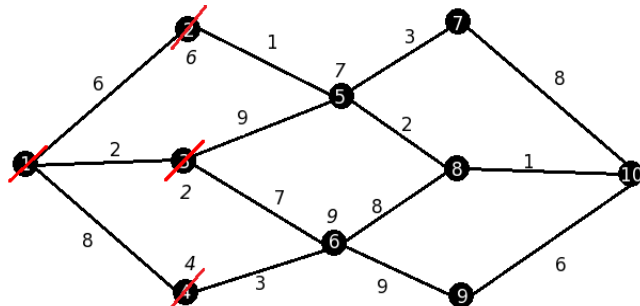
and 4 are 6, 2, and 8, respectively. We label the vertex the tentative distance from 1 to the vertex. Since we have considered all the neighbors of 1, it is marked and we never consider it again. In the following figure, we have the tentative distance to that vertex in italics.



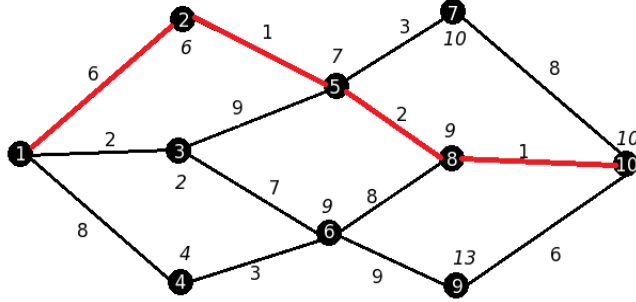
Now, we consider the neighbors of 2, 3, and 4. We start with the vertex with least tentative distance, in this case, vertex 3. We consider its unmarked neighbors and label them with their tentative distance. There are two neighbors, 4 and 5. We label those vertices with their tentative distances by adding the tentative distance from 3 to the weight of the edge connecting them. We now mark vertex 3 as we have considered all its neighbors. We have the following graph



Now, we consider the next vertex with minimum tentative distance, that is, vertex 2. We consider all its unmarked neighbors, which is only vertex 5. We compute the tentative distance from 1 to 5 by summing the tentative distance of 2 to the edge weight connecting them. We see that it is less than the current tentative distance for 3. Thus, we overwrite it and mark 2. We continue the same for vertex 4 and are left with the following



We continue this process until each vertex is marked, leaving us with the following



## 2.2 Fuzzy Numbers

*Fuzzy numbers* are a generalization of the real numbers where the value refers not to one specific number, but to a range of numbers. Fuzzy numbers can be defined in many different ways, for more information about the subject, see [1] [5]. For our research, we limited ourselves to triangular and trapezoidal fuzzy numbers. A triangular fuzzy number is a tuple  $(a, b, c)$  where  $a < b < c$  and a membership function  $\mu$  defined as

$$\mu(x; a, b, c) = \begin{cases} \frac{x-a}{b-a} & : a \leq x \leq b \\ \frac{c-x}{c-b} & : b \leq x \leq c \\ 0 & : otherwise \end{cases}$$

We similarly define a trapezoidal fuzzy number as a tuple  $(a, b, c, d)$  where  $a < b \leq c < d$  and membership function

$$\mu(x; a, b, c, d) = \begin{cases} \frac{x-a}{b-a} & : a \leq x \leq b \\ 1 & : b \leq x \leq c \\ \frac{d-x}{d-c} & : c \leq x \leq d \\ 0 & : otherwise \end{cases}$$

We can see that if a trapezoidal number defined as  $(a, b, c, d)$  with  $b = c$ , then this is also a triangular number. We see in Figure 1 examples of fuzzy numbers. We can intuitively think of the fuzzy number on the left as about 2.5 and the fuzzy number on the right as about 2 – 6.

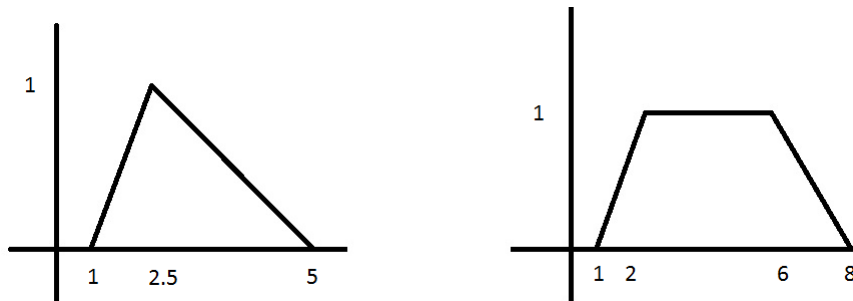


Figure 1: An example of a fuzzy numbers. On the left, we have a triangular fuzzy number; on the right a trapezoidal number.

A fair question to ask is whether its possible to define the basic mathematical operations (addition, subtraction, multiplication, division) on fuzzy numbers. Given the scope of this paper, we are only interested in addition on trapezoidal fuzzy numbers, given by

$$x_1 + x_2 = (a_1, b_1, c_1, d_1) + (a_2, b_2, c_2, d_2) = (a_1 + a_2, b_1 + b_2, c_1 + c_2, d_1 + d_2). \quad (1)$$

We see immediately that given this definition of addition, the set of trapezoidal fuzzy numbers is closed under addition.

### 3 Shortest Path Problem in Graphs with Fuzzy Weights

Now, we extend the shortest path problem to graphs with fuzzy edge weights. Given a graph  $G = (V, E, W)$  where  $V$  is the set of vertices,  $E$  is the set of edges, and  $W$  is a fuzzy weight function defined on the edges of  $G$ , we wish to determine the shortest path between two distinct vertices in the graph. Since the edge weights are fuzzy numbers, we must be clear about what we mean by the shortest path. We define the set  $P$  as the set of all paths from a start vertex  $s$  to a terminal vertex  $t$ . We define a function  $l(p)$  as the length of path  $p$ . Then

$$sp(s, t) = \min_{p \in P} l(p)$$

defines the shortest path. In order to use this definition, we need a method of comparing fuzzy number. We can do this in several ways.

One method is by defuzzification. *Defuzzification* is the process of taking a fuzzy and defining a crisp (non-fuzzy) number. For trapezoidal fuzzy numbers, we define the *expected value* as

$$E(x; a, b, c, d) = \frac{1}{4}(a + b + c + d).$$

This is simply the average of the four control points. We can use the total ordering of the real numbers as a comparison metric. That is,

$$x_1 < x_2 \iff E(x_1) < E(x_2).$$

For example, if  $x_1 = (1, 2, 3, 4)$  and  $x_2 = (5, 6, 7, 8)$  then  $x_1 < x_2 \iff 2.5 < 6.5$ , which is true. Furthermore, we can easily modify Dijkstra's algorithm by using our comparison and sum operators. Lines 10 and 21 of Algorithm 1 should use  $d(x)$  to defuzzify the distances before comparison. Line 20 should use the sum method for fuzzy numbers defined in equation 1.

One major problem with this method is that it does not incorporate the degree of uncertainty of the fuzzy edge weights being visited. For example, if  $x_1 = (85, 90, 110, 115)$  and  $x_2 = (0, 99, 101, 200)$  both have expected value 100. We present two different approaches to solving this problem.

Let the uncertainty of a fuzzy number be define as  $u(x) = d - a$ . Then, we ask, is there a path with minimal uncertainty. This is easily solved by setting the weight of each edge as the uncertainty and using Dijkstra's algorithm to solve it.

Perhaps we wish to find a path shorter than  $l$  with minimal uncertainty. This is an example of the multi-objective shortest path problem, which is known to run in polynomial time.

There are at least two different ways to solve this problem. First, we can modify Dijkstra’s algorithm to incorporate more than one constraint. Second, we can resort to the standard LP formulation for the shortest path problem and suitably modifying the objective function. The interested reader should consult [6] for more information.

## 4 Stochastic Shortest Path with Recourse

We now turn to the stochastic shortest path problem with recourse. Given a graph  $G = (V, \mathcal{A})$ , we define the weights in a stochastic manner. We are given a set of all possible sets of edge weights called *realizations* to use the terminology in [4]. An example of such sets is the following:

	$r_1$	$r_2$	$r_3$
$e_1$	1	4	5
$e_2$	8	8	5
$e_3$	7	3	2
$e_4$	3	5	6
$e_5$	1	2	7

Each column represents a separate realization  $r_i \in \mathcal{R}$ . Furthermore, each realization is assigned a probability of being the actual realization, denoted  $p_i$ . Given an entity starting at node  $s$ , we wish to find the expected shortest path to  $t$ . The entity starts at node  $s$  and learns the weights of its neighbors. Using this information, we can eliminate some realizations that are incompatible with how the entity sees the world. It navigates the world, navigating nodes and eliminating realizations until there is only one realization, whereby we may assign the weights given by that realization to the graph and use the shortest path algorithm from the current vertex to the destination.

We define an *information set* as a set of realizations that are still possible, given the information the entity has acquired so far. We denote this set by  $\mathcal{I}$  and  $\mathcal{I}_l$  denotes the set of all information sets with  $l$  realizations. Before traversing the graph, we start with  $\mathcal{I} = \mathcal{R}$ . We let  $c^{r_k}(i, j)$  be the cost of edge  $(i, j)$  under realization  $r_k$ . Now, we define the set

$$\mathcal{A}_I^d = \{(i, j) \in \mathcal{A} \mid c^{r_1}(i, j) = c^{r_2}(i, j), \forall r_1, r_2 \in I\}$$

which is set of all edges with weights common to all realizations. We define  $\mathcal{N}_I^c(i)$  as the set of all information collecting nodes of vertex  $i$  given the information set  $I$ . An *information collecting node* is a node that reduces the size of the set  $I$  by giving information that allows for the elimination of some  $r_i \in I$ .

We now know enough about the problem to give the dynamic programming solution. This variant of the shortest path problem, called R-SSPPR, was analyzed in depth by George Polychronopoulos and John Tsitsiklis and more information about the run time analysis and correctness can be found in [4]. The following algorithm comes directly from [3] and is present in algorithms 2 and 3. More information about its derivation can be found in [3, 4].

---

**Algorithm 2** Dynamic Programming Algorithm to solve R-SSPPR

---

```
1:  $\forall r \in \mathcal{R}$  solve the shortest path problem from  $s$  to  $t$ 
2: for  $l = 2, \dots, R$  do
3:   for  $\forall I \in \mathcal{I}_l$  do
4:     for  $i \in \mathcal{N}$  do
5:        $V(i, I) = \min \left\{ \min_{j \in \mathcal{N}_I(i)} \{c(i, j) + V(j, I)\}, \min_{j \in \mathcal{N}_I^c(i)} \{c(i, j) + E[V(j, h(j, I))]\} \right\}$ 
6:     end for
7:   end for
8: end for
```

---

---

**Algorithm 3** Solution to the recursive step in Algorithm 2 for a given  $I$ 

---

```
1:  $V(t, I) \leftarrow 0$ 
2:  $V(i, I) \leftarrow \infty$ 
3:  $i \in \mathcal{N} - \{t\}$ 
4:  $\mathcal{P} \leftarrow \emptyset$ 
5:  $\mathcal{T} \leftarrow \mathcal{N}$ 
6: for  $i \in \mathcal{N} = \{t\}$  do
7:    $V(i, I) \leftarrow \infty$ 
8: end for
9: while  $\mathcal{P} \neq \mathcal{N}$  do
10:  Find node  $i^*$  with minimum label
11:   $\mathcal{P} \leftarrow \mathcal{P} \cup \{i^*\}$ 
12:   $\mathcal{T} \leftarrow \mathcal{T} - \{i^*\}$ 
13:  if  $(i, i^*) \in \mathcal{A}_I^q$  then
14:    if  $i \in \mathcal{N}_I^c$  then
15:      if  $V(i, I) > c(i, i^*) + E[V(i^*, h(i^*, I))]$  then
16:         $V(i, I) \leftarrow c(i, i^*) + E[V(i^*, h(i^*, I))]$ 
17:      end if
18:    else
19:      if  $V(i, I) > c(i, i^*) + V(i^*, I)$  then
20:         $V(i, I) \leftarrow c(i, i^*) + V(i^*, I)$ 
21:      end if
22:    end if
23:  end if
24: end while
```

---



## 5 Conclusion

We have shown two different alternatives to the deterministic shortest path problem, with applications to GPS path-finding systems. If we can present traffic levels as fuzzy numbers, then we can use these weights instead of crisp weights. Since traffic levels are ever changing, the need for some form of stochastic edge weights is necessary. We can get better routes to destinations by incorporating domain knowledge into the problem. With the R-SSPPR problem, we can define a set of possible edge weights and in doing so requires the entity to find the shortest path in a graph without knowing which set it is. This is a much more realistic representation of the problem, where the entity (a vehicle) can observe the current traffic level, road conditions, or construction zones as it progresses to its destination. It may have some domain knowledge in the form of edge sets, this corresponding to knowing what time it is. If the entity knows that taking a certain route is not optimal (rush-hour, construction zone, etc.), then it can form a new path and continue along it. We can modify the R-SSPPR problem to incorporate fuzzy edge weights, giving a flexible solution that allows for optimal path finding under a wide variety of conditions.

## References

- [1] James J. Buckley and Esfandiar Eslami. *An Introduction to Fuzzy Logic and Fuzzy Sets*. 2002.
- [2] Edsger W. Dijkstra. A note on two problems in connexion with graphs. *Numerische Mathematik*, 1:269–271, 1959.
- [3] George H. Polychronopoulos and John N. Tsitsiklis. Stochastic and dynamic shortest distance problems. *DSpace*, 1992.
- [4] George H. Polychronopoulos and John N. Tsitsiklis. Stochastic shortest path problems with recourse. *Networks*, 27:133–143, 1996.
- [5] William Siler and James J. Buckley. *Fuzzy Expert Systems and Fuzzy Reasoning*. Wiley, 2005.
- [6] Zbigniew Tarapata. Selected multicriteria shortest path problems: An analysis of complexity, models and adaptation of standard algorithms. *Int. J. Appl. Math. Comput. Sci.*, 17:269–287, June 2007.