Advanced Navigation of (Electric) Vehicles in Street Networks

Sabine Storandt

University of Stuttgart
Germany

SELF LOCALIZATION VIA PATH SHAPES

[Funke, Storandt; GIS 2011]

Present-Day Localization Schemes

Scheme	Requirements	Drawback
GPS	GPS-receiver, free sight to $\geq$ 4 satellites	signal blockage
GSM	mobile phone, geocoded antennas	too imprecise
WPS	active internet connection geocoded wifi-access points	too imprecise

New Scheme: Path Shapes

BASIC IDEA Use movement pattern to determine the own position in the street network.

GIVEN street network $G(V, E)$, shape of driven trajectory

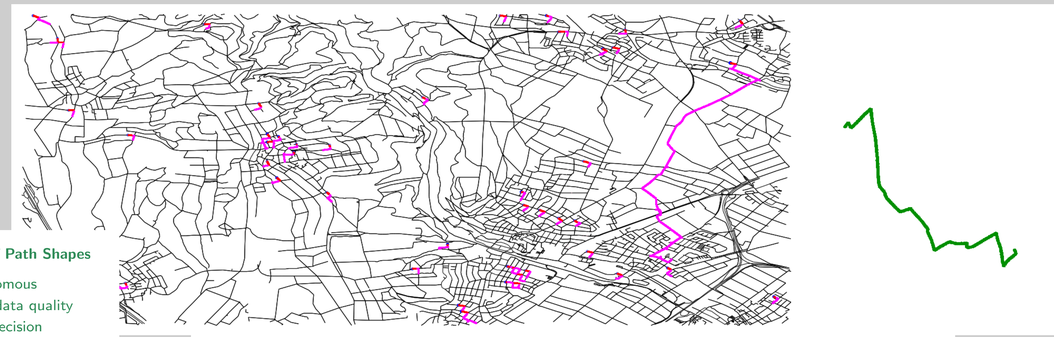
GOAL find the (unique) starting vertex where the sequence of movements implied by the shape is possible $\Rightarrow$ derive the own position in G

BASIC QUESTIONS

1. How to gather shape data?
2. How long is the shape required for uniqueness?
3. How to answer queries efficiently?

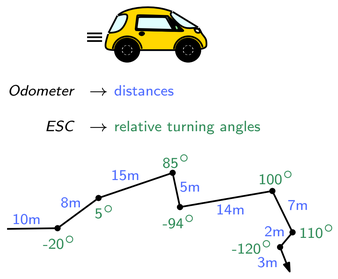
Adavantages of Path Shapes

- fully autonomous
- consistent data quality
- GPS-like precision
- efficient query answering



Path shape (green) and its (long) matches in the map for a fuzzy equality model. Blue circles indicate path origins. While several matches for the first few meters are found, the whole sequence of movements is only once performable in the map.

1. Gathering Shape Data



2. MA - Massachusetts, GER - Germany

		avg query time (sec)	avg prefix length (m)	max prefix length (m)
exact	MA	1.31	279	4480
	GER	90.24	182	1351
imprecise	MA	1.85	641	23497
	GER	95.35	287	5047
very imprecise	MA	3.73	1413	41036
	GER	144.45	946	78796

3. Shapes as Strings

IDEA: consider shape encoding as concatenated string, letters = angles

ADVANTAGE: pattern text search possible in linear time

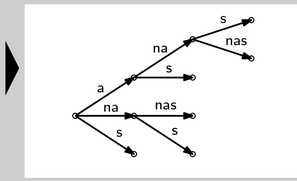
APPROACH: build generalised suffix tree (GST) upon 'text' representing the street network

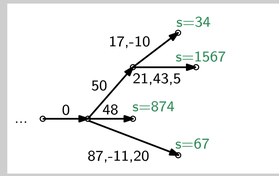
GST represents all suffixes of a set of strings

- each edge is labelled with a non-empty string
- no inner node with degree 1
- each suffix of a string corresponds to unique tree path
- pattern search via tree traversal

ADAPTION TO PATH SHAPES: store for each node which indicates a unique prefix the respective source vertex $s \in V$

Preprocessing interleaves Dijkstra's algorithm and online GST construction.

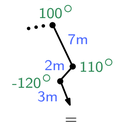




Complete GST on 'ananas' (above) and a small cutout of a GST for path shapes (below).

Encode Shape

- sample into pieces of 1m
- store for each piece relative angle towards previous piece



code length = path length in m

=

...,100,0,0,0,0,0,0,110,0,-120,0,0

Shape-Preserving Dijkstra (SPD)

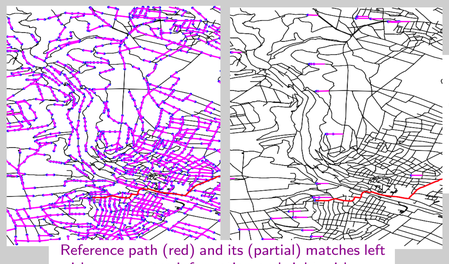
INPUT: start vertex s, reference path shape P

APPROACH: only push a node w into the PQ if the shape of the path $s, \cdots, w$ equals a prefix of P $\Rightarrow$ **Result:** all (partial) matches

Two shapes are **equal** if their angle sequence is equal. In practice imprecise comparison is used, which allows small deviations.

The **unique prefix of** P is the second longest match in the map plus the next code sign

PROBLEM: have to check for every node $s = v \in V$ whether P is possible starting at to assure uniqueness of P (if so) $\Rightarrow$ non-practical query times

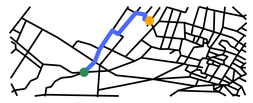


Reference path (red) and its (partial) matches left without compass information and right with.

Fusion with other information
e.g. height data, compass data or area location
$\Rightarrow$ improved query times and precision

QUERY ANSWERING:

1. traverse GST to find source vertex $s \in V$
2. start SPD in s to retrieve the whole path and the actual position



$\Rightarrow$ decreased query times up to factor 3000

PLANNING ENERGY-EFFICIENT ROUTES

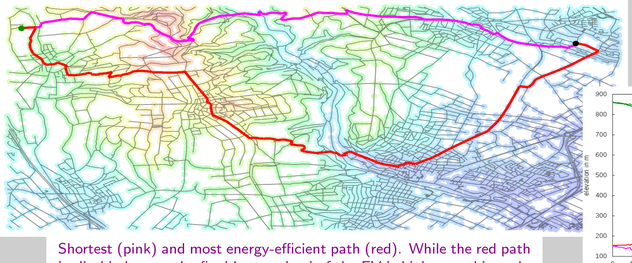
[Eisner, Funke, Storandt; AAAI 11]

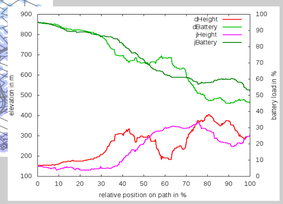
Motivation

- Electric Vehicles (EVs) are battery-powered.
- The cruising range is restricted by the battery's capacity.
- Loading stations are sparse.
- Reloading takes several hours.

Choosing the most energy-efficient route (instead of the shortest one)

- is eco-friendly
- allows to exploit the whole cruising range
- might save recharging inbetween





Shortest (pink) and most energy-efficient path (red). While the red path is slightly longer, the final battery load of the EV is higher on this path.

GIVEN

- street network $G(V, E)$, $|V| = n$, $|E| = m$
- costs $c : E \rightarrow \mathbb{R}$ representing energy consumption
- M maximal battery load of the electric vehicle (EV)

GOAL Compute path p from $s \in V$ to $t \in V$ with minimal energy consumption for initial battery load $I \leq M$.

PROBLEMS

1. partly negative edge eosts
   (due to the recuperation capability of EVs)
   $\Rightarrow$ Dijkstra's algorithm can not be applied
2. battery constraints
   battery load on every node in p must be
   ≥ 0 (no running out of energy) and
   $\leq M$ (no overcharging)

SOLUTION

STEP 1: Model battery contraints into FIFO edge cost functions
Bellman-Ford applicable $\Rightarrow$ query time $\mathcal{O}(nm)$

STEP 2: Apply Johnson's Shifting technique to get rid of negative edge costs
Dijkstra applicable $\Rightarrow$ query time $\mathcal{O}(n \log n + m)$

STEP 3: Apply speed-up technique
Contraction Hierarchies (CH)

$n = 5.5$M, $m = 11$M	query time (sec)
Bellman-Ford	66.91
Dijkstra	3.43
CH	0.18

STEP 1: Modelling Battery Constraints into Edge Cost Functions

(A) Never running out of energy

Battery load b_v of any node v on a feasible path must fulfill $b_v \geq 0$.

If for an edge $e = (v, w)$ yields $b_v < c(e)$ we can not use e!

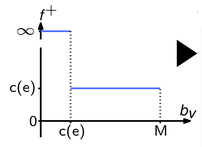
$$f_e^+(b_v) = \begin{cases} \infty & b_v < c(e) \\ c(e) & otherwise \end{cases}$$

(B) No overcharging

Battery load b_v of any node v on a feasible path must fulfill $b_v \leq M$.

If for an edge $e = (v, w)$ yields $b_v - c(e) > M$ we can only recuparate energy up to $b_w = M$!

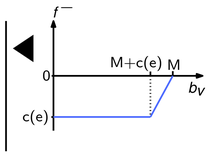
$$f_e^-(b_v) = \begin{cases} c(e) & b_v < M + c(e) \\ b_v - M & otherwise \end{cases}$$



Unified Representation

$$f_e(b_v) = \begin{cases} \infty & b_v < l \\ c(e) & b_v \in [l, u] \\ b_v - u + c(e) & b_v > u \end{cases}$$

fulfills FIFO-property
$\Rightarrow$ Bellman-Ford applicable



STEP 2: Getting Rid of negative Edge Costs

Johnson's Shifting Technique

Theorem (Johnson 1977)
There exists a potential function $P : V \rightarrow \mathbb{R}$ such that $\forall e = (v, w) : c'(e) = c(e) + P(v) - P(w) \geq 0$ and the shortest path structure does not change under the transformation!

Potential function P can be determined via a single run of Bellman-Ford's algorithm in $\mathcal{O}(nm)$.

Adaption to Edge Cost Functions

Modified Transformation
$f'(b_v) = f(b_v - P(v)) + P(v) - P(w)$

Modified potential function retrieval
Bellman-Ford in graph with costs $c(e) = \min_{0 \leq b \leq M} f_e(b)$.

Modified Query Answering
Initialization: $I' = I + P(s)$
Result: $b_t = b'_t - P(t)$

only non-negative edge costs in transformed graph G'
$\Rightarrow$ Dijkstra applicable

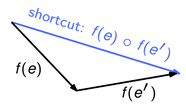
STEP 3: Further speed-up via Contraction Hierarchies (CH)

CH
$\rightarrow$ graph preprocessing technique [Geisberger et al. 2008]
basic idea: add shortcuts to the graph
$\rightarrow$ adaptable to edge cost functions [Batz et al. 2009]

Problem:
Requires chaining of edge cost functions!
In general drastic growth of

- function complexity
- space-consumption

Might prohibit usability of CH...



Theorem (EFS 2011)
The descriptive complexity of our cost function $f_p(b_v)$ on a path $p = v, \cdots$ is bounded.

+ $f()$ is closed under chaining
+ chaining can be performed in $\mathcal{O}(1)$

$\Rightarrow$ CH applicable in our scenario

TAKING INTO ACCOUNT LOADING STATIONS

[Storandt, Funke; AAAI 12]

A **Loading Station (LS)** is a node $l \in V$, that leads to a full battery ($b_l = M$), whenever it is visited.

The set of all LSs is called $L \subseteq V$.

GOAL Compute path from $s \in V$ to $t \in V$ with minimal number of necessary recharging events.

APPROACH Build auxiliary graph Q upon loading stations in a preprocessing step.

$Q(L, E')$

- L set of loading stations
- for $l, l' \in L$: $(l, l') \in E'$ if l' is *reachable* from l

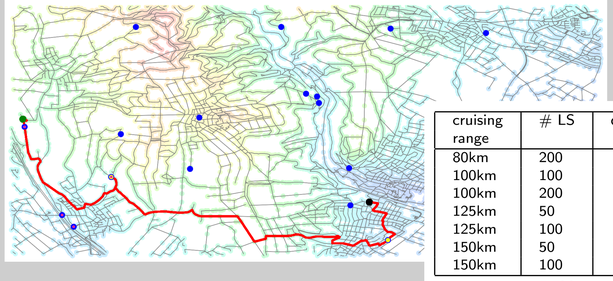
A node $w \in V$ is **reachable** from $v \in V$, if there exists a path from v to w which obeys the battery constraints and requires no recharging inbetween.

QUERY ANSWERING

1. Compute $L_S \subseteq L$ set of reachable LSs from s.
   time $\mathcal{O}(n \log n + m)$
2. Compute $L_t \subseteq L$ set of LSs t can be reached from.
   time $\mathcal{O}(n \log n + m)$
3. Run BFS from L_S in Q until the first node in L_t is visited.
   time $\mathcal{O}(n)$
4. Backtrack respective path.
   time $\mathcal{O}(n)$

Total run time $\mathcal{O}(n \log n + m)$

Path where recharging is necessary twice to reach the target (black) from the source (green). Loading stations are marked blue.



cruising range	# LS	query time (sec)
80km	200	0.17
100km	100	0.32
100km	200	0.32
125km	50	0.47
125km	100	0.45
150km	50	0.53
150km	100	0.51

$n = 5.5$M, $m = 11$M