

HISTORY OF MOVEMENT IN NETWORKS

Data Model

Motivation

In many applications, moving entities move along networks

- cars, vehicles on roads, highways
- trains, underground, buses on public transport networks, even air planes
- people along roads, hiking trails, paths, pedestrian zones
- vessels along rivers, canals

Such movement can be described by the previous model. Advantages of a specialized model?

- network should be modeled explicitly
 - easier formulation of queries
 - more efficient execution
- describe movements relative to network, not the embedding space
 - factor out geometry, descriptions of moving object much more compact
 - discovering relationships between moving objects and parts of the network much simpler and more efficient

Further Requirements

- describe static and moving network positions and regions
 - gas station, construction area, car, traffic jam
- handle interactions between **network, network space, space**
 - **network**: roads, junctions
 - **network space**: vehicles, gas stations, traffic jams
 - **space**: forests, city areas, rivers, ...
 - network - network space: On which highway is this car?
 - network space - space: At what times is the vehicle within the fog area?
 - network - space: Return the part of the network that lies within forest X.
- adequate model of network
 - concept of paths
 - simple and dual routes
 - transitions at junctions
- consistent with model for unconstrained movement; reuse concepts

Network Model

First idea: directed graph.

However, **routes (or paths)** are the conceptual entities in real life:

- cities are organized by **streets**, not blocks. Street names, street addresses, for example.
- **highways**, not pieces of highway between junctions

Further advantage:

- descriptions of movements much more compact than in a graph model.

Network Definitions

Route:

$(id, l, c, kind, start)$

where $id \in \underline{int}$, $l \in \underline{real}$, $c \in \underline{line}$,

$kind \in \{simple, dual\}$, $start \in \{smaller, larger\}$

Route measure:

(rid, d)

where $rid \in \underline{int}$, $d \in \underline{real}$,

and (rid, l, c, k, s) is a route such that $0 \leq d \leq l$

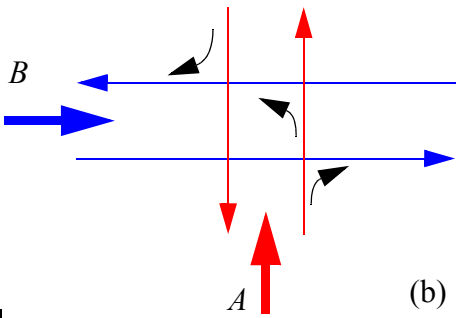
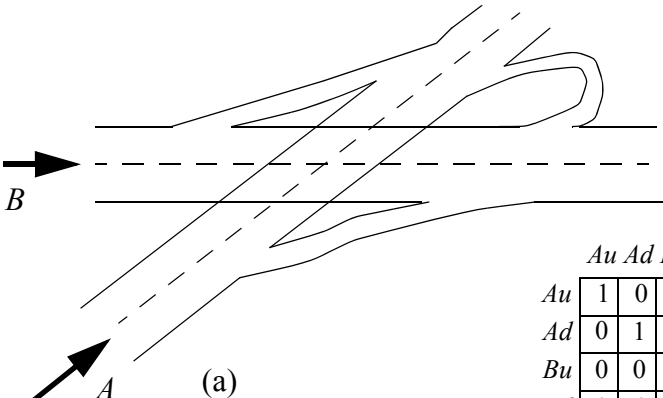
Junction:

(rm_1, rm_2, cc)

where $rm_1 = (r_1, d_1)$ and $rm_2 = (r_2, d_2)$ are route measures ($r_1 \neq r_2$)

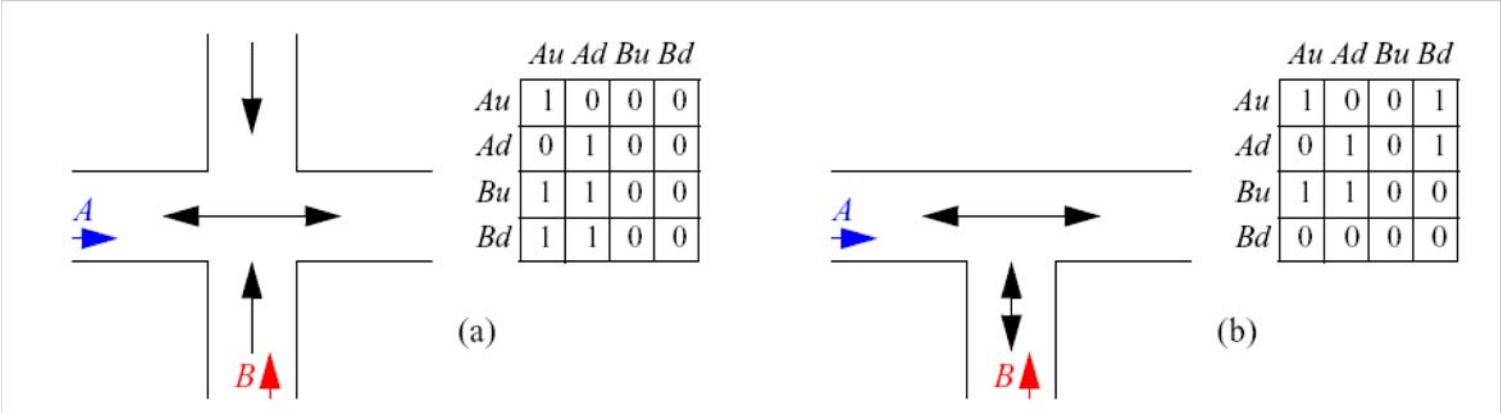
and $cc \in \underline{int}$ is a connectivity code

Connectivity codes:



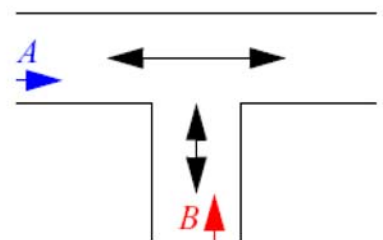
	<i>Au</i>	<i>Ad</i>	<i>Bu</i>	<i>Bd</i>
<i>Au</i>	1	0	1	1
<i>Ad</i>	0	1	0	1
<i>Bu</i>	0	0	1	0
<i>Bd</i>	0	0	0	1

(c)



(a)

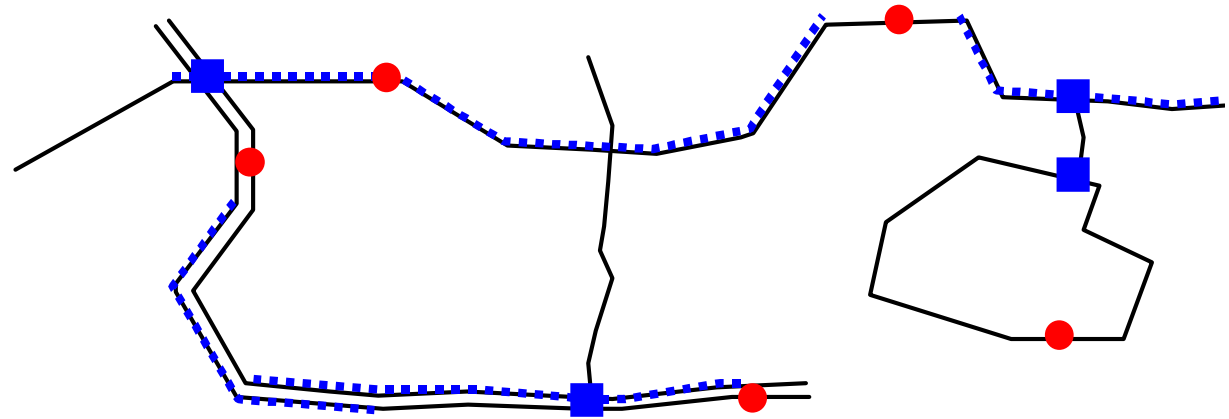
	<i>Au</i>	<i>Ad</i>	<i>Bu</i>	<i>Bd</i>
<i>Au</i>	1	0	0	0
<i>Ad</i>	0	1	0	0
<i>Bu</i>	1	1	0	0
<i>Bd</i>	1	1	0	0



(b)

	<i>Au</i>	<i>Ad</i>	<i>Bu</i>	<i>Bd</i>
<i>Au</i>	1	0	0	1
<i>Ad</i>	0	1	0	1
<i>Bu</i>	1	1	0	0
<i>Bd</i>	0	0	0	0

Data Types



Three static types:

- network: the network
- gpoint: a position in the network
- gline: a “region” (part) of the network

Domains of data types:

D_{network} is a set of all possible networks

Let $N = \{N_1, \dots, N_k\}$ be the set of networks present in the database.

$$D_{\text{gpoint}} = \{(i, gp) \mid 1 \leq i \leq k \wedge gp \text{ is either a route location or } \perp\}$$

$$D_{\text{gline}} = \{(i, gl) \mid 1 \leq i \leq k \wedge gl \text{ is a network region}\}$$

Operations

- Operations to Interface with Relations and Standard Types
 - relational views of a network: converting a network to relations
 - constructing a network from relations
 - accessing gpoint and gline values
 - constructing gpoint and gline values
- Operations according to [GBE+00]
 - type system extended
 - concept of spaces extended

Operations on non-temporal types

- generic operations (stuff inherited from GBE)
- added: interaction between network objects and space
- added: interaction between network and space
- added: interaction between network (routes) and network objects

Operations on temporal types

- lifting
 - generic operations (stuff inherited from GBE)
- Some more operations
 - network operations
 - auxiliary (time intervals, now, etc.)

Operations to Interface with Relations and Standard Types

Relational views of a network: converting a network to relations

network → rel routes, junctions, sections

Result schemas:

```
(route: int, length: real, curve: line, dual: bool, startsSmaller: bool)
(route1: int, meas1: real, route2: int, meas2: real, cc: int, pos: point)
(route: int, meas1: real, meas2: real, dual: bool, curve: line)
```

Example:

Assume a city road network **Hagen** and a road relation

```
road(name: string, route: int, roadLevel: int)
```

Query: “Find all sections of Bahnhofstrasse longer than 500 meters!”

```
SELECT *
FROM sections(Hagen) AS s, road AS r
WHERE r.name = 'Bahnhofstrasse' AND s.route = r.route AND
      (s.meas2 - s.meas1) > 0.5
```

Operations According to [GBE+00]

Type system extended:

	→ <i>BASE</i>	<u><i>int, real, string, bool</i></u>
	→ <i>SPATIAL</i>	<u><i>point, points, line, region</i></u>
	→ <i>GRAPH</i>	<u><i>gpoint, gline</i></u>
	→ <i>TIME</i>	<u><i>instant</i></u>
<i>BASE</i> ∪ <i>SPATIAL</i> ∪ <i>GRAPH</i>	→ <i>TEMPORAL</i>	<u><i>moving, intime</i></u>
<i>BASE</i> ∪ <i>TIME</i>	→ <i>RANGE</i>	<u><i>range</i></u>

Results in types:

- *moving(gpoint)*, *moving(gline)*
- *intime(gpoint)*, *intime(gline)*

Operations According to [GBE+00]

Concept of spaces extended:

			<i>point type</i>	<i>point set types</i>
<i>1D Spaces</i>	<i>discrete</i>	<i>Integer</i>	<i>int</i>	<i>range(int)</i>
		<i>Boolean</i>	<i>bool</i>	<i>range(bool)</i>
		<i>String</i>	<i>string</i>	<i>range(string)</i>
	<i>continuous</i>	<i>Real</i>	<i>real</i>	<i>range(real)</i>
		<i>Time</i>	<i>instant</i>	<i>periods</i>
<i>2D Space</i>		<i>2D</i>	<i>point</i>	<i>points, line, region</i>
		<i>Graph</i>	<i>gpoint</i>	<i>gline</i>

Operations According to [GBE+00] - Non-Temporal Operations

(a) Many operations inherited from [GBE+00], for example:

$\pi \times \sigma$	\rightarrow <u>bool</u>	inside
$\sigma_1 \times \sigma_2$	\rightarrow <u>bool</u>	inside, intersects
$\pi \times \pi$	\rightarrow <u>real</u>	distance

results in:

<u>gpoint</u> \times <u>gline</u>	\rightarrow <u>bool</u>	inside
<u>gline</u> \times <u>gline</u>	\rightarrow <u>bool</u>	intersects
<u>gpoint</u> \times <u>gpoint</u>	\rightarrow <u>real</u>	distance

(b) Define further operations to treat interaction between network, network objects and space, e.g.

<u>gpoint</u>	\rightarrow <u>point</u>	in_space
<u>gline</u>	\rightarrow <u>line</u>	in_space
<u>network</u> \times <u>point</u>	\rightarrow <u>gpoint</u>	in_network
<u>network</u> \times <u>points</u>	\rightarrow <u>gline</u>	in_network
<u>network</u> \times <u>line</u>	\rightarrow <u>gline</u>	in_network
<u>network</u> \times <u>region</u>	\rightarrow <u>gline</u>	in_network

Operations According to [GBE+00] - Temporal Operations

(a) Lifting: applies to all the non-temporal operations defined before. For example:

$\underline{gpoint} \times \underline{gline}$	$\rightarrow \underline{bool}$	inside
$\underline{moving}(gpoint) \times \underline{gline}$	$\rightarrow \underline{moving}(bool)$	inside
$\underline{gpoint} \times \underline{moving}(gpoint)$	$\rightarrow \underline{moving}(bool)$	inside
$\underline{moving}(gpoint) \times \underline{moving}(gpoint)$	$\rightarrow \underline{moving}(bool)$	inside
\underline{gpoint}	$\rightarrow \underline{point}$	in_space
$\underline{moving}(gpoint)$	$\rightarrow \underline{moving}(gpoint)$	in_space

(b) Many generic operations for temporal types inherited from [GBE+00]. For example:

$\underline{moving}(\alpha)$	$\rightarrow \underline{periods}$	deftime
$\underline{moving}(\alpha) \times \underline{instant}$	$\rightarrow \underline{intime}(\alpha)$	atinstant
$\underline{moving}(\alpha) \times \underline{periods}$	$\rightarrow \underline{moving}(\alpha)$	atperiods
$\underline{moving}(gpoint)$	$\rightarrow \underline{periods}$	deftime
$\underline{moving}(gpoint) \times \underline{instant}$	$\rightarrow \underline{intime}(gpoint)$	atinstant
$\underline{moving}(gpoint) \times \underline{periods}$	$\rightarrow \underline{moving}(gpoint)$	atperiods

Further Operations

For example, some network specific operations:

gpoint × real

→ gline

circle, out_circle, in_circle

gpoint × gpoint

→ gline

shortest_path

Some Example Queries

Parcel Delivery

```
road(name: string, route: int)
postman(name: string, trip: mgpoint)
city_area(name: string, reg: region)
```

“Which places did Bob visit between 9:00 am and 11:00 am of last Saturday?”

```
LET Saturday9to11 = period(hour(2003,8,9,9), hour(2003, 8, 9, 10));

SELECT trajectory(atperiods(trip, Saturday9to11))
FROM postman
WHERE name = 'Bob'
```

Some Example Queries

Parcel Delivery

```
road(name: string, route: int)
postman(name: string, trip: mgpoint)
city_area(name: string, reg: region)
```

“For the last week, who visited the Marktring area most often?”

```
LET LastWeek = period(day(2003, 8, 4), day(2003, 8, 9));
LET Marktring = ELEMENT(
  SELECT reg FROM city_area WHERE name = 'Marktring');

SELECT name, no_components(
  at(atperiods(trip inside Marktring, LastWeek), true)) AS no_times
FROM postman
ORDER BY no_times
FIRST 1
```

Some Example Queries

Vehicles on Highway Networks

```
highway(no: int, route: int)
vehicle(licence: string, trip: mgpoint)
...
```

“How does traffic density at km 140 of highway 45 of the network change through the day?”

```
LET yesterday = ...;
LET highway45 = ELEMENT(SELECT route FROM highway WHERE no = 45);
LET location = gpoint(GermanHighways, highway45, 140, up);

LET passing_times =
  SELECT hour(inst(initial(at(atperiods(v.trip, yesterday), location))))
         AS hour
  FROM vehicle AS v
  WHERE atperiods(v.trip, yesterday) passes location;

SELECT hour, COUNT(*) AS no_vehicles
FROM passing_times
GROUP_BY hour
```

Some Example Queries

Vehicles on Highway Networks

```
highway(no: int, route: int)
vehicle(licence: string, trip: mgpoint)
...
speed_limit(limit: int, stretch: gline)
```

“Find vehicles that exceeded the speed limit by more than 20%; when and where did that occur?”

Some Example Queries

Vehicles on Highway Networks

```
highway(no: int, route: int)
vehicle(licence: string, trip: mgpoint)
...
speed_limit(limit: int, stretch: gline)
```

“Find vehicles that exceeded the speed limit by more than 20%; when and where did that occur?”

```
SELECT v.licence,
       deftime(speed(at(v.trip, s.stretch)) when[. > s.limit * 1.2])
       AS time
       trajectory(speed(at(v.trip, s.stretch)) when[. > s.limit * 1.2])
       AS place
FROM vehicle AS v, speed_limit AS s
WHERE v.trip passes s.stretch
      AND max(rangevalues(speed(at(v.trip, s.stretch)))) > s.limit * 1.2
```