Travel-Time Prediction using Gaussian Process Regression: A Trajectory-Based Approach

IBM Tokyo Research Lab.
Tsuyoshi Idé

Contents

- **Problem setting**
- **Background**
- **Formulation**
- **Implementation**
- **Experiment**

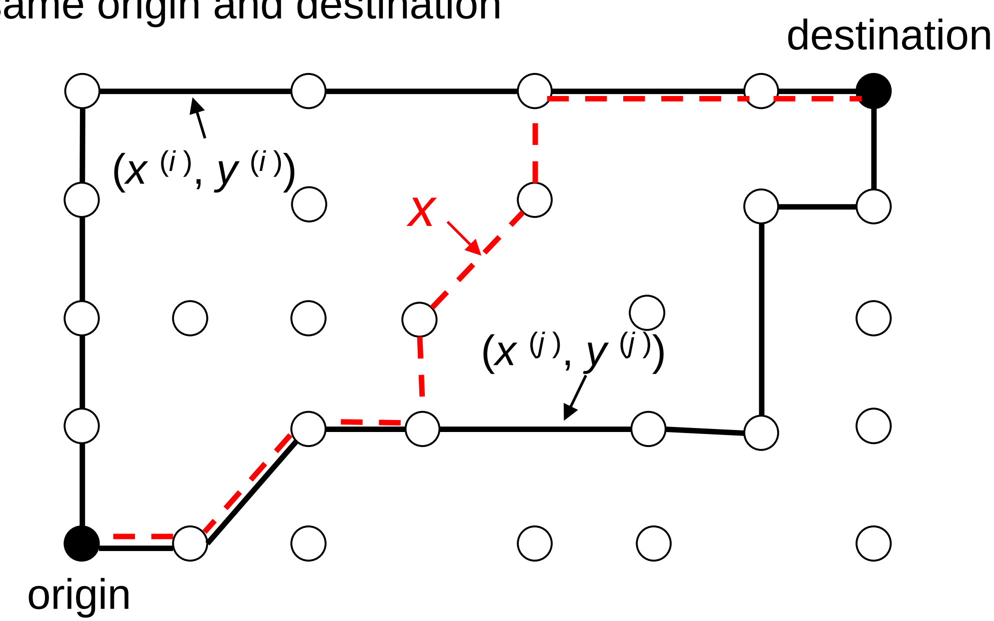
Problem setting:
Predict travel time along arbitrary path

- **Given traffic history data, find a p.d.f.** $p(y|x, \mathcal{D})$

  travel time (y) — input path (x)

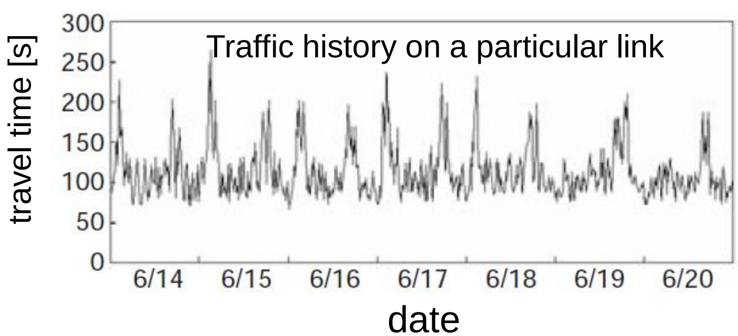
- **Traffic history data is a set of (path, travel time) :**
  - $$\mathcal{D} \equiv \{(x^{(n)}, y^{(n)})|n = 1, 2, ..., N\}$$
    - Assuming all the paths in *D* share the same origin and destination

- **Link**
  road segment between neighboring intersections
- **Path**
  sequence of links



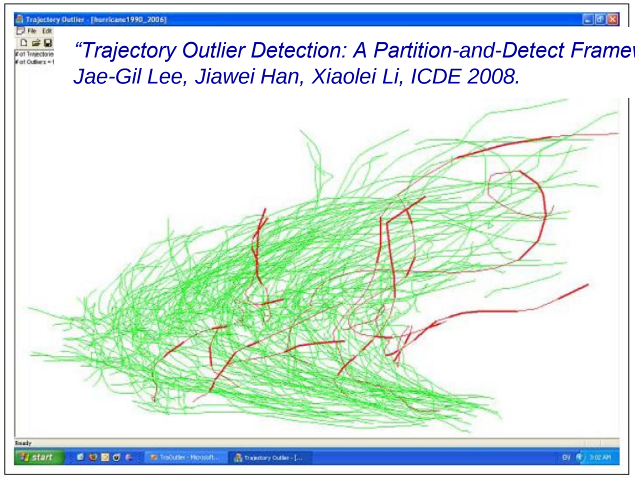
Background (1/2): Traditional time-series modeling is not useful for low-traffic links

- **Traditional approach: time-series modeling for particular link**
  - Construct an AR model or a variant model for computing travel time as a function of time
- **Limitation: hard to model low-traffic links**
  - Time-series modeling needs a lot of data for individual links
  - However, a path includes low-traffic links in general
    - many side roads have little traffic



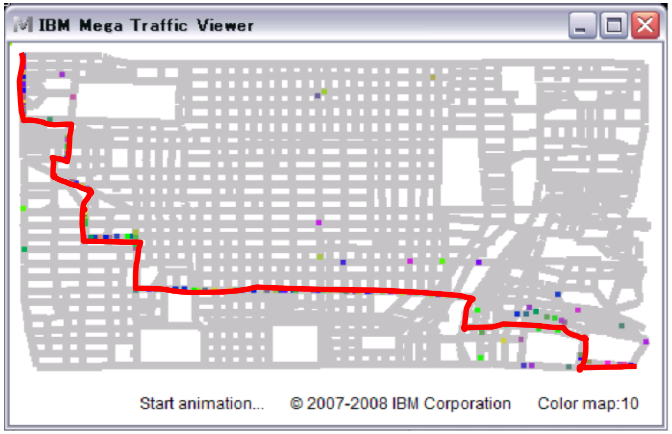
Background (2/2): Trajectory mining is an emerging research field

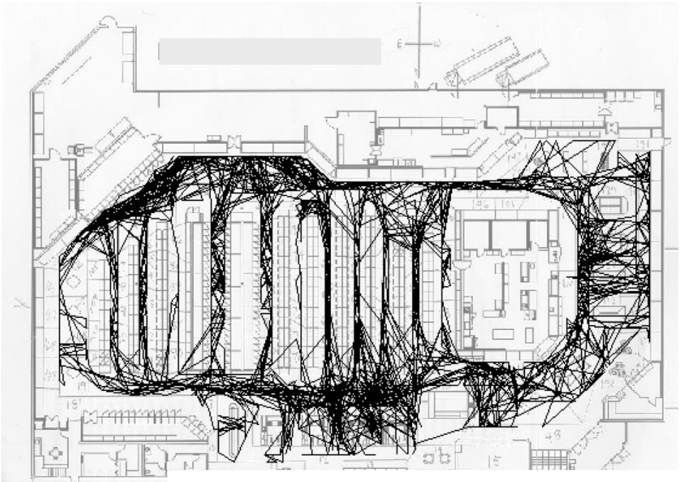
- **Hurricane trajectory analysis**
  - Clustering and outlier detection for trajectories
- **Shopping path analysis**
  - Analyzing shipping paths in stores for marketing
- **Travel time prediction** (this work)
  - Predicting travel time for each trajectory



"Trajectory Outlier Detection: A Partition-and-Detect Framework", Jae-Gil Lee, Jiawei Han, Xiaolei Li, ICDE 2008.

Fig. 9. Trajectory outliers for Hurricane (small).

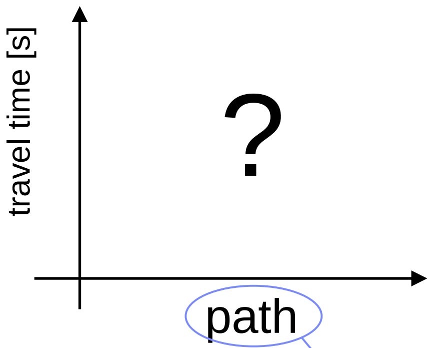


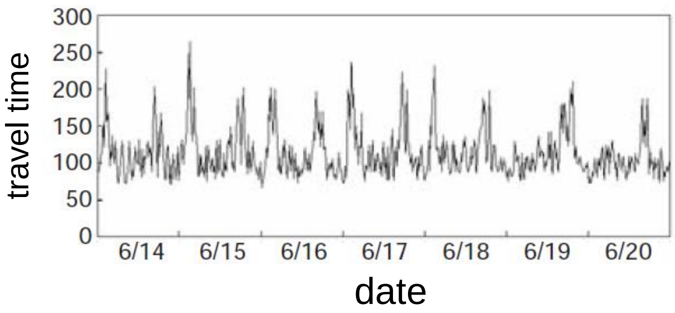


"An exploratory look at supermarket shopping paths", Jeffrey S. Larson, et al. , 2005.

Our problem can be thought of as a non-standard regression problem, where input *x* is not a vector but a path

- **Our problem: input = path (or trajectory)**
- **Conventional: input = time (real value)**





Generally includes low-traffic links
→ time-series modeling is hard due to lack of data.

Our solution

- Use string kernel for computing similarity between trajectories
- Use Gaussian process regression for probabilistic prediction

(Review)
Comparing standard regression with kernel regression

- **Standard regression explicitly needs input vectors**
  - Input = data matrix (design matrix)

$$\mathsf{X} = \left[\boldsymbol{x}^{(1)}, \cdots, \boldsymbol{x}^{(N)}\right]$$

dimensionality of input space

# of samples

- **Kernel regression needs only similarities**
  - Input = kernel matrix
    - i.e. only similarities matter

$$\mathsf{K} = \begin{bmatrix} k(x^{(1)}, x^{(1)}) & \cdots & k(x^{(1)}, x^{(N)}) \\ \vdots & \ddots & \vdots \\ k(x^{(N)}, x^{(1)}) & \cdots & k(x^{(N)}, x^{(N)}) \end{bmatrix}$$

# of samples

# of samples

Formulation (1/4): Employing string kernels for similarity between paths

- **Each path is represented as a sequences of symbols**
  - The "symbol" can be link ID
    - e.g. the 3rd sample may look like

$$x^{(3)} = (25020201, 24021102, \underline{222020101}, 258020001, \ldots)$$

link ID

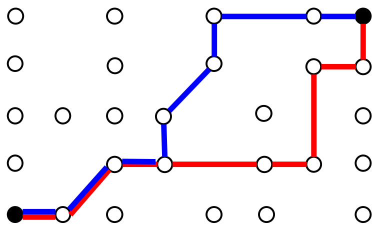
- **String kernel is a natural measure for similarity between strings**
  - We used *p*-spectrum kernel [Leslie 02]

$$k_p(x^{(i)}, x^{(j)}) \equiv \beta \sum_{\boldsymbol{u} \in \Sigma^p} N_{\boldsymbol{u}}(x^{(i)}) N_{\boldsymbol{u}}(x^{(j)})$$

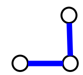
Set of subsequences of *p* consecutive symbols

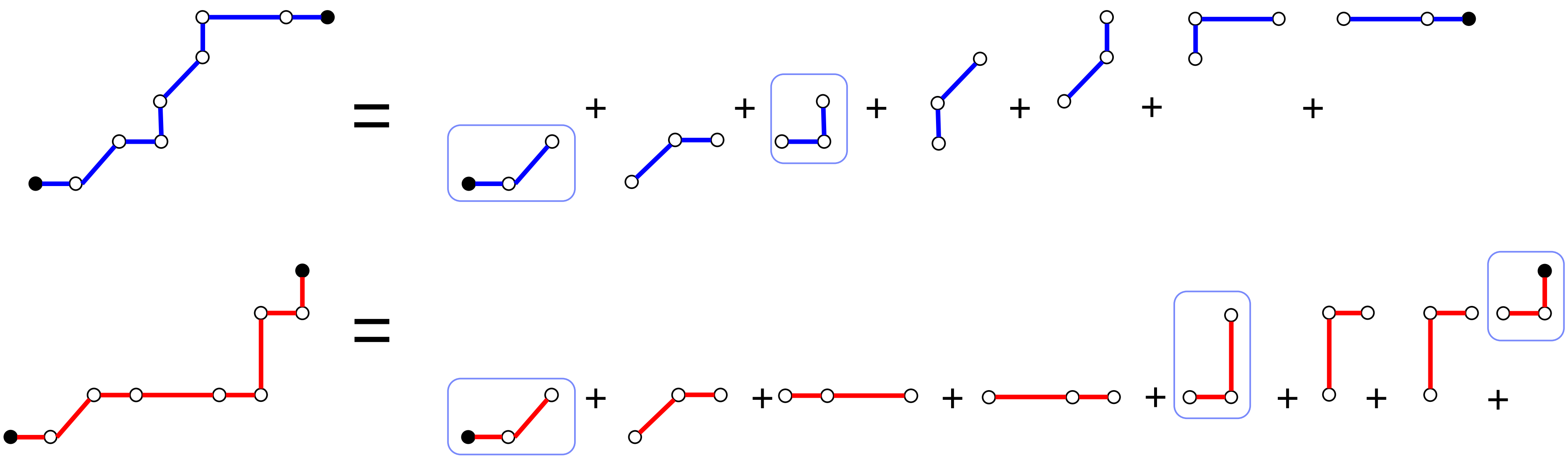
# of occurrences of a subsequence ***u*** in a path $x^{(i)}$

Formulation (2/4): Intuitions behind *p*-spectrum kernel – “split-and-compare”



- **Step 1: Split each path into subsequences**
- **Step 2: Sum up number of co-occurrences**

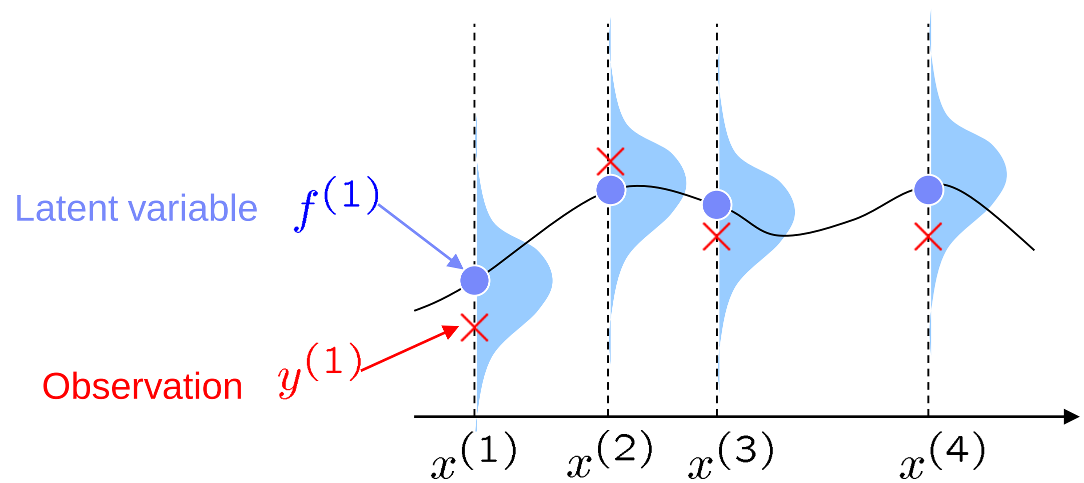
- **Example: *p* = 2, alphabet = {north, south, east, west}**
  - If ***u*** =  = (east, north) , *N*$_{\boldsymbol{u}}$(*blue*) = *2 and N*$_{\boldsymbol{u}}$(*red*) = *3.*



Formulation (3/4): Employing Gaussian process regression (GPR). Two assumptions of GPR

- **Assumption 1: Observation noise is Gaussian**
  - $$p(y^{(n)}|x^{(n)}) = \mathcal{N}(y^{(n)}|f^{(n)}, \sigma^2)$$
- **Assumption 2: Prior distribution of latent variables is also Gaussian**
  - $$p(\boldsymbol{f}_N) = \mathcal{N}(\boldsymbol{f}_N|\mathbf{0}, \mathsf{K})$$
    - Close points favor similar values of the latent variable
      - i.e. “underlying function should be smooth”

$\mathsf{K}_{i,j}$: similarity between path *i* and *j*



Formulation (4/4): Employing Gaussian process regression (GPR). Predictive distribution $p(y|x,\mathcal{D})$ is analytically obtained

- **Predictive distribution is also Gaussian**
  - (See the paper for derivation)

$$p(y|x,\mathcal{D}) = \mathcal{N}(y|m(x), s^2(x))$$

$$m(x) \equiv \boldsymbol{k}^\top \mathsf{C}^{-1} \boldsymbol{y}_N$$

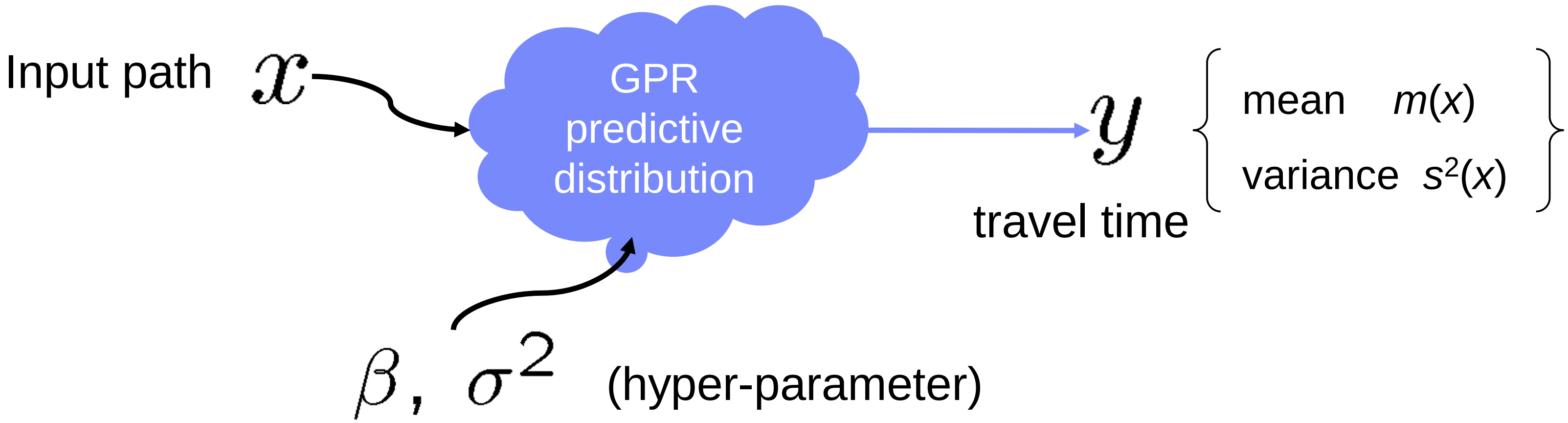
$$s(x)^2 \equiv \sigma^2 + k(x,x) - \boldsymbol{k}^\top \mathsf{C}^{-1} \boldsymbol{k}$$

$$\boldsymbol{f}_N \equiv [f^{(1)}, f^{(2)}, ..., f^{(N)}]^\top$$

$$\boldsymbol{y}_N \equiv [y^{(1)}, y^{(2)}, ..., y^{(N)}]^\top$$

$$\boldsymbol{k} \equiv [k(x,x^{(1)}), k(x,x^{(2)}), ..., k(x,x^{(N)})]^\top$$

$$\mathsf{C} \equiv \mathsf{K} + \sigma^2 \mathsf{I}_N$$



Implementation (1/2): Hyper-parameters are determined from the data

- **Find $\beta,\ \sigma^2$ so that marginal likelihood is maximized**
  - Log marginal likelihood (log-evidence):

$$\psi(\sigma,\beta) \equiv \ln \int \mathrm{d}\boldsymbol{f}_N \ p(\boldsymbol{f}_N) \prod_{n=1}^{N} p(y^{(n)}|f_n)$$

- **We can derive fixed-point equations for** σ^2 and $\gamma \equiv \sigma^2/\beta$
  - No need to use gradient method in 2D space
  - Alternately solve $\frac{\partial \psi}{\partial \gamma} = 0$ and $\frac{\partial \psi}{\partial \beta} = 0$
    - Cholesky factorization is needed at each iteration
      - More efficient algorithm → future work

Implementation (2/2): Algorithm summary

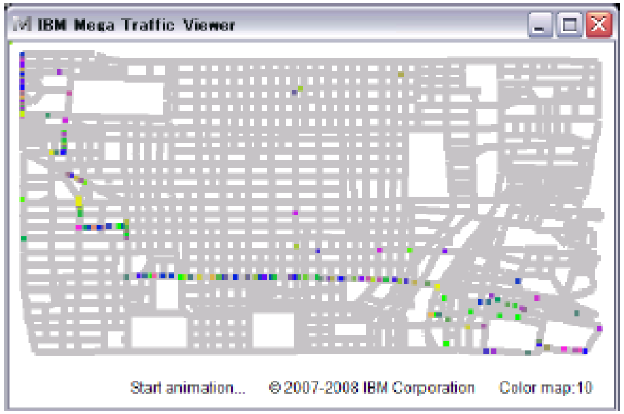
In the *test phase*, we precompute the Cholesky factor L, where $\mathsf{C} = \mathsf{L}\mathsf{L}^\top$, and its inverse L^{-1} as a side product of the Cholesky factorization. We also precompute a vector $\boldsymbol{h} \equiv \mathsf{L}^{-1}\boldsymbol{y}_N$.

1. Input: Path x (and precomputed L^{-1} and $\boldsymbol{h}$).
2. Algorithm:
   - Compute $\boldsymbol{l} \equiv \mathsf{L}^{-1}\boldsymbol{k}$.
   - Compute $m = \bar{y} + \boldsymbol{h}^\top \boldsymbol{l}$.
   - Compute $s^2 = \sigma^2 + k(x, x) - \boldsymbol{l}^\top \boldsymbol{l}$.
3. Output: Predictive mean m and variance s^2.

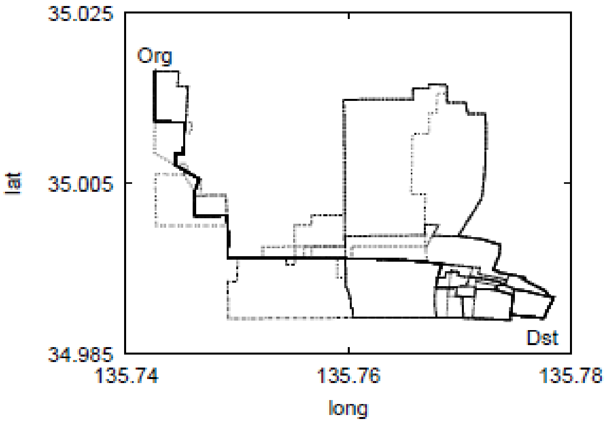
Experiment (1/4): Generating traffic simulation data on an actual map

- **We used IBM Mega Traffic Simulator**
  - Agent-based simulator which allows modeling complex behavior of individual drivers
  - Generated traffic on actual Kyoto City map

- **Data generation procedure: simulating sensible drivers**
  - Pick one of top N_0 shortest paths for a given OD pair
  - Inject the car at the origin with Poisson time interval
  - Determine vehicle speed at every moment as a function of legal speed limit and vehicular gaps
    - Give waiting time τ at each intersection
  - Upon arrival, compute travel time by adding up transit times of all the links



(a) Screenshot of simulator.



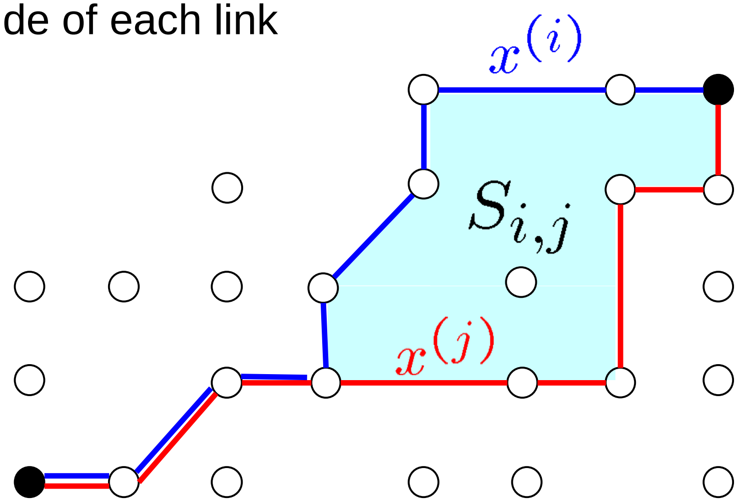
(b) Sample route paths.

Experiment (2/4):
We compare three different kernels

- **ID kernel**
  - p-spectrum kernel whose alphabet Σ is a set of link IDs themselves
    - $k_p(x^{(i)}, x^{(j)}) \equiv \beta \sum_{\boldsymbol{u} \in \Sigma^p} N_{\boldsymbol{u}}(x^{(i)}) N_{\boldsymbol{u}}(x^{(j)})$
    - *p* is an input parameter
- **Direction kernel**
  - *p*-spectrum kernel whose alphabet is the direction of each link
    - North, South, East, West
      - These are determined from longitude and latitude of each link
- **Area kernel**
  - Based on enclosing area S between trajectory pairs

$$k^{\text{area}}(x^{(i)}, x^{(j)}) \equiv \beta \, \mathrm{e}^{-S_{i,j}}$$

  - Can be thought of as a counterpart of standard distances (Euclid distance etc.)



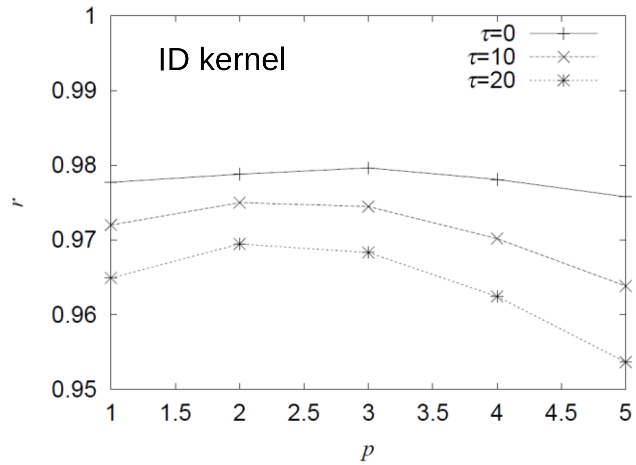
Experiment (3/4): Correlation coefficient as evaluation metric

- **Evaluation metric *r* :**
  correlation coefficient between predicted and actual values
  - $$r \equiv \frac{\sum_{n=1}^{N_{\rm test}} (y^{(n)} - \bar{y})(m(x^{(n)}) - \bar{m})}{\sqrt{\sum_{n=1}^{N_{\rm test}} (y^{(n)} - \bar{y})^2 \sum_{l=1}^{N_{\rm test}} (m(x^{(l)}) - \bar{m})^2}}$$

- **We used *N* = 100 paths for training, and the rest for testing**
  - Total N_0 = 132 paths were generated
  - Compare different intersection waiting times $\tau = 0, 10, 20$
  - Compare different lengths of substring $p = 1, 2, .., 5$

Experiment (4/4): String kernel showed good agreement with actual travel time

- **Comparing different substring lengths (ID and direction kernels)**
  - *p* = 2 gave the best result when $\tau > 0$
    - Major contribution comes from individual links, but turning patterns at intersections also matter
- **Comparing different kernels**
  - ID kernel is the best in terms of high *r* and small variance
  - Area kernel doesn't work
    - The "shapes" of trajectories shouldn't be directly compared



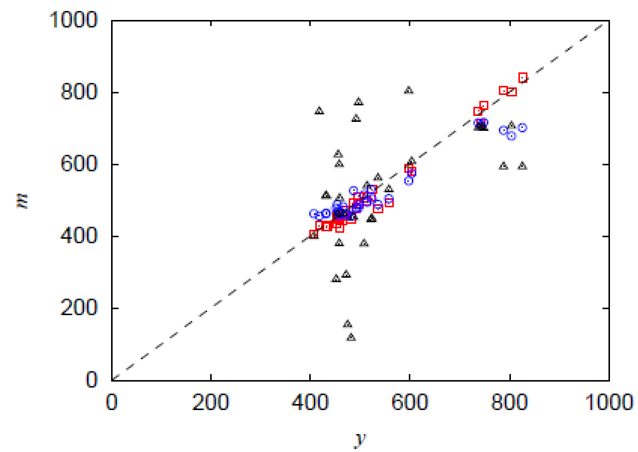


Table 1: r and averaged s^2 values for different kernels ($\tau = 10$).

	ID	direction	area
r	**0.980**	0.933	0.059
$\sqrt{s^2}$	**4.5**	10.0	10.3

Figure 7: Comparison between the predicted (m) and actual (y) travel times with the ID kernel (□), direction kernel (○), and the area kernel (△). The dashed line represents $y = m$, showing perfect agreement.

Summary

- **We formulated the task of travel-time prediction as the problem of trajectory mining**

- **We Introduced two new ideas**

Use of string kernels as a similarity metric between trajectories

Use of Gaussian process regression for travel-time prediction

- **We tested our approach using simulation data and showed good predictability**

Thank You!