A Critical-Time-Point Approach for All-start-time Lagrangian Shortest Paths

V. Gunturi  E. Nunes  K. Yang  S. Shekhar

Dept of Computer Science and Engineering
University of Minnesota

Symposium on Spatial and Temporal Databases 2011
Outline

1. Introduction
2. Basic Concepts and Problem Definition
3. Computational Structure
4. Analytical and Experimental Evaluation
5. Conclusion
1 Introduction

2 Basic Concepts and Problem Definition

3 Computational Structure

4 Analytical and Experimental Evaluation

5 Conclusion
Dynamic nature of Transportation networks

Traffic during non-rush hours

Traffic during Rush hours

Daily Traffic Speed

Time of Average Weekday
NAVTEQ dataset
We can save up to 30% in travel time by considering the dynamic nature.

Reduction in travel time leads to lower fuel consumption, and greenhouse emissions.

Leads to the possibility of Eco-routing.
We can save up to 30% in travel time by considering the dynamic nature.

Reduction in travel time leads to lower fuel consumption, and greenhouse emissions.

Leads to the possibility of Eco-routing.
Problem Instance

- Determine set of shortest paths between University and MSP airport
- Over an interval 7:00am-12:00 noon

<table>
<thead>
<tr>
<th>Time</th>
<th>Preferred Routes</th>
</tr>
</thead>
<tbody>
<tr>
<td>7:30am</td>
<td>Via Hiawatha</td>
</tr>
<tr>
<td>8:30am</td>
<td>Via Hiawatha</td>
</tr>
<tr>
<td>9:30am</td>
<td>via 35W</td>
</tr>
<tr>
<td>10:30am</td>
<td>via 35W</td>
</tr>
</tbody>
</table>
Challenges

Non-stationary ranking of paths

Violation of stationary assumption dynamic programming

<table>
<thead>
<tr>
<th>Time</th>
<th>Preferred Routes</th>
</tr>
</thead>
<tbody>
<tr>
<td>7:30am</td>
<td>Via Hiawatha</td>
</tr>
<tr>
<td>8:30am</td>
<td>Via Hiawatha</td>
</tr>
<tr>
<td>9:30am</td>
<td>via 35W</td>
</tr>
<tr>
<td>10:30am</td>
<td>via 35W</td>
</tr>
</tbody>
</table>
Challenges

Non-stationary ranking of paths

Violation of stationary assumption dynamic programming

<table>
<thead>
<tr>
<th>Time</th>
<th>Preferred Routes</th>
</tr>
</thead>
<tbody>
<tr>
<td>7:30am</td>
<td>Via Hiawatha</td>
</tr>
<tr>
<td>8:30am</td>
<td>Via Hiawatha</td>
</tr>
<tr>
<td>9:30am</td>
<td>via 35W</td>
</tr>
<tr>
<td>10:30am</td>
<td>via 35W</td>
</tr>
</tbody>
</table>

Another instance... Consider the shortest path between MSP and Austin (TX)

<table>
<thead>
<tr>
<th>Time</th>
<th>Route</th>
<th>Flight Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>8:30am</td>
<td>via Detroit</td>
<td>6 hrs 31 mins</td>
</tr>
<tr>
<td>9:10am</td>
<td>direct flight</td>
<td>2 hrs 51 mins</td>
</tr>
<tr>
<td>11:00am</td>
<td>via Memphis</td>
<td>4 hrs 38 mins</td>
</tr>
<tr>
<td>11:30am</td>
<td>via Atlanta</td>
<td>6 hrs 28 mins</td>
</tr>
<tr>
<td>2:30pm</td>
<td>direct flight</td>
<td>2 hrs 51 mins</td>
</tr>
</tbody>
</table>
Challenges

Non-stationary ranking of paths

Violation of stationary assumption dynamic programming

Another instance... Consider the shortest path between MSP and Austin (TX)

<table>
<thead>
<tr>
<th>Time</th>
<th>Preferred Routes</th>
</tr>
</thead>
<tbody>
<tr>
<td>7:30am</td>
<td>Via Hiawatha</td>
</tr>
<tr>
<td>8:30am</td>
<td>Via Hiawatha</td>
</tr>
<tr>
<td>9:30am</td>
<td>via 35W</td>
</tr>
<tr>
<td>10:30am</td>
<td>via 35W</td>
</tr>
</tbody>
</table>

Non-FIFO nature of the network

Violates the no wait assumption of Dijkstra/A*
The related work can be classified into:

- FIFO - A* based - Chabini et al. and Kanoulas et al.
- non-FIFO - Our work.
1 Introduction

2 Basic Concepts and Problem Definition

3 Computational Structure

4 Analytical and Experimental Evaluation

5 Conclusion
Lagrangian Vs Eulerian perspectives

Snapshot model: Eulerian view

Time t=0

Time t=1

Time t=2

Time t=3

Time t=4
Question

- What is the shortest path from A to D for start time $t = 0$?
- Is it $<A,C,D>$ or $<A,B,D>$ or both?
Lagrangian Vs Eulerian perspectives

Snapshot model: Eulerian view

- What is the shortest path from A to D for start time $t = 0$?
- Is it $<A,C,D>$ or $<A,B,D>$ or both?

<table>
<thead>
<tr>
<th>Path</th>
<th>Total travel time at</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>t=0</td>
</tr>
<tr>
<td>$&lt;A,C,D&gt;$</td>
<td>4</td>
</tr>
<tr>
<td>$&lt;A,B,D&gt;$</td>
<td>6</td>
</tr>
</tbody>
</table>
### Problem: All start time Lagrangian Shortest Path (ALSP)

#### Input
- A spatio-temporal (ST) network $G = (V, E)$.
- A source, destination pair.
- A discrete time interval over which the shortest path is to be determined.

#### Output
- A set of routes between source and destination.
- Each route is associated with set of time instants.

#### Objective
- Each route in output is shortest for its corresponding time instants.

#### Constraints
- The length of the time horizon over which the ST network is considered is finite.
- The edge travel time function is a discrete time series.
Running example

INPUT

(a) ST network

(b) Source = A, (c) Destination = D, (d) Time interval = [0 3]

OUTPUT

<table>
<thead>
<tr>
<th>Path</th>
<th>Total travel time at</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>t=0</td>
</tr>
<tr>
<td>&lt;A,C,D&gt;</td>
<td>4</td>
</tr>
<tr>
<td>&lt;A,B,D&gt;</td>
<td>6</td>
</tr>
</tbody>
</table>

- Path \(< A, C, D >\) is shortest for times \([0 1]\) and path \(< A, B, D >\) is shortest for time \([2 3]\).
Naive solution: Run shortest path for each start time

**Snapshot model**

- Time $t=0$
  - Node A to B: 3
  - Node A to C: 1
  - Node B to D: 1
  - Node C to D: 3

- Time $t=1$
  - Node A to B: 3
  - Node A to C: 1
  - Node B to D: 1
  - Node C to D: 3

- Time $t=2$
  - Node A to B: 1
  - Node A to C: 2
  - Node B to D: 2
  - Node C to D: 4

- Time $t=3$
  - Node A to B: 1
  - Node A to C: 3
  - Node B to D: 2
  - Node C to D: 4

- Time $t=4$
  - Node A to B: 2
  - Node A to C: 3
  - Node B to D: 1
  - Node C to D: 3

**Using Time expanded graph**

- Run Dijkstra’s for each start time
- Here, run Dijkstra’s from A0, A1, A2, A3.
Naive solution: Run shortest path for each start time

Snapshot model

Using time aggregated graph

- Use generalized version of label correcting/correcting algorithms.
- **Modified Best Start Time algorithm (MBEST)**
  - Best Start Time algorithm (BEST) (George et al. SSTD '07): Finds best start time in an interval.
  - Generalized for ALSP problem.
Limitations of Naive approach

- Computationally inefficient
- Scalability:
  - # start-times = 1000 ?
  - Size of graph = 1 billion nodes/edges ?

Is there a way to avoid re-computation for each start time?
1 Introduction

2 Basic Concepts and Problem Definition

3 Computational Structure

4 Analytical and Experimental Evaluation

5 Conclusion
Idea of *critical-time point* approach

Observation

It's enough to compute shortest path at start time and recompute at critical time points.

- In this case, compute shortest paths at times $t = 0$ and $t = 2$. 

<table>
<thead>
<tr>
<th>Path</th>
<th>Start-times</th>
</tr>
</thead>
<tbody>
<tr>
<td>$&lt;A,C,D&gt;$</td>
<td>0,1</td>
</tr>
<tr>
<td>$&lt;A,B,D&gt;$</td>
<td>2,3</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Path</th>
<th>Total travel time at</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$t=0$</td>
</tr>
<tr>
<td>$&lt;A,C,D&gt;$</td>
<td>4</td>
</tr>
<tr>
<td>$&lt;A,B,D&gt;$</td>
<td>6</td>
</tr>
</tbody>
</table>
**Proof of Sketch**

- Divide the given non-stationary interval into disjoint stationary intervals.
- Run a dynamic programming based approach for each interval.

**Implementation Sketch**

- Each candidate path is associated with a cost-function.
- Critical points are determined by computing the intersections points between the cost-functions.
Meeting Non-FIFO challenge

Transforming Travel time to Earliest arrival time

Consider Travel time series of edge (A,B)

\[
\begin{bmatrix}
3 & 3 & 1 & 1
\end{bmatrix}
\]
\[t = 0 \quad 1 \quad 2 \quad 3\]

STEP 1

Add time index to travel time to get arrival time series

\[
\begin{bmatrix}
3+0 & 3+1 & 1+2 & 1+3
\end{bmatrix}
\]
\[t = 0 \quad 1 \quad 2 \quad 3\]

STEP 2

Earliest Arrival time series

\[
\begin{bmatrix}
3 & 3 & 3 & 4
\end{bmatrix}
\]

Compare each value to the ones on its right and choose the min

\[
\begin{bmatrix}
3 & 4 & 3 & 4
\end{bmatrix}
\]
Transformed time aggregated graph (TAG)

PROPERTY: Earliest arrival time TAG is **FIFO**.
Path function computation

- Why Path Function?
  - To compute Critical time points where path ranking may change.
- What are path functions?
  - Earliest arrival time at the end node.

How to compute them?

**Example Diagram**

- **EAT of AC**
  - t= 0 1 2 3 4
  - [1 2 4 5 7]
  - If we depart A at t=0, we reach C at t=1

- **EAT of CD**
  - t= 0 1 2 3 4
  - [3 4 4 6 6]
  - If we depart C at t=1, we reach D at t=4

- **Path function of <A,C,D>**
  - t= 0 1 2 3
  - [4 4 6 6]
  - Thus if we depart A at t=0 we reach D at t=4 through path <A,C,D>
Path Functions

**Property:** Earliest arrival time TAG is *FIFO*.
Basic concepts for **Critical-time-point**

- **Non-critical times**: Path ranking can’t change.
- **Critical-time-points**: Time point where path ranking may change.

**Observation**

Path ranking cannot change at *non critical-time-points*. 

The ranking changes at t=2. Thus t=2 becomes a **Critical time point**.
Two ways to compute the critical time points:

**Pre-computation:**
- Enumerate all the possible paths.
- Can be inefficient except for very sparse graphs with few paths.

**On the fly**
- Do we know a shortest path algorithm which enumerates and prunes the paths?
Two ways to compute the critical time points:

**Pre-computation:**
- Enumerate all the possible paths.
- Can be inefficient except for very sparse graphs with few paths.

**On the fly**
- Do we know a shortest path algorithm which enumerates and prunes the paths?
- Recall Dijkstra’s → enumerates and prunes.
Two ways to compute the critical time points:

**Pre-computation:**
- Enumerate all the possible paths.
- Can be inefficient except for very sparse graphs with few paths.

**On the fly**
- Do we know a shortest path algorithm which enumerates and prunes the paths?
- Recall Dijkstra’s → enumerates and prunes.
- Can also be generalized to find critical time points on fly.
- *Critical Time point All start time Lagrangian shortest paths Solver (CTAS) uses this.*
- Alternatively generalize A* and other label correcting approaches.
1 Introduction

2 Basic Concepts and Problem Definition

3 Computational Structure

4 Analytical and Experimental Evaluation

5 Conclusion
Correctness and Completeness

Correctness
The correctness proof for CTAS algorithm is similar to that of Dijkstra’s algorithm.

Completeness
Completeness of CTAS is based on Lemma 1.

Lemma
CTAS does not recompute for any non-critical time points.
Experimental Setup

Goals

- Evaluate # of shortest paths re-computations avoided by CTAS.
- Evaluate the performance of CTAS by varying different parameters.

Parameters

- Length of start time interval (Lambda).
- Length of path (in terms of travel time).
- Rush vs Non-Rush hours characteristics.

Candidates

- CTAS algorithm.
- MBEST: A generalized version of a label correcting algorithm developed for finding best start time.
Experiments were carried on a real dataset containing the highway road network of Hennepin county, Minnesota.

- The dataset contained 1417 nodes and 3754 edges.
- The data contained travel times for each edge at time quanta of 15mins.
Metrics of evaluation

**Metric 1: # Re-computation avoided**
- *length of start time interval – recomputations performed.*

**Metric 2: Performance of CTAS**
- *speedup ratio = \( \frac{MBEST\ runtime}{CTAS\ runtime} \)* was calculated for each run.
# Re-computations saved

- Variable parameters: Travel time of path
- Fixed parameters:
  - Different values of start time interval: 100, 200
  - Network size: 1417 nodes, 3754 edges

- Shorter paths ⇒ More savings.
- Non rush hours ⇒ fewer critical time points and thus fewer re-computations.
Performance of CTAS: Effect of length of start time interval

- Variable parameters: length of start time interval (lambda)
- Fixed parameters
  - Different lengths of path: 30, 40
  - Network size: 1417 nodes, 3754 edges

- Longer time interval $\Rightarrow$ Longer runtime of CTAS.
- Runtime of CTAS increases slowly due to fewer increase in critical time points.
Performance of CTAS: Effect of path length

- Variable parameters: length of path
- Fixed parameters
  - Different values of start time interval: 100, 200
  - Network size: 1417 nodes, 3754 edges

- Longer paths ⇒ Longer runtime of CTAS.
- Runtime of MBEST does not change.
Conclusion

- Introduced the notion of *critical time points*.
- Designed CTAS: Computes critical time points on fly.
- Correctness and completeness of CTAS.
- Experimental evaluation using real datasets.

Future Work

- Develop cost models.
- Generalize other shortest path algorithms for critical time points.

Acknowledgment

- NSF III-CXT, US-DOD
- SSTD Reviewers
- Spatial database group at University of Minnesota
- Kim Koffolt
- Navteq, http://www.navteq.com