

Finding Time-dependent Shortest Paths over Large Graphs

Alberto Presta

10-07-2019

DSSC - Algorithmic Design exam

Finding
Time-
dependent
Shortest
Paths over
Large Graphs

Alberto
Presta

Introduction
to the
problem

Existing
solutions

New Dijkstra
Based
Algorithm

Time
complexity

Non-FIFO
Graphs

presentation's schedule

- 1 Introduction to the problem
- 2 Existing solutions
- 3 New Dijkstra Based Algorithm
- 4 Time complexity
- 5 Non-FIFO Graphs

Finding
Time-
dependent
Shortest
Paths over
Large Graphs

**Alberto
Presta**

Introduction
to the
problem

Existing
solutions

New Dijkstra
Based
Algorithm

Time
complexity

Non-FIFO
Graphs

- 1 TSDP (time-dependent shortest path problem): to find the optimal path (with minimum travel time) from a source to a destination, over a time-dependent graph.
- 2 We concentrate on finding the *least total travel time* (LTT) from source node v_s to a dest. node v_e with starting time $t \in [t_s, t_e]$, chosen from the user. Such a query is called an LTT *query*, denoted as $\text{LTT}(v_s, v_e)$.
- 3 More focused in on a specific class of graphs, called FIFO time-dependent graphs.

Definition

A *time-dependent graph* is defined as $G_T(V, E, W)$ (or G_T for short):

- 1 $V = \{v_i\}$ is a set of nodes.
- 2 $E \subseteq V \times V$ is a set of edges.
- 3 W is a set of positive valued functions.

For every edge $(v_i, v_j) \in E$, there is a function $\omega_{i,j}(t)$, where t is a time variable in a domain T : It specifies how much time it takes to travel from v_i to v_j , if departing v_i at time t .

Problem Definition

Definition

Given a time-dependent graph $G_T(V, E, W)$ and a $LTT(v_s, v_e, T)$, where $v_s, v_e \in V$ and $T \in \tau$ is a starting-time interval, the *time-dependent shortest path* (TDSP) problem is to minimize LTT:

$$g_{p^*}(t^*) - t^* = \min_{p, \omega(\cdot), t} g_p(t) - t$$

where p^* is the path $v_s - v_e$, $\omega^*(v_i)$ is the waiting time in v_i and t^* is the best starting time, with which results in the minimum travel time $g_p(t) - t$.

Finding
Time-
dependent
Shortest
Paths over
Large Graphs

Alberto
Presta

Introduction
to the
problem

Existing
solutions

New Dijkstra
Based
Algorithm

Time
complexity

Non-FIFO
Graphs

Existing Solutions

three different types of algorithm:

- ① Discrete-time algorithm.
- ② *Bellman-Ford* based algorithm.
- ③ A^* algorithm.

observation

Main challenge: edge delays are different function of departure times, the $v_s - v_e$ path with the least total travel time changes in a complicated manner as the starting time changes

Finding
Time-
dependent
Shortest
Paths over
Large Graphs

Alberto
Presta

Introduction
to the
problem

Existing
solutions

New Dijkstra
Based
Algorithm

Time
complexity

Non-FIFO
Graphs

Discrete-time algorithm

- 1 Find approximate LTT by globally discretizing time interval into time points.
- 2 Given a graph $G_T(V, E, W)$, we discretize the starting-time interval $T = [t_s, t_e]$ into k points and constructs a static graph $G'_T(V', E', W')$ by making k copies of each node and each edge.
- 3 For each edge (v'_i, v'_j) , $w'_{i,j}$ is equal to the value of $w_{i,j}(t)$ on a fixed time point.
- 4 static-single-source shortest path problem on $G'_T(V', E', W')$.

observation

Increasing k deteriorates the efficiency of discrete-time approaches, since G'_t is k times larger than G_t .

Finding
Time-
dependent
Shortest
Paths over
Large Graphs

Alberto
Presta

Introduction
to the
problem

Existing
solutions

New Dijkstra
Based
Algorithm

Time
complexity

Non-FIFO
Graphs

Bellman-Ford Based Algorithm

Definition

Let us give the following definitions:

- ① $g_l(t)$ the *earliest arrival time* function at node v_l , from source v_s , for starting time t .
- ② $h_{k,l}(t)$ the *earliest arrival time* at v_l , from source v_s via edge (v_k, v_l) , for starting time t .

how does the algorithm work?

- ① It updates $g_l(t)$ and $h_{k,l}(t)$ until they converge to the correct values.
- ② it returns the best starting time t^* and the optimal path p^* .
- ③ Time complexity is $O(|V| \cdot |E| \cdot \alpha(T))$ where $\alpha(T)$ is the time required in a function operation in T .

Finding
Time-
dependent
Shortest
Paths over
Large Graphs

Alberto
Presta

Introduction
to the
problem

Existing
solutions

New Dijkstra
Based
Algorithm

Time
complexity

Non-FIFO
Graphs

Bellman-Ford Based Algorithm

Pseudo-code

```
1: for all  $v_l \in V$  do  $g_l \leftarrow \infty$  for  $t \in T$ ;
2: for all  $(v_k, v_l) \in E$  do  $h_{k,l} \leftarrow \infty$  for  $t \in T$ ;
3:  $g_s(t) \leftarrow t$  for  $t \in T$ ;
4: repeat
5:   for all  $(v_k, v_l) \in E$  do  $h_{k,l} \leftarrow g_k(t) + w_{k,l}(g_k(t))$ ;
6:   for all  $v_l \in V$  do  $g_l(t) \leftarrow \min_{v_k \in N(v_l)} h_{k,l}(t)$ ;
7: until all functions  $g_l(t)$  are unchanged;
8: return  $(t^* \leftarrow \arg \min_{t \in T} g_e(t) - t, p^*)$ 
```

Finding
Time-
dependent
Shortest
Paths over
Large Graphs

Alberto
Presta

Introduction
to the
problem

Existing
solutions

New Dijkstra
Based
Algorithm

Time
complexity

Non-FIFO
Graphs

A* Algorithm

Main idea: maintain a priority queue of all paths to be expanded.

- 1 Let p_k be a path from v_s to v_k (there can be multiple paths of this type in the queue).
- 2 Each path is associated with a function $f_{p_k}(t) = g_{p_k}(t) + d_{k,e} - t$.
- 3 In each iteration we pick the path which $\min_t \{f_{p_i}(t)\}$ is minimum and we extend it with one more edge (v_i, v_j) : New path is added in priority queue and the old one is deleted.
- 4 In worst case, all possible paths are enumerated and time/space complexity is exponential with respect the size of G_T .

Finding
Time-
dependent
Shortest
Paths over
Large Graphs

Alberto
Presta

Introduction
to the
problem

Existing
solutions

New Dijkstra
Based
Algorithm

Time
complexity

Non-FIFO
Graphs

First of all, we introduce some important notations:

- ① $G_T(V, E, W)$: time-dependent graph (G_T).
- ② $w_{i,j}(t)$: edge-delay function for $(v_i, v_j) \in E$.
- ③ v_s, v_e, T : source, destination and starting-time interval.
- ④ p^* : optimal path from v_s to v_e .
- ⑤ t^* : optimal starting time.
- ⑥ $\omega^*(v_i)$: optimal waiting time at node v_i .
- ⑦ $g_i(t)$: $v_s - v_i$ earliest arrival-time function.
- ⑧ $g_p(t)$: arrival-time function (along path p).
- ⑨ $\alpha(T)$: time/space required to maintain a function or to manipulate a function operation over time interval T

FIFO time-dependent graph

Introduction

We focus in answering LTT queries in an FIFO (*First-in* and *Firt-out*) time-dependent graph G_T , where no waiting time is needed in optimal solution.

Definition

Time-dependent graph G_T is a FIFO graph, iff every edge (v_i, v_j) has FIFO property. An edge (v_i, v_j) has FIFO property, iff $w_{i,j}(t_{t_0}) \leq t_{\Delta} + w_{i,j}(t_0 + t_{\Delta})$ for $t_{\Delta} \geq 0$.

Theorem

For a given LTT query on a FIFO time-dependent graph G_T , there exists an optimal path p^ along which the optimal waiting time is 0 for every v_i on p^* .*

Finding
Time-
dependent
Shortest
Paths over
Large Graphs

Alberto
Presta

Introduction
to the
problem

Existing
solutions

New Dijkstra
Based
Algorithm

Time
complexity

Non-FIFO
Graphs

New Dijkstra Based Algorithm

Organization of the algorithm

Answering LTT query can be done in two decoupled steps:

- ① *time-refinement*: for every node $v_i \in V$ to compute the earliest arrival time $g_i(t)$, departing from v_s at any starting time $t \in T$.
- ② *path-selection*: we select from first step one of the paths from v_s to v_e , which matches the optimal travel time $g_e(t^*) - t^*$.

These two steps are grouped in the so called **TWO-STEP-LTT**.

Finding
Time-
dependent
Shortest
Paths over
Large Graphs

Alberto
Presta

Introduction
to the
problem

Existing
solutions

New Dijkstra
Based
Algorithm

Time
complexity

Non-FIFO
Graphs

```
TWO-STEP-LTT( $G_T(V, E, W), v_s, v_e, T$ ):  
1:  $\{g_i(t)\} \leftarrow \text{timeRefinement}(G_T, v_s, v_e, T)$ ;  
2: if  $\neg (g_e(t) = \infty \text{ for the entire } [t_s, t_e])$  then  
3:    $t^* \leftarrow \arg \min_{t \in T} \{g_e(t) - t\}$ ;  
4:    $p^* \leftarrow \text{pathSelection}(G_T, g_i(t), v_s, v_e, t^*)$ ;  
5:   return  $(t^*, p^*)$ ;  
6: else return 0;
```

Time-refinement

How does the algorithm works?

The first step is dominating factor in terms of computational cost. We compute $g_i(t)$ for every node $v_i \in V$, through this recursive equation:

$$g_i(t) = \min_{v_j \in N(v_i), \omega(v_j)} (g_j(t) + \omega(v_j)) + w_{i,j}(g_j(t) + \omega(v_j))$$

where $N(v_i) = \{v_j | (v_i, v_j) \in E\}$

Definition

We say that function $g_i(t)$ is *well-refined* in a starting-time subinterval I_i , if it specifies the earliest arrival time at v_i from v_s for any starting time $t \in I_i$.

Finding
Time-
dependent
Shortest
Paths over
Large Graphs

Alberto
Presta

Introduction
to the
problem

Existing
solutions

New Dijkstra
Based
Algorithm

Time
complexity

Non-FIFO
Graphs

How does the algorithm works?

- 1 We refine $g_i(t)$, incrementally in the given starting-time interval $T = [t_s, t_e]$.
- 2 $\forall v_i \in V$, let $I_i = [t_s, \tau_i] \subseteq T$ be a starting-time subinterval, where $\tau_i \in T = [t_s, t_e]$.
- 3 We "incrementally" refine $g_i(t)$ to a largest starting-time subinterval $I'_{i,i}$, in a way that $g_i(t)$ is always well-refined.
- 4 We go ahead until $g_e(t)$ is well refined in the entire starting-time interval T .

Time-refinement

Pseudo-code

Algorithm 3 *timeRefinement* ($G_T(V, E, W)$, v_s, v_e, T)

Input: a time-dependent graph G_T , a query LTT(v_s, v_e, T) - source v_s , destination v_e , and starting-time interval $T = [t_s, t_e]$;

Output: $\{g_i(t) | v_i \in V\}$ - all earliest arrival-time functions.

```
1:  $g_s(t) \leftarrow t$  for  $t \in T$ ;  $\tau_s \leftarrow t_s$ ;
2: for each  $v_i \neq v_s$  do
3:    $g_i(t) \leftarrow \infty$  for  $t \in T$ ;  $\tau_i \leftarrow t_s$ ;
4: Let  $Q$  be a priority queue initially containing pairs,  $(\tau_i, g_i(t))$ ,
   for all nodes  $v_i \in V$ , ordered by  $g_i(\tau_i)$  in ascending order;
5: while  $|Q| \geq 2$  do
6:    $(\tau_i, g_i(t)) \leftarrow \text{dequeue}(Q)$ ;
7:    $(\tau_k, g_k(t)) \leftarrow \text{head}(Q)$ ;
8:    $\Delta \leftarrow \min\{w_{f,i}(g_k(\tau_k)) \mid (v_f, v_i) \in E\}$ ;
9:    $\tau'_i \leftarrow \max\{t \mid g_i(t) \leq g_k(\tau_k) + \Delta\}$ ;
10:  for each  $(v_i, v_j) \in E$  do
11:     $g'_j(t) \leftarrow g_i(t) + w_{i,j}(g_i(t))$  for  $t \in [\tau_i, \tau'_i]$ ;
12:     $g_j(t) \leftarrow \min\{g_j(t), g'_j(t)\}$  for  $t \in [\tau_i, \tau'_i]$ ;
13:     $\text{update}(Q, (\tau_j, g_j(t)))$ ;
14:   $\tau_i \leftarrow \tau'_i$ ;
15:  if  $\tau_i \geq t_e$  then
16:    if  $v_i = v_e$  then
17:      return  $\{g_i(t) | v_i \in V\}$ ;
18:    else
19:       $\text{enqueue}(Q, (\tau_i, g_i(t)))$ ;
20: return  $\{g_i(t) | v_i \in V\}$ .
```

Finding
Time-
dependent
Shortest
Paths over
Large Graphs

Alberto
Presta

Introduction
to the
problem

Existing
solutions

New Dijkstra
Based
Algorithm

Time
complexity

Non-FIFO
Graphs

More on Time-refinement algorithm

- ① We use a priority Queue Q , which initially contains pairs $(\tau_i, g_i(t))$ for all nodes v_i in ascending order of $g_i(\tau_i)$.
- ② While loop conducts time-refinement for every node v_i in G_T .
- ③ Line 6-9 and line 14: we update starting-time interval refinement, by finding the minimum travel time from v_f to fixed v_i (line 8) and by calculating τ' (line 9).
- ④ line 11-14: we use the well-refined $g_i(t)$ to refine arrival-time functions $g_j(t)$ in starting-time subintervals $[\tau_i, \tau'_i]$ of all v_i 's outgoing neighbors.

Finding
Time-
dependent
Shortest
Paths over
Large Graphs

Alberto
Presta

Introduction
to the
problem

Existing
solutions

New Dijkstra
Based
Algorithm

Time
complexity

Non-FIFO
Graphs

How does the algorithm works?

- 1 We determine the predecessor of every node in p^* backward from v_e to v_s based on $\{g_i(t)\}$ and $t^* \in T$.
- 2 This algorithm takes five inputs: graph G_t , all the earliest arrival-time functions $\{g_i(t)\}$, the optimal starting time $t^* \in T$, source v_s and destination v_e .
- 3 The predecessor of v_j is determined as v_i , if $g_j(t^*) = g_i(t^*) + w_{i,j}(g_i(t^*))$ for $(v_i, v_j) \in E$.

Pseudo-Code

PathSelection($G_T(V, E, W), \{g_i(t)\}, v_s, v_e, t^*$):

```
1:  $v_j \leftarrow v_e$ ;  
2:  $p^* \leftarrow \emptyset$ ;  
3: while  $v_j \neq v_s$  do  
4:   for each  $(v_i, v_j) \in E$  do  
5:     if  $g_i(t^*) + w_{i,j}(g_i(t^*)) = g_j(t^*)$  then  
6:        $v_j \leftarrow v_i$ ; break;  
7:      $p^* \leftarrow (v_i, v_j) \cdot p^*$ ;  
8: return  $p^*$ ;
```

Finding
Time-
dependent
Shortest
Paths over
Large Graphs

Alberto
Presta

Introduction
to the
problem

Existing
solutions

New Dijkstra
Based
Algorithm

Time
complexity

Non-FIFO
Graphs

A running example

Our query is $\text{LTT}(v_1, v_4, T = [0, 60])$.

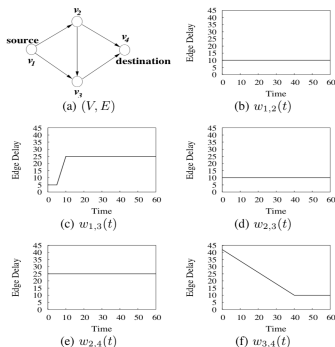


Figure: Our Time-dependent graph, which we use as example

Finding
Time-
dependent
Shortest
Paths over
Large Graphs

Alberto
Presta

Introduction
to the
problem

Existing
solutions

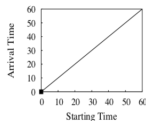
New Dijkstra
Based
Algorithm

Time
complexity

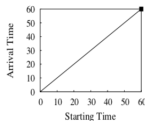
Non-FIFO
Graphs

A running example

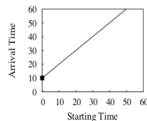
First Iteration:



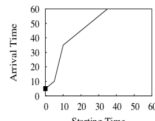
(a) $g_1(t)$



(b) $g_1(t)$

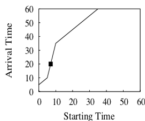


(c) $g_2(t)$

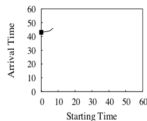


(d) $g_3(t)$

Second Iteration:



(e) $g_3(t)$



(f) $g_4(t)$

Finding
Time-
dependent
Shortest
Paths over
Large Graphs

Alberto
Presta

Introduction
to the
problem

Existing
solutions

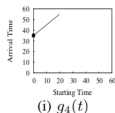
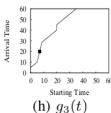
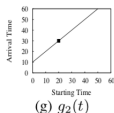
New Dijkstra
Based
Algorithm

Time
complexity

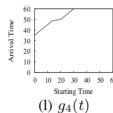
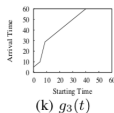
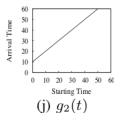
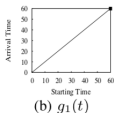
Non-FIFO
Graphs

A running example

Third iteration:



... after 11 iterations we have that all functions are well-refined!



Finding
Time-
dependent
Shortest
Paths over
Large Graphs

Alberto
Presta

Introduction
to the
problem

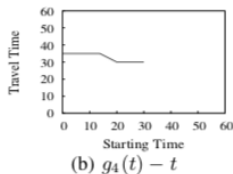
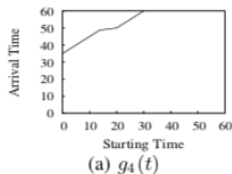
Existing
solutions

New Dijkstra
Based
Algorithm

Time
complexity

Non-FIFO
Graphs

A running example



Solution

$t^* = 20$ and $p^* = (v_1, v_2)(v_2, v_3)(v_3, v_4)$.

Finding
Time-
dependent
Shortest
Paths over
Large Graphs

Alberto
Presta

Introduction
to the
problem

Existing
solutions

New Dijkstra
Based
Algorithm

Time
complexity

Non-FIFO
Graphs

Time complexity of two-Step-LTT

Given a graph G_T with n nodes and m edges in total, consider query $\text{LTT}(v_s, v_e, T)$.

Theorem

The time complexity of the timeRefinement algorithm is $O((n \cdot \log n + m)\alpha(T))$.

Theorem

The time complexity of PathSelection is $O(m\alpha(T))$.

Theorem

the time-complexity of Two-step-LTT is $O((n \cdot \log n + m)\alpha(T))$.

Finding
Time-
dependent
Shortest
Paths over
Large Graphs

Alberto
Presta

Introduction
to the
problem

Existing
solutions

New Dijkstra
Based
Algorithm

Time
complexity

Non-FIFO
Graphs

Solution for Non-FIFO Graphs

How to find an optimal LTT over a non-FIFO graphs?

- 1 We can transform a non-FIFO graphs G'_T into a FIFO graph G_T , where both V and E remain unchanged.
- 2 Find optimal path p^* found in G_T can be converted into a optimal path p'^* for G'_T , by inserting some waiting time on each node.

Finding
Time-
dependent
Shortest
Paths over
Large Graphs

Alberto
Presta

Introduction
to the
problem

Existing
solutions

New Dijkstra
Based
Algorithm

Time
complexity

Non-FIFO
Graphs

solution for non-FIFO Graphs

IDEA

For each $w'_{i,j}(t)$ in the non-FIFO graph, we define $w_{i,j}(t)$ to construct a FIFO graph:

$$w_{i,j} = \Delta_{i,j}(t) + w'_{i,j}(t + \Delta_{i,j}(t))$$

$\Delta_{i,j}(t)$ is the optimal waiting time to traverse edge (v_i, v_j) , if arriving at v_i at time t .

Then in the FIFO optimal path p^* we add the term $\Delta_{i,j}(t)$ at node v_i , for $1 \leq i \leq k - 1$, where t is the arrival time at node v_i along path p^* in G_T for starting time t^* .

Finding
Time-
dependent
Shortest
Paths over
Large Graphs

Alberto
Presta

Introduction
to the
problem

Existing
solutions

New Dijkstra
Based
Algorithm

Time
complexity

Non-FIFO
Graphs