t-CSA: A fast and flexible CSA Implementation

TSRK Prasad*, Kartik Sathyanarayanan†, Sukriti Tiwari‡, Neena Goveas§ and Bharat Deshpande¶
Department of Computer Science & Information Systems
BITS Pilani - K K Birla Goa Campus, Goa, India - 403726
{*tsrkp, †f2013037, ‡f2013086, §neena, ¶bmd} @goa.bits-pilani.ac.in

***Abstract*—Interactive journey planning on public transport networks is a challenge. We humans cope with the journey planning challenge by intuitively searching timetables for patterns. Connection scan algorithm (CSA) is a promising algorithm to automate interactive journey planning by searching timetables for best possible journey plan. We propose t-CSA algorithm as an extension of CSA. Our contribution is two fold; Our first contribution is the inclusion of direct connection table (DCTable) in CSA to accelerate the discovery of frequent direct connections. Our second contribution is the inclusion of all known multi-modal connections between nearby stations in journey planning by using other means table (OMTable). OMTable permits customization of journey plan as per the transport facilities available to a user. Our algorithm has an average response time of 0.7ms when run on timetables of Indian public transport networks which includes both Indian railways and bus networks. We find that our algorithm makes interactive journey planning applications feasible.**

I. Introduction

In India, public transport networks are the most preferred means of transport. Ideally commuters would like to plan their journeys well in advance. This journey planning requirement from commuters gives rise to the theoretical problem of searching for a shortest path equivalent on timetables of public transport networks. Earlier, researchers in this field used graph models to represent timetables [1], [2]. Recent efforts explore more compact non-graph representations for timetables [3]–[6]. Connection scan algorithm (CSA) is one such non-graph based approach that searches timetables in linear time [7].

We propose an extended CSA which incorporates two table lookups. The first table lookup accelerates queries for direct connections. Search queries for stations on the same route also get benefit out of the proposed direct connection table (*DCTable*) lookup. The second table lookup connects nearby stations using footpath or any other known means of transport (for example, cabs); this table is named *other means table (OMTable)*. DCTable reduces the average response time of the queries, where as OMTable adds the multi-modal private transport facilities to known public transport facilities.

The rest of the paper is organized as follows. Sec II provides summary of the literature. Sec III defines key terms used in the rest of the paper. Sec IV details basic CSA (adopted from [7]). Sec V explains the two proposed extensions to CSA and presents an improved algorithm (named t-CSA). Sec VI describes the data sets from Indian public transport networks and shows the performance of t-CSA on these data sets. Sec VII contains conclusions.

II. Related Work

Algorithm researchers have made noteworthy contributions to improve real-time transit query performance on public transport networks. Initial research efforts focused on adoption of best algorithms developed for routing on road networks to the public transport networks [2], [8]. Over the years, researchers have discovered significant differences between the two problems, namely routing on road networks *vs* best itinerary on public transport networks. Bast et al. provides an overview of these differences and a summary of query speed-up techniques for public transport (timetable) networks [9]. Bast et al. also provide a detailed survey of algorithms used on road networks and public transport networks [2].

Over the years, researchers converged onto time-expanded (TE) graphs and time-dependent (TD) graphs as two appropriate graph-based models for representing public transport networks (also known as timetable networks) [1], [10]. In TE graphs, each arrival and departure event is represented as a vertex and the travel time as a weighted edge [11]. TE graph model was originally proposed by Schultz et al. and then extended by Müller-Hannemann et al. [11], [12]. The widely used General Transit Feed Specification (GTFS) is a TE graph based representation of timetable [13].

Delling et al. contract the arrival and departure nodes of TE graphs completely [14]. They also parameterize the contraction of transfer nodes. Bast et al. modify the edges of TE graph model to represent periodic trips. The resulting transfer patterns algorithm requires costly preprocessing but provides query responses in a few milliseconds [15].

Orda et al. propose time dependent (TD) graph model and give shortest path algorithm [16], [17]. Brodal et al. use TD graphs for executing fast itinerary search queries [18]. In basic TD graphs model, vertices represent stations and edges represent trips, routes. In expanded TD graph model (TD+), even routes have station-specific vertices; such expanded TD models have been referred to as train route graphs [1]. Geisberger et al. proposes station graph model which is an application of contraction hierarchies preprocessing technique to TD graph [19].

Connection Scan Algorithm (CSA), RAPTOR, GBR, and FBS are the non-graph based approaches to transit routing [3]–[6]. CSA represents timetables using connections; Strasser et al. shows equivalence between running CSA and computing shortest path queries on TE-graphs. *RAPTOR*, short for Round-

based Public Transit Routing, utilizes trips and routes for itinerary computation and is equivalent to queries on TD-graphs [4], [6]. *GBR*, short for guidebook routing, utilizes network flow-based approach to generate time independent itineraries for a given timetable [5]. *FBS*, short for frequency-based search, shows a way to extract and represent frequent-trips from timetables [4].

CSA uses connection as a fundamental unit for decomposing timetables and searching these decomposed timetables. Such a clean representation allows quick update of searchable timetables for possible delays, cancellations and special trips on busy occasions. An update of timetables only requires changing effected connections of CSA. CSA has been proven to be faster than RAPTOR, TE and TD-graph based algorithms for one-to-one queries [2], [3]. Only transfer patterns is faster than CSA, but transfer patterns technique requires costly preprocessing [2], [15].

Our work adopts techniques outlined in CSA, and transfer patterns papers [3], [15]. We propose an extension to CSA using DCTable. DCTable approach is similar to direct connections of transfer patterns [15]. Such a direct connection table approach has given rise to transit node routing (TNR) algorithm – the best performing algorithm for road networks in terms of query response time [20]. The OMTable extends footpath table of CSA to incorporate frequent trips. We include private transport networks in OMTable; Consequently, connection array gets augmented with private transport networks. Similar approaches such as GBR and FBS affirm the need to concisely represent frequent connections of a timetable. Our other means table (OMTable) can incorporate private transit facilities available to users, thus bridging the gap between public and private transport networks.

III. Problem Statement

Definitions

Public transport networks contain a set of stations (*S*) where vehicles stop to pickup or drop-off passengers. These vehicles cover a well-defined sequence of stations; The arrival and departure times of a vehicle at a station is well known. One instance of a vehicle starts at one station, goes through a well-defined sequence of stations and stops at a station. The vehicle arrives at each station at arrival time (t_{arr}) and departs at the departure time (t_{dep}). The sequence of stations including the first and last stations along with their respective arrival and departure times is defined as a *trip*. A mathematical formulation of a trip is shown in Figure 1.

The arrival and departure times at a station S_i are indicated as (t_{arr_i}, t_{dep_i}). The arrival time at the starting station and the departure time at the ending station are undefined. The time difference $(t_{dep_i} - t_{arr_i})$ is the waiting time at i^{th}-station [3].

In public transport networks vehicles go on a trip, return to the starting station (through another trip) and repeat such a loop many times during one time period (a day / a week / a fortnight); we call such trips as *loop trips*. In such cases, we consider the onward and return trips as two distinct trips.

	S_1	S_2	...	S_n	VehID
t_1	(–,t_{dep_1})	(t_{arr_2},t_{dep_2})	...	(t_{arr_n},–)	ID_1
	Connection (C)				
t_2	(t_{arr_1},–)	(t_{arr_2},t_{dep_2})	...	(–,t_{dep_n})	ID_1

Fig. 1. Sample trips t_1 and t_2 undertaken by vehicle identified as ID_1. The trips t_1 and t_2 together form a loop trip.

An interesting variation on loop trips is a *circular trip* where a vehicle starts at station S_1, goes through a series of stations S_i and terminates the trip at S_1 again. For circular trips, the following timing constraint holds.

$\forall i, j \in [2, n]$ and $j > i$,

$$t_{dep_1} < t_{arr_i} < t_{dep_i} < t_{arr_j} < t_{dep_j} < t_{arr_1}$$

Two typical events can happen in public transport system. One, a vehicle makes multiple loop trips covering same stations with each trip having different arrival and departure times; Two, between busy stations, multiple vehicles cover the same sequence of stations with each vehicle undertaking multiple trips. *Route* (r) is a set of all trips that cover the exact same sequence of stations and satisfy first-in-first-out (FIFO) property [3].

$$r = \{t_1, t_2, \ldots, t_m\}$$

FIFO property guarantees strict time-ordering of all trips belonging to a route. A common sense way of saying this is, "*on a route, waiting never pays off.*"

By definition, all routes of a public transport system are mutually disjoint, i.e., no two routes contain a common trip. Since trips belonging to different routes can have overlapping stations, routes may have a few common stations.

We can define *timetable* for a public transport system as a tuple (π, S, T, R, F). π denotes time period of a timetable; often this time period is a day or a week. *S* is the set of all stations, *T* is the set of all trips, *R* is the set of all routes and *F* is a footpath table containing walking times between nearby stations [6].

A *connection* is a fundamental unit of timetable [7]. A connection is defined as

$$C = (S_{dep}, t_{dep}, S_{arr}, t_{arr}, tr)$$

where,

tr = unique trip identifier

other variables as defined before

We can rephrase trip as a sequence of connections. For a trip covering *n* stations, we will have *(n-1)* connections.

$$t = C_1 C_2 \ldots C_{n-1}$$
$$\text{where } C_i = (S_i, t_{dep_i}, S_{i+1}, t_{arr_{(i+1)}}, tr)$$

Connection Array

In the previous section, we have shown a way to transform timetables into elementary connections. We know that each connection has a start time. Based on the start times, we can

Query: $S_1@00:10 \rightarrow S_5$
Itinerary:
1) Wait for 5 minutes at S_1.
2) Take $t_2 = C_4C_5C_6$ from S_1 to S_4 and reach S_4 by $02:10$.
3) Wait for 30 minutes at S_4. (transfer from t_2 to t_3)
4) Take C_8 (part of t_3) and reach S_5 at $03:15$.

Fig. 2. A sample query and the generated itinerary.

sort the connections in ascending order and place the resulting connection sequence in an array. This array is called *connection array*. In a connection array, let i and j be the positions of two connections, then $t_{dep}(C_i) \leq t_{dep}(C_j)\ \forall\ i < j$.

Timetables of public transport systems tend to be periodic; Often, the time period is a day, a week or a fortnight. We can easily incorporate the periodicity by making the following adjustments to the connection array.

1) Connection array becomes a circular list.
   i.e., after last connection in the array, we can pursue the first connection in the connection array to determine the travel itinerary.
2) Arrival and departure times of each connection need to be adjusted based on the number of passes through the connection array.

$$t'_{arr} = i \times 1440 + t_{arr}$$

$$t'_{dep} = i \times 1440 + t_{dep}$$

where, $i = 0, 1, 2, \ldots, k$ is the number of passes made through the connection array. For practical purposes, we can limit k to 3 days even for country-wide timetables.

Feasible Itinerary

An itinerary (I) is made up of a sequence of connections, $I = C_1C_2 \ldots C_h$ with consecutive connections satisfying the condition $S_{arr}(C_i)$ is equal to $S_{dep}(C_{i+1})$. Each of the consecutive connections of an itinerary may belong to the same trip. If two consecutive connections are not part of the same trip, a *transfer* is required at the corresponding station.

A representative query from user and the itinerary generated using connection array are shown in Figure 2. The itinerary shown in Figure 2 is feasible because the commuter can catch C_8 after C_6. If C_8 were to depart anytime before $02:10$, the itinerary would have become in-feasible.

Transfer Times

In simple connection model, *transfer time* – the time required for alighting a connection of one trip and boarding connection of another trip (*trip transfer*) – has been assumed to be zero. But in reality, transfer times are non-zero. The transfer time is often dependent on connections, station at which transfer occurs and the time of day. The transfer time is zero for a commuter not undertaking the trip transfer at a station. We can represent transfer time as

$$\begin{aligned} t_{tr}(C_i, C_j) &= 0 && \text{if } C_i, C_j \in \text{ same trip} \\ &= f(C_i, C_j) && \text{else} \end{aligned}$$

We can approximate $f(C_i, C_j)$ as

$$f_1(C_i, C_j) \approx f_2(S_{arr}(C_i), t) \approx f_3(S_{arr}(C_i))$$
where, $f_1 : C_i \times C_j \rightarrow T, f_2 : S \times T \rightarrow T, f_3 : S \rightarrow T$

Function f_3 is sufficient to model transfer times in most practical scenarios. In the rest of the paper, we will use function f_3 to model transfer times.

With non-zero transfer times, a transfer C_i to C_j is feasible if only if

$$\begin{aligned} S_{arr}(C_i) &= S_{dep}(C_j) \text{ and} \\ t_{dep}(C_j) &\geq t_{arr}(C_i) + t_{tr}(C_i, C_j) \\ &\approx t_{arr}(C_i) + t_{tr}(S_{arr}(C_i)) \end{aligned}$$

IV. Connection Scan Algorithm (CSA)

In this section, we give an outline of algorithm that operate on a connection array to produce travel itinerary. This algorithm have been called a connection scan algorithm (CSA) [7]. A basic version of CSA is given in Algorithm 1. Algorithm 1 requires connection array (CA), starting station (S_1), start time (t_s), ending station (S_2), and station array (SA) as inputs. If possible, the Algorithm 1 produces a feasible itinerary (I). Algorithms 1 and 2 use a few temporary variables during their operation. These temporary variables are as follows: IC is an array to hold incoming connections for each station; d is an array to hold the earliest arrival time for each station; C_i is a temporary connection variable to hold the connection under consideration.

Algorithm 1 Connection Scan Algorithm (CSA)

Input: CA, S_1, t_s, S_2, SA
Output: I

```
1:                                   ▷ Temporary variables: IC, d, C_i
2: for all s_i ∈ SA do
3:     IC[s_i] ← null
4:     d[s_i] ← ∞
5: end for
6:                                   ▷ Compute itinerary
7: for all C_i ∈ CA do
8:     if t_dep(C_i) > d[S_dep(C_i)] and
         t_arr(C_i) < d[S_arr(C_i)] then
9:         d[S_arr(C_i)] ← t_arr(C_i)
10:        IC[S_arr(C_i)] ← C_i
11:    end if
12: end for
13: I ←ITINERARY(IC)
14: return I
```

Algorithm 1 operates as follows. The arrays IC and d are initialized. The for loop given in the lines 7–12 iterates through each connection and tries to settle the arrival station of a connection. If the relaxation function (d) of arrival station

Algorithm 2 Fetch itinerary through backtracking (ITINERARY)

Input: IC
Output: I

1: ▷ Temporary variable: C_i
2: $I \leftarrow null$
3: $C_i \leftarrow IC[S_2]$
4: **while** C_i **do** ▷ valid connection
5: I.PREPEND(C) ▷ add at the beginning
6: $C_i \leftarrow IC[S_{dep}(C_i)]$
7: **end while**
8: **return** I

can be modified, the corresponding entries are updated in the incoming connections array (IC). After processing all the connections, Algorithm 1 uses Algorithm 2 to extract an optimal itinerary from IC array.

V. EXTENSIONS TO CSA

Fast Lookups

CSA gives fast query response (<2 ms) even on country-wide multi-mode timetables. We reduce the queries times even further by storing answers to queries of direct connections that can be answered directly from timetable. Such direct connections are the norm in long-distance bus networks. For these scenarios, direct lookup is optimal.

We copy all such direct connections cases from timetables and place them in *DCTable*, short for direct connections table. Entries in the DCTable are stored as as connection tuples. Any incoming query is first checked against a match in DCTable; Upon a match in the DCTable, all possible direct connections/itineraries between two stations specified in the query are returned. If an answer is not found, then CSA is run. DCTable's role can be augmented as a cache for popular queries.

Footpaths and Other Means

In case two public transport stations are nearby, the passengers can alight a connection at one station, walk to the nearby station on footpath and board a new connection. For all nearby stations of the public transport network, we can maintain a table of walk times. A sample set of entries are shown in Table I. f_{start} denotes starting station, f_{end} denotes ending station and t_w indicates walk time.

TABLE I
FOOTPATHS TABLE.

f_{start}	f_{end}	t_w
S_i	S_j	t_{w1}
S_j	S_k	t_{w2}
S_i	S_k	$t_{w3} \leq t_{w1} + t_{w2}$

All entries are transitively closed which enables a direct lookup of all possible walks between nearby stations. If the entries were not transitively closed, we would have been forced to run shortest path algorithm on the table for each query, an expensive proposition.

No single public transport network provides best any-station to any-station connectivity at all times. Our footpath table is a more effective way of reaching nearby stations. We can segregate all known means of transport into public transport system with timetable and everything else (*other means*). Popular other means could be footpaths, autos, taxis, car pools or a helpful drop by a friend. The transport by other means can be summarized into a table named *other means table* (OMTable). Entries in OMTable will be a tuple of the form (S_i,S_j,travel time, details). The format of OMTable is given in Table II.

TABLE II
OTHER MEANS TABLE.

S_{start}	S_{end}	t_{omt}	Details
S_i	S_j	10 *min*	$walk$
S_i	S_k	30 *min*	cab

Connection array is a very efficient representation for aperiodic connections. Most timetables have high-frequency trips between popular stations, say a bus every 5 minutes between S_i and S_j. One such timetable entry leads to 288 connections in the connection array. Such routes have been called as guidebook routes or transfer patterns [4], [5], [15]. We can efficiently represent such cases in OMTable.

t-CSA Algorithm

CSA outlined in Algorithm 1 assumes the zero transfer times. It also assumes that an itinerary can be found on or before we reach the end of connection array (CA). End of CA is also the stop condition for the loop given in lines 7 – 12 of Algorithm 1. We can redesign the basic CSA to overcome these three limitations.

An enhanced version of CSA (named, *t*-CSA) that incorporates modifications is shown in Algorithm 3. Algorithm 3 builds on Algorithm 1. Over and above the inputs required by Algorithm 1, Algorithm 3 requires four additional inputs. These additional inputs are: the maximum number of travel days (n), station transfer times array (TT), direct connections table ($DCTable$) and other means table ($OMTable$). C_j, C_{last}, itr, n_ca, t_{dep},t_{arr} and $settled$ are the newly introduced temporary variables. C_j denotes the first connection in the connection array with a starting time $t_{dep}(C_j) > t_s$. C_{last} holds the last connection in the connection scan array. itr specifies the number of times the connection array needs to be scanned for creating a feasible itinerary. n_ca, t_{arr} and t_{dep} are the temporary variables used to enhance readability. $settled$ indicates a change in earliest arrival estimate at a station.

The first major addition to Algorithm 3 over Algorithm 1 is the $DCTable$ code given in lines 2 – 5. $DCTable$ is consulted first; only in case the query fails on $DCTable$, will the CSA be run. The function call on line 12 returns C_j. In abstract terms, CONNSEARCH performs a binary search on connection array to find the connection C_j that satisfies the condition.

$$min \;\; t_{dep}(C_j) - t_s$$

With Constraints:

$$t_{dep}(C_j) - t_s > 0$$
$$C_j \in CA$$

The code block in lines 13 – 46 iterates through connection array required number of times, examines all eligible connections and tries to settle stations whenever relaxation is possible. The code in lines 20 – 24 considers realistic station transfer times. Another feature addition is *settling* the nearby stations of a recently *settled* station by using the entries of other means table ($OMTable$). The corresponding code is in lines 33 – 41 of Algorithm 3. Early termination condition on line 42 helps us discard all connections whose departure times are greater than the current estimate of the earliest arrival time at the destination station.

VI. Experiments

Data Sets

We utilize transport data available from Indian public transport networks. The characteristics of the networks selected are summarized in Figure 3. The #Stations and #Connections columns indicate the number of stations and connections respectively. The train data has been obtained from Indian Railways timetable [21]. This railway timetable network is the largest Indian public transport network utilized by us. The flight data has been generated from the domestic flight schedules released by Directorate General of Civil Aviation (DGCA), India [22]. The bus data has been obtained from numerous public and private bus transport operators (for one such example, namely GSRTC, see [23]).

Query Times

Algorithms 1, 2 and 3 have been coded in Python2.7 programming language and tested on a commodity laptop. The laptop has 2-core 64-bit Intel i5-2430M processor with 256KB of L2 cache and 3MB of L3 cache; the operating system is 64-bit Ubuntu. We run queries on complete Indian network (has 2264 stations and 59555 connections). We randomly generate 1000 queries with randomly chosen source, destination pairs and a random start time. Our Python2.7 implementation of Algorithm 1 gives an average query response time 0.5ms. Algorithm 3 gives an average query response time of 0.7ms. The direct connection queries on $DCTable$ execute very fast (worst case time: $1\mu s$; average time: $0.5\mu s$). These numbers compare favorably with average response time of 1.8ms for London metropolitan data reported in [3].

VII. Conclusions

We use connection as a building block to model timetables of public transport networks. We propose t-CSA which adds direct connection table ($DCTable$) and other means table ($OMTable$) to connection scan algorithm. $DCTable$ stores itinerary details of stations that are directly connected through a trip. $OMTable$ stores details of alternative (footpath and miscellaneous modes of transport) connections between

Algorithm 3 An enhanced CSA (t-CSA)

Input: CA, S_1, t_s, S_2, SA, n, TT, $DCTable$, $OMTable$
Output: I

1: ▷ Temporary variables: IC, d, C_i, C_j, C_{last}, itr, n_ca, t_{dep}, t_{arr}, $settled$, C_{temp}
2: ▷ try direct connections first
3: **if** $I \leftarrow$ SearchDCT($DCTable, S_1, t_s, S_2$) **then**
4: **return** I
5: **end if**
6: ▷ Compute number of iterations
7: $C_{last} \leftarrow CA[CA.length - 1]$
8: $n\_ca \leftarrow t_{dep}(C_{last})$ **mod** 1440
9: $itr \leftarrow n$ **mod** n_ca
10: Initialize IC and d as per lines 2-5 of Algorithm-1
11: ▷ Pick first connection with $t_{dep}(C) > t_s$
12: $C_j \leftarrow$ ConnSearch(CA, t_s)
13: `search:`
14: **for all** $day \leftarrow 1, 2, \ldots, (itr + 1)$ **do**
15: **if** $day == 2$ **then**
16: $C_j \leftarrow C_0$
17: **end if**
18: **for all** $C_i \in CA \ni C_i \geqslant C_j$ **do**
19: $settled \leftarrow false$
20: **if** $tr(C_i) == tr(IC[S_{dep}(C_i)])$ **then**
21: $t_{tr} \leftarrow 0$
22: **else**
23: $t_{tr} \leftarrow TT[S_{dep}(C_i)]$
24: **end if**
25: $t_{dep} \leftarrow t_{dep}(C_i) + day \times 1440$
26: $t_{arr} \leftarrow t_{arr}(C_i) + day \times 1440$
27: **if** $t_{dep} > d[S_{dep}(C_i)] + t_{tr}$ **and** $t_{arr} < d[S_{arr}(C_i)]$ **then**
28: $settled \leftarrow true$
29: $d[S_{arr}(C_i)] \leftarrow t_{arr}(C_i)$
30: $IC[S_{arr}(C_i)] \leftarrow C_i$
31: **end if**
32: ▷ use OMTable to settle nearby stations
33: **if** $settled$ **then**
34: **for all** $S_{arr}(C_i), S_i \in OMTable$ **do**
35: **if** $d[S_j] > d[S_{arr}(C_i)] + t_{omt}[S_{arr}(C_i), S_j]$ **then**
36: $d[S_j] \leftarrow d[S_{arr}(C_i)] + t_{omt}[S_{arr}(C_i), S_j]$
37: $C_{temp} \leftarrow (S_{arr}(C_i), t_{arr}(C_i), S_j, t_{arr}(C_i) + t_{omt}[S_{arr}(C_i), S_j], om)$
38: $IC[S_j] \leftarrow C_{temp}$
39: **end if**
40: **end for**
41: **end if**
42: **if** $t_{dep} > d[S_2]$ **then**
43: **break** `search`
44: **end if**
45: **end for**
46: **end for**
47: $I \leftarrow$ Itinerary(IC)
48: **return** I

Name of Network	Mode of Transport	#Stations	#Connections	CA size
Indian Railways (IR)	train	2136	44876	0.36 MB
Indian Flights (IFlights)	flight	80	2501	0.02 MB
Long Distance Buses (LDB)	bus	147	12219	0.1 MB
Complete Network (India)	multi-mode	2264	59555	0.52 MB

Fig. 3. Summary of timetables obtained from Indian public transport networks.

nearby stations. We find that t-CSA extends the capabilities of CSA significantly.

We also demonstrate the implementation of extended CSA on timetables of Indian public transport networks. Our t-CSA eliminates the need to run CSA on direct connection queries. Since direct connection queries are the bulk of queries on timetables, t-CSA significantly accelerates interactive journey planner applications.

ACKNOWLEDGMENT

The authors would like to thank Atul Agrawal, Shaik Asifullah, Ishaan Bansal, Kartika Nair and Sushrut Sivaramakrishnan for help in data collection.

We thank Indian Railways, Director General of Civil Aviation, and GSRTC for making their timetables available in the public domain.

REFERENCES

[1] E. Pyrga, F. Schulz, D. Wagner, and C. Zaroliagis, "Efficient models for timetable information in public transportation systems," *ACM Journal of Experimental Algorithmics (JEA)*, vol. 12, pp. 2–4, 2008.

[2] H. Bast, D. Delling, A. Goldberg, M. Müller-Hannemann, T. Pajor, P. Sanders, D. Wagner, and R. F. Werneck, "Route planning in transportation networks," *arXiv preprint arXiv:1504.05140*, 2015.

[3] J. Dibbelt, T. Pajor, B. Strasser, and D. Wagner, "Intriguingly simple and fast transit routing," in *Proceedings of the 12th International Symposium on Experimental Algorithms (SEA'13)*, Lecture Notes in Computer Science, vol. 7933, pp. 43–54, Springer Berlin Heidelberg, 2013.

[4] H. Bast and S. Storandt, "Frequency-based search for public transit," in *Proceedings of the 22nd ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*, pp. 13–22, ACM, 2014.

[5] H. Bast and S. Storandt, "Flow-based guidebook routing," in *2014 Proceedings of the 16th Workshop on Algorithm Engineering and Experiments (ALENEX'14)*, pp. 155–165, SIAM, 2014.

[6] D. Delling, T. Pajor, and R. F. Werneck, "Round-based public transit routing," *Transportation Science*, vol. 49, no. 3, pp. 591–604, 2015.

[7] B. Strasser, "Delay-robust stochastic routing in timetable networks," Master's thesis, Department of Informatics, Institute of Theoretical Informatics, KIT, Karlsruhe, Germany, July 2012.

[8] R. Geisberger, P. Sanders, D. Schultes, and D. Delling, "Contraction hierarchies: Faster and simpler hierarchical routing in road networks," in *7th International Workshop on Experimental Algorithms (WEA'08)*, Lecture Notes in Computer Science, vol. 5038, pp. 319–333, Springer Berlin Heidelberg, 2008.

[9] H. Bast, "Car or public transport – two worlds," in *Efficient Algorithms*, Lecture Notes in Computer Science, vol. 5760, pp. 355–367, Springer Berlin Heidelberg, 2009.

[10] M. Müller-Hannemann, F. Schulz, D. Wagner, and C. Zaroliagis, "Timetable information: Models and algorithms," in *Algorithmic Methods for Railway Optimization*, Lecture Notes in Computer Science, vol. 4359, pp. 67–90, Springer Berlin Heidelberg, 2007.

[11] F. Schulz, D. Wagner, and K. Weihe, "Dijkstra's algorithm on-line: An empirical case study from public railroad transport," *ACM Journal of Experimental Algorithmics*, vol. 5, Dec. 2000.

[12] M. Müller-Hannemann and K. Weihe, "Pareto shortest paths is often feasible in practice," in *5th International Workshop on Algorithm Engineering*, Lecture Notes in Computer Science, vol. 2141, pp. 185–197, Springer Berlin Heidelberg, 2001.

[13] "General Transit Feed Specification (GTFS)," [Online]. Available: https://developers.google.com/transit/?hl=en, accessed: 2015-10-30.

[14] D. Delling, T. Pajor, and D. Wagner, "Engineering time-expanded graphs for faster timetable information," in *Robust and Online Large-Scale Optimization*, Lecture Notes in Computer Science, vol. 5868, pp. 182–206, Springer Berlin Heidelberg, 2009.

[15] H. Bast, E. Carlsson, A. Eigenwillig, R. Geisberger, C. Harrelson, V. Raychev, and F. Viger, "Fast routing in very large public transportation networks using transfer patterns," in *Proceedings of the 18th Annual European Symposium on Algorithms (ESA'10)*, Lecture Notes in Computer Science, vol. 6346, pp. 290–301, Springer Berlin Heidelberg, 2010.

[16] A. Orda and R. Rom, "Shortest-path and minimum-delay algorithms in networks with time-dependent edge-length," *Journal of the ACM (JACM)*, vol. 37, no. 3, pp. 607–625, Jul. 1990.

[17] A. Orda and R. Rom, "Minimum weight paths in time-dependent networks," *Networks*, vol. 21, no. 3, pp. 295–319, 1991.

[18] G. S. Brodal and R. Jacob, "Time-dependent networks as models to achieve fast exact time-table queries," *Electronic Notes in Theoretical Computer Science*, vol. 92, pp. 3–15, 2004.

[19] R. Geisberger, "Contraction of timetable networks with realistic transfers," in *Proceedings of the 9th International Symposium on Experimental Algorithms (SEA10)*, Lecture Notes in Computer Science, vol. 6049, pp. 71–82, Springer Berlin Heidelberg, 2010.

[20] H. Bast, S. Funke, and D. Matijevic, "Ultrafast shortest-path queries via transit nodes," vol. 74, pp. 175–192, American Mathematical Society Providence, RI, 2009.

[21] Indian Railways, "Trains/fare/accommodation availability between important stations," [Online]. Available: http://www.indianrail.gov.in/between_Imp_Stations.html, accessed: 2015-10-30.

[22] Directorate General of Civil Aviation, "Domestic flight schedule," [Online]. Available: http://dgca.nic.in/dom_flt_schedule/flt_index.htm, accessed: 2015-10-30.

[23] GSRTC, "Search & book tickets," [Online]. Available: http://www.gsrtc.in/site/, accessed: 2015-10-30.