Supporting Material of "CityChrone: an interactive platform for transport analysis and planning in urban systems".

Indaco Biazzo[1]

Politecnico di Torino, Corso Duca degli Abruzzi 24, 10129, Torino, Italy,
indaco.biazzo@polito.it,
WWW home page: http://indacobiazzo.me

1 Supporting Material

1.1 Connections Scan Algorithm (CSA)

A timetable defines a public transit network. A timetable consists of a set of stops, connections, and (possibly) footpaths. The stops are the points on maps where a traveler can enter or exit from a vehicle. A connection c represents a vehicle departing from the stop $s_{dep}(c)$ at time $t_{dep}(c)$ and arriving at the stop $s_{arr}(c)$ at time $t_{arr}(c)$ without intermediate halt. The connection set is an array of four tuples representing the movements of the vehicles. A footpath consists of the time walking distance between two stops. Footpaths allow transfers between stops. Give the timetable it is possible to compute the time needed to reach all the other stops given a starting time t_0 at the stop s_{start}. We label each stop s_i with its arrival time $\tau[s_i]$ and we set them all at starting to infinity $\tau[s_i] = \infty$ except for the starting stop $\tau[s_{start}] = t_0$ that we set to its starting time. The array containing the connections is ordered by the $t_{dep}(c)$. A connection is defined as *reachable* if the time $t_{dep}(c)$ of starting stop $s_{dep}(c)$ of the connection c is equal or after the time $\tau[s_{dep}(c)]$ of the stop $s_{dep}(c)$ that has been reached.
The Connections Scan Algorithm (CSA) scans all the connections $\{c\}$, testing if c is *reachable*. If c can be reached and if the arrival time $\tau[s_{arr}(c)]$ is larger of the arrival time $t_{arr}(c)$ of the connection, the connection is relaxed, meaning that the $\tau[s_{arr}(c)]$ is update to the earlier arrival time $t_{arr}(c)$. After the entire array of connections is scanned the labels tau contain the earliest arrival time for each stop, starting from s_{start}. The pseudo-code of the CSA algorithm is shown in Fig.1
In the above description of the algorithm, we did not consider footpaths. In order to include footpaths between stops, each time the algorithm relaxes a connection, it checks all the outgoing footpath $f[s_{arr}]$ of $s_{arr}(c)$ and updates the time of the neighbors accordingly. The algorithm requires, to ensure correctness that the footpaths have to be closed under transitivity. This means that if there is a footpath from stop s_a to stop s_b and a footpath from the stop s_b to the stop s_c there must be a footpath between s_a and s_c. So for every connected component of stops connected by footpaths, we need all the footpaths connecting them. The

number of them grows quadratically with the number of stops that belong to the connected component. For our purposes, we want realistic footpaths between stops, allowing walking times between stops up to 15 minutes. In the end, all the stops of a city belong to the same connected component. Usually, the number of stops is of order 10^4, then the number of footpaths to consider are 10^8 that is too large for any practical use because each time a connection relax, in the CSA algorithm, all the outgoing footpaths have to be checked. The next subsection describes a new version of the algorithm that can solve the earliest arrival time problem without imposing closeness by transitivity of footpaths between stops.

for all stops s **do**
  $\tau[s] \leftarrow \infty$
end for
$\tau[s_{start}] \leftarrow t_0$
for all connections c increasing by $t_{dep}(c)$ **do**
  if $\tau[s_{dep}(c)] \leq t_{dep}(c)$ **then**
    if $\tau[s_{arr}(c)] > t_{arr}(c)$ **then**
      $\tau[s_{arr}(c)] \leftarrow t_{arr}(c)$
      for all footpaths f from $s_{arr}(c)$ **do**
        $\tau[f_{arr}] \leftarrow \min\{\tau[f_{arr}], \tau[s_{arr}(c)] + f_{dur}\}$
      end for
    end if
  end if
end for

Fig. 1: Connection Scan algorithm.

1.2 Intransitive Connections Scan algorithm (ICSA)

The new algorithm we propose can correctly solve the earliest arrival time problem on public transport networks considering footpaths between stops without imposing the closeness under transitivity. For each stop s_i in the database, we consider a subset of stops $\{s_j\}$ reachable by footpaths without restriction. The journeys computed by the CSA algorithm could be incorrect. The error is caused by the update of the arrival time of stops throw the footpath from neighbor stops. Consider the case, we have just relaxed a connection c, i.e. the arrival time $\tau[s_i(c)]$ of the arrival stop $s_i(c)$ is updated, see Fig.2. Now we check the stops $\{s_j\}$ reachable by footpaths from $s_i(c)$. Then the arrival time $\tau[s_j^*]$ of a neighbour stop s_j^* is updated. The s_j^* is also connected by footpaths to other stops $\{s_k\}$, see Fig.2. They are not updated because the arrival time's updating ends with the first set of neighbor stops (and they are not closed under transitivity). Now it could happen that the

The algorithm that we propose, the Intransitive Connections Scan algorithms (ICSA), overcome this problem with a minor modification of the original algorithm, maintaining the running time and its simplicity. The key is to consider

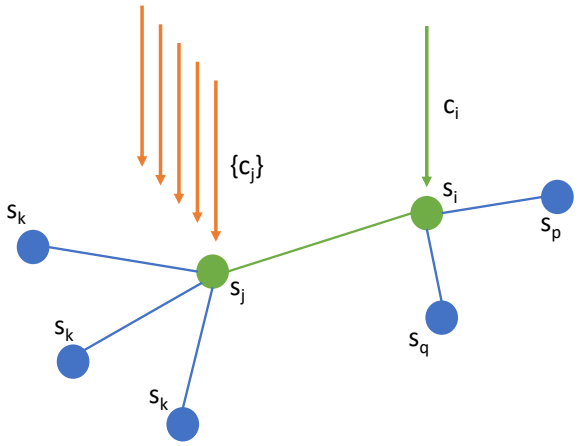

Fig. 2: The figure shows the updating process of the CSA algorithm and the possible error when the closeness by transitivity of the footpaths between stops is not preserved. First a connection c arriving at the stop s_i relaxes, i.e. update the arrival time $\tau[s_i]$ of s_i. Then the arrival time $\tau[s_j^*]$ of the neighbor stop s_j^* is update through the footpath. It is therefore possible that all the other connection $\{c_j\}$ arriving to s_j^* do not relax and the neighbour $\{s_k\}$ are never update by footpaths from s_j^* causing possible errors.

two labels τ and τ^f for the arrival time for each stops. τ represents the arrival time to the stop with the relaxation of one connection and τ^f the update of the arrival time due by footpaths. The ICSA algorithm is the same as CSA except for three modifications:

1. a connection is considered reachable if its starting time $t_{dep}(c)$ is after or equal either to arrival time of the stops $\tau[s_{dep}(c)]$ or the arrival time $\tau^f[s_{dep}(c)]$.
2. Both the CSA and ICSA update the arrival time of the stops in two ways, by direct relaxation of the connections or by footpaths. The ICSA algorithms update the arrival time $\tau[s]$ of the stop s when it is updated by connections and the arrival time $\tau^f[s]$ when footpaths update it.
3. After the complete scanning of the connections array, the arrival time taken for each stop s is the best arrival time between the two labels $tau[s]$, $tau^f[s]$.

A pseudo-code implementation scheme is shown in fig.3.

for all stops s **do**
  $\tau[s] \leftarrow \infty$
  $\tau^f[s] \leftarrow \infty$
end for
$\tau[s_{start}] \leftarrow t_0$
for all connections c increasing by $t_{dep}(c)$ **do**
  if $\tau[s_{dep}(c)] \leq t_{dep}(c)$ or $\tau^f[s_{dep}(c)] \leq t_{dep}(c)$ **then**
    if $\tau[s_{arr}(c)] > t_{arr}(c)$ **then**
      $\tau[s_{arr}(c)] \leftarrow t_{arr}(c)$
      for all footpaths f from $s_{arr}(c)$ **do**
        $\tau^f[f_{arr}] \leftarrow \min\{\tau^f[f_{arr}], \tau[s_{arr}(c)] + f_{dur}\}$
      end for
    end if
  end if
end for
for all stops s **do**
  $\tau[s] \leftarrow min(\tau[s], \tau^f[s])$
end for

Fig. 3: Intransitive Connection Scan algorithm (ICSA).

ICSA solves the problems to find the earliest arrival time to stops, given a set of connections and footpaths connecting them, without the constraint of the closeness under transitivity of footpaths. We check the performance of the CSA algorithms against the new ICSA algorithm on the quality of solutions and running times in three different cities, Helsinki, Rome, and Berlin. For each pair of stops (s_i, s_j) in the city's database, we compute the time between them, with CSA $t^{CSA}_{s_i,s_j}$ and ICSA $t^{ICSA}_{s_i,s_j}$. For each stop, we add footpaths to all the other stops reachable in 15 minutes by walk. The footpaths considered are not closed under transitivity, so the CSA algorithm gives incorrect results. The first value, visible in table 1, is the relative average difference of travel time of all journeys

$\sum s_i, s_j : i \neq j \frac{t^{CSA}_{s_i,s_j} - t^{ICSA}_{s_i,s_j}}{t^{ICSA}_{s_i,s_j}}$ starting at 7am. The difference, on average, between the two algorithms, is minimal, less than one percent for all the three cities considered. Then we consider the worst case scenario: for each stop, we consider the worst relative average travel difference with all the other stops in the city; we average this quantity between all the stops in the city. In this case, the difference between the two algorithms is high; for instance, for Helsinky, on average, for each stop, there is a journey computed by ICSA with a travel time 68% smaller than computed by CSA. The relative running time difference between CSA and CSA is very small; ICSA results to be slower than CSA, but with a difference smaller than 10%, for the cities analized, table 1.

Table 1: **Performance comparison between CSA and ICSA.**

city	travel time diff	worst travel time diff	run time diff	stops
Helsinki	0.37%	68.5%	8.21%	7633
Rome	0.04%	17.4%	6.67%	8585
Berlin	0.77%	71.6%	-0.4%	4863

The ICSA can be considered an excellent method to solve the earliest arrival time problem, when dense footpaths connections between stops are present, maintaining almost the same performance as CSA.