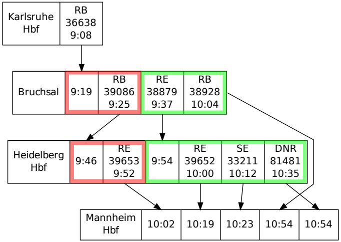
Karlsruhe Institute of Technology

Connection Scan

Julian Dibbelt, Thomas Pajor, Ben Strasser, Dorothea Wagner |

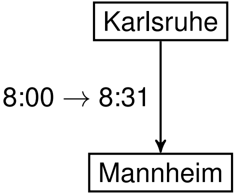
KIT - INSTITUTE OF THEORETICAL INFORMATICS - CHAIR PROF. DR. WAGNER



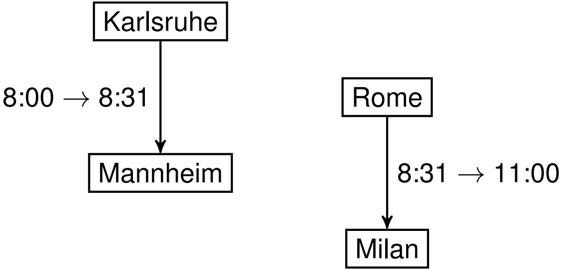
What is a Timetable?

Karlsruhe

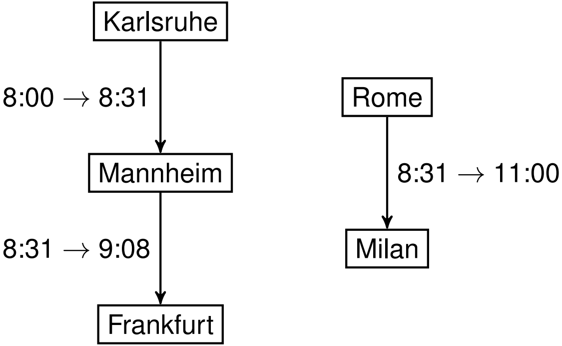
What is a Timetable?



What is a Timetable?



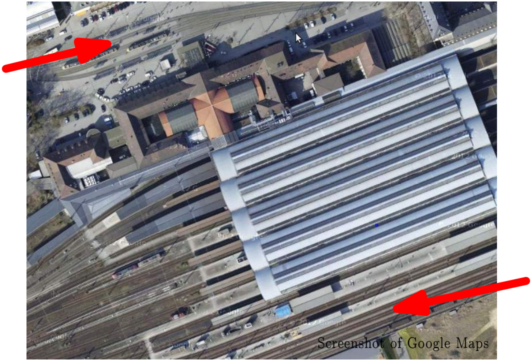
What is a Timetable?



What is a Timetable?

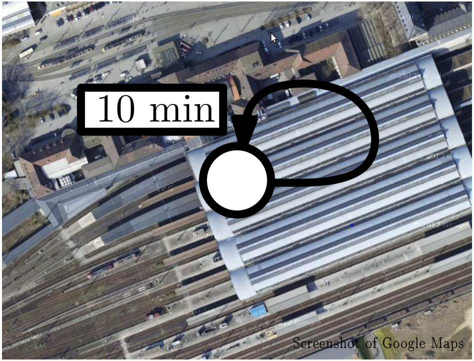
A train station

What is a Timetable?



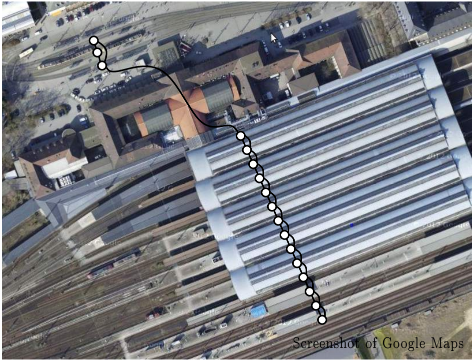
Can the user do this instantely?

What is a Timetable?



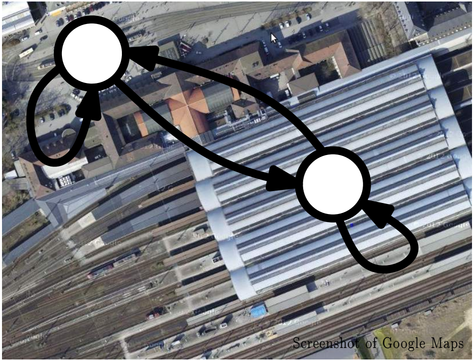
One stop with minimum change time 10 min

What is a Timetable?



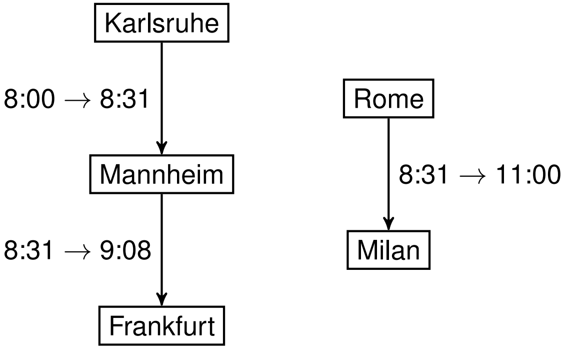
Many stops with footpaths
No minimum change times

What is a Timetable?

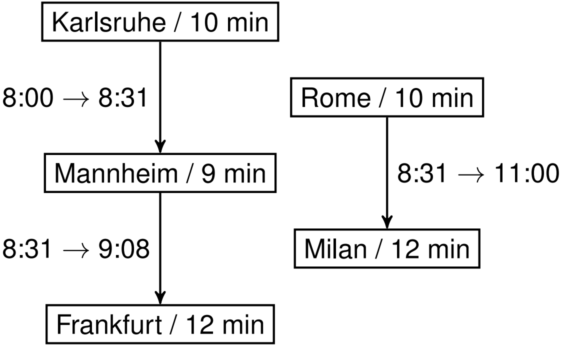


Mix change times and footpaths

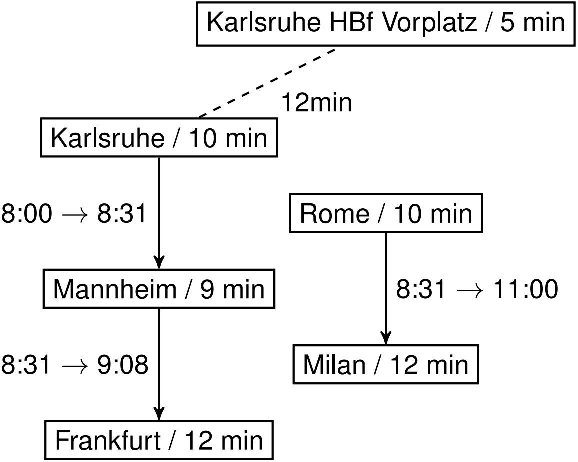
What is a Timetable?



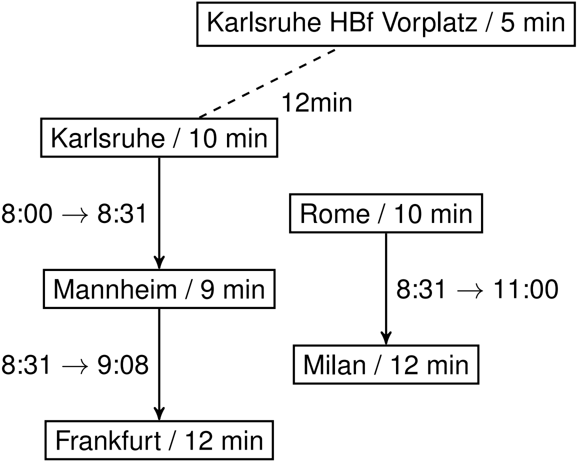
What is a Timetable?



What is a Timetable?



What is a Timetable?



For simplicity: We ignore footpaths in this lecture.

What is a Timetable?

- A timetable contains stops, connections, trips.
- A connection is a train that drives from one stop to another one without intermediate halt.
- A trip is a sequence of connections operated by the same train.
- A connection has a departure and arrival time and a departure and arrival stop and a trip ID.
- The timetable is **aperiodic**: Connections do not repeat.
- A path through a timetable is called journey.
- Switching between trains of different trips is a transfer.
- Transfers require time. This is formalized as following:
  - Stops have a minimum change time.
    Each transfer(within the stop) needs at least this amount of time.
  - Footpaths exist with a constant walking time between adjacent stops. (The footpath graph usually is highly disconnected, i.e., main station platforms are connected to subway platforms, but not neighbouring tram stops on the same line.)

Basic Connection Scan

Earliest Arrival Time Problem

Input: Ordered list of connections, source stop, source time, target stop

Output: Earliest (on time) arrival time

change time (in min)		4	10	5	7	3	
arrival time	⋯	$+\infty$	$+\infty$	$+\infty$	$+\infty$	$+\infty$	⋯

connections orderd by departure time	⋯	depstop	arrstop	deptime	arrtime	tripid	depstop	arrstop	deptime	arrtime	tripid	depstop	arrstop	deptime	arrtime	tripid	⋯

is trip reachable?	⋯	F	F	⋯

Basic Connection Scan

Earliest Arrival Time Problem

Input: Ordered list of connections, source stop, source time, target stop

Output: Earliest (on time) arrival time

change time (in min)		4	10	5	7	3
arrival time	⋯	$+\infty$	$+\infty$	$+\infty$	$+\infty$	$+\infty$	⋯

connections orderd by departure time	⋯	dep	arr	9:00	9:25	tripid	dep	arr	9:15	9:45	tripid	dep	arr	9:25	9:55	tripid	⋯

is trip reachable?	⋯	F	F	⋯

Basic Connection Scan

Karlsruhe Institute of Technology

Earliest Arrival Time Problem

Input: Ordered list of connections, source stop, source time, target stop
Output: Earliest (on time) arrival time

change time (in min)		4	10	5	7	3	
arrival time	…	$+\infty$	8:00	$+\infty$	$+\infty$	$+\infty$	…

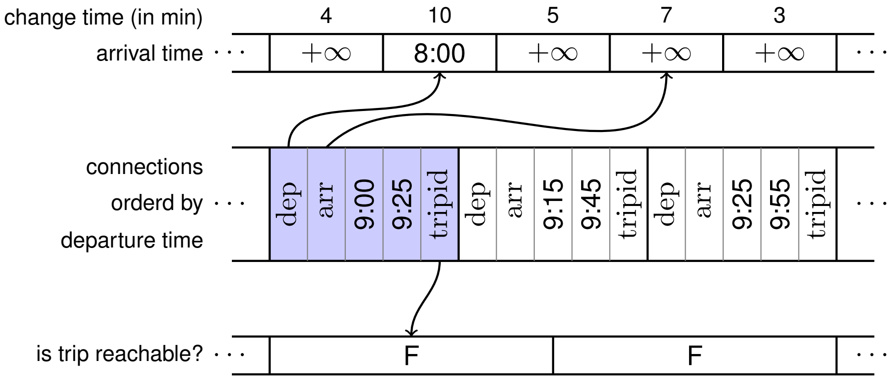
connections orderd by departure time																
…	dep	arr	9:00	9:25	tripid	dep	arr	9:15	9:45	tripid	dep	arr	9:25	9:55	tripid	…

is trip reachable?				
	…	F	F	…

Basic Connection Scan

Earliest Arrival Time Problem

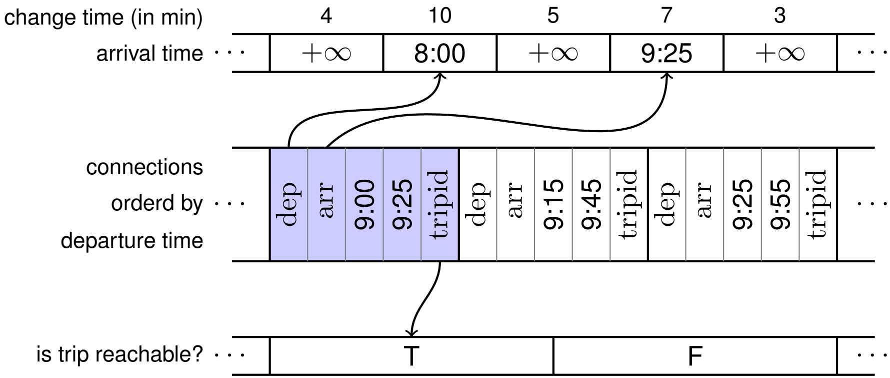
Input: Ordered list of connections, source stop, source time, target stop
Output: Earliest (on time) arrival time



Basic Connection Scan

Earliest Arrival Time Problem

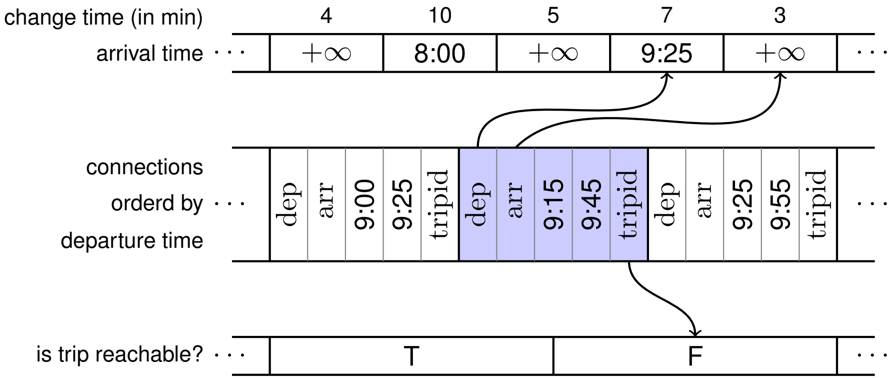
Input: Ordered list of connections, source stop, source time, target stop
Output: Earliest (on time) arrival time



Basic Connection Scan

Earliest Arrival Time Problem

Input: Ordered list of connections, source stop, source time, target stop
Output: Earliest (on time) arrival time

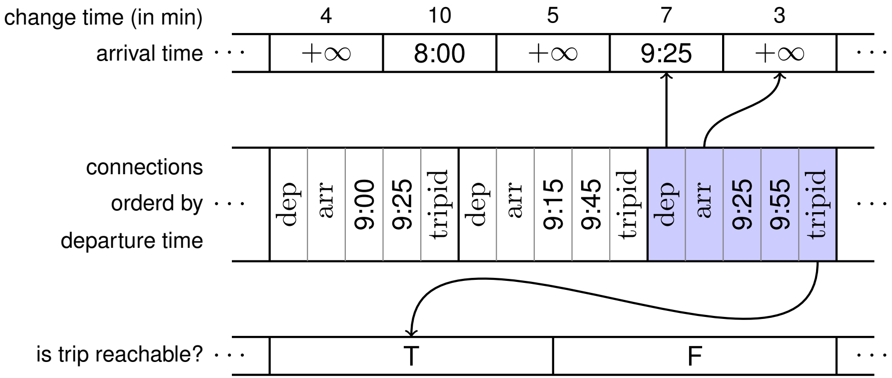


Basic Connection Scan

Earliest Arrival Time Problem

Input: Ordered list of connections, source stop, source time, target stop

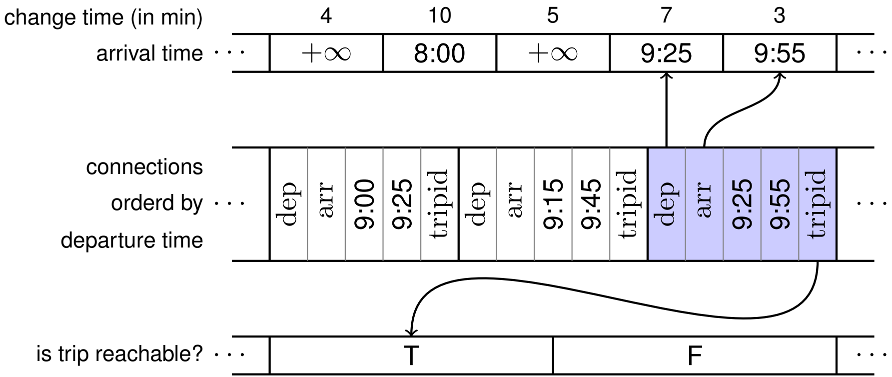
Output: Earliest (on time) arrival time



Basic Connection Scan

Earliest Arrival Time Problem

Input: Ordered list of connections, source stop, source time, target stop
Output: Earliest (on time) arrival time



Basic Connection Scan

Earliest Arrival Time Problem

Input: Ordered list of connections, source stop, source time, target stop
Output: Earliest (on time) arrival time

change time (in min)		4	10	5	7	3	
arrival time	⋯	$+\infty$	8:00	$+\infty$	9:25	9:55	⋯

connections orderd by departure time	⋯	dep	arr	9:00	9:25	tripid	dep	arr	9:15	9:45	tripid	dep	arr	9:25	9:55	tripid	⋯

is trip reachable?	⋯	T	F	⋯

Startcriterion

Observation: Trains departing before the source time are never needed.

$\Rightarrow$ Do a binary search to determine the first connection that departs no earlier than the source time and start the scan there.

Stopcriterion

So far: We solve the one-to-all problem.

Question: Can we do better given a target stop t, i.e., solve the one-to-one problem?

Stopcriterion

So far: We solve the one-to-all problem.

Question: Can we do better given a target stop t, i.e., solve the one-to-one problem?

Observation: Trains departing after the arrival time at t are never useful.

$\Rightarrow$ Abort the scan if the time at t is not bigger than the departure time of the current connection.

Profile Queries

Problem: The user often does not know his departure or arrival time.
Solution: Provide journeys for a whole time range.

Profile Queries

Problem: The user often does not know his departure or arrival time.
Solution: Provide journeys for a whole time range.

Karlsruhe Hbf Leipzig Hbf	dep arr	15:00 20:18	2
Karlsruhe Hbf Leipzig Hbf	dep arr	16:00 20:46	0
Karlsruhe Hbf Leipzig Hbf	dep arr	18:01 22:55	1
Karlsruhe Hbf Leipzig Hbf	dep arr	18:51 00:10	2
Karlsruhe Hbf Leipzig Hbf	dep arr	18:51 00:47	1
Karlsruhe Hbf Leipzig Hbf	dep arr	19:01 00:10	3
Karlsruhe Hbf Leipzig Hbf	dep arr	19:01 00:47	2

Screenshot of bahn.de

Profile Queries

Problem: The user often does not know his departure or arrival time.
Solution: Provide journeys for a whole time range.

source stop

target stop

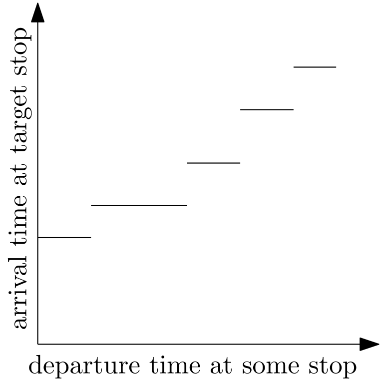
minimum departure time

maximum arrival time

Karlsruhe Hbf Leipzig Hbf	dep arr	15:00 20:18	2
Karlsruhe Hbf Leipzig Hbf	dep arr	16:00 20:46	0
Karlsruhe Hbf Leipzig Hbf	dep arr	18:01 22:55	1
Karlsruhe Hbf Leipzig Hbf	dep arr	18:51 00:10	2
Karlsruhe Hbf Leipzig Hbf	dep arr	18:51 00:47	1
Karlsruhe Hbf Leipzig Hbf	dep arr	19:01 00:10	3
Karlsruhe Hbf Leipzig Hbf	dep arr	19:01 00:47	2

Screenshot of bahn.de

Profile Connection Scan

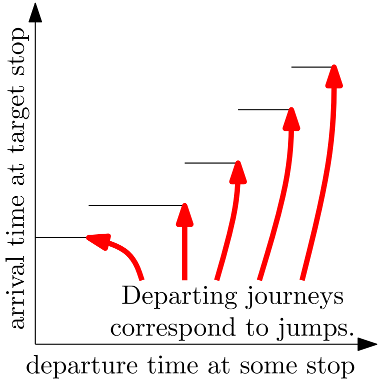


Earliest Arrival Backward Profile Problem

Input: Timetable, target stop t

Output: (full) st-Profile for every stop s (except t)

Profile Connection Scan



Earliest Arrival Backward Profile Problem

Input:	Timetable, target stop t
Output:	(full) st-Profile for every stop s (except t)

Profile Connection Scan

Core Idea: Dynamic Programming

At the arrival of each connection the user has three options:

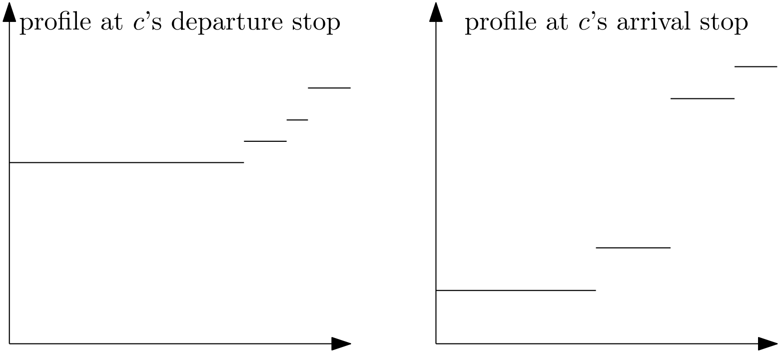
- He remains seated.
- He exits the train and waits for another one.
- He finishes his journey (only valid at target stop).

Profile-Representation: Store a profile function as dynamic array of ordered $(\mathrm{deptime}, \mathrm{arrtime})$-pairs.

Note: The pairs are simultaneously ordered increasing by $\mathrm{deptime}$ and $\mathrm{arrtime}$.

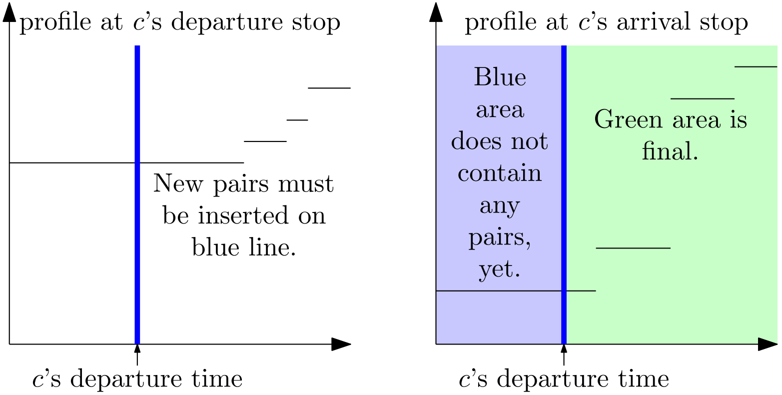
Profile Connection Scan

For every connection c decreasing by departure time:



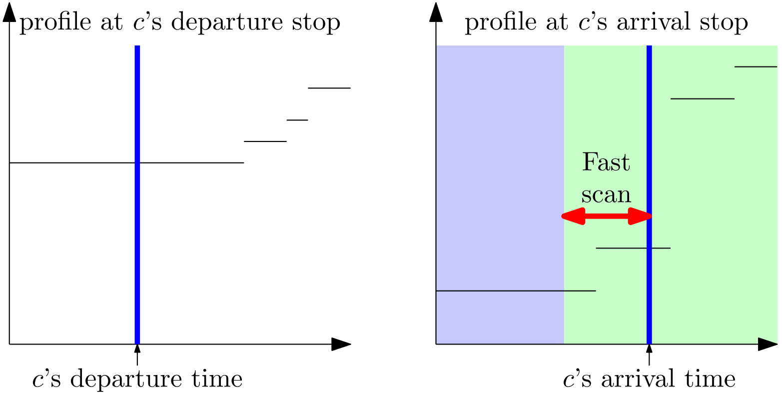
Profile Connection Scan

For every connection *c* decreasing by departure time:



Profile Connection Scan

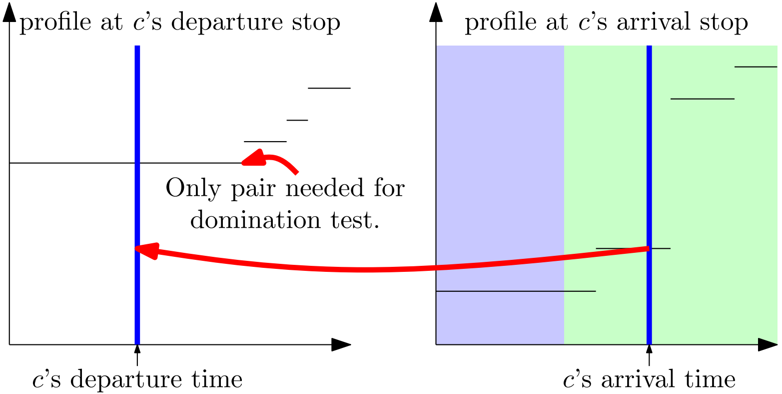
For every connection c decreasing by departure time:



In practice: very short linear scan . (Can be made constant, see later.)

Profile Connection Scan

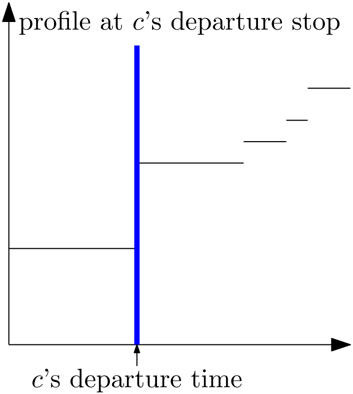
For every connection *c* decreasing by departure time:

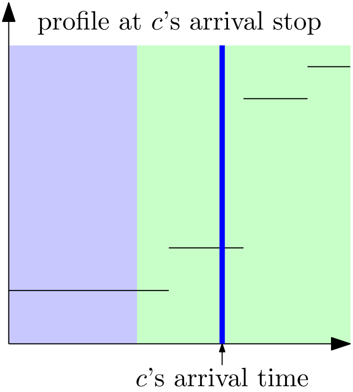


Test if new jump is above or below existing one.

Profile Connection Scan

For every connection c decreasing by departure time:





Insert new jump if below. (In the example it is below.)

Profile Connection Scan

$P(s)$: profile of stop s, an array of pairs;
$T(t)$: earliest arrival time of trip t, a single timestamp;

$P(s) \leftarrow \{(\infty, \infty)\}$ for every stop s;
$T(t) \leftarrow \infty$ for every trip t;
for *every connection c by decreasing* c_{deptime} **do**
    $t \leftarrow$ evaluate $P(c_{\mathrm{arrstop}})$ at $c_{\mathrm{arrtime}} + \mathrm{minchange}(c_{\mathrm{arrstop}})$;
    $t \leftarrow \min(t, T(c_{\mathrm{tripid}}))$;
    if $c_{\mathrm{arrstop}} =$ *target stop* **then**
        $t \leftarrow \min(t, c_{\mathrm{arrtime}})$;
    $T(c_{\mathrm{tripid}}) \leftarrow t$;
    if $t < P(c_{\mathrm{depstop}})[\mathrm{front}].\mathrm{arrtime}$ **then**
        Insert $(c_{\mathrm{deptime}}, t)$ pair at the front of $P(c_{\mathrm{depstop}})$;

How to Evaluate Profiles?

Evaluating a profile at moment t consists of finding the first pair (d, a) such that $t \leq d$.

Option 1:

- $c_{\mathrm{arrtime}} - c_{\mathrm{deptime}}$ is small compared to the time horizon of the whole profile.
- A linear scan works very good in practice.
- Less than 2 pairs touched on average.
- Quasi $O(1)$.

How to Evaluate Profiles?

Option 2: Modify the algorithm slightly. Replace:

---

if $t < P(c_{\text{depstop}})[\text{front}].\text{arrtime}$ **then**
    Insert (c_{deptime}, t) pair at the front of $P(c_{\text{depstop}})$;

---

with

Insert $(c_{\text{deptime}}, \min(t, P(c_{\text{depstop}})[\text{front}].\text{arrtime}))$ pair
    at the front of $P(c_{\text{depstop}})$;

- The departure times are now independent of the target stop.
- The same pairs with the same departure times (but different arrival times) are generated in the same order for each algorithm execution.
- In a quick preprocessing step: Store for each connection the position c_{eval} of the corresponding pair.
- True $O(1)$ evaluation $\rightarrow$ Complete running time in $O(\#\text{connections})$

Profile Connection Scan

$P(s)$: profile of stop s;
$T(t)$: earliest arrival time of trip t, a single integer;
c_{eval} : pair ID needed at evaluation, independent of the target stop;

$P(s) \leftarrow \{(\infty, \infty)\}$ for every stop s;
$T(t) \leftarrow \infty$ for every trip t;
for *every connection c by decreasing* c_{deptime} **do**
    $t \leftarrow P(c_{\text{arrstop}})[c_{\text{eval}}]$;
    $t \leftarrow \min(t, T(c_{\text{tripid}}))$;
    if $c_{\text{arrstop}} =$ *target stop* **then**
        $t \leftarrow \min(t, c_{\text{arrtime}})$;
    $T(c_{\text{tripid}}) \leftarrow t$;
    Insert $(c_{\text{deptime}}, \min(t, P(c_{\text{depstop}})[\text{front}].\text{arrtime}))$ pair
        at the front of $P(c_{\text{depstop}})$;

Restricted Range

The input also contains:

- a minimum departure time d
- a maximum arrival time a

Optimization:

- Scan only connections c with $d \leq c_{\mathrm{deptime}} \leq a$.
- This subarray can be determined using two binary searches.

Restricted Range

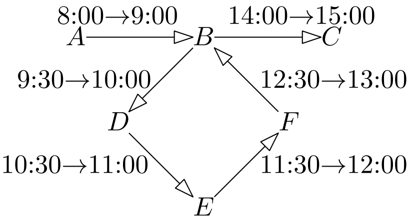
The input also contains:

- a minimum departure time d
- a maximum arrival time a
- a source stop s
- a target stop t

Optimization:

- For every stop x determine the earliest arrival time $\tau(x)$ from s with source time d. This can be done using a basic connection scan, restricted to the same subrange.
- Before processing a connection c in the profile scan check whether $\tau(c_{\text{depstop}}) \leq c_{\text{deptime}}$. If this does not hold skip the connection, as it can not be part of any desired journey.

Problems with Earliest Arrival Time



The user wants to get from *A* to *C*.

Earliest arrival journeys:

1. $A \to B \to C$
2. $A \to B \to D \to E \to F \to B \to C$

The user wants journey 1.

However, journey 2 is also optimal according to arrival time.

→One solution: Minimize Number of Transfers

Problems with Earliest Arrival Time

Easy solution: Chose among all earliest arrival journeys one with a minimum number of transfers.

Algorithm modification: Add a small constant to the arrival time returned by the $P(c_{\mathrm{arrstop}})$ evaluation.

Good enough in most networks.

Problems with Earliest Arrival Time

Complex solution: Compute all Pareto-optimal journeys.

Pareto-optimal: A journey is Pareto-optimal if no other journey exists that is faster and uses fewer transfers.

Algorithm modification:

- We are only interested in journeys with at most n train entries (i.e. $n - 1$ transfers).
- Replace all arrival times with $(a_0, a_1, a_2 \ldots a_n)$-tuples. a_i indicates the earliest arrival time when entering at most i trains.
- Inserting a pair consits of entering a train.
  $\rightarrow$ Shift all components, i.e.,
  $\mathrm{shift}(a_0, a_1, \ldots, a_n) = (\infty, a_0, a_1, \ldots, a_{n-1})$
- Minimum is replace with componentwise minimum.

Pareto Profile Connection Scan

$P(s)$: profile of stop s;
$T(t)$: earliest arrival time of trip t;
c_{eval} : pair ID needed at profile evaluation;

$P(s) \leftarrow \{(\infty, (\infty, \infty \ldots \infty))\}$ for every stop s;
$T(t) \leftarrow (\infty, \infty \ldots \infty)$ for every trip t;
for *every connection c by decreasing* c_{deptime} **do**
    $t \leftarrow P(c_{\mathrm{arrstop}})[c_{\mathrm{eval}}]$;
    $t \leftarrow \min(t, T(c_{\mathrm{tripid}}))$;
    if $c_{\mathrm{arrstop}} =$ *target stop* **then**
        $t \leftarrow \min(t, (c_{\mathrm{arrtime}}, c_{\mathrm{arrtime}} \ldots c_{\mathrm{arrtime}}))$;
    $T(c_{\mathrm{tripid}}) \leftarrow t$;
    Insert $(c_{\mathrm{deptime}}, \min(\mathrm{shift}(t), P(c_{\mathrm{depstop}})[\mathrm{front}].\mathrm{arrtime}))$ pair
        at the front of $P(c_{\mathrm{depstop}})$;

SIMD/SSE

- SIMD : Single Instruction Multiple Data
- Special CPU instructions that work on fixed length vectors of integers (or floats) in a single CPU cycle.

- Available on all mordern x86 processors.
- 128bit registers à 4×32 bit or à 8×16 bit integers.

- The x86 implementation is called SSE.
- SIMD is the general concept.

SIMD/SSE

Without SSE:

```
int a[8],b[8],c[8];
c[0]=a[0]+b[0];    c[1]=a[1]+b[1];    c[2]=a[2]+b[2];
c[3]=a[3]+b[3];    c[4]=a[4]+b[4];    c[5]=a[5]+b[5];
c[6]=a[6]+b[6];    c[7]=a[7]+b[7];
```

8 time units used

With SSE:

```
__m128i a[2],b[2],c[2];
c[0]=_mm_add_epi32(a[0],b[0]);
c[1]=_mm_add_epi32(a[1],b[1]);
```

2 time units used

Speedup of 4 only observed for compute-bound algorithms.

SIMD/SSE

Without SSE:

```
short a[8],b[8],c[8];
c[0]=a[0]+b[0];    c[1]=a[1]+b[1];    c[2]=a[2]+b[2];
c[3]=a[3]+b[3];    c[4]=a[4]+b[4];    c[5]=a[5]+b[5];
c[6]=a[6]+b[6];    c[7]=a[7]+b[7];
```

8 time units used

With SSE:

```
__m128i a,b,c;
c=_mm_add_epi16(a,b);
```

1 time units used

Speedup of 8 only observed for compute-bound algorithms.

SIMD/SSE

Simple if-else-constructs are also supported.

Without SSE:

```
short a[8],b[8],c[8];
for(int i=0; i<8; ++i)
    c[i] = a[i] < b[i] ? a[i] : b[i];
```

With SSE:

```
__m128i a,b,c;
c=_mm_blendv_epi8(a,b,_mm_cmplt_epi16(a,b));
```

Or use the minimum instruction:

```
__m128i a,b,c;
c=_mm_min_epi16(a,b);
```

Pareto Profile Connection Scan

The arrival time vector operations can all be done using SSE-instructions.

Time Compression

Problem: Algorithm is memory-bound.
The memory is nearly completely filled with arrival times.
$\rightarrow$ Compress arrival times.

Observation: Not at every second a train leaves and departs.

Idea: For every stop compute an ordered array $t_0, t_1, \ldots, t_n$ of the time points at which a train departs or arrives.

- Indices respect time order, i.e., $t_i < t_j \iff i < j$.
- Often the indices fit into 16 bit integers.
- Propagate the indices instead of the time points.

Problems with Delays

Karlsruhe Hbf	Fr, 31.05.13	dep 08:00	10:30	2	ICE, EC, ES	Connection i
Roma Termini	Fr, 31.05.13	arr 18:30				

Station/Stop	Date	Time			Platform	Products	
Karlsruhe Hbf	Fr, 31.05.13	dep	08:00		2	ICE 5	Intercity-Express
Zürich HB	Fr, 31.05.13	arr	11:00		10		Bordrestaurant
Transfer time 9 min.							→ Adjust the transfer time
Zürich HB	Fr, 31.05.13	dep	11:09		5	EC 17	Eurocity
Milano Centrale	Fr, 31.05.13	arr	14:50	approx. +25			Subject to compulsory reser available
Transfer time 20 min.		Connecting train may not be reached in time.					→ Adjust the transfer time
Milano Centrale	Fr, 31.05.13	dep	15:10			ES 9541	EuroStar Italia
Roma Termini	Fr, 31.05.13	arr	18:30				Subject to compulsory reser wheelchairs

composed out of several screenshots of bahn.de, specific situation was not observed

Problems with Delays

Karlsruhe Hbf	Fr, 31.05.13	dep	08:00	10:30	2	ICE, EC, ES		Connection i
Roma Termini	Fr, 31.05.13	arr	18:30					

Station/Stop	Date	Time		Platform	Products	
Karlsruhe Hbf	Fr, 31.05.13	dep	08:00	2	ICE 5	Intercity-Express
Zürich HB	Fr, 31.05.13	arr	11:00	10		Bordrestaurant
Transfer time 9 min.						→ Adjust the transfer time
Zürich HB	Fr, 31.05.13	dep	11:09	5	EC 17	Eurocity
Milano Centrale	Fr, 31.05.13	arr	14:50 approx. +25			Subject to compulsory reser available
Transfer time 20 min.		Connecting train may not be reached in time.				→ Adjust the transfer time
Milano Centrale	Fr, 31.05.13	dep	15:10		ES 9541	EuroStar Italia
Roma Termini	Fr, 31.05.13	arr	18:30			Subject to compulsory reser wheelchairs

composed out of several screenshots of bahn.de, specific situation was not observed

25 min delay vs 20 min transfer

Problems with Delays

Karlsruhe Hbf	Fr, 31.05.13	dep	08:00	10:30	2	ICE, EC, ES	Connection i
Roma Termini	Fr, 31.05.13	arr	18:30				

Station/Stop	Date	Time			Platform	Products	
Karlsruhe Hbf	Fr, 31.05.13	dep	08:00		2	ICE 5	Intercity-Express
Zürich HB	Fr, 31.05.13	arr	11:00		10		Bordrestaurant
Transfer time 9 min.							→ Adjust the transfer time
Zürich HB	Fr, 31.05.13	dep	11:09		5	EC 17	Eurocity
Milano Centrale	Fr, 31.05.13	arr	14:50	approx. +25			Subject to compulsory reser available
Transfer time 20 min.		Connecting train may not be reached in time.					→ Adjust the transfer time
Milano Centrale	Fr, 31.05.13	dep	15:10			ES 9541	EuroStar Italia
Roma Termini	Fr, 31.05.13	arr	18:30				Subject to compulsory reser wheelchairs

composed out of several screenshots of bahn.de, specific situation was not observed

What if 9 min are not enough?

Problems with Delays

Karlsruhe Hbf Roma Termini	Fr, 31.05.13 Fr, 31.05.13	dep 08:00 arr 18:30	10:30	2	ICE, EC, ES	Connection i

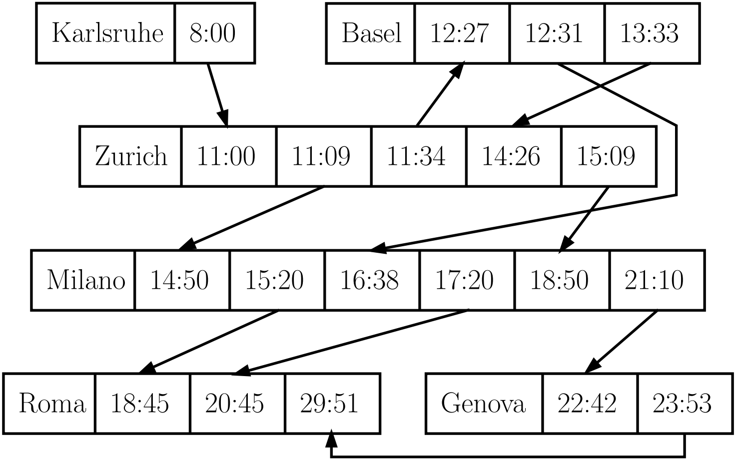
Station/Stop	Date	Time		Platform	Products	
Karlsruhe Hbf	Fr, 31.05.13	dep	08:00	2	ICE 5	Intercity-Express
Zürich HB	Fr, 31.05.13	arr	11:00	10		Bordrestaurant
Transfer time 9 min.						Adjust the transfer time
Zürich HB	Fr, 31.05.13	dep	11:09	5	EC 17	Eurocity
Milano Centrale	Fr, 31.05.13	arr	14:50 approx. +25			Subject to compulsory reser available
Transfer time 20 min.		Connecting train may not be reached in time.				Adjust the transfer time
Milano Centrale	Fr, 31.05.13	dep	15:10		ES 9541	EuroStar Italia
Roma Termini	Fr, 31.05.13	arr	18:30			Subject to compulsory reser wheelchairs

composed out of several screenshots of bahn.de, specific situation was not observed

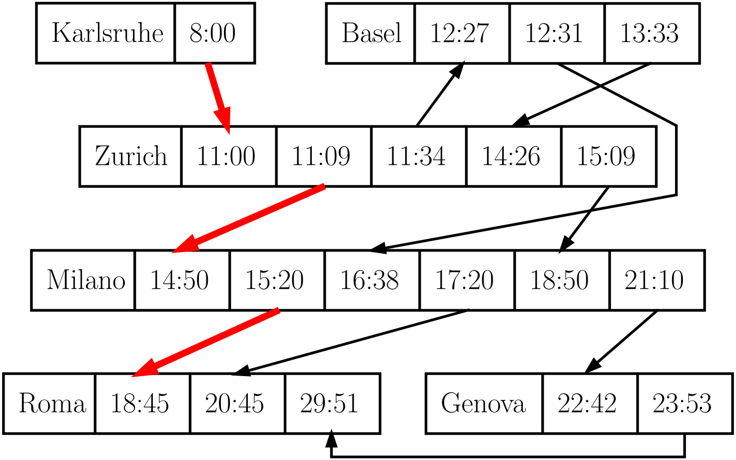
. . . but perhaps they are enough . . .

→ Backup journeys are needed

Decision Graph

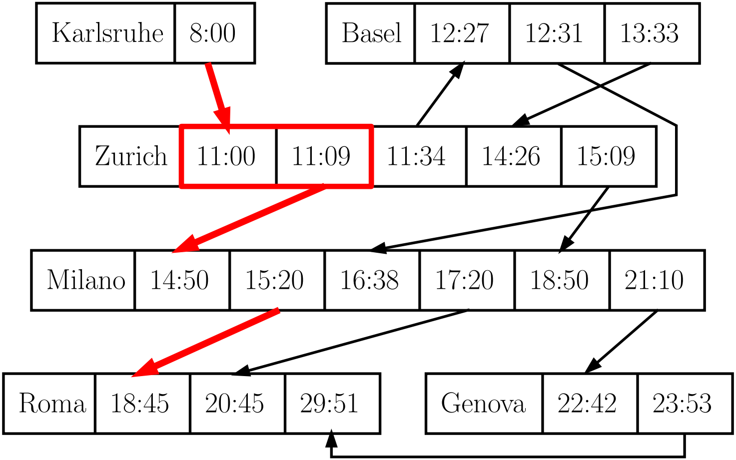


Decision Graph



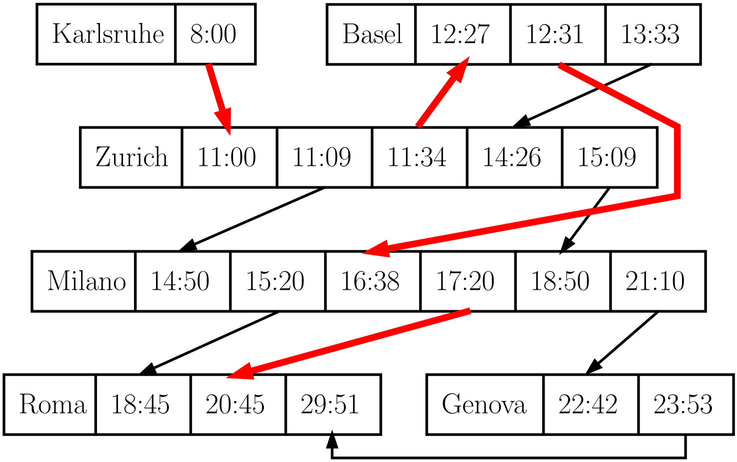
If all goes well

Decision Graph



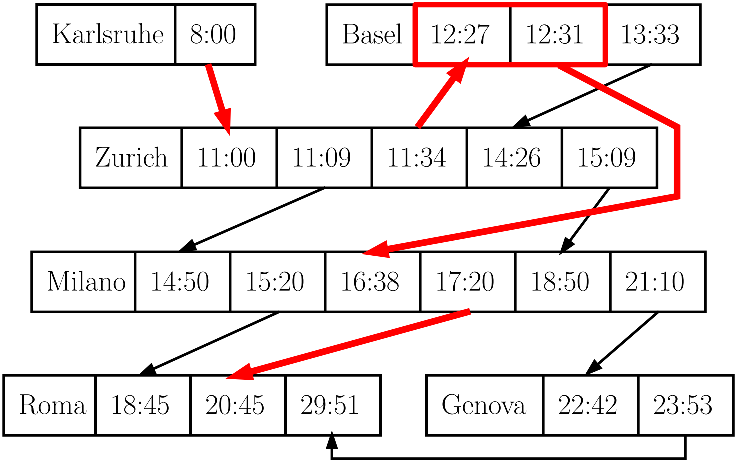
9 min change time → likely to fail

Decision Graph



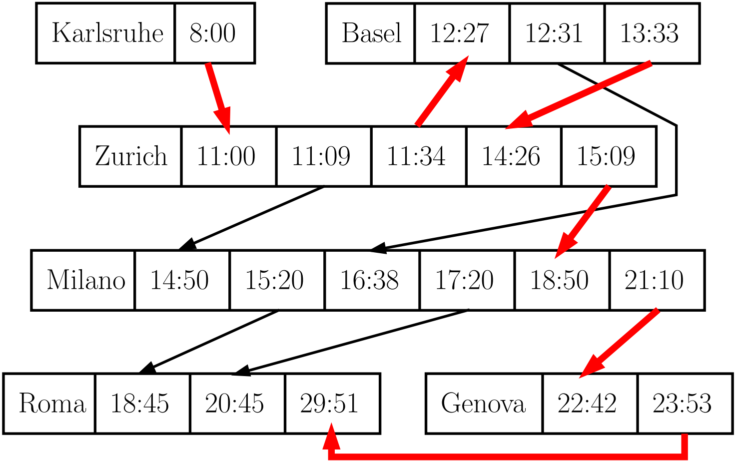
The backup journey

Decision Graph



Tight change on the backup → Backup for the backup needed

Decision Graph

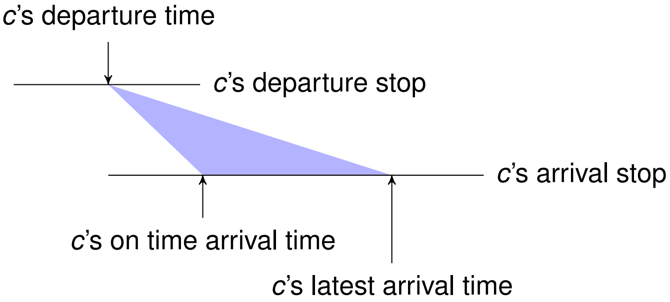


Backup of the Backup

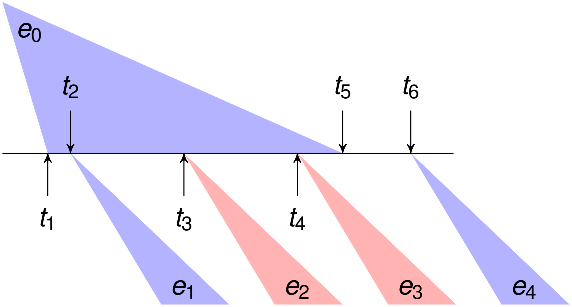
Delay Model

Assumptions:

- Connections arrive with a random delay.
- Connections have a maximum delay.
- Distributions are known.
- All random variables are independent.
- Connections always depart on time.

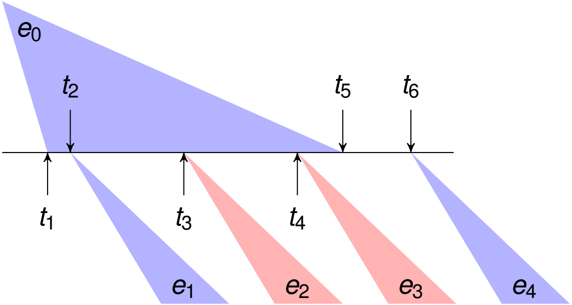


Expected Arrival Time



$e_0 \ldots e_4$:	expected arrival times
$t_1 \ldots t_6$:	fixed points int time
blue:	in the decision graph
red:	not in the decision graph

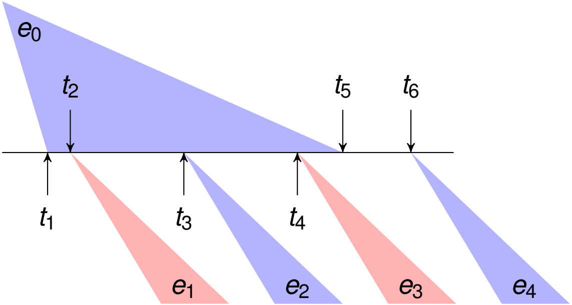
Expected Arrival Time



t: actual arrival time

$$e_0 = P(t_1 \leq t \leq t_2) \cdot e_1 + P(t_2 \leq t \leq t_5) \cdot e_4$$

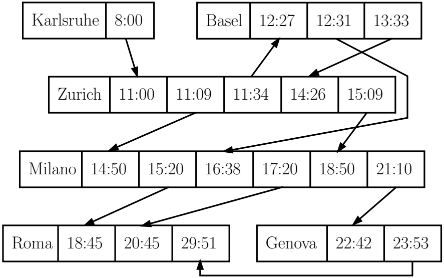
Expected Arrival Time



t: actual arrival time

$$e_0 = P(t_1 \leq t \leq t_3) \cdot e_2 + P(t_3 \leq t \leq t_5) \cdot e_4$$

Minimum Expected Arrival Time (MEAT)

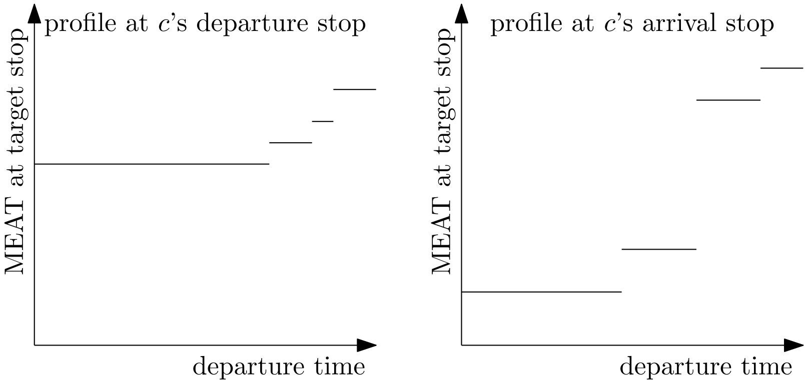


Computing Decision Graphs

Input: Timetable, delay probabilities, target stop

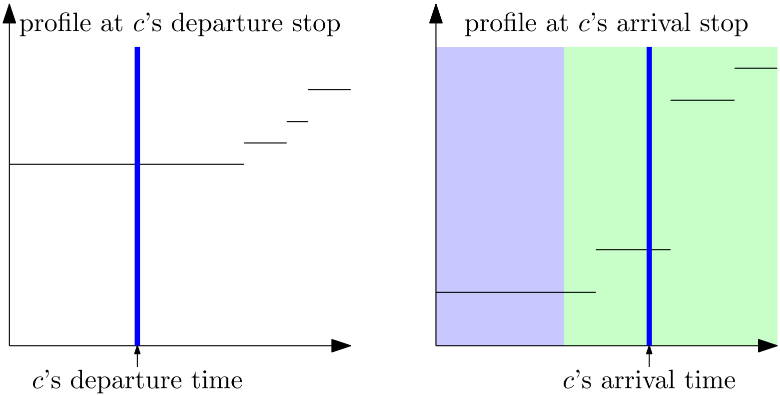
Output: Subset of connections with *minimum expected arrival time* for every source stop and time

MEAT Connection Scan



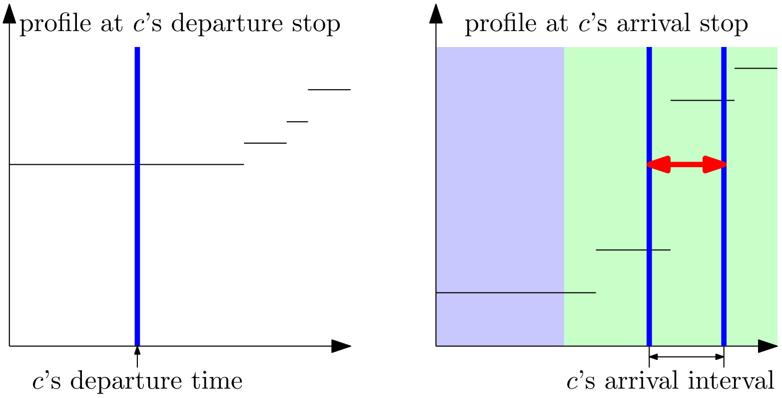
Profile maps depature time onto MEAT at target.

MEAT Connection Scan



Profile maps depature time onto MEAT at target.

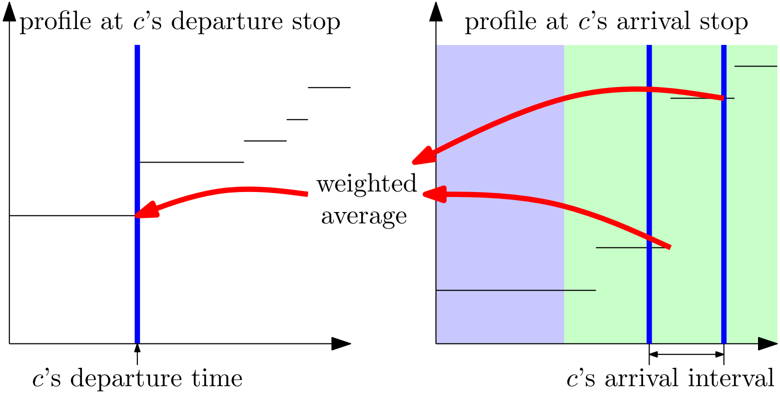
MEAT Connection Scan



Scan a bit further to collect all relevant trains.

This scan is not $O(1)$.

MEAT Connection Scan



Average weighted by probability of a getting train.
Inserting is $O(1)$ just as before.

Running Times

London instance with 4 850 431 connections.

Earliest Arrival One-to-One:

- Time-Expanded: 64.4 ms
- Time-Dependent: 10.9 ms
- Connection Scan: 2.0 ms

Earliest Arrival One-to-All:

- Time-Expanded: 876.2 ms
- Time-Dependent: 18.9 ms
- Connection Scan: 9.7 ms

(Time-Dependent can be made slightly faster, with ideas not covered in this course.)

Running Times

Non-Pareto Profile All-to-One:

- Self-Pruning-Connection-Setting : 1 262 ms
- Connection Scan: 177 ms
- + constant eval: 134 ms
- + time compress: 104 ms

Pareto Profile All-to-One (journeys with at most 8 trains):

- RAPTOR : 1 179 ms
- Connection Scan: 255 ms
- + SSE: 221 ms

MEAT: 272 ms