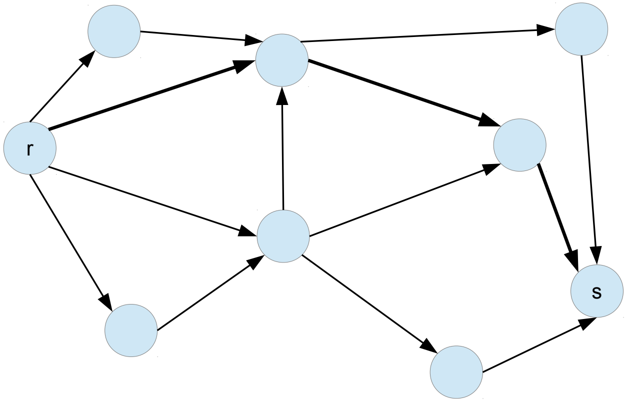
Time-dependent shortest paths

CE 392D

TDSP CONCEPTS

In a network, we have a traveler leaving an origin node r at time t_0, heading for destination node s. What is the fastest route there, taking into account the changes in link travel times during the trip?



This is called the time-dependent shortest path (TDSP) problem. It is the converse to network loading.

One odd twist of shortest path problems: it's not much harder to find the shortest path from r to s than to find many shortest paths at the same time. Two broad approaches:

One-to-all:	Find the shortest paths from node r to *all* destination nodes, leaving at time t_0.
All-to-one:	Find the shortest paths from *all* origin nodes to node s, arriving at time t_a.

Which one is more appropriate? When would we have known departure times, and when would we have known arrival times?

Today, we'll focus on the one-to-all problem. The all-to-one problem can be solved in a symmetric way, just starting at a destination and working backwards (rather than starting at the origin and working forwards).

We add a few additional assumptions to start:

- Travelers choose routes to minimize travel time.
- Travel times on all links obey the first-in, first-out (FIFO) property: $t < t' \Rightarrow t + \tau_{ij}(t) < t' + \tau_{ij}(t')$
- Waiting is not allowed at nodes. (Not necessary in FIFO networks, but we'll come back to this later.)
- The network is *strongly connected*: for any departure time, there is at least one path from every origin to every destination.

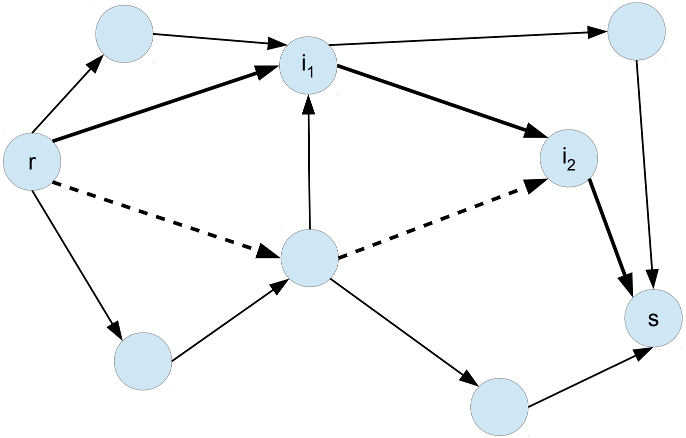
When might these assumptions be violated? Is there an easy way to check the FIFO property?

One-to-all TDSP relies on **Bellman's Principle**, which lets us re-use information between different origins and destinations:

If $\pi^* = [r, i_1, i_2, \ldots, i_n, s]$ is a shortest path from r to s at time t_0, then the subpath $[r, i_1, \ldots, i_k]$ is a shortest path from r to i_k, also leaving at time t_0.

The upshot: we don't have to consider the *entire* route from r to d at once. Instead, we can break it up into smaller, easier problems.

Why does Bellman's principle hold?



Under the FIFO assumption, if there is a shorter path from r to i_k, I could "splice" that into π^* and obtain a shorter path from r to s.

ONE-TO-ALL TDSP ALGORITHM IN FIFO NETWORKS

This is Section 11.2.1 in the book, and is based on a modified version of Dijkstra's algorithm.

We associate two pieces of information with each node i:

- A label L_i storing the travel time on the shortest known path from r to i (∞ if we don't know any yet)
- A predecessor q_i storing the second-to-last node on the shortest known path from r to i. (-1 if we don't know any yet.)

We also maintain a list of finalized nodes F, to which we are sure we have found the shortest path.

Why are the predecessors enough to tell us the entire shortest path?

This algorithm has two phases: an **initialization** and an **iteration**.

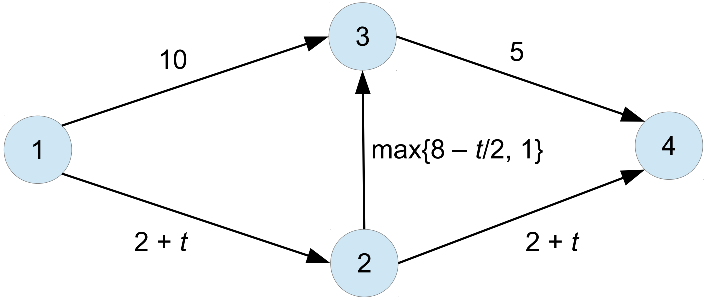
Given: Origin node r, departure time t_0
Find: Shortest paths to all nodes $i \neq r$

```
Initialization:
    Set $L_r = t_0$ and $L_i = \infty$ for all $i \neq r$.
    Set $q_i = -1$ for all $i$.
    Set $F = \emptyset$.

Iteration:
Repeat the following steps until $F$ contains all nodes:
    Let $i$ be the unfinalized node with the least $L_i$ label.
    Finalize $i$ by adding it to $F$.
    For each link $(i, j)$ leaving node $i$, do the following:
        If $L_i + \tau_{ij}(L_i) < L_j$ then set $L_j = L_i + \tau_{ij}(L_i)$ and $q_j = i$.
```

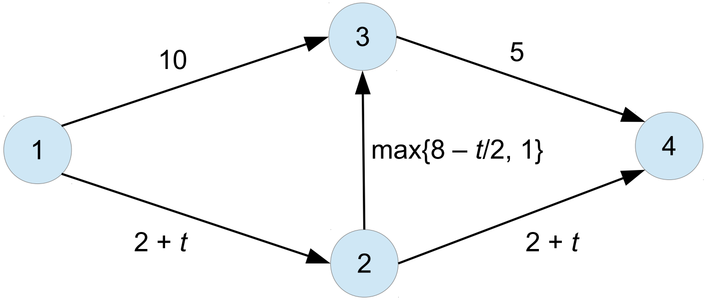
EXAMPLE

After initialization



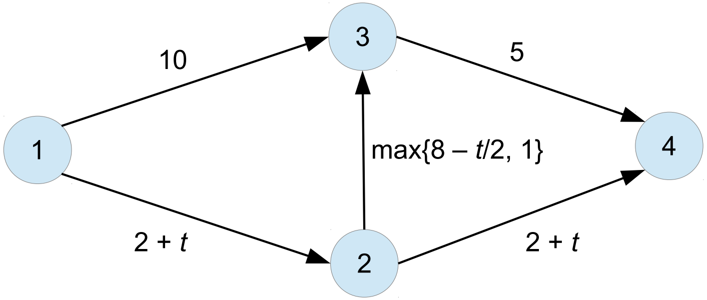
Node	1	2	3	4
L	4	∞	∞	∞
q	-1	-1	-1	-1

After iteration 1



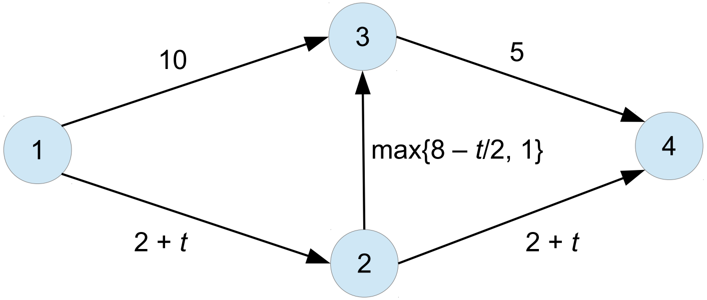
Node	**1**	2	3	4
L	**4**	10	14	∞
q	**-1**	1	1	-1

After iteration 2



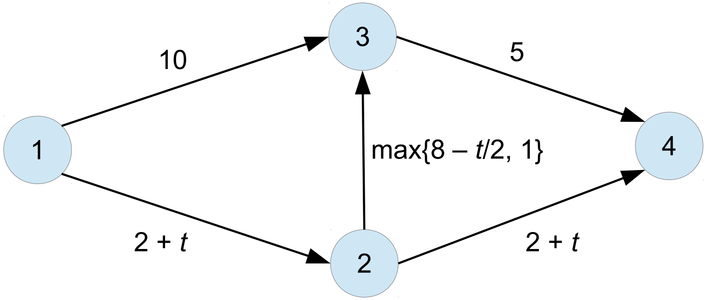
Node	**1**	**2**	3	4
L	**4**	**10**	13	22
q	**-1**	**1**	2	2

After iteration 3



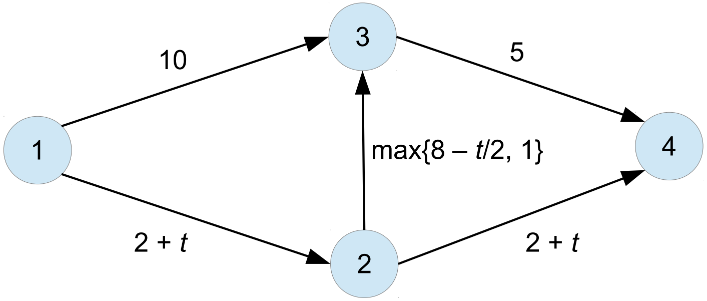
Node	**1**	**2**	**3**	4
L	**4**	**10**	**13**	18
q	**-1**	**1**	**2**	3

After iteration 4



Node	1	2	3	4
L	**4**	**10**	**13**	**18**
q	**-1**	**1**	**2**	**3**

What would be the instantaneous shortest path?



Node	1	2	3	4
L	4	∞	∞	∞
q	-1	-1	-1	-1

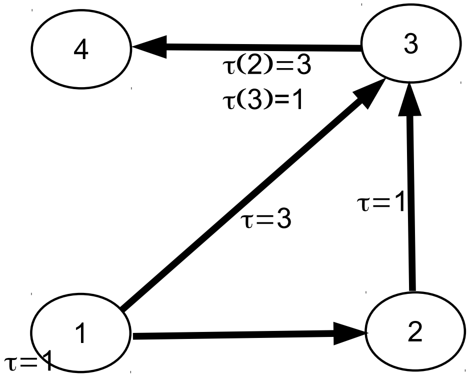
MORE GENERAL TDSP

More general versions of the time-dependent shortest path problem:

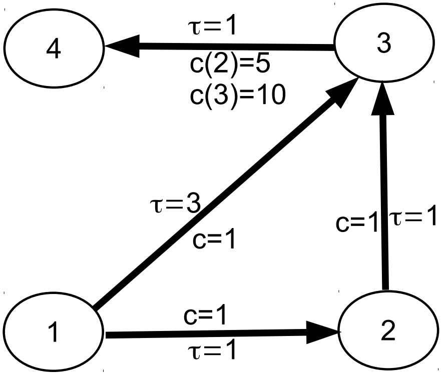
- Non-FIFO links
- Link cost is different from link travel time

Bellman's principle can fail in these cases unless we are more careful.

Scenario 1



Costs and times are constant when no time index is given.

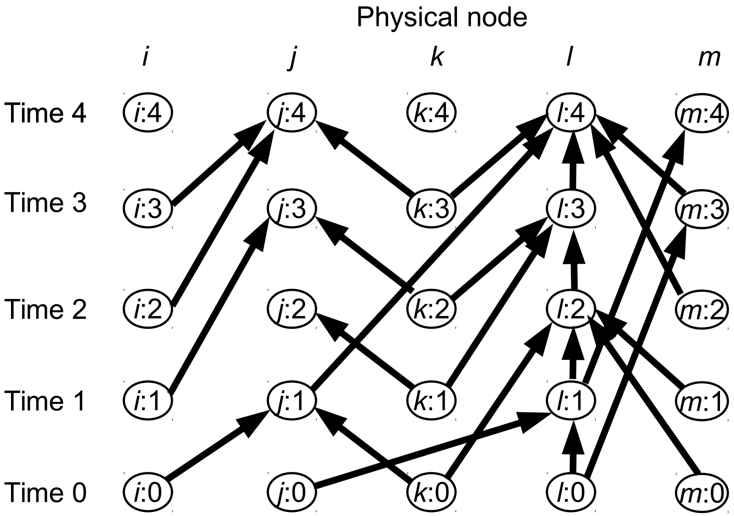


Costs and times are constant when no time index is given.

A time-expanded network contains a copy of each node, for each (discrete) time step.

Each "node" in the time-expanded network represents a physical node *at a specific instant in time.* We use the notation $i : t$ for this.

Each link in the time-expanded network represents leaving a physical node at a certain point in time, and arriving at another physical node at another point in time.



In a FIFO network, links between the same physical node will never cross.

Bellman's principle holds *in the time-expanded network*.

Furthermore, the time-expanded network is *acyclic*, so we can find shortest paths more efficiently than before by moving through the network in increasing order of time.

In this algorithm, we use $c_{ij}(t)$ to refer to the cost of link (i, j); it need not be the same as the travel time.

We use $\bar{q}$ as the "backnode" (which now includes the time for leaving the previous node, since it refers to a time-expanded node.)

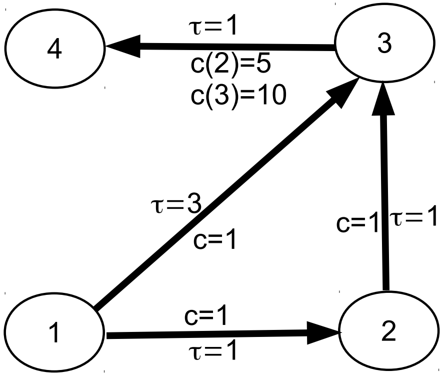
Initialization:

Set $L_r^{t_0} = 0$ and $L_i^t = \infty$ for all $i \neq r$.
Set $\bar{q}_i^t = -1$ for all i, t.
Set t to the departure time t_0.

Iteration:

```
Repeat the following steps until $t$ is the last time interval:
    For each time-expanded link $(i : t, j : t')$, perform the following steps:
        Set $L_j^{t'} = \min \left\{ L_j^{t'}, L_i^t + c_{ij}(t) \right\}$.
        If $L_j^{t'}$ changed in the previous step, update $\bar{q}_j^{t'} \leftarrow i : t$.
    Increase $t$ by 1.
```

Example



Costs and times are constant when no time index is given.

ALL DEPARTURE TIMES

We will need to find shortest paths for each origin, destination, and departure time.

The algorithms thus far picked one origin r and departure time t_0, and found shortest paths to all destinations s.

Another alternative is to pick one origin r and destination s, and find shortest paths for all departure times t.

This approach will naturally lend itself to a departure time choice model.

Key differences

L_i^t now is the cost of the best-known path from i to the *destination* s, starting at time t.

$\bar{q}_i^t$ now is a *forward*node, showing the next node in this best-known path.

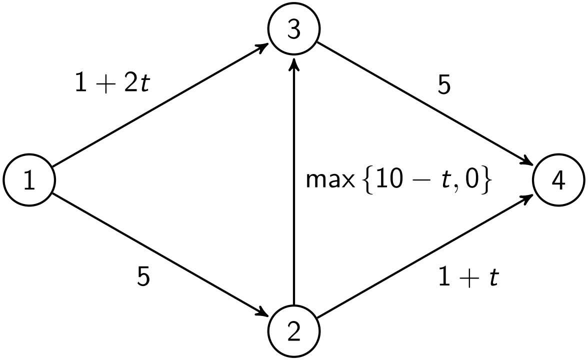
Backnodes no longer work, because multiple departure times could end up arriving at the same node simultaneously.

Initialization:

Set $L_s^t = 0$ for all t and $L_i^t = \infty$ for all $i \neq s$ and all t.
Set $\bar{q}_i^t = -1$ for all i, t.
Set t to the time horizon T.

Iteration:

```
Repeat the following steps until $t = 0$:
    For each time-expanded node $(i : t)$ and link $(h : t', i : t)$, perform the following steps:
        Set $L_h^{t'} = \min \left\{ L_h^{t'}, L_i^t + c_{hi}(t') \right\}$.
        If $L_h^{t'}$ changed in the previous step, update $\bar{q}_h^{t'} \leftarrow i : t$.
    Decrease $t$ by 1.
```



1
2
3
4
$1+2t$
5
5
$\max\{10-t,0\}$
$1+t$

t	L_1^t	L_2^t	L_3^t	L_4^t	$\bar{q}_1^t$	$\bar{q}_2^t$	$\bar{q}_3^t$	$\bar{q}_4^t$
20	∞	∞	∞	0	-1	-1	-1	-1
19	∞	∞	∞	0	-1	-1	-1	-1
18	∞	∞	∞	0	-1	-1	-1	-1
17	∞	∞	∞	0	-1	-1	-1	-1
16	∞	∞	∞	0	-1	-1	-1	-1
15	∞	5	5	0	-1	3	4	-1
14	∞	5	5	0	-1	3	4	-1
13	∞	5	5	0	-1	3	4	-1
12	∞	5	5	0	-1	3	4	-1
11	∞	5	5	0	-1	3	4	-1
10	10	5	5	0	2	3	4	-1
9	10	6	5	0	2	3	4	-1
8	10	7	5	0	2	3	4	-1
7	10	8	5	0	2	3	4	-1
6	10	7	5	0	2	4	4	-1
5	10	6	5	0	2	4	4	-1
4	11	5	5	0	2	4	4	-1
3	12	4	5	0	2	4	4	-1
2	10	3	5	0	3	4	4	-1
1	8	2	5	0	3	4	4	-1
0	6	1	5	0	3	4	4	-1

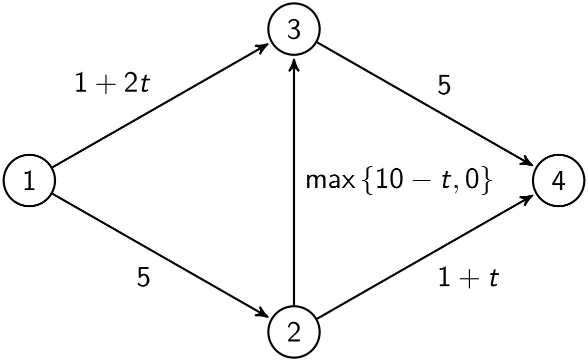
In many cases, drivers can choose their departure times, not just their routes.

Define a function $f(t)$ giving the "cost" of arriving at the destination at time t; drivers wish to minimize the sum of path cost and $f(t)$.

One example: if t^* is the preferred arrival time, then

$$f(t) = \alpha[t^* - t]^+ + \beta[t - t^*]^+$$

The only change needed to the previous algorithm is to initialize L_s^t to $f(t)$ instead of 0!



Assume that the arrival time cost is

$$f(t) = 2[t-10]^+ + [10-t]^+$$

t	L_1^t	L_2^t	L_3^t	L_4^t	$\bar{q}_1^t$	$\bar{q}_2^t$	$\bar{q}_3^t$	$\bar{q}_4^t$
20	∞	∞	∞	20	-1	-1	-1	-1
19	∞	∞	∞	18	-1	-1	-1	-1
18	∞	∞	∞	16	-1	-1	-1	-1
17	∞	∞	∞	14	-1	-1	-1	-1
16	∞	∞	∞	12	-1	-1	-1	-1
15	∞	25	25	10	-1	3	4	-1
14	∞	23	23	8	-1	3	4	-1
13	∞	21	21	6	-1	3	4	-1
12	∞	19	19	4	-1	3	4	-1
11	∞	17	17	2	-1	3	4	-1
10	30	15	15	0	2	3	4	-1
9	28	16	13	1	2	3	4	-1
8	26	17	11	2	2	3	4	-1
7	24	18	9	3	2	3	4	-1
6	22	13	7	4	2	4	4	-1
5	20	8	5	5	2	4	4	-1
4	21	6	6	6	2	4	4	-1
3	22	7	7	7	2	4	4	-1
2	14	8	8	8	3	4	4	-1
1	9	9	9	9	3	4	4	-1
0	10	10	10	10	3	4	4	-1