MONASH University

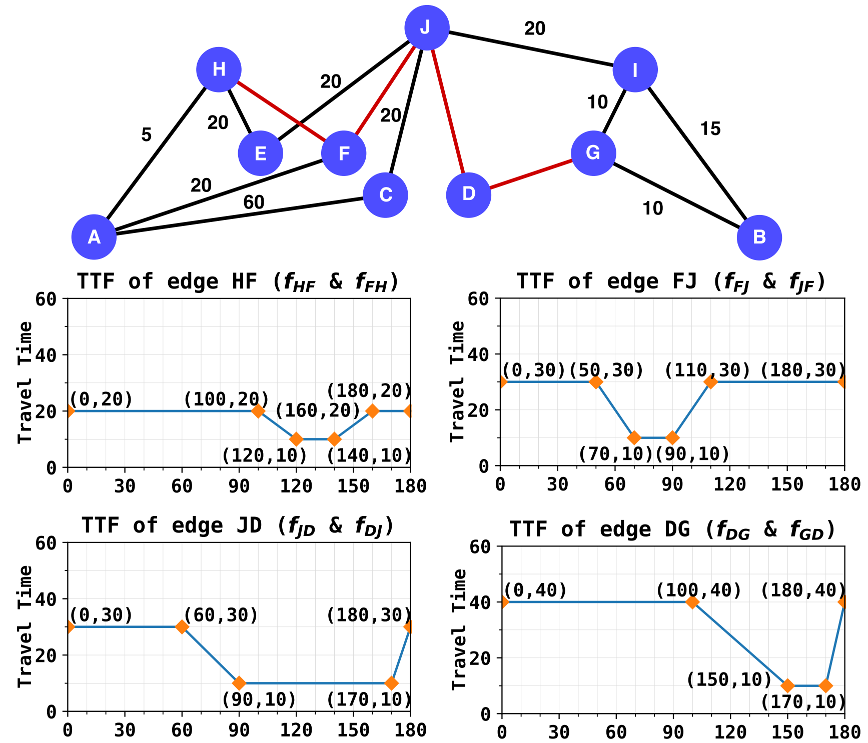
Improving Time-Dependent Contraction Hierarchy ICAPS 2022

Bojie Shen, Muhammad Aamir Cheema, Daniel D. Harabor, Peter J. Stuckey

Monash University

Time Dependent Road Network

- Time Dependent Road Network:
  - A directed graph:
    - A set of vertices: V.
    - A set of edges: $E \subseteq V \times V$.
    - The travel cost of each edge $e \in E$ is represented as a travel time function f (TTF).
      - Each TTF follows FIFO property (i.e., $f(t') + t' \geq f(t) + t \mid \forall e \in E$ and $\forall t' > t \in T$).
  - Objective:
    - Given a start s and destination d.
    - Find the fastest path that minimize the travel time.



An example of an undirected time-dependent graph. TTFs of the red edges are shown below the graph, and the travel cost of the other edges are constant

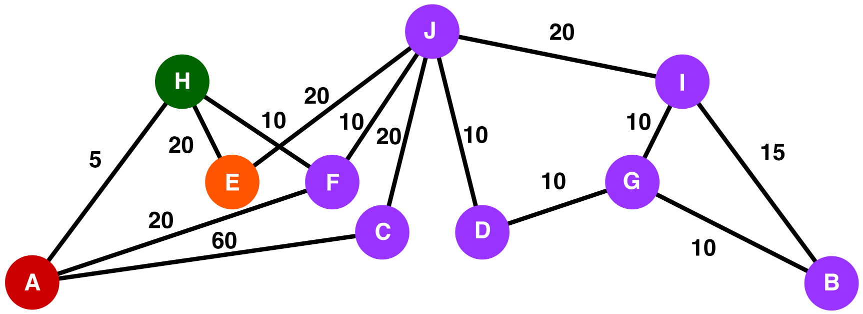
Background
Compressed Path Databases (CPD) Heuristic [1]

- Compressed Path Databases

Compressed Path Databases (CPD) Heuristic [1]

- Compressed Path Databases
  - Construction:
    - First move table:

Ordering	H	A	F	J	I	B	G	D	C	E
H	*	A	F	F	F	F	F	F	F	E
A	H	*	H	H	H	H	H	H	H	H
J	F	F	F	*	I	D	D	D	C	E

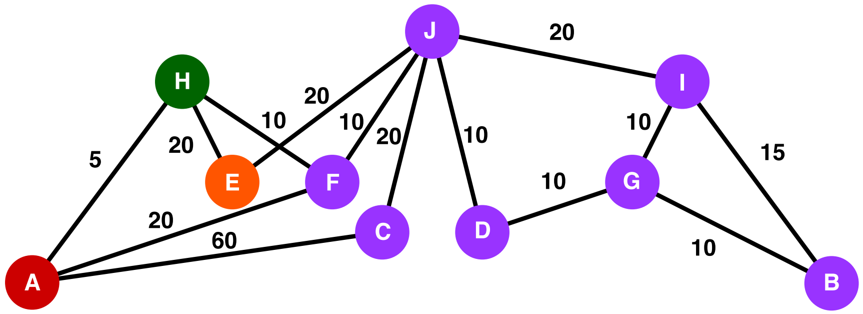


From the source node H, the first move on the optimal path to any node are A (red), E(orange) and F (purple).

Background
Compressed Path Databases (CPD) Heuristic [1]

- Compressed Path Databases
  - Construction:
    - First move table:
      - [A – J]: indicates the optimal first move.

Ordering	H	A	F	J	I	B	G	D	C	E
H	*	A	F	F	F	F	F	F	F	E
A	H	*	H	H	H	H	H	H	H	H
J	F	F	F	*	I	D	D	D	C	E

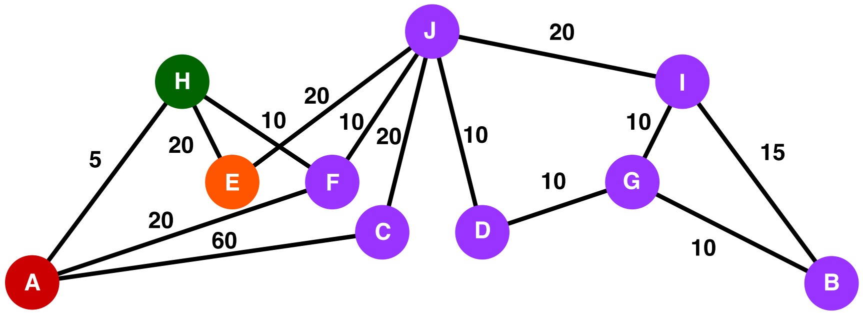


From the source node H, the first move on the optimal path to any node are A (red), E(orange) and F (purple).

Background
Compressed Path Databases (CPD) Heuristic [1]

- Compressed Path Databases
  - Construction:
    - First move table:
      - [A – J]: indicates the optimal first move.
      - *: wildcard symbol [2].

Ordering	H	A	F	J	I	B	G	D	C	E
H	*	A	F	F	F	F	F	F	F	E
A	H	*	H	H	H	H	H	H	H	H
J	F	F	F	*	I	D	D	D	C	E

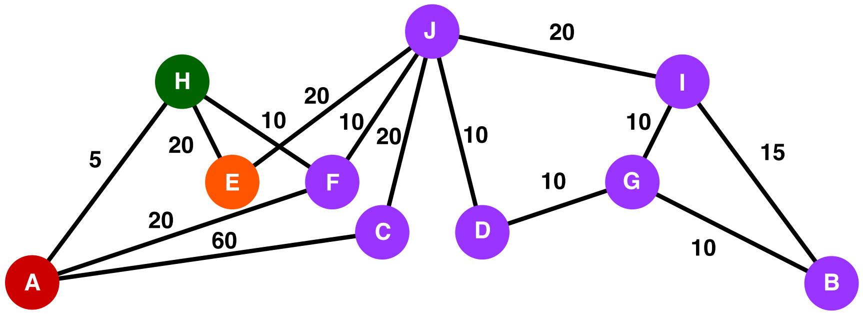


From the source node H, the first move on the optimal path to any node are A (red), E(orange) and F (purple).

Background
Compressed Path Databases (CPD) Heuristic [1]

- Compressed Path Databases
  - Construction:
    - First move table:
      - [A – J]: indicates the optimal first move.
      - *: wildcard symbol [2].
    - Compression:
      - Depth first search order [3].

Ordering	H	A	F	J	I	B	G	D	C	E
H	*	A	F	F	F	F	F	F	F	E
A	H	*	H	H	H	H	H	H	H	H
J	F	F	F	*	I	D	D	D	C	E

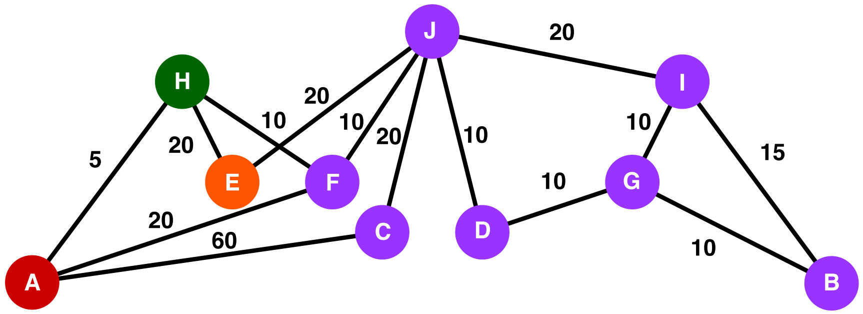


From the source node H, the first move on the optimal path to any node are A (red), E(orange) and F (purple).

Compressed Path Databases (CPD) Heuristic [1]

- Compressed Path Databases
  - Construction:
    - First move table:
      - [A – J]: indicates the optimal first move.
      - *: wildcard symbol [2].
    - Compression:
      - Depth first search order [3].
      - Run length encoding [3] (i.e. Row H: 1A; 3F; 10E).

Ordering	H	A	F	J	I	B	G	D	C	E
H	*	A	F	F	F	F	F	F	F	E
A	H	*	H	H	H	H	H	H	H	H
J	F	F	F	*	I	D	D	D	C	E

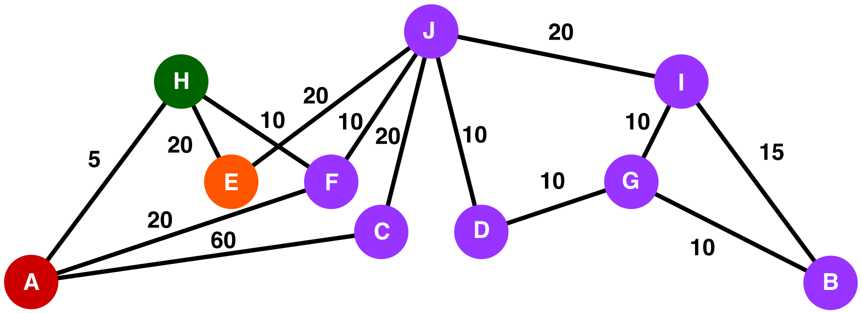


From the source node H, the first move on the optimal path to any node are A (red), E(orange) and F (purple).

Background
Compressed Path Databases (CPD) Heuristic [1]

- Compressed Path Databases
  - Construction:
    - First move table:
      - [A – J]: indicates the optimal first move.
      - *: wildcard symbol [2].
    - Compression:
      - Depth first search order [3].
      - Run length encoding [3] (i.e. Row H: 1A; 3F; 10E).
  - First move extraction:
    - A binary search over compressed RLE string.

Ordering	H	A	F	J	I	B	G	D	C	E
H	*	A	F	F	F	F	F	F	F	E
A	H	*	H	H	H	H	H	H	H	H
J	F	F	F	*	I	D	D	D	C	E

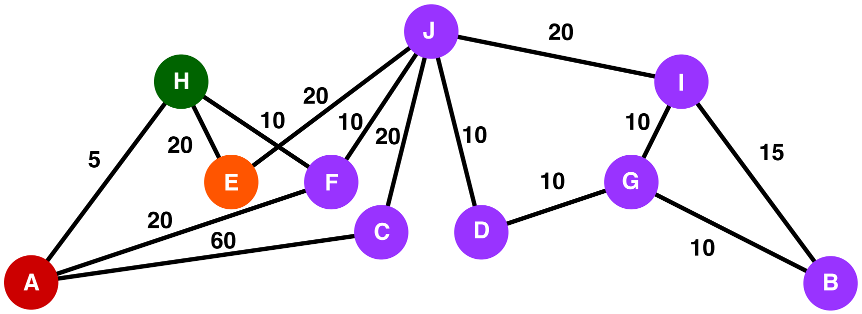


From the source node H, the first move on the optimal path to any node are A (red), E(orange) and F (purple).

Background
Compressed Path Databases (CPD) Heuristic [1]

- Compressed Path Databases
  - Construction:
    - First move table:
    - Compression:
  - First move extraction:
    - A binary search over compressed RLE string.
- Reverse Path Databases

Ordering	H	A	F	J	I	B	G	D	C	E
H	*	A	F	F	F	F	F	F	F	E
A	H	*	H	H	H	H	H	H	H	H
J	F	F	F	*	I	D	D	D	C	E

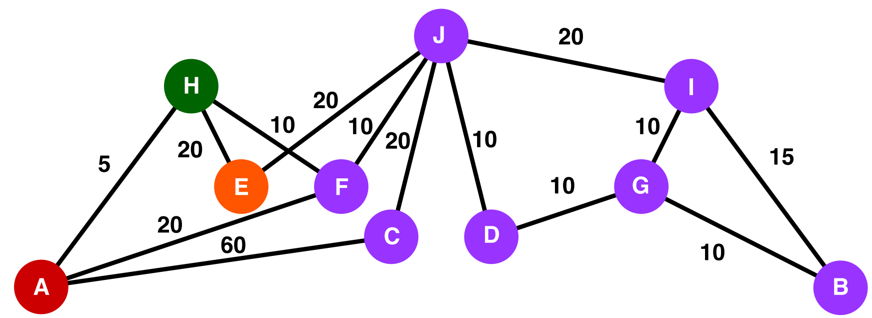


From the source node H, the first move on the optimal path to any node are A (red), E(orange) and F (purple).

Background
Compressed Path Databases (CPD) Heuristic [1]

- Compressed Path Databases
  - Construction:
    - First move table:
    - Compression:
  - First move extraction:
    - A binary search over compressed RLE string.
- Reverse Path Databases
  - Construction:
    - Reverse first move table [4].

Ordering	H	A	F	J	I	B	G	D	C	E
H	*	A	F	F	F	F	F	F	F	E
A	H	*	H	H	H	H	H	H	H	H
J	F	F	F	*	I	D	D	D	C	E



From the source node H, the first move on the optimal path to any node are A (red), E(orange) and F (purple).

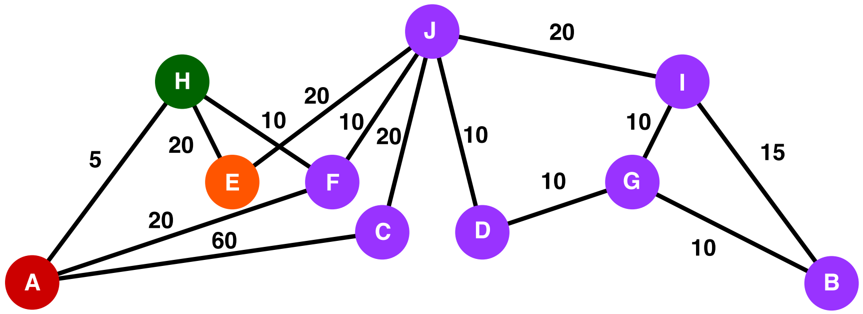
Ordering	H	A	F	J	I	B	G	D	C	E
H	H	H	H	F	J	G	D	J	J	H
A	A	A	H	F	J	G	D	J	J	H
J	F	H	J	J	J	G	D	J	J	J

A reverse first move table which records the for every d ∈ V , the first move on the shortest path from d to s

Background
Compressed Path Databases (CPD) Heuristic [1]

- Compressed Path Databases
  - Construction:
    - First move table:
    - Compression:
  - First move extraction:
    - A binary search over compressed RLE string.
- Reverse Path Databases
  - Construction:
    - Reverse first move table [4].
  - First move extraction:
    - Accessing the first move in O(1).

Ordering	H	A	F	J	I	B	G	D	C	E
H	*	A	F	F	F	F	F	F	F	E
A	H	*	H	H	H	H	H	H	H	H
J	F	F	F	*	I	D	D	D	C	E



From the source node H, the first move on the optimal path to any node are A (red), E(orange) and F (purple).

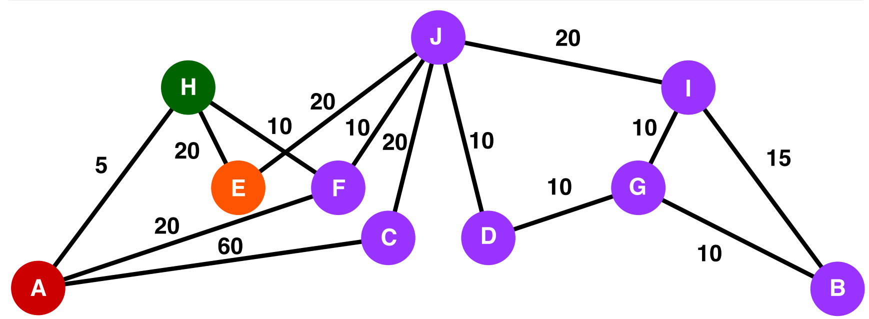
Ordering	H	A	F	J	I	B	G	D	C	E
H	H	H	H	F	J	G	D	J	J	H
A	A	A	H	F	J	G	D	J	J	H
J	F	H	J	J	J	G	D	J	J	J

A reverse first move table which records the for every d ∈ V , the first move on the shortest path from d to s

Background
Compressed Path Databases (CPD) Heuristic [1]

- Compressed Path Databases
  - Construction:
    - First move table:
    - Compression:
  - First move extraction:
    - A binary search over compressed RLE string.
- Reverse Path Databases
  - Construction:
    - Reverse first move table [4].
  - First move extraction:
    - Accessing the first move in O(1).
- Heuristic:
  - The shortest path extracted from CPD between s and d is a valid lower-bound.

Ordering	H	A	F	J	I	B	G	D	C	E
H	*	A	F	F	F	F	F	F	F	E
A	H	*	H	H	H	H	H	H	H	H
J	F	F	F	*	I	D	D	D	C	E



From the source node H, the first move on the optimal path to any node are A (red), E(orange) and F (purple).

Ordering	H	A	F	J	I	B	G	D	C	E
H	H	H	H	F	J	G	D	J	J	H
A	A	A	H	F	J	G	D	J	J	H
J	F	H	J	J	J	G	D	J	J	J

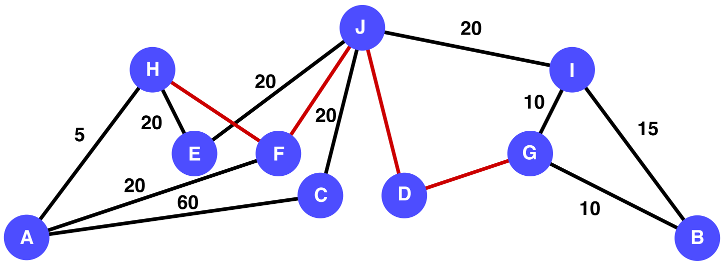
A reverse first move table which records the for every d ∈ V , the first move on the shortest path from d to s

Background
Time Dependent Contraction Hierarchy (TCH) [5]

- Time Dependent Contraction Hierarchy:

Background
Time Dependent Contraction Hierarchy (TCH) [5]

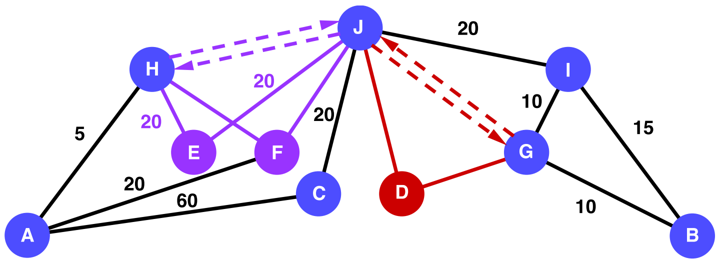
- Time Dependent Contraction Hierarchy:
  - Construction:
    - Apply a total lex order L.



The lex order L is the alphabetical order shown in the figure.

Time Dependent Contraction Hierarchy (TCH) [5]

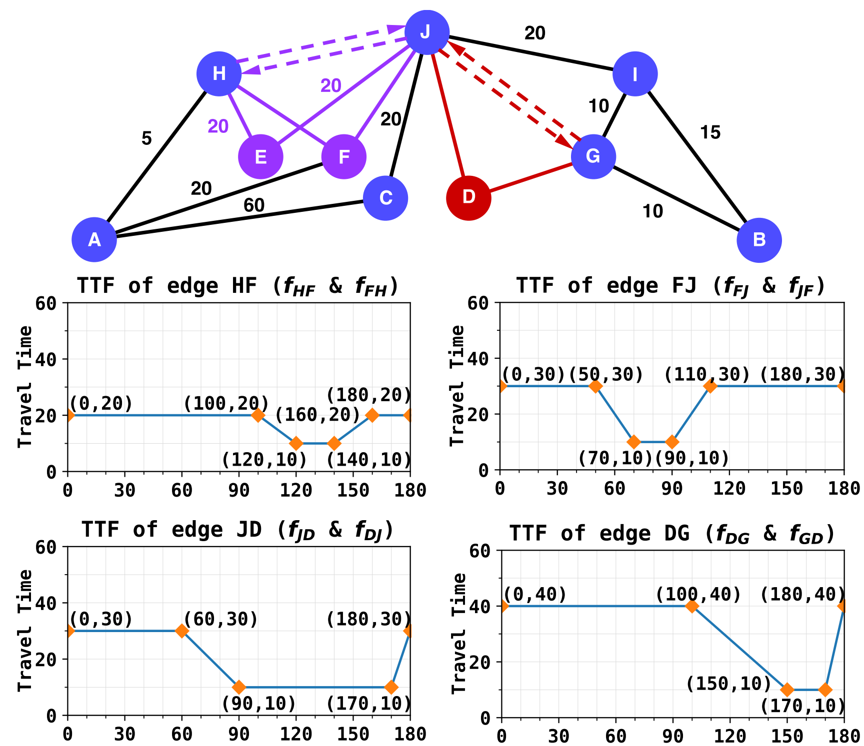
- Time Dependent Contraction Hierarchy:
  - Construction:
    - Apply a total lex order L.
    - Contraction:
      - W.r.t. L, choose the least node v from the graph.



We show the result for contracting nodes E and F in purple, and D in red. Dashed edge are the shortcut edges.

Background
Time Dependent Contraction Hierarchy (TCH) [5]

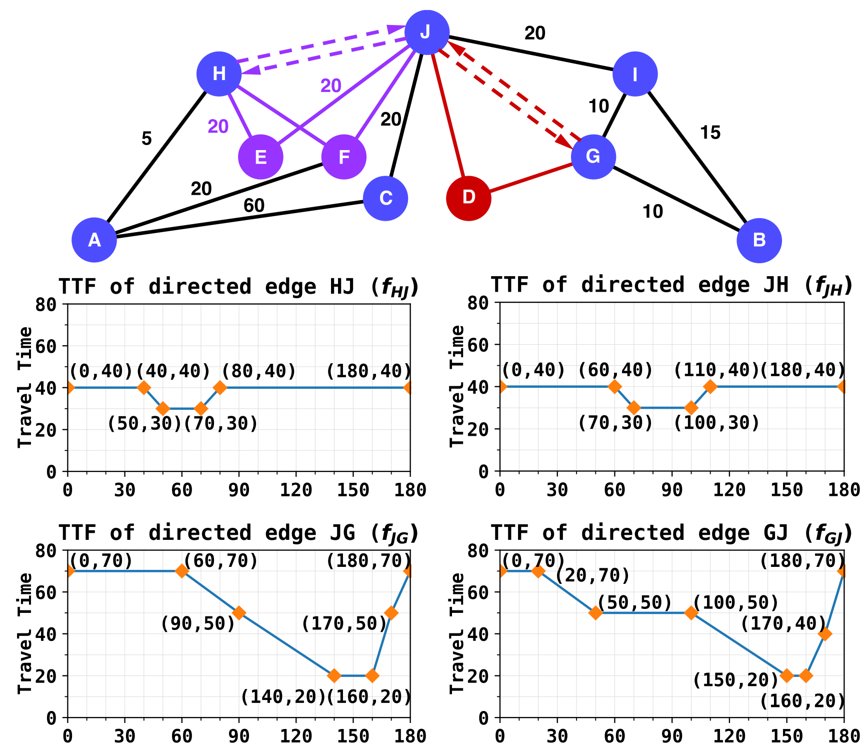
- Time Dependent Contraction Hierarchy:
  - Construction:
    - Apply a total lex order L.
  - Contraction:
    - W.r.t. L, choose the least node v from the graph.
    - Add a shortcut edge (u, w) between each pair of in-neighbour u and out-neighbour w of v:
      - » v $<_L$ u & v $<_L$ w.
      - » ∃ t ∈ T, v ∈ sp(u,w,t).



We show the result for contracting nodes E and F in purple, and D in red. Dashed edge are the shortcut edges.

Background
Time Dependent Contraction Hierarchy (TCH) [5]

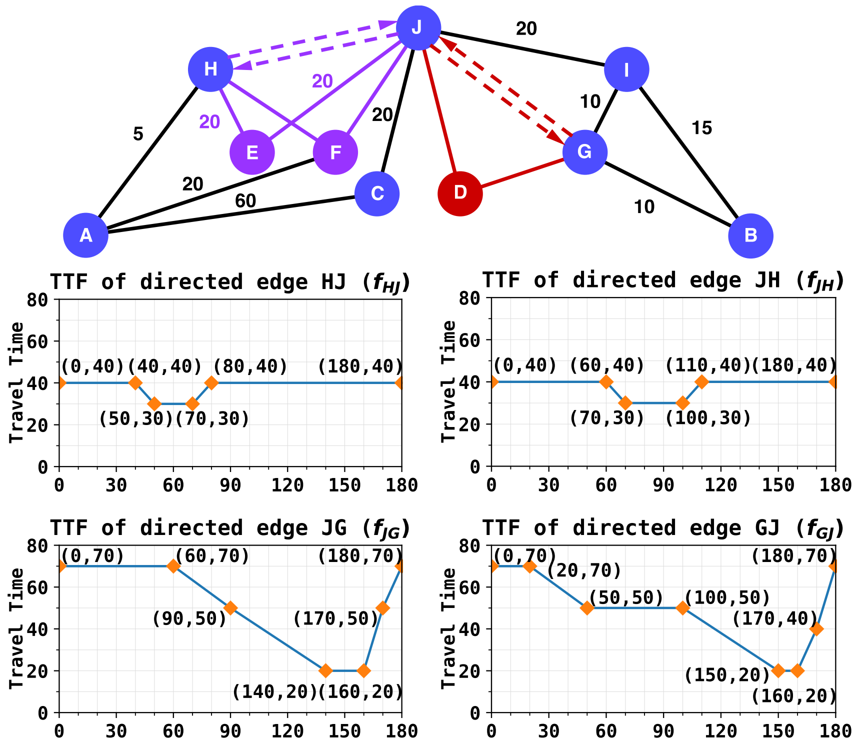
- Time Dependent Contraction Hierarchy:
  - Construction:
    - Apply a total lex order L.
    - Contraction:
      - W.r.t. L, choose the least node v from the graph.
      - Add a shortcut edge (u, w) between each pair of in-neighbour u and out-neighbour w of v:
        - » v $<_L$ u & v $<_L$ w.
        - » ∃ t ∈ T, v ∈ sp(u,w,t).
      - When add a shortcut edge (u,w) the TTF is computed as
        - » $fuw = fuv \circ fvw$ or $fuw = \min(fuw', fuv \circ fvw)$ if parallel edges existed.



We show the result for contracting nodes E and F in purple, and D in red. Dashed edge are the shortcut edges and their corresponding TTFs are shown in the figure below.

Background
Time Dependent Contraction Hierarchy (TCH) [5]

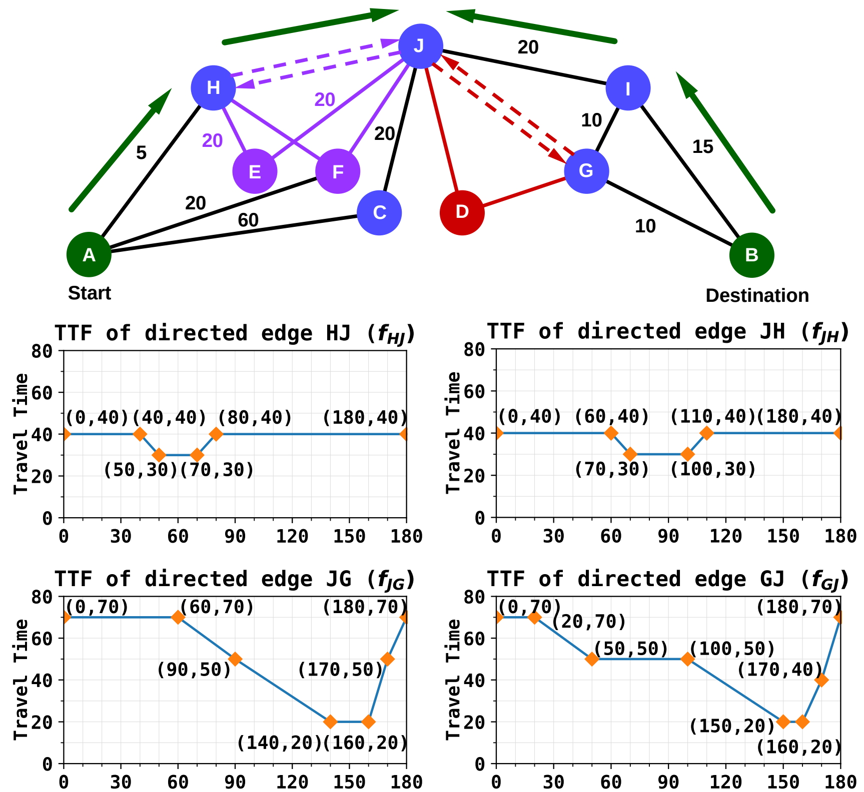
- Time Dependent Contraction Hierarchy:
  – Construction:
  – Search algorithm (BTCH):



We show the result for contracting nodes E and F in purple, and D in red. Dashed edge are the shortcut edges and their corresponding TTFs are shown in the figure below.

Background
Time Dependent Contraction Hierarchy (TCH) [5]

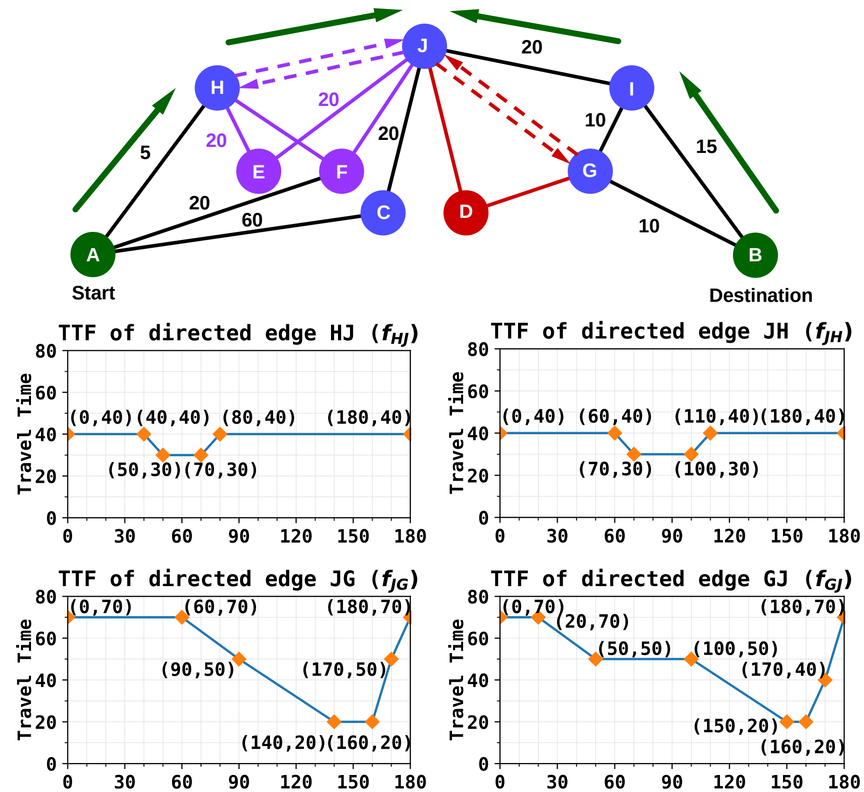
- Time Dependent Contraction Hierarchy:
  - Construction:
  - Search algorithm (BTCH):
    - Bi-directional search:



We show the result for contracting nodes E and F in purple, and D in red. Dashed edge are the shortcut edges and their corresponding TTFs are shown in the figure below.

Background
Time Dependent Contraction Hierarchy (TCH) [5]

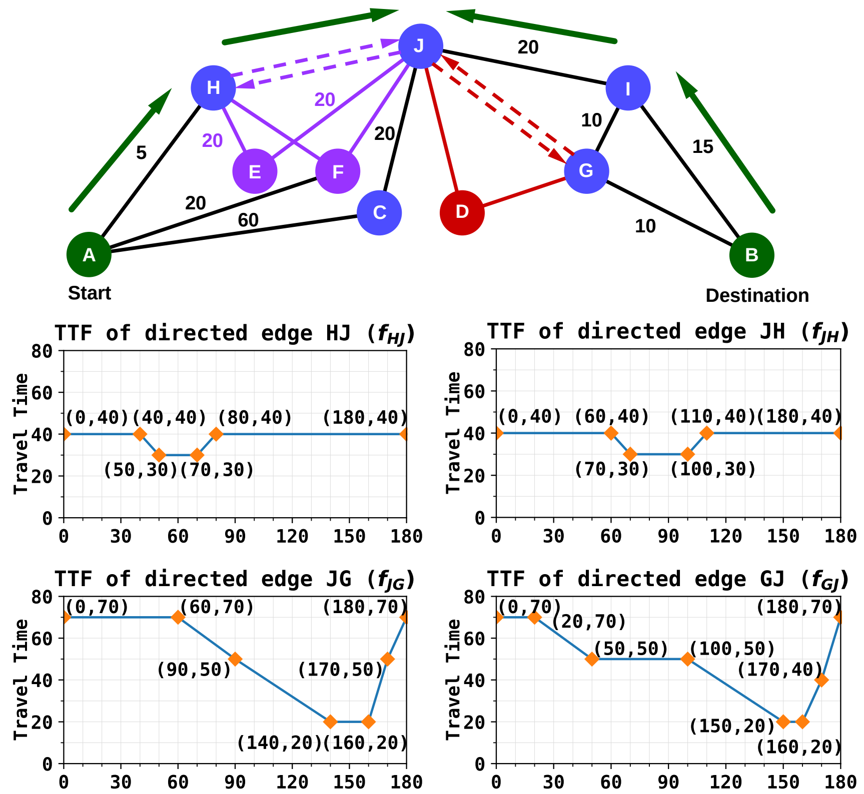
- Time Dependent Contraction Hierarchy:
  - Construction:
  - Search algorithm (BTCH):
    - Bi-directional search:
      - Forward direction:
        - » Run a time-dependent Dijkstra search considering only the outgoing edges in E↑.



We show the result for contracting nodes E and F in purple, and D in red. Dashed edge are the shortcut edges and their corresponding TTFs are shown in the figure below.

Background
Time Dependent Contraction Hierarchy (TCH) [5]

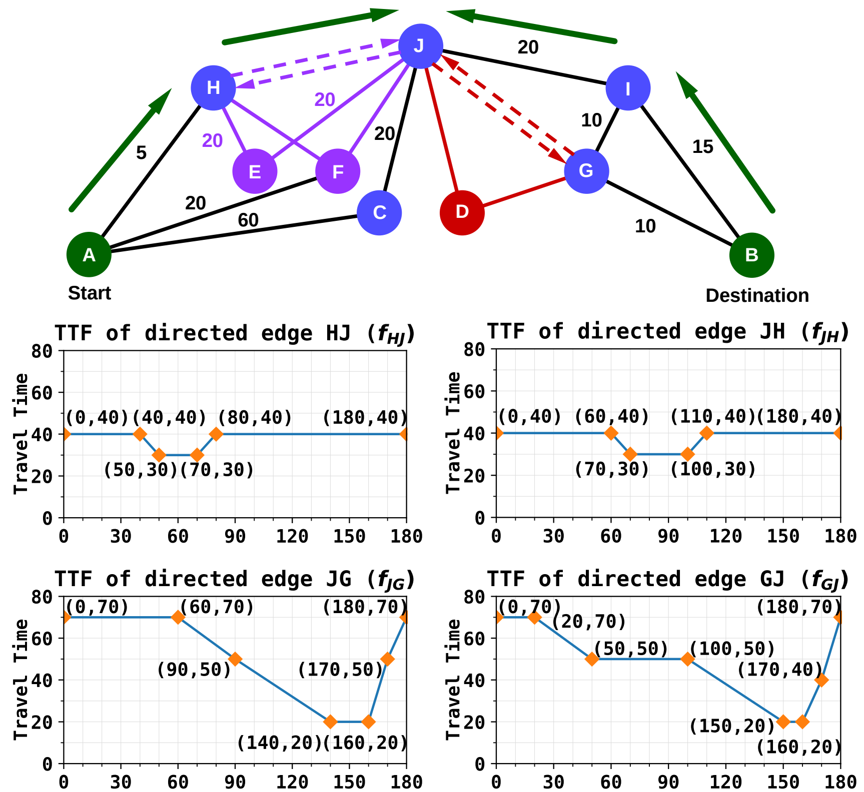
- Time Dependent Contraction Hierarchy:
  - Construction:
  - Search algorithm (BTCH):
    - Bi-directional search:
      - Forward direction:
        - » Run a time-dependent Dijkstra search considering only the outgoing edges in E↑.
      - Backward direction:
        - » Run a static Dijkstra search considering only the incoming edges in E↓. Meanwhile, mark the edge traversed as Etrv.



We show the result for contracting nodes E and F in purple, and D in red. Dashed edge are the shortcut edges and their corresponding TTFs are shown in the figure below.

Background
Time Dependent Contraction Hierarchy (TCH) [5]

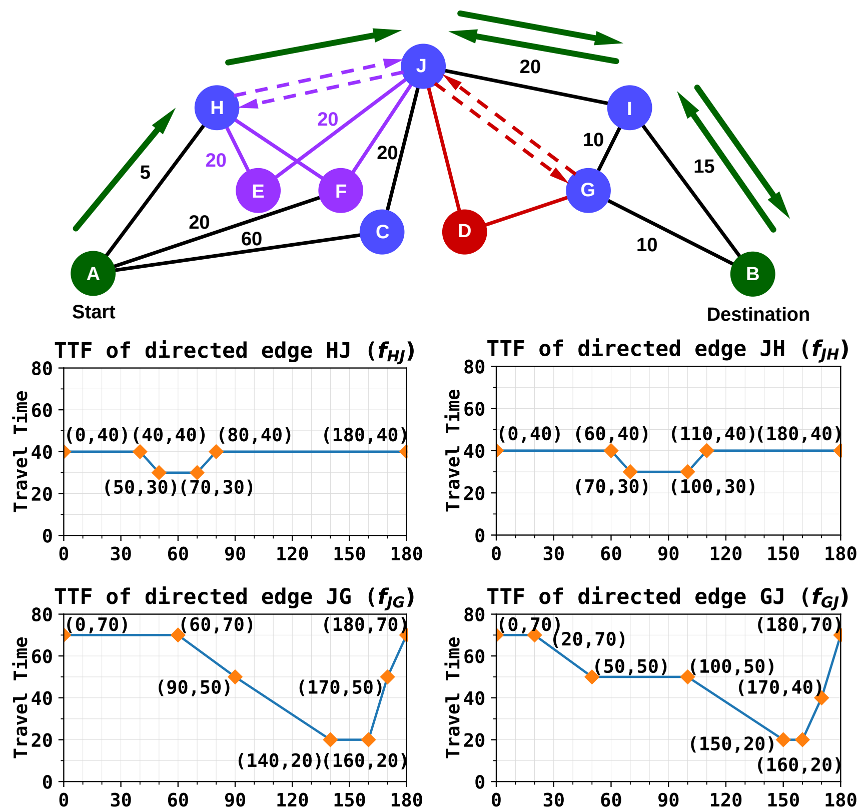
- Time Dependent Contraction Hierarchy:
  - Construction:
  - Search algorithm (BTCH):
    - Bi-directional search:
      - Forward direction:
        - » Run a time-dependent Dijkstra search considering only the outgoing edges in E↑.
      - Backward direction:
        - » Run a static Dijkstra search considering only the incoming edges in E↓. Meanwhile, mark the edge traversed as Etrv.
      - When the search meet at k:
        - » Compute the upper-bound
          - fwd(s,k) + upper(k,d)
        - » Compute the lower-bound
          - fwd(s,k) + lower(k,d)



We show the result for contracting nodes E and F in purple, and D in red. Dashed edge are the shortcut edges and their corresponding TTFs are shown in the figure below.

Background
Time Dependent Contraction Hierarchy (TCH) [5]

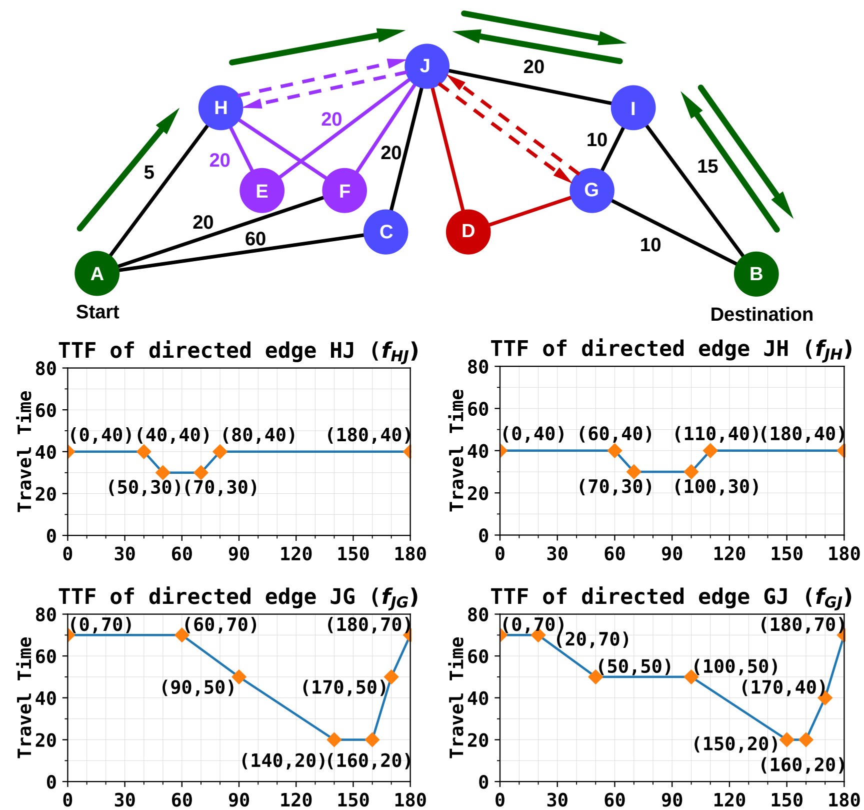
- Time Dependent Contraction Hierarchy:
  - Construction:
  - Search algorithm (BTCH):
    - Bi-directional search:
    - Forward search:
      - Inserting each apex nodes k back to queue.



We show the result for contracting nodes E and F in purple, and D in red. Dashed edge are the shortcut edges and their corresponding TTFs are shown in the figure below.

Background
Time Dependent Contraction Hierarchy (TCH) [5]

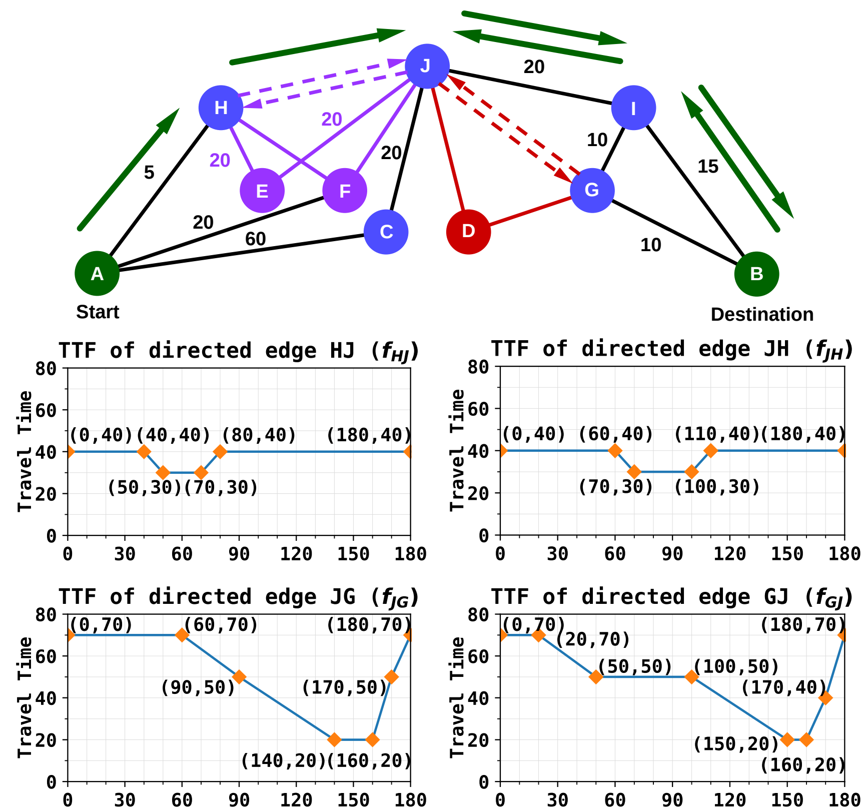
- Time Dependent Contraction Hierarchy:
  - Construction:
  - Search algorithm (BTCH):
    - Bi-directional search:
    - Forward search:
      - Inserting each apex nodes k back to queue.
      - Continue the forward time-dependent Dijkstra search by considering only the edges Etrv.



We show the result for contracting nodes E and F in purple, and D in red. Dashed edge are the shortcut edges and their corresponding TTFs are shown in the figure below.

Time Dependent Contraction Hierarchy (TCH) [5]

- Time Dependent Contraction Hierarchy:
  - Construction:
  - Search algorithm (BTCH):
    - Bi-directional search:
    - Forward search:
      - Inserting each apex nodes k back to queue.
      - Continue the forward time-dependent Dijkstra search by considering only the edges Etrv.
    - Extract the path and unpack.



We show the result for contracting nodes E and F in purple, and D in red. Dashed edge are the shortcut edges and their corresponding TTFs are shown in the figure below.

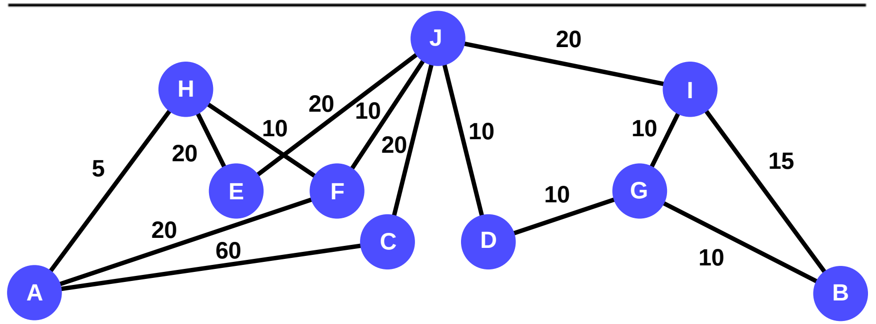
Our Approach
Improving TCH using Heuristic

- Combing BTCH with landmark:

Our Approach
Improving TCH using Heuristic

- Combing BTCH with landmark:
  - Bi-directional search:
    - landmark heuristic [6]:

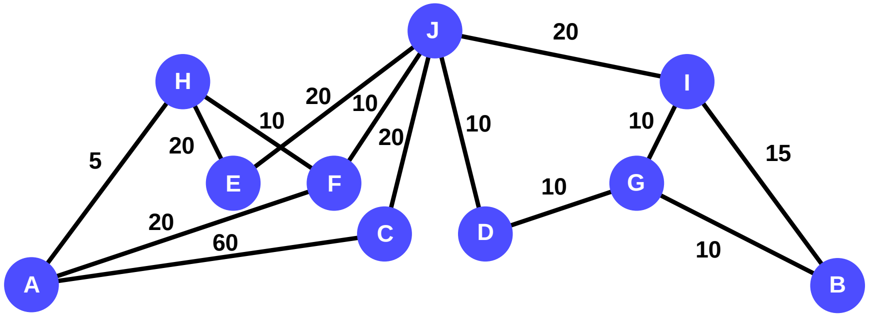
Ordering	H	A	F	J	I	B	G	D	C	E
d(A,_), d(_,A)	5	0	15	25	45	55	45	35	45	25
d(B,_), d(_,B)	50	55	40	30	15	0	10	20	50	50
d(J,_), d(_,J)	20	25	10	0	20	30	20	10	20	20



An example of static graph, where the travel cost of each edge is the free-flow cost of corresponding TTF.

- Combing BTCH with landmark:
  - Bi-directional search:
    - landmark heuristic [6]:
      - landmark(vi,vj) = max l∈L {max (d(vi,l) − d(vj,l), d(l,vi) − d(l,vj))}

Ordering	H	A	F	J	I	B	G	D	C	E
d(A,_), d(_,A)	5	0	15	25	45	55	45	35	45	25
d(B,_), d(_,B)	50	55	40	30	15	0	10	20	50	50
d(J,_), d(_,J)	20	25	10	0	20	30	20	10	20	20

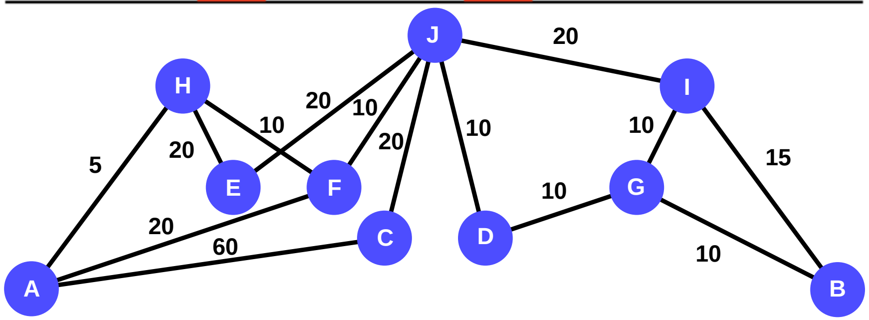


An example of static graph, where the travel cost of each edge is the free-flow cost of corresponding TTF.

Our Approach
Improving TCH using Heuristic

- Combing BTCH with landmark:
  - Bi-directional search:
    - landmark heuristic [6]:
      - landmark(vi,vj) = max l∈L {max (d(vi,l) − d(vj,l), d(l,vi) − d(l,vj))}
      - E.g., landmark(H,I) = max(|5 - 45|, |50 - 15|, |20 - 20|) = 35

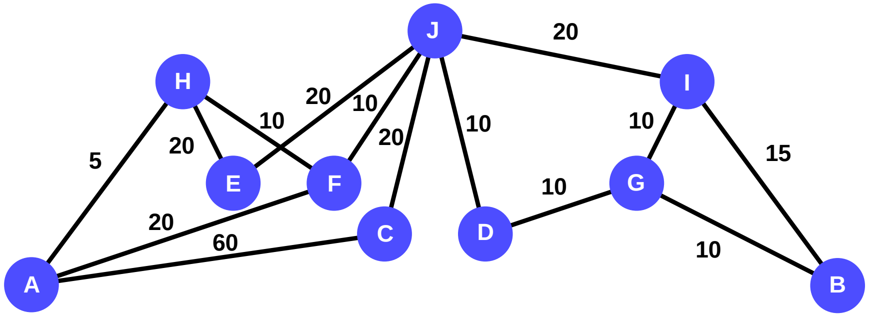
Ordering	H	A	F	J	I	B	G	D	C	E
d(A,_), d(_,A)	5	0	15	25	45	55	45	35	45	25
d(B,_), d(_,B)	50	55	40	30	15	0	10	20	50	50
d(J,_), d(_,J)	20	25	10	0	20	30	20	10	20	20



An example of static graph, where the travel cost of each edge is the free-flow cost of corresponding TTF.

- Combing BTCH with landmark:
  - Bi-directional search:
    - landmark heuristic [6]:
      - landmark(vi,vj) = max l∈L {max (d(vi,l) − d(vj,l), d(l,vi) − d(l,vj))}
      - E.g., landmark(H,I) = max(|5 - 45|, |50 - 15|, |20 - 20|) = 35
  - Forward search:
    - Heuristic:
      - Directly reuse the lower-bound computed by backward search.

Ordering	H	A	F	J	I	B	G	D	C	E
d(A,_), d(_,A)	5	0	15	25	45	55	45	35	45	25
d(B,_), d(_,B)	50	55	40	30	15	0	10	20	50	50
d(J,_), d(_,J)	20	25	10	0	20	30	20	10	20	20



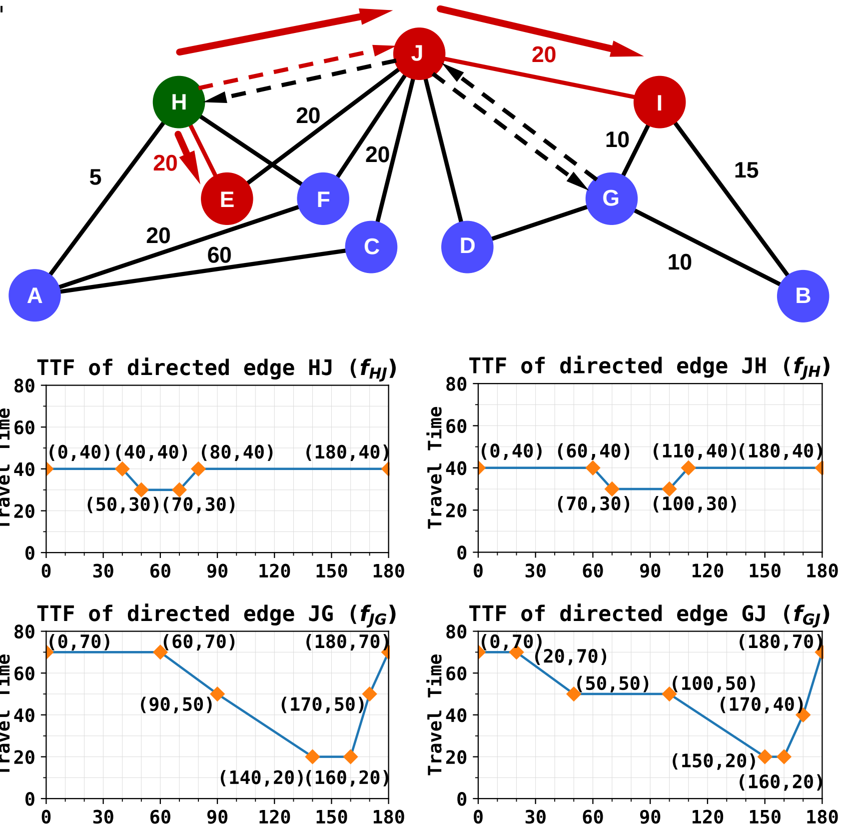
An example of static graph, where the travel cost of each edge is the free-flow cost of corresponding TTF.

Our Approach
Improving TCH using Heuristic

- Combing BTCH with landmark:
- Forward TCH Search with CPD-based Heuristic:

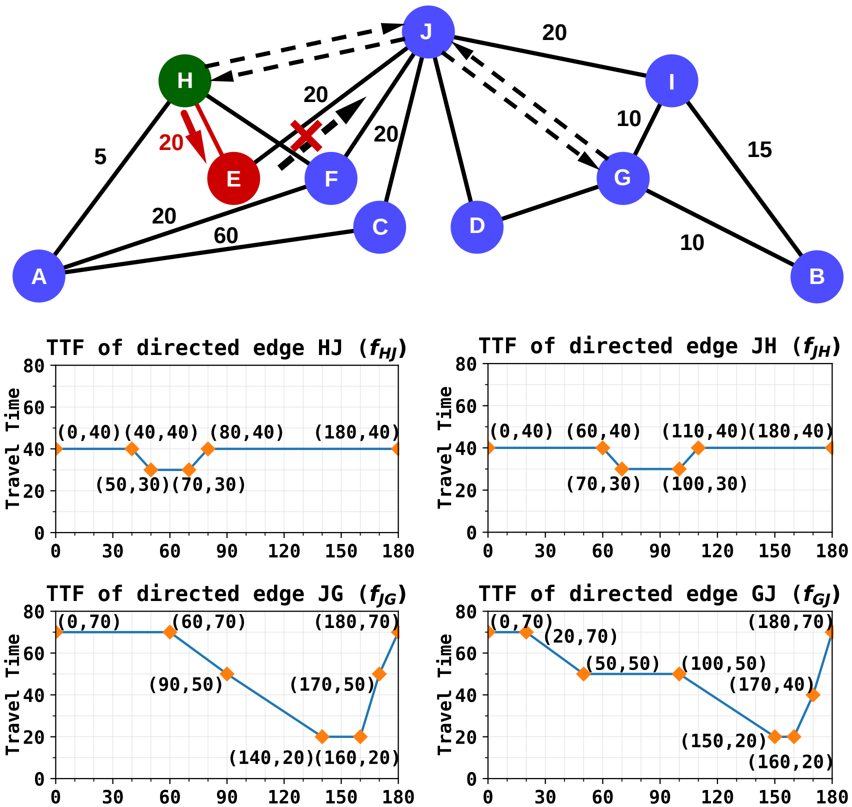
Our Approach
Improving TCH using Heuristic

- Combing BTCH with landmark:
- Forward TCH Search with CPD-based Heuristic:
  - Forward TCH Search:
    - Up-then-Down Policy.



Deconstructing the TCH-path gives following three cases: (i) up TCH-path: <H, J>; (ii) up-down TCH-path: <H, J, I>; (iii) down TCH-path: <H, E>.

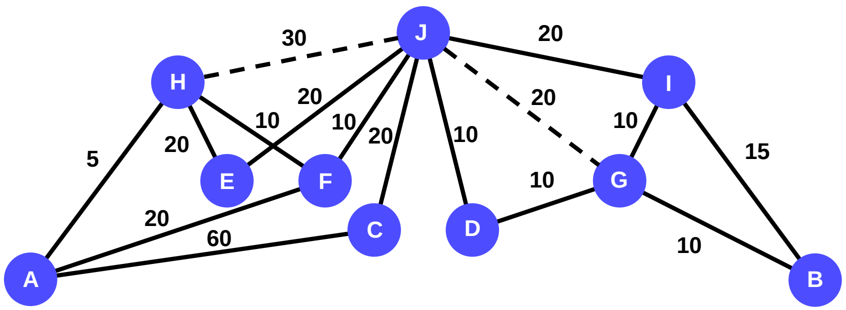
- Combing BTCH with landmark:
- Forward TCH Search with CPD-based Heuristic:
  - Forward TCH Search:
    - Up-then-Down Policy.



The up-then-down policy: the up successor J of E is pruned, because the predecessor H is lexically larger than E. (i.e., h $>_L$ E).

Our Approach
Improving TCH using Heuristic

- Combing BTCH with landmark:
- Forward TCH Search with CPD-based Heuristic:
  - Forward TCH Search:
    - Up-then-Down Policy
  - Heuristic:
    - TCH-CPD heuristic.

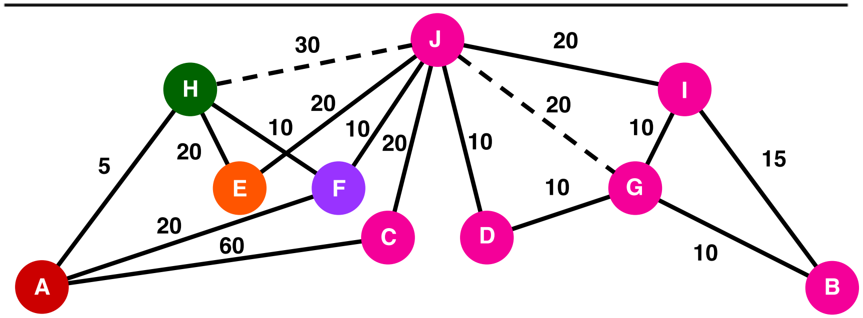


An example of static TCH, where the travel cost of each edge is the free-flow cost of corresponding TTF.

Our Approach
Improving TCH using Heuristic

- Combing BTCH with landmark:
- Forward TCH Search with CPD-based Heuristic:
  - Forward TCH Search:
    - Up-then-Down Policy.
  - Heuristic:
    - TCH-CPD heuristic.
      - Construction:
        - » Run a modified Dijkstra search to compute first move on the optimal TCH-path.

Ordering	H	A	F	J	I	B	G	D	C	E
H	*	A	F	J	J	J	J	J	J	E
A	H	*	H	F	F	F	F	F	F	H
J	H	F	F	*	I	G	G	D	C	E

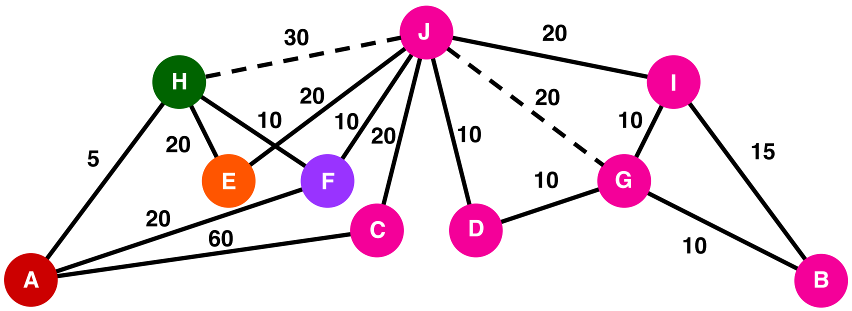


From the source node H, the optimal first move to any node are A (red), E(orange), F (purple) and J (pink)

Our Approach
Improving TCH using Heuristic

- Combing BTCH with landmark:
- Forward TCH Search with CPD-based Heuristic:
  - Forward TCH Search:
    - Up-then-Down Policy.
  - Heuristic:
    - TCH-CPD heuristic.
      - Construction:
        - » Run a modified Dijkstra search to compute first move on the optimal TCH-path.
        - » Compress by following the same procedures as CPD.

Ordering	H	A	F	J	I	B	G	D	C	E
H	*	A	F	J	J	J	J	J	J	E
A	H	*	H	F	F	F	F	F	F	H
J	H	F	F	*	I	G	G	D	C	E

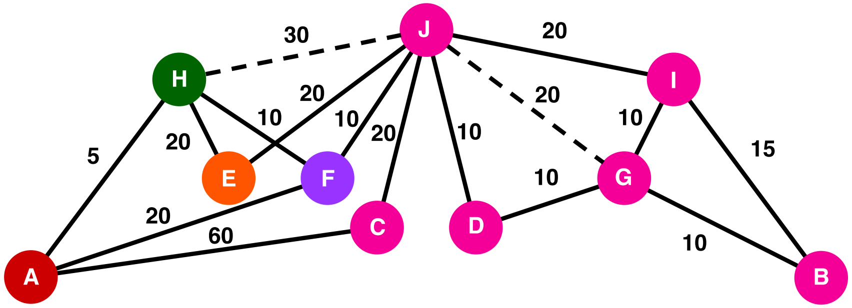


From the source node H, the optimal first move to any node are A (red), E(orange), F (purple) and J (pink)

Our Approach
Improving TCH using Heuristic

- Combing BTCH with landmark:
- Forward TCH Search with CPD-based Heuristic:
  - Forward TCH Search:
    - Up-then-Down Policy.
  - Heuristic:
    - TCH-CPD heuristic.
  - Enhancement:

Ordering	H	A	F	J	I	B	G	D	C	E
H	*	A	F	J	J	J	J	J	J	E
A	H	*	H	F	F	F	F	F	F	H
J	H	F	F	*	I	G	G	D	C	E

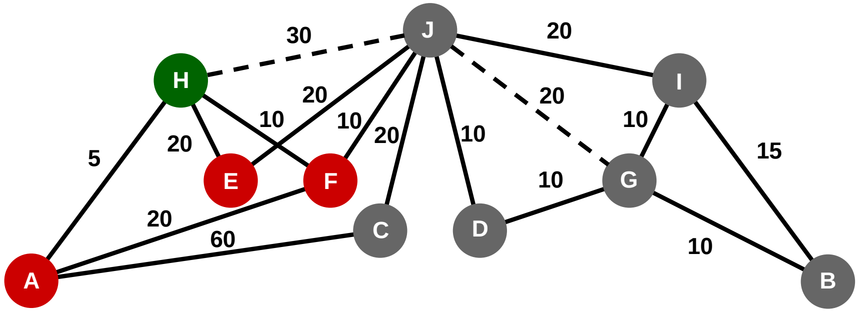


From the source node H, the optimal first move to any node are A (red), E(orange), F (purple) and J (pink)

Our Approach
Improving TCH using Heuristic

- Combing BTCH with landmark:
- Forward TCH Search with CPD-based Heuristic:
  - Forward TCH Search:
    - Up-then-Down Policy.
  - Heuristic:
    - TCH-CPD heuristic.
  - Enhancement:
    - Downward successor pruning:
      - Reachability Oracle.

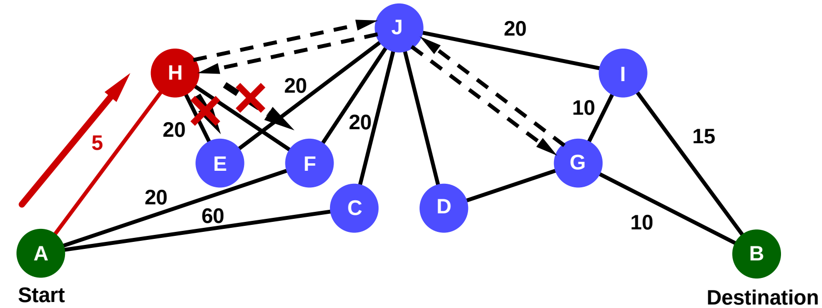
Ordering	H	A	F	J	I	B	G	D	C	E
H	*	T	T	F	F	F	F	F	F	T
A	F	*	F	F	F	F	F	F	F	F
J	T	T	T	*	T	T	T	T	T	T

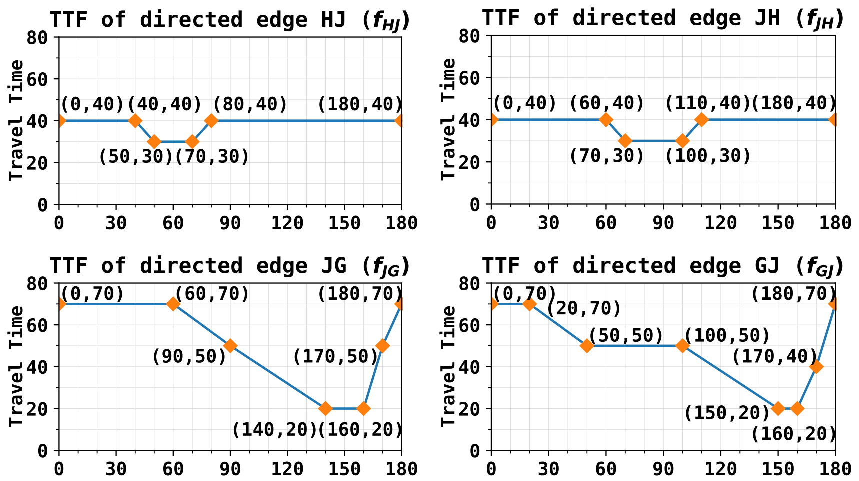


From the source node H, the downward reachable and non-reachable nodes are colored in red and grey respectively.

Our Approach
Improving TCH using Heuristic

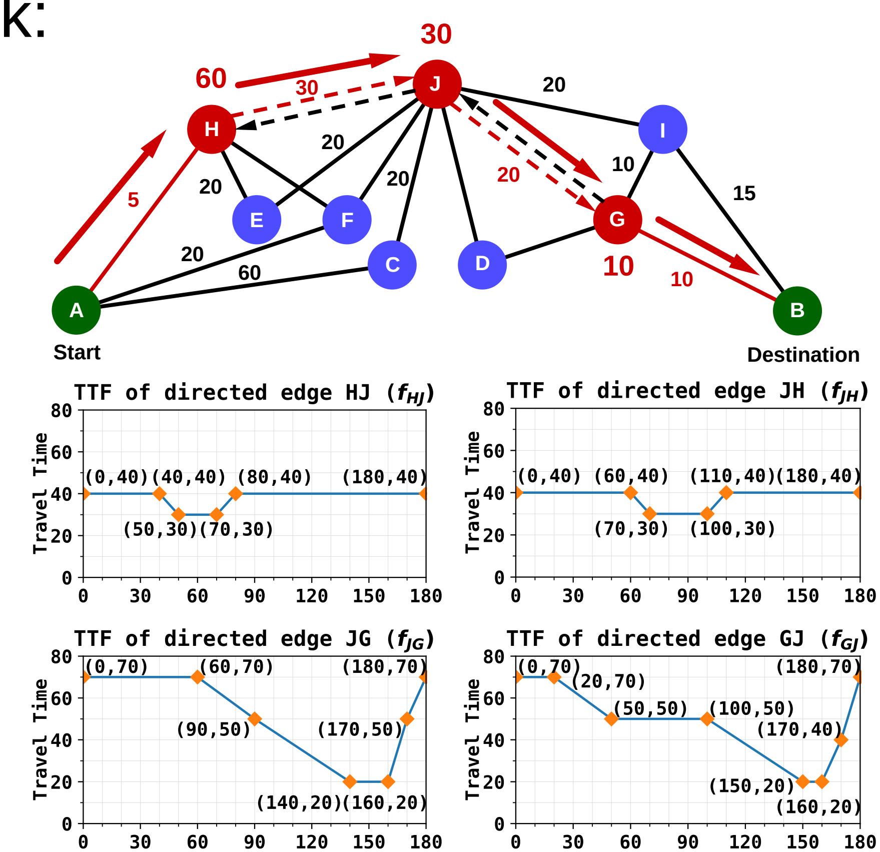
- Combing BTCH with landmark:
- Forward TCH Search with CPD-based Heuristic:
  - Forward TCH Search:
    - Up-then-Down Policy.
  - Heuristic:
    - TCH-CPD heuristic.
  - Enhancement:
    - Downward successor pruning:
      - Reachability Oracle.
      - During the search, prune the unreachable down successors.





The down successor E and F are pruned, because there exists no down path from E or F can reach the destination B.

- Combing BTCH with landmark:
- Forward TCH Search with CPD-based Heuristic:
  - Forward TCH Search:
    - Up-then-Down Policy.
  - Heuristic:
    - TCH-CPD heuristic.
  - Enhancement:
    - Downward successor pruning:
      - Reachability Oracle.
      - During the search, prune the unreachable down successors.
    - Cost Caching.

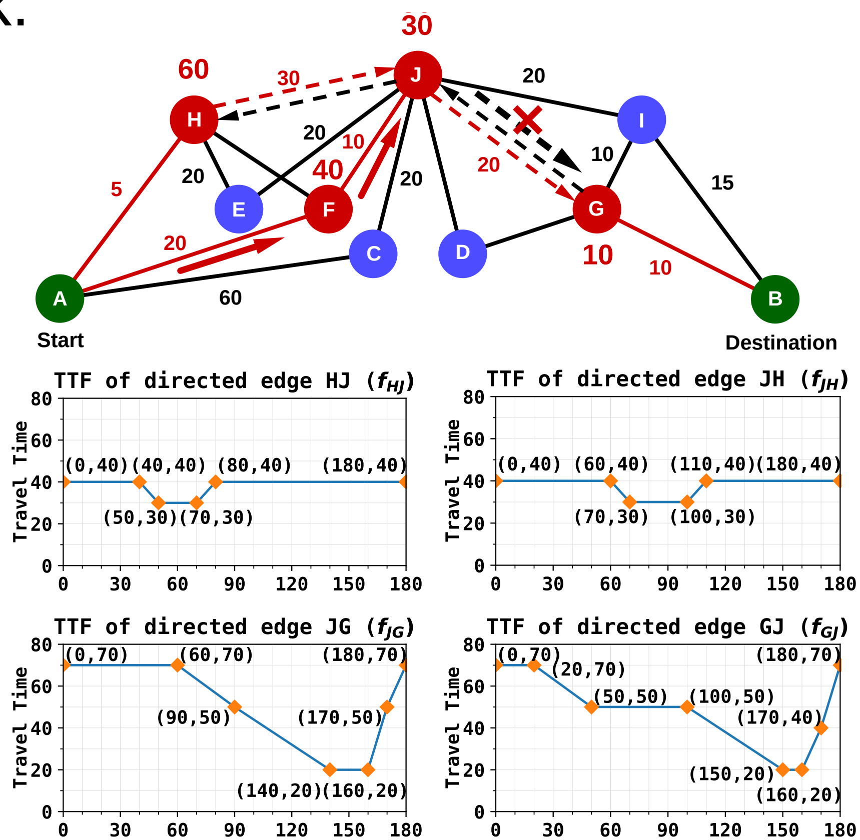


When compute the heuristic, we cache distance on each node when extracts the path from H to B.

Our Approach
Improving TCH using Heuristic

MONASH University

- Combing BTCH with landmark:
- Forward TCH Search with CPD-based Heuristic:
  - Forward TCH Search:
    - Up-then-Down Policy.
  - Heuristic:
    - TCH-CPD heuristic.
  - Enhancement:
    - Downward successor pruning:
      - Reachability Oracle.
      - During the search, prune the unreachable down successors.
    - Cost Caching.

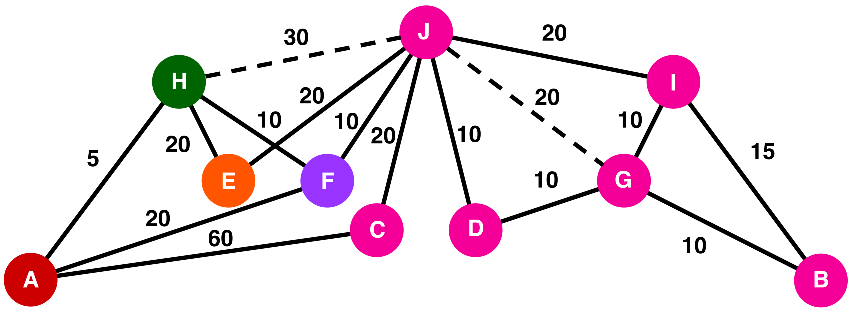


When compute heuristic for node F, the path extraction terminate at J as we already recorded the distance from J to B.

Our Approach
Improving TCH using Heuristic

- Combing BTCH with landmark:
- Forward TCH Search with CPD-based Heuristic:
  - Forward TCH Search:
    - Up-then-Down Policy.
  - Heuristic:
    - TCH-CPD heuristic.
  - Enhancement:
    - Downward successor pruning:
      - Reachability Oracle.
      - During the search, prune the unreachable down successors.
    - Cost Caching.
    - Reverse TCH path database.

Ordering	H	A	F	J	I	B	G	D	C	E
H	*	A	F	J	J	J	J	J	J	E
A	H	*	H	F	F	F	F	F	F	H
J	H	F	F	*	I	G	G	D	C	E



From the source node H, the optimal first move to any node are A (red), E(orange), F (purple) and J (pink)

Ordering	H	A	F	J	I	B	G	D	C	E
H	H	H	H	H	J	G	J	J	J	H
A	A	A	H	F	J	G	J	J	J	H
J	J	H	J	J	J	G	J	J	J	J

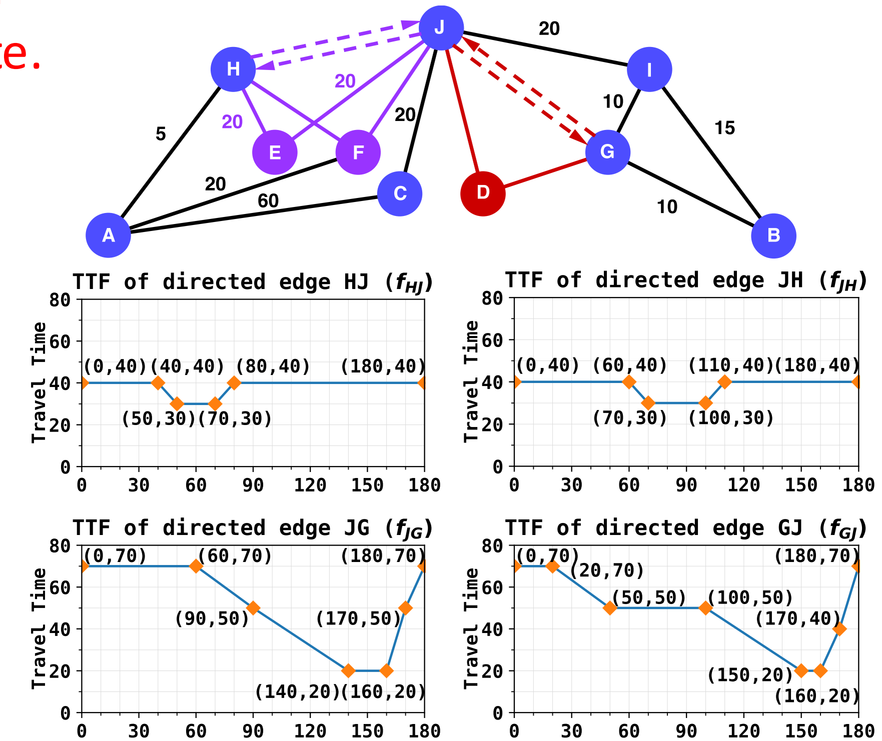
A reverse first move table which records the for every d ∈ V , the first move on the shortest path from d to s

Our Approach
Improving TCH by splitting the time domain

- Drawbacks of TCH:

Improving TCH by splitting the time domain

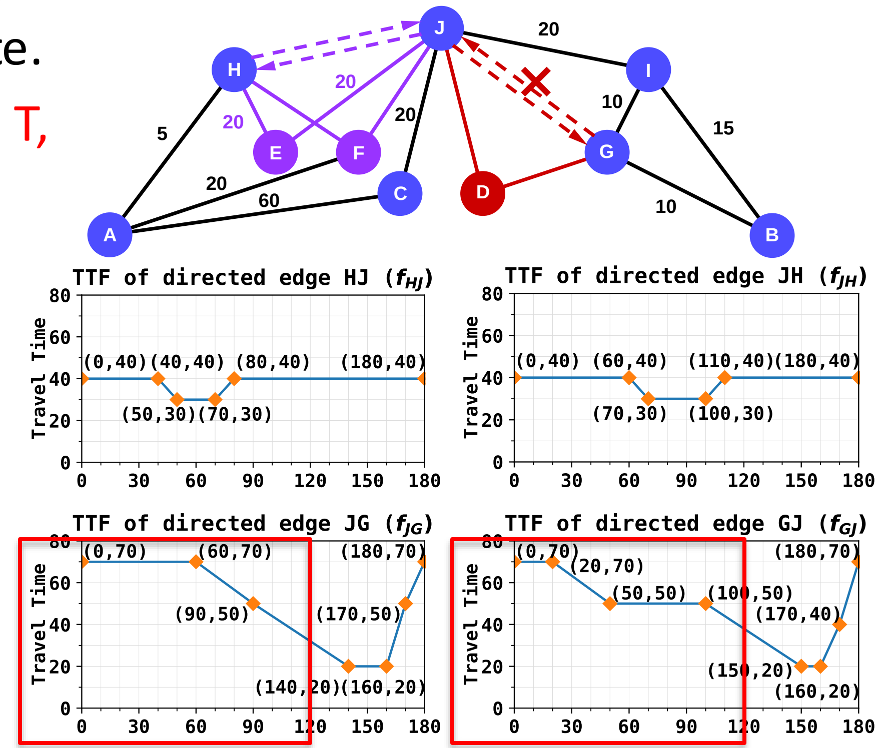
- Drawbacks of TCH:
  - TTF stores all interpolate points for entire T, thus are harder to evaluate.



We show the result for contracting nodes E and F in purple, and D in red. Dashed edge are the shortcut edges and their corresponding TTFs are shown in the figure below.

Improving TCH by splitting the time domain

- Drawbacks of TCH:
  - TTF stores all interpolate points for entire T, thus are harder to evaluate.
  - Shortcut edge are added for entire T, may be unnecessary for T' ⊆ T.



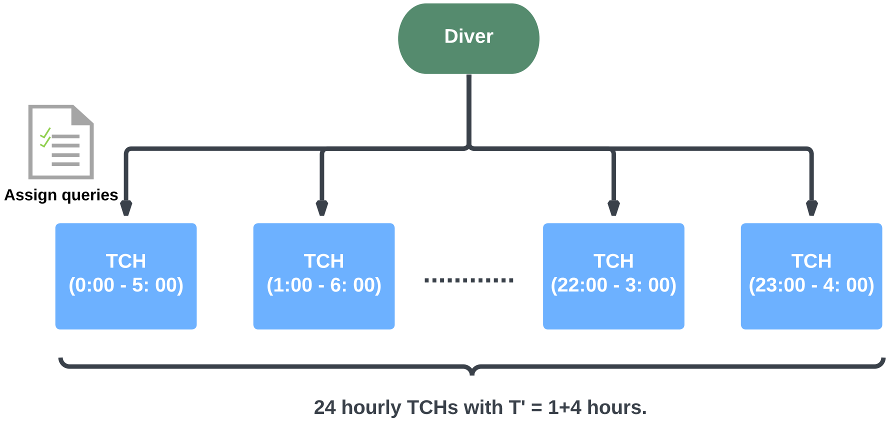
Assume a TCH is built for T = [0, 120], contracting the node D cannot add the shortcut JG, as <J,I,G> is a shorter path than <J,D,G> (i.e., 20 + 10 < min(fJD ∘ fDG)).

Our Approach
Improving TCH by splitting the time domain

- Drawbacks of TCH:
- Single layer TCH (STCH):

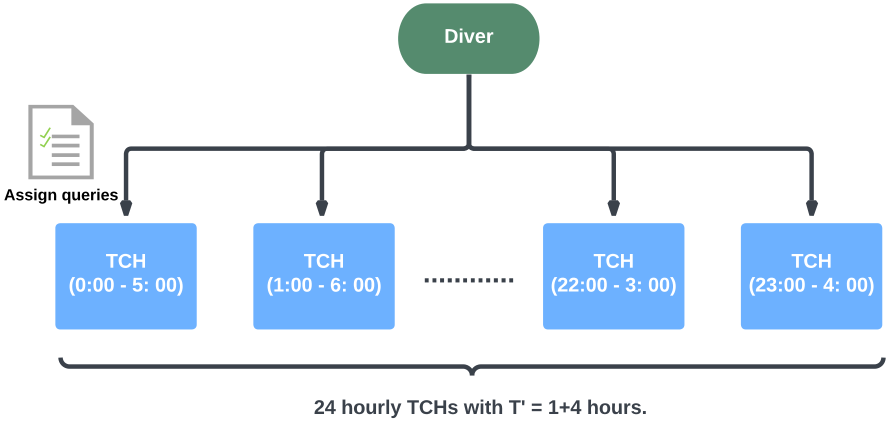
Our Approach
Improving TCH by splitting the time domain

- Drawbacks of TCH:
- Single layer TCH (STCH):
  - Divide the time domain T into n buckets.



Our Approach
Improving TCH by splitting the time domain

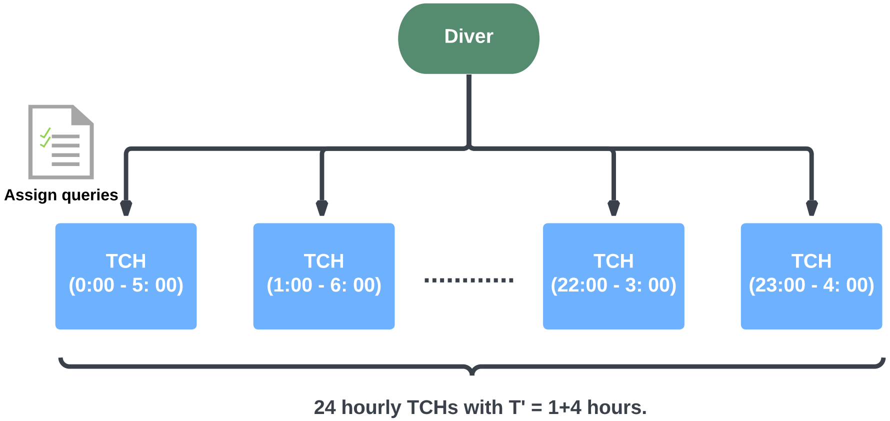
- Drawbacks of TCH:
- Single layer TCH (STCH):
  - Divide the time domain T into n buckets.
  - Built a TCH for each bucket:
    - T’ = T/n + U.



Our Approach
Improving TCH by splitting the time domain

MONASH University

- Drawbacks of TCH:
- Single layer TCH (STCH):
  - Divide the time domain T into n buckets.
  - Built a TCH for each bucket:
    - T’ = T/n + U.
  - Query:
    - Assign queries to the corresponding TCH.

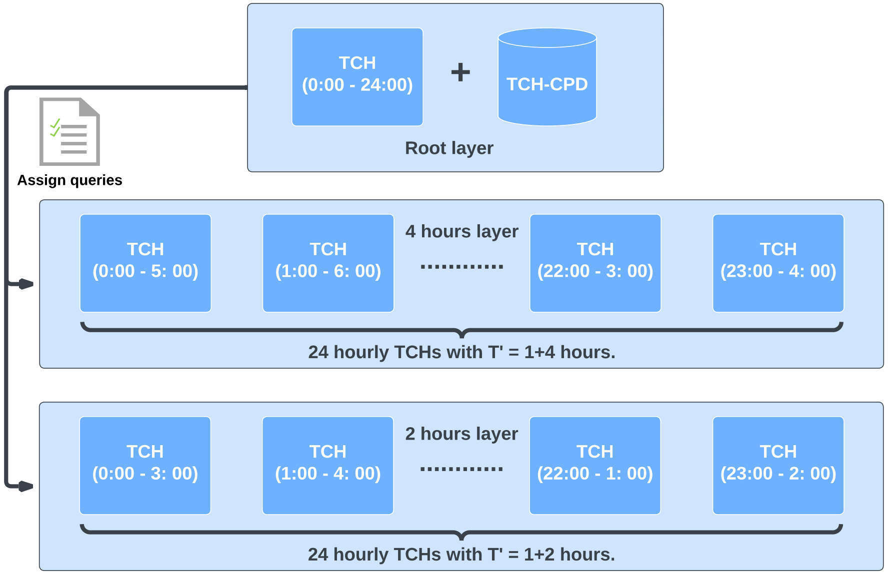


Our Approach
Improving TCH by splitting the time domain

- Drawbacks of TCH:
- Single layer TCH (STCH):
- Multi layer TCH (MTCH):

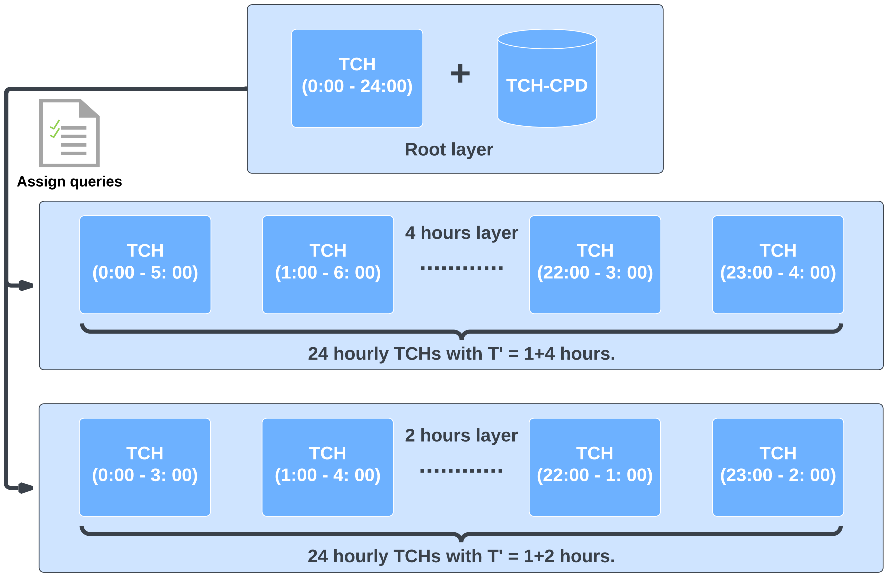
Our Approach
Improving TCH by splitting the time domain

- Drawbacks of TCH:
- Single layer TCH (STCH):
- Multi layer TCH (MTCH):
  - Root layer:
    - Build a full TCH and TCH-CPD.



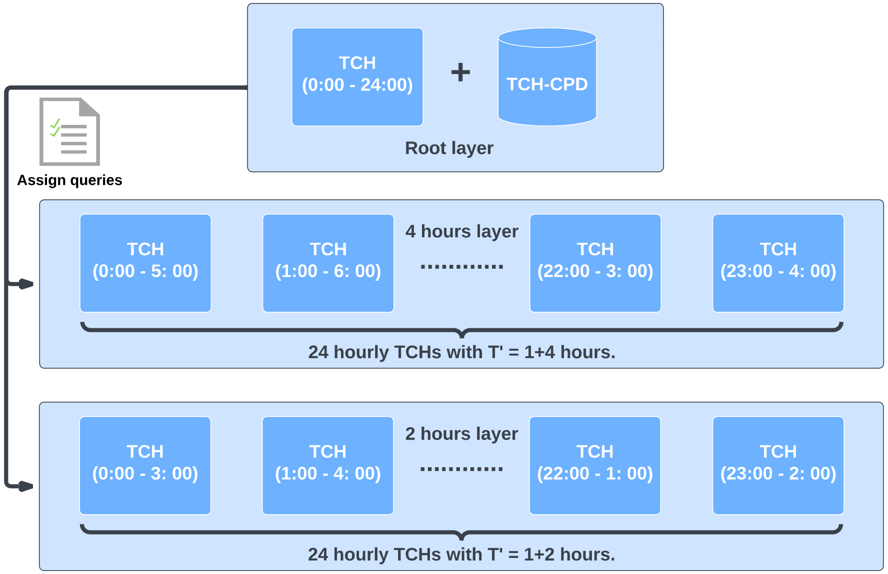
Our Approach
Improving TCH by splitting the time domain

- Drawbacks of TCH:
- Single layer TCH (STCH):
- Multi layer TCH (MTCH):
  - Root layer:
    - Build a full TCH and TCH-CPD.
  - For each layer under root layer:
    - Build STCH with any arbitrary U.



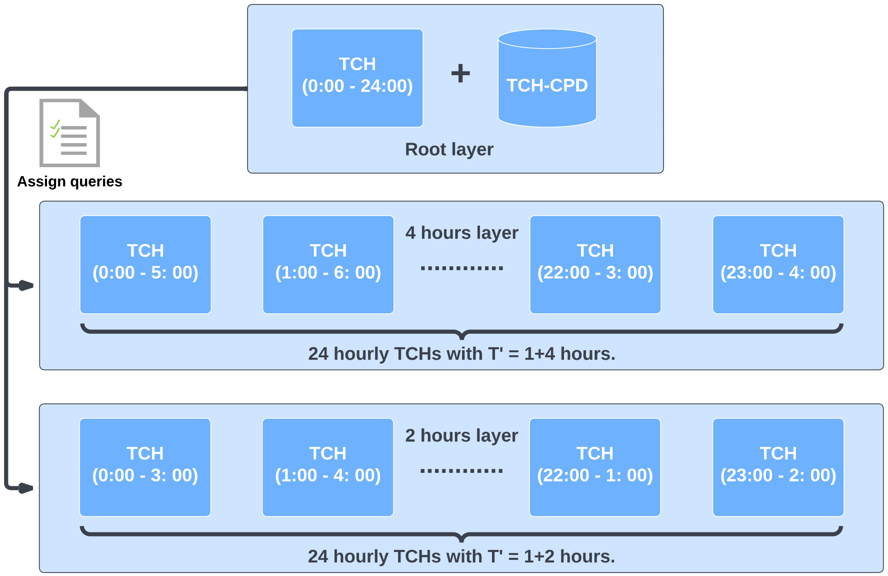
Our Approach
Improving TCH by splitting the time domain

- Drawbacks of TCH:
- Single layer TCH (STCH):
- Multi layer TCH (MTCH):
  - Root layer:
    - Build a full TCH and TCH-CPD.
  - For each layer under root layer:
    - Build STCH with any arbitrary U.
  - Query:
    - TCH-CPD to compute an upper-bound U(s,d).



Our Approach
Improving TCH by splitting the time domain

- Drawbacks of TCH:
- Single layer TCH (STCH):
- Multi layer TCH (MTCH):
  - Root layer:
    - Build a full TCH and TCH-CPD.
  - For each layer under root layer:
    - Build STCH with any arbitrary U.
  - Query:
    - TCH-CPD to compute an upper-bound U(s,d).
    - Assign queries to the TCH with minimal T' that covers [t, t+U(s,d)]

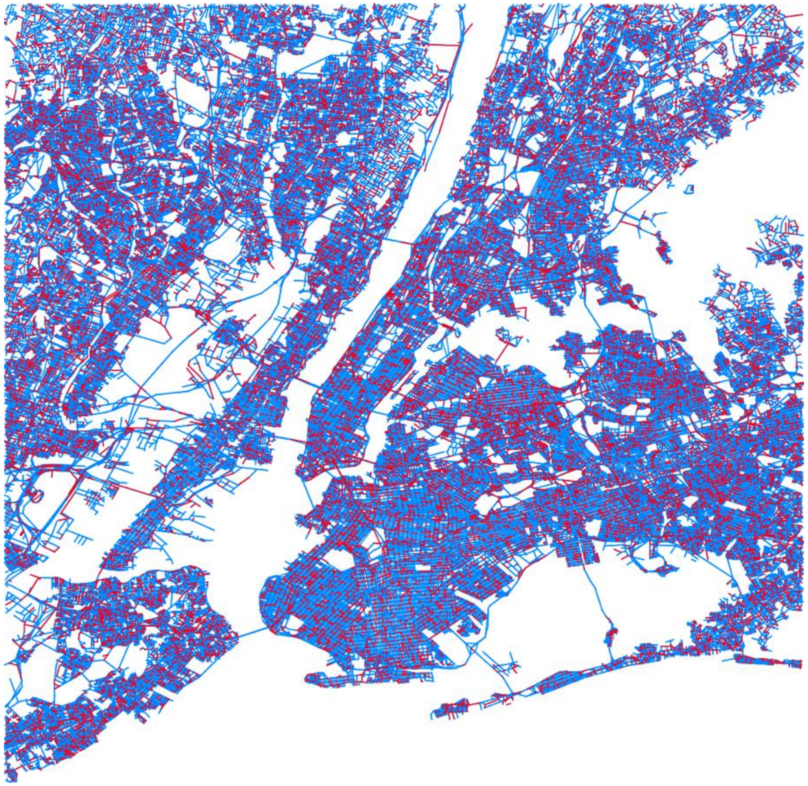


Our Approach
Experimental Results

- Experimental Results:

Our Approach
Experimental Results

- Experimental Results:
  - Benchmarks:
    - Real-world time-dependent road network [7].



Real-world dataset: New York. The edges that have TTF are colored in red.

MONASH University

- Experimental Results:
  - Preprocessing cost:

Map	#V	#E	Build Time (Mins)								Memory (MB)							
			TCH				STCH		MTCH		TCH				STCH		MTCH	
			-	CPD	TCPD	RTPD	-	TCPD	-	TCPD	-	CPD	TCPD	RTPD	-	TCPD	-	TCPD
NY	96k	260k	1.72	2.92	2.94	3.21	2.99	31.87	9.04	95.12	269	346	353	9596	1279	3286	3193	9198

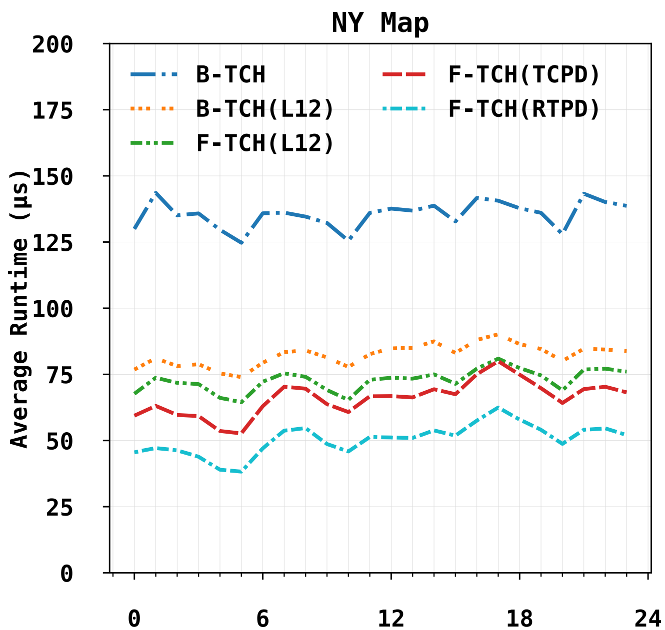
Our Approach
Experimental Results

- Experimental Results:
  - Preprocessing cost:

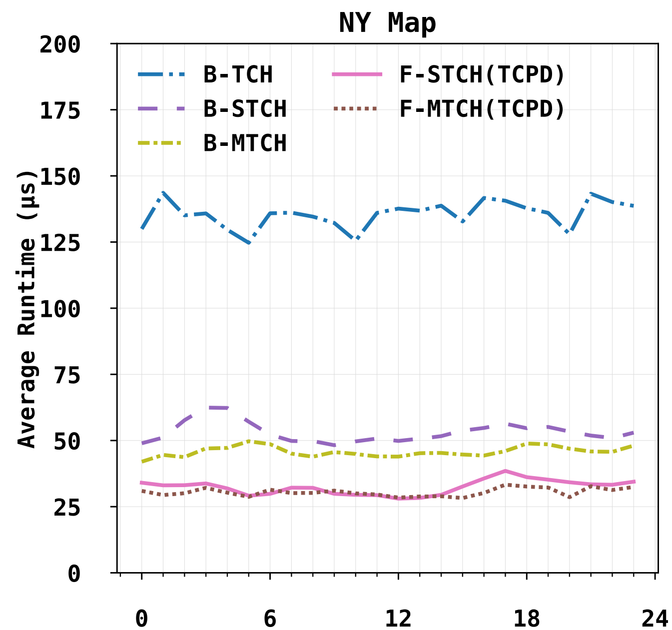
Map	#V	#E	Build Time (Mins)								Memory (MB)							
			TCH				STCH		MTCH		TCH				STCH		MTCH	
			-	CPD	TCPD	RTPD	-	TCPD	-	TCPD	-	CPD	TCPD	RTPD	-	TCPD	-	TCPD
NY	96k	260k	1.72	2.92	2.94	3.21	2.99	31.87	9.04	95.12	269	346	353	9596	1279	3286	3193	9198

  - Query performance:

Heuristic Search



Time Domain Splitting



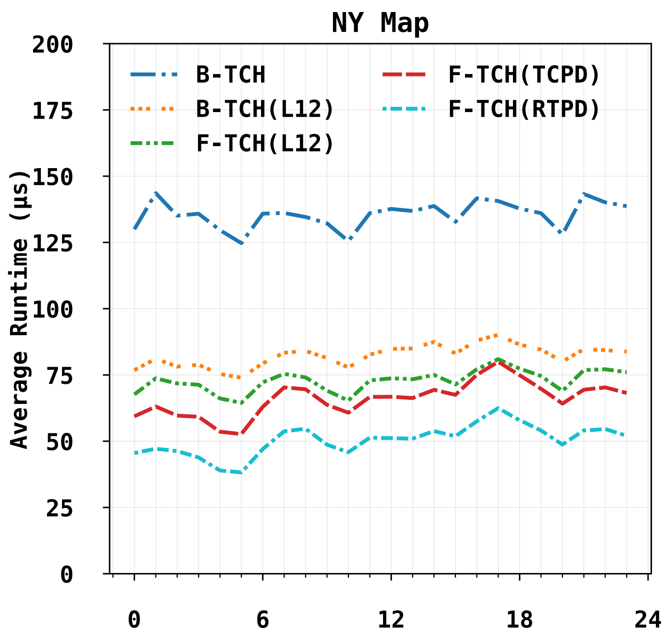
Our Approach
Experimental Results

- Experimental Results:
  - Preprocessing cost:

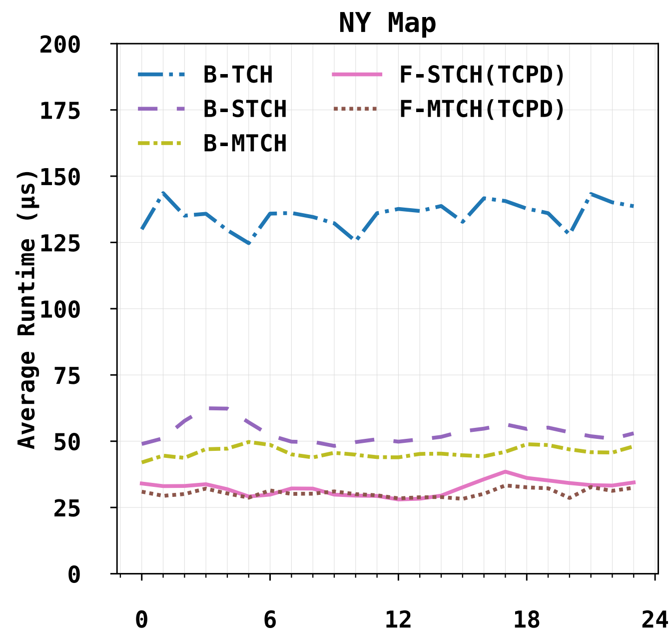
Map	#V	#E	Build Time (Mins)								Memory (MB)							
			TCH				STCH		MTCH		TCH				STCH		MTCH	
			-	CPD	TCPD	RTPD	-	TCPD	-	TCPD	-	CPD	TCPD	RTPD	-	TCPD	-	TCPD
NY	96k	260k	1.72	2.92	2.94	3.21	2.99	31.87	9.04	95.12	269	346	353	9596	1279	3286	3193	9198

  - Query performance:

Heuristic Search



Time Domain Splitting



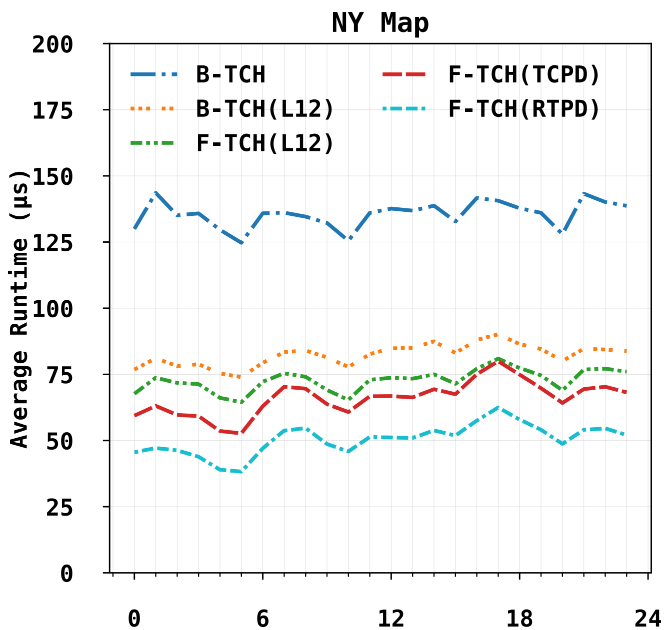
Our Approach
Experimental Results

- Experimental Results:
  - Preprocessing cost:

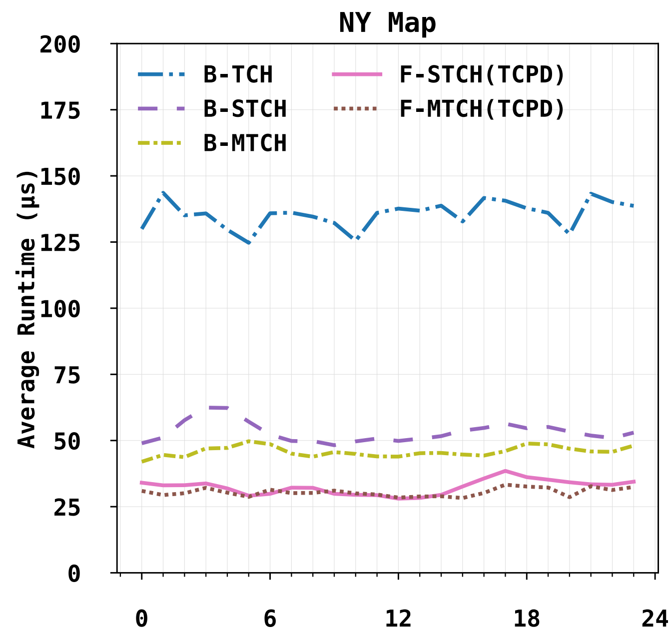
Map	#V	#E	Build Time (Mins)									Memory (MB)							
			TCH				STCH		MTCH			TCH				STCH		MTCH	
			-	CPD	TCPD	RTPD	-	TCPD	-	TCPD		-	CPD	TCPD	RTPD	-	TCPD	-	TCPD
NY	96k	260k	1.72	2.92	2.94	3.21	2.99	31.87	9.04	95.12		269	346	353	9596	1279	3286	3193	9198

  - Query performance:

Heuristic Search



Time Domain Splitting



Reference

[1] Bono, M.; Gerevini, A. E.; Harabor, D. D.; and Stuckey, P. J. 2019. Path Planning with CPD Heuristics. In *Proceedings of the Twenty-Eighth International Joint Conference on Artifi- cial Intelligence, IJCAI 2019, Macao, China, August 10-16, 2019*, 1199–1205. ijcai.org.

[2] M. Chiari, S. Zhao, A. Botea, A. E. Gerevini, D. Harabor, A. Saetti, M. Salvetti, P. J. Stuckey, Cutting the size of compressed path databases with wildcards and redundant symbols, in: Proceedings of the Twenty-Ninth International Conference on Automated Planning and Scheduling, ICAPS 2018, Berkeley, CA, USA, July 11-15, 2019, AAAI Press,5852019, pp. 106–113.

[3] B. Strasser, D. Harabor, A. Botea, Fast first-move queries through run-length encoding, in: Proceedings of the Seventh Annual Symposium on Combinatorial Search, SOCS 2014, Prague, Czech Republic, 15-17 August 2014, AAAI Press, 2014.

[4] Mahe´o, A.; Zhao, S.; Afzaal, H.; Harabor, D.; Stuckey, P. J.; and Wallace, M. 2021. Customised Shortest Paths Using a Distributed Reverse Oracle. In *Proceedings of the Four- teenth International Symposium on Combinatorial Search, SOCS 2021, Virtual Conference [Jinan, China], July 26-30, 2021*, 79–87. AAAI Press.

[5] Batz, G. V.; Delling, D.; Sanders, P.; and Vetter, C. 2009. Time-Dependent Contraction Hierarchies. In *Proceedings of the Eleventh Workshop on Algorithm Engineering and Ex- periments, ALENEX 2009, New York, New York, USA, Jan- uary 3, 2009*, 97–105. SIAM.

Reference

[6] Delling, D.; and Wagner, D. 2007. Landmark-Based Rout- ing in Dynamic Graphs. In *Experimental Algorithms, 6th International Workshop, WEA 2007, Rome, Italy, June 6-8, 2007, Proceedings*, volume 4525 of *Lecture Notes in Com- puter Science*, 52–65. Springer.

[7] https://uofi.app.box.com/v/NYC-traffic-estimates.

Thank you for listening