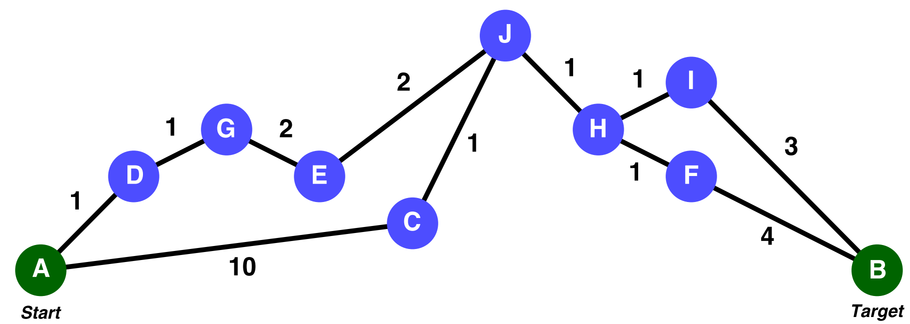
Contracting and Compressing Shortest path Databases ICAPS 2021

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

Monash University

Shortest Path Problem in Road Network

- Road Network:
  - A directed or undirected graph
    - A set of vertices: V.
    - A set of edges: $E \subseteq V \times V$.
    - Each edge $e \in E$ has a non-negative weight $w(e)$, which can represent travel time, distance and so on.
- Shortest Path Problem:
  - Given a start s and target t.
  - Find an optimal path which minimize the sum of weight on each edge.



An example of undirected road network graph. The start and target are shown in green, and the weight of each edge are shown in the figure correspondingly.

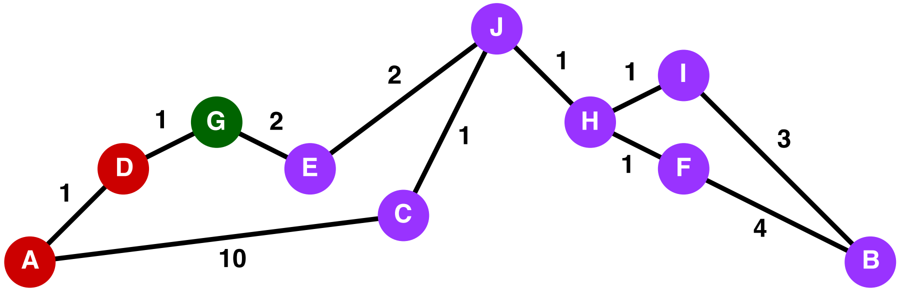
Compressed Path Databases (CPD) [1]

- Compressed Path Databases:

Background
Compressed Path Databases (CPD) [1]

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

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

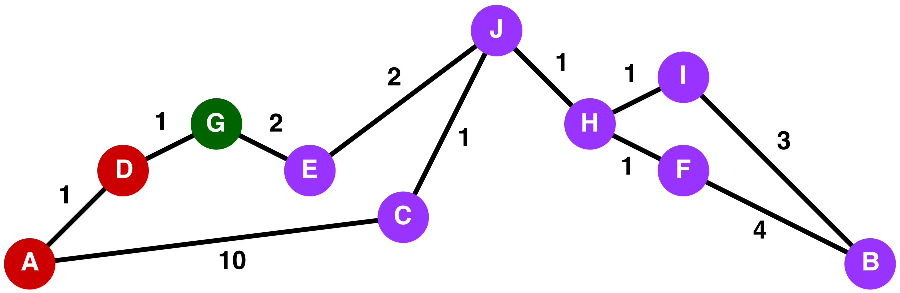


From the source node G, the optimal first move to any node colored red (resp. purple) is D (resp. E).

Compressed Path Databases (CPD) [1]

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

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

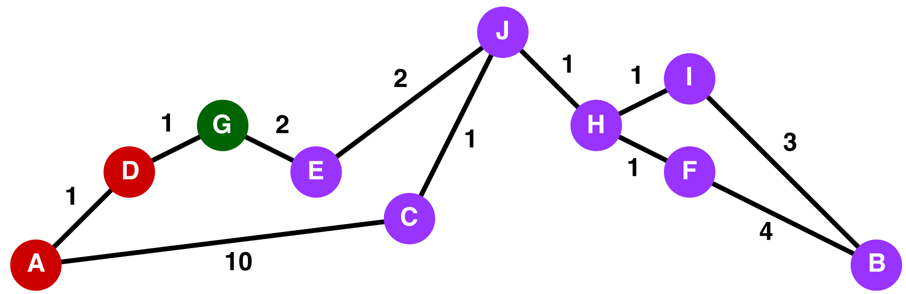


From the source node G, the optimal first move to any node colored red (resp. purple) is D (resp. E).

Background
Compressed Path Databases (CPD) [1]

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

Ordering	G	D	A	C	J	E	H	F	B	I
G	$\boxed{*}$	D	D	E	E	E	E	E	E	E
J	E	E	E	C	*	E	H	H	H	H
I	H	H	H	H	H	H	H	H	B	*

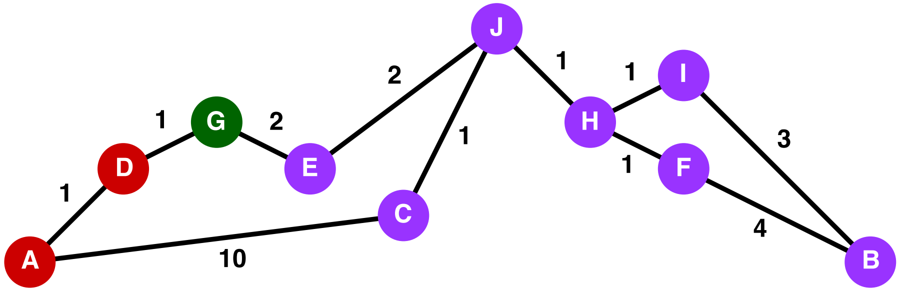


From the source node G, the optimal first move to any node colored red (resp. purple) is D (resp. E).

Compressed Path Databases (CPD) [1]

- Compressed Path Databases:
  - Construction:
    - First move table:
      - [A – J]: indicates the optimal first move.
      - *: wildcard symbol [2].
    - Compression:
      - Depth first search order [1].

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

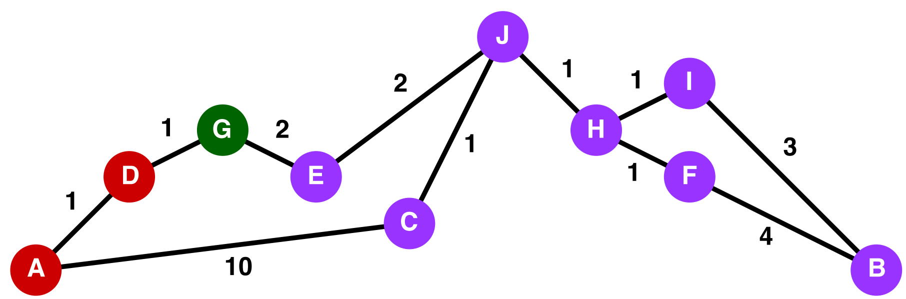


From the source node G, the optimal first move to any node colored red (resp. purple) is D (resp. E).

Compressed Path Databases (CPD) [1]

- Compressed Path Databases
  - Construction:
    - First move table:
      - [A – J]: indicates the optimal first move.
      - *: wildcard symbol [2].
    - Compression:
      - Depth first search order [1].
      - Run length encoding [1] (i.e. Row G: 1D; 4E).

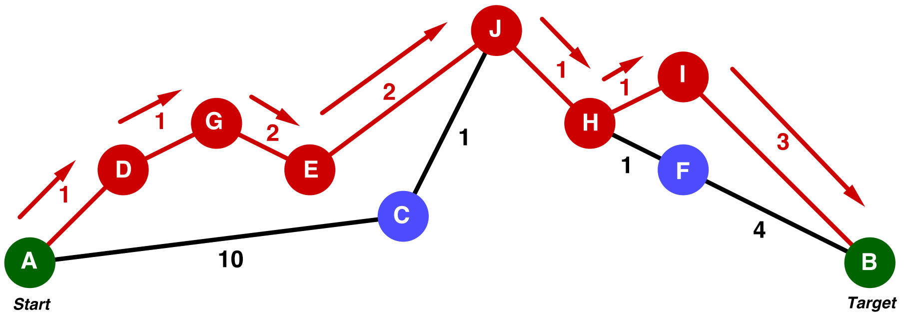
Ordering	G	D	A	C	J	E	H	F	B	I
G	*	D	D	E	E	E	E	E	E	E
J	E	E	E	C	*	E	H	H	H	H
I	H	H	H	H	H	H	H	H	B	*



From the source node G, the optimal first move to any node colored red (resp. purple) is D (resp. E).

Background
Compressed Path Databases (CPD) [1]

- Compressed Path Databases
  - Construction:
    - First move table:
      - [A – J]: indicates the optimal first move.
      - *: wildcard symbol [2].
    - Compression:
      - Depth first search order [1].
      - Run length encoding [1] (i.e. Row G: 1D; 4E).
  - Query:
    - From a given s, CPD recursively extract the optimal first move until reaches t.



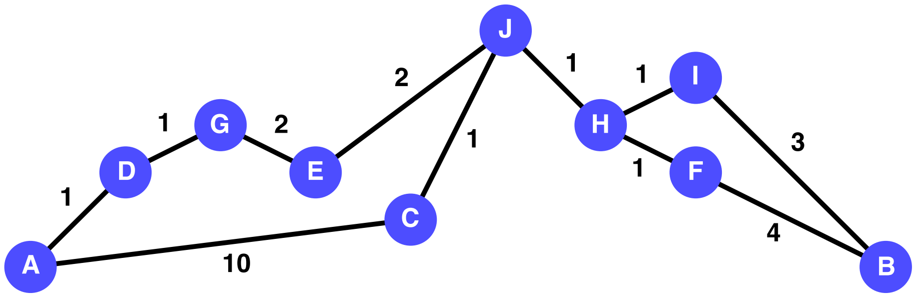
Exacting the shortest path from CPD, the shortest path from start to target is shown in red.

Contraction Hierarchy (CH) [3]

- Contraction Hierarchy:

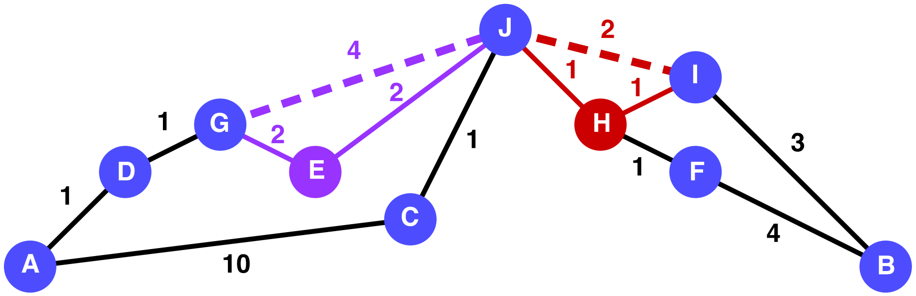
Contraction Hierarchy (CH) [3]

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



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

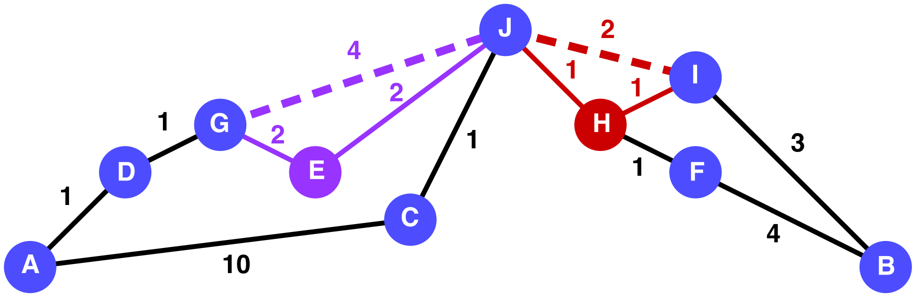
- Contraction Hierarchy:
  - Construction:
    - Apply a total lex order L.
    - Contraction:
      - W.r.t. L, choose the least node v from the graph.



The result of contracting E (resp. H) in purple (resp. red). Dashed edges indicate shortcut edges.

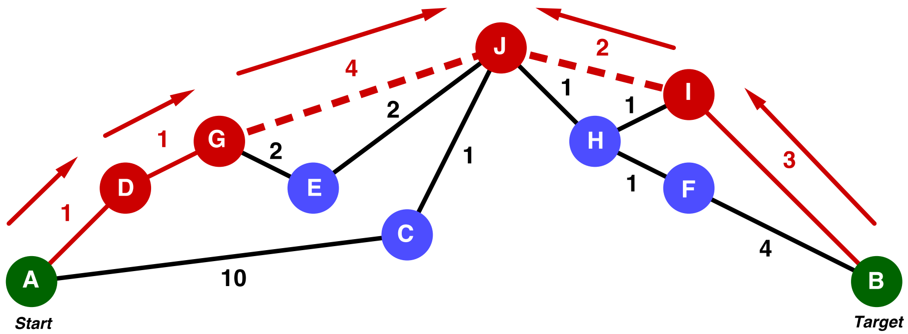
Contraction Hierarchy (CH) [3]

- 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.
        - » v ∈ sp(u,w).



The result of contracting E (resp. H) in purple (resp. red). Dashed edges indicate shortcut edges.

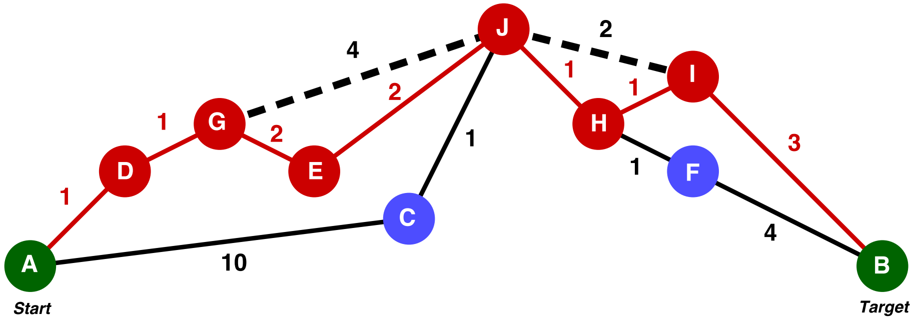
- Contraction Hierarchy:
  - Construction:
    - Apply a total lex order L.
    - Contraction:
      - W.r.t. L, choose the least node v from the graph.
      - Add a shortcut edge (u, w) between each pair of in-neighbour u and out-neighbour w of v:
        - » v $<_L$ u & v $<_L$ w.
        - » v ∈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



Bi-directional Dijkstra search from start and target, the shortest ch-path is shown in red.

Background
Contraction Hierarchy (CH) [3]

- 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.
        - » v ∈ sp(u,w).
  - Query:
    - Bi-directional Dijkstra search.
    - Unpack the path.



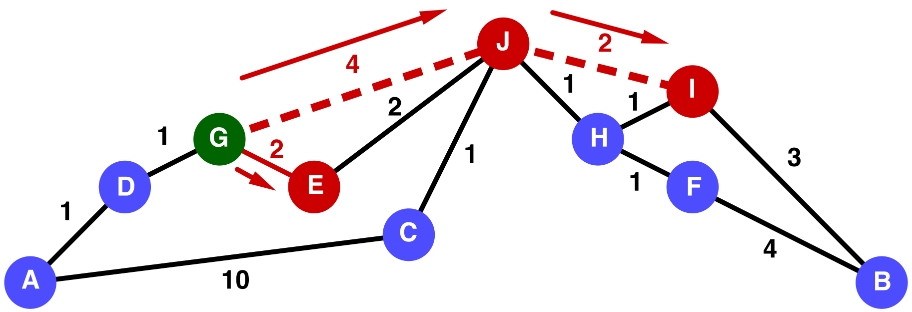
Unpacking the CH path, the final shortest path is shown in red.

Our Approach:
CH-based Compressed Path Databases (CH-CPD)

- CH-CPD:

Our Approach:
CH-based Compressed Path Databases (CH-CPD)

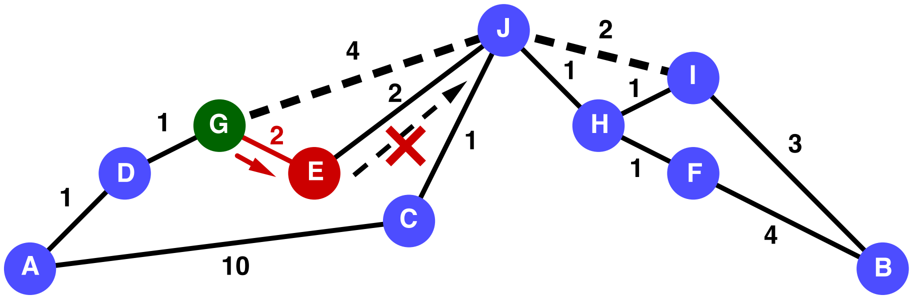
- CH-CPD:
  - Construction:
    - Modified Dijkstra:
      - Up-then-down policy.



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

Our Approach:
CH-based Compressed Path Databases (CH-CPD)

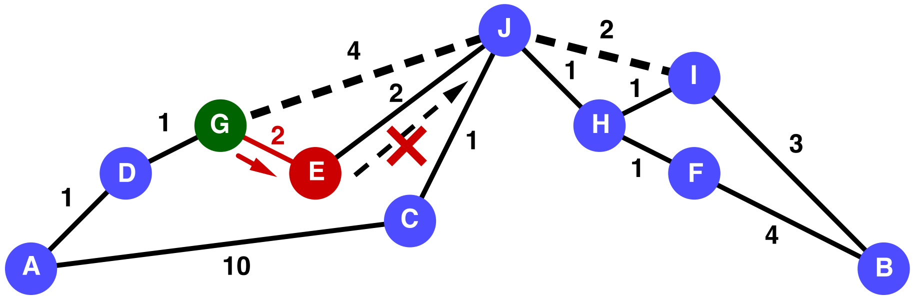
- CH-CPD:
  - Construction:
    - Modified Dijkstra:
      - Up-then-down policy.



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

Our Approach:
CH-based Compressed Path Databases (CH-CPD)

- CH-CPD:
  - Construction:
    - Modified Dijkstra:
      - Up-then-down policy.
    - Distance table enhancement:

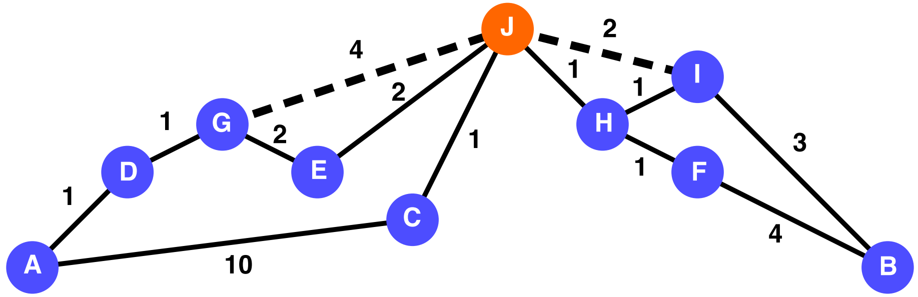


The up-then-down policy: the up successor J of E is pruned, because the predecessor G is lexically larger than E. (i.e., G $>_L$ E).

Our Approach:
CH-based Compressed Path Databases (CH-CPD)

- CH-CPD:
  - Construction:
    - Modified Dijkstra:
      - Up-then-down policy.
    - Distance table enhancement:
      - Caching:
        - » Cache the distance table for top n% of CH-nodes.

Ordering	G	D	A	C	J	E	H	F	B	I
FirstMove(J,_)	G	G	G	C	*	E	H	H	I	I
$d(J, \_)$	4	5	6	1	0	2	1	2	5	2

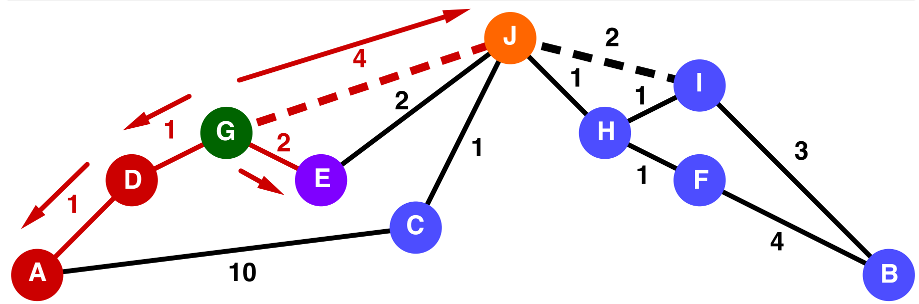


Distance table enhancement: We cache the all-pairs distance when compute the first moves for node J.

Our Approach:
CH-based Compressed Path Databases (CH-CPD)

- CH-CPD:
  - Construction:
    - Modified Dijkstra:
      - Up-then-down policy.
    - Distance table enhancement:
      - Caching:
        - » Cache the distance table for top n% of CH-nodes.
      - Pruning:

Ordering	G	D	A	C	J	E	H	F	B	I
$d(\mathtt{J}, \_)$	4	5	6	1	0	2	1	2	5	2
FirstMove(G,_)	*	D	D	-	J	E	-	-	-	-
$g(\mathtt{G}, \_)$	0	1	2	∞	4	2	∞	∞	∞	∞

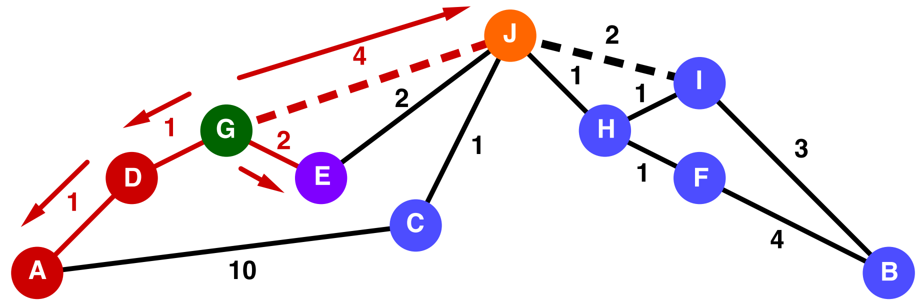


Constructing CPD on top of CH. The source node G is highlighted as green. The first move on the optimal path from source node to any node are D, E and J shown as red, purple and orange, respectively

Our Approach:
CH-based Compressed Path Databases (CH-CPD)

- CH-CPD:
  - Construction:
    - Modified Dijkstra:
      - Up-then-down policy.
    - Distance table enhancement:
      - Caching:
        - » Cache the distance table for top n% of CH-nodes.
      - Pruning:
        - » Relax tentative distance.

Ordering	G	D	A	C	J	E	H	F	B	I
$d(J, \_)$	4	5	6	1	0	2	1	2	5	2
FirstMove(G,_)	*	D	D	-	J	E	-	-	-	-
$g(G, \_)$	0	1	2	5	4	2	5	6	9	6

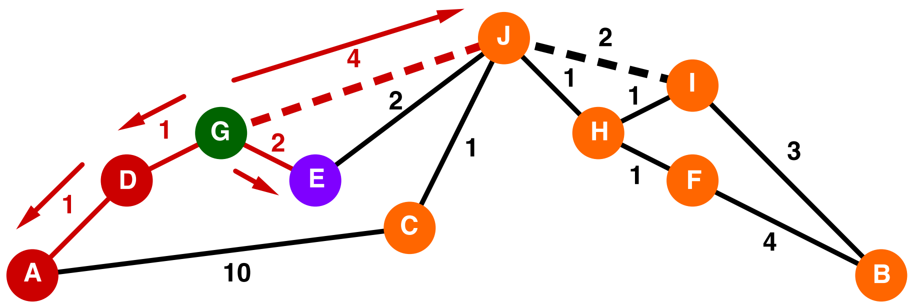


Constructing CPD on top of CH. The source node G is highlighted as green. The first move on the optimal path from source node to any node are D, E and J shown as red, purple and orange, respectively

Our Approach:
CH-based Compressed Path Databases (CH-CPD)

- CH-CPD:
  - Construction:
    - Modified Dijkstra:
      - Up-then-down policy.
    - Distance table enhancement:
      - Caching:
        - » Cache the distance table for top n% of CH-nodes.
      - Pruning:
        - » Relax tentative distance.
        - » Update first move table.

Ordering	G	D	A	C	J	E	H	F	B	I
$d(J, \_)$	4	5	6	1	0	2	1	2	5	2
FirstMove(G,_)	*	D	D	J	J	E	J	J	J	J
$g(G, \_)$	0	1	2	5	4	2	5	6	9	6



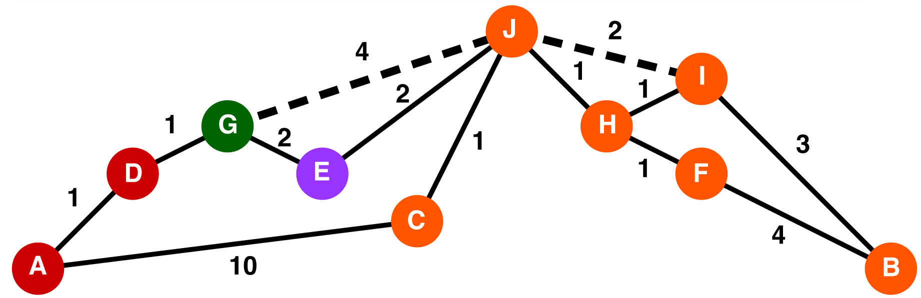
Constructing CPD on top of CH. The source node G is highlighted as green. The first move on the optimal path from source node to any node are D, E and J shown as red, purple and orange, respectively

Our Approach:
CH-based Compressed Path Databases (CH-CPD)

MONASH University

- CH-CPD:
  - Construction:
    - Modified Dijkstra:
      - Up-then-down policy.
    - Distance table enhancement:
      - Caching:
        - » Cache the distance table for top n% of CH-nodes.
      - Pruning:
        - » Relax tentative distance.
        - » Update first move table.

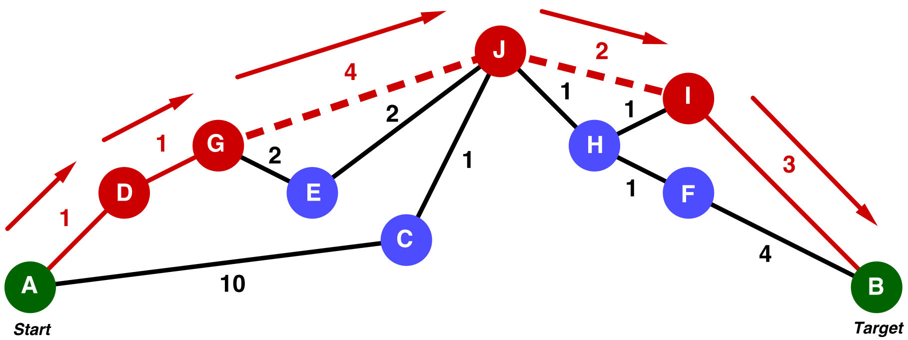
Ordering	G	D	A	C	J	E	H	F	B	I
G	*	D	D	J	J	E	J	J	J	J
J	G	G	G	C	*	E	H	H	I	I
I	J	J	J	J	J	J	H	H	B	*



Constructing CPD on top of CH. The source node G is highlighted as green. The first move on the optimal path from source node to any node are D, E and J shown as red, purple and orange, respectively

Our Approach:
CH-based Compressed Path Databases (CH-CPD)

- CH-CPD:
  - Construction:
    - Modified Dijkstra:
      - Up-then-down policy.
    - Distance table enhancement:
      - Caching:
        - » Cache the distance table for top n% of CH-nodes.
      - Pruning:
        - » Relax tentative distance.
        - » Update first move table.
  - Query:
    - Extract and unpack the CH-path from CH-CPD.



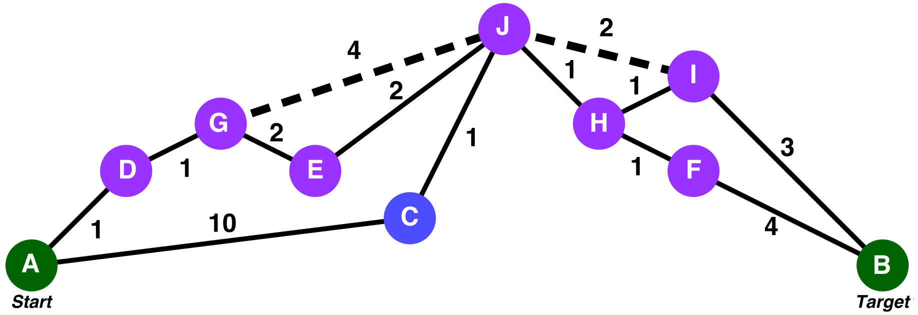
Exacting the shortest ch-path from CH-CPD, the shortest ch-path from start to target is shown in red.

Our Approach:
Partial CH-CPD

- Partial CH-CPD:

Our Approach:
Partial CH-CPD

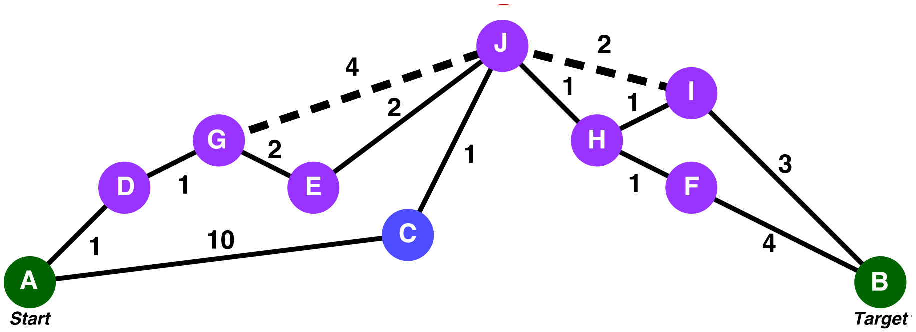
- Partial CH-CPD:
  - Construction:
    - Select top n% of CH nodes to construct a CH-CPD.



The partial CH-CPD nodes are D-J which shown in purple color.

Our Approach:
Partial CH-CPD

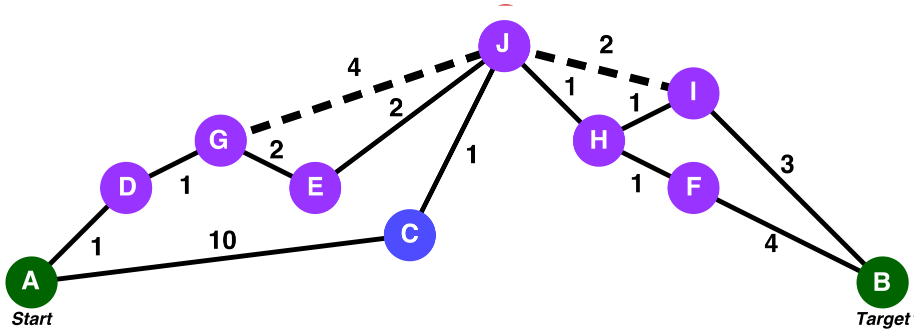
- Partial CH-CPD:
  - Construction:
    - Select top n% of CH nodes to construct a CH-CPD.
  - Bi-directional CPD search:



The partial CH-CPD nodes are D-J which shown in purple color.

Our Approach:
Partial CH-CPD

- Partial CH-CPD:
  - Construction:
    - Select top n% of CH nodes to construct a CH-CPD.
  - Bi-directional CPD search:
    - Incremental exploration:
      - A* search:
        - » $Search_s$ & $Search_t$.

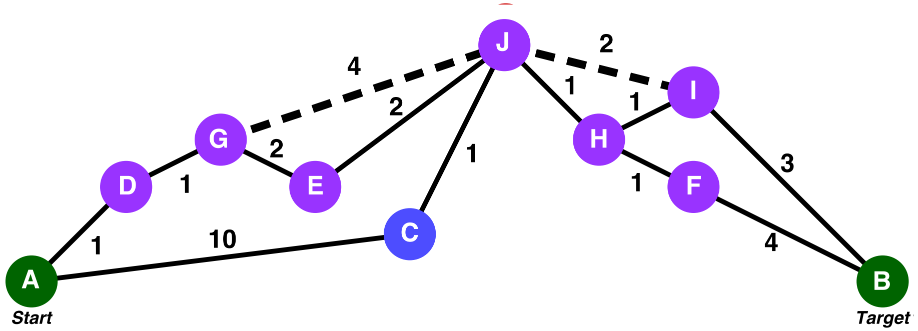


The partial CH-CPD nodes are D-J which shown in purple color.

Our Approach:
Partial CH-CPD

MONASH University

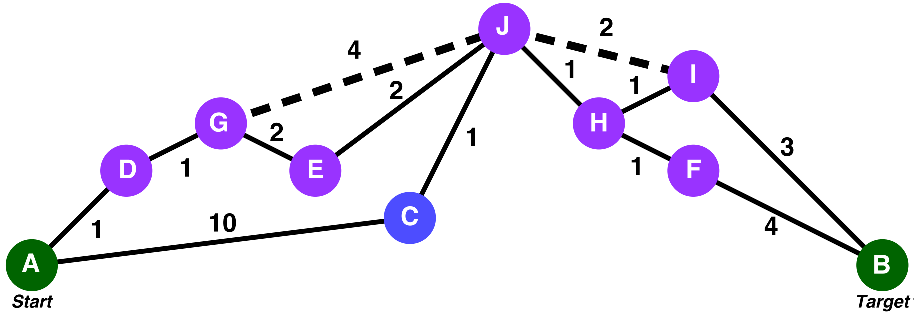
- Partial CH-CPD:
  - Construction:
    - Select top n% of CH nodes to construct a CH-CPD.
  - Bi-directional CPD search:
    - Incremental exploration:
      - A* search:
        - » Search_s & Search_t .
        - » Landmark heuristics [4].



The partial CH-CPD nodes are D-J which shown in purple color.

Our Approach: Partial CH-CPD

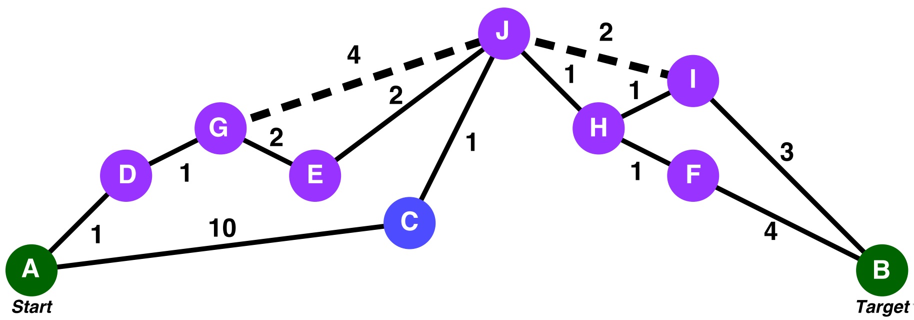
- Partial CH-CPD:
  - Construction:
    - Select top n% of CH nodes to construct a CH-CPD.
  - Bi-directional CPD search:
    - Incremental exploration:
      - A* search:
        - » $Search_s$ & $Search_t$.
        - » Landmark heuristics [4].
      - V_s & V_t .



The partial CH-CPD nodes are D-J which shown in purple color.

Our Approach:
Partial CH-CPD

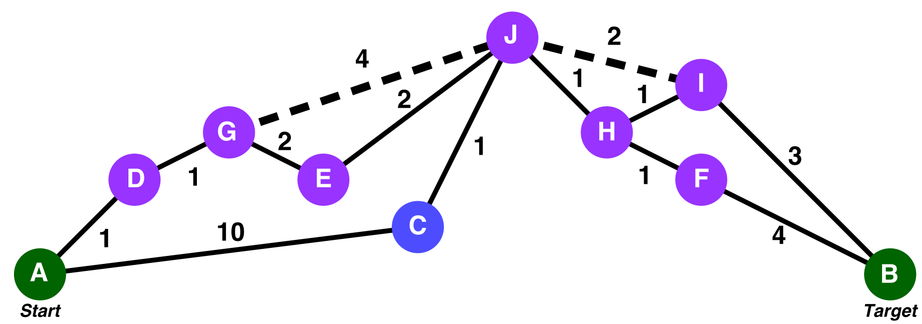
- Partial CH-CPD:
  - Construction:
    - Select top n% of CH nodes to construct a CH-CPD.
  - Bi-directional CPD search:
    - Incremental exploration:
      - A* search:
        - » $Search_s$ & $Search_t$.
        - » Landmark heuristics [4].
      - V_s & V_t .
      - Shortest path (sp).



The partial CH-CPD nodes are D-J which shown in purple color.

Our Approach: Partial CH-CPD

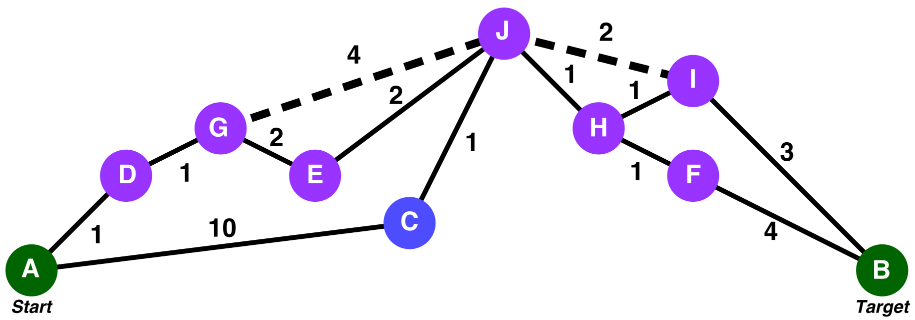
- Partial CH-CPD:
  - Construction:
    - Select top n% of CH nodes to construct a CH-CPD.
  - Bi-directional CPD search:
    - Incremental exploration:
      - A* search:
        - » $Search_s$ & $Search_t$.
        - » Landmark heuristics [4].
      - V_s & V_t .
      - Shortest path (sp).
    - Pruning:



The partial CH-CPD nodes are D-J which shown in purple color.

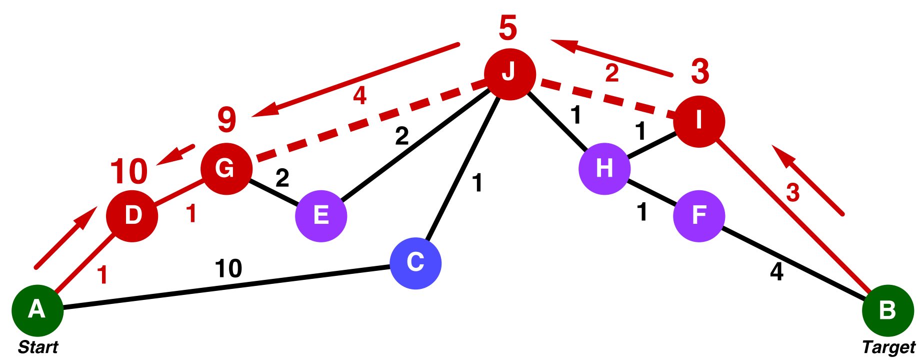
Our Approach: Partial CH-CPD

- Partial CH-CPD:
  - Construction:
    - Select top n% of CH nodes to construct a CH-CPD.
  - Bi-directional CPD search:
    - Incremental exploration:
      - A* search:
        - » $Search_s$ & $Search_t$.
        - » Landmark heuristics [4].
      - V_s & V_t .
      - Shortest path (sp).
    - Pruning:
      - Distance-based pruning:
        - » $g(s, v_s) + landmark(v_s, v_t) + g(v_t, t) \geq |sp|$.



The partial CH-CPD nodes are D-J which shown in purple color.

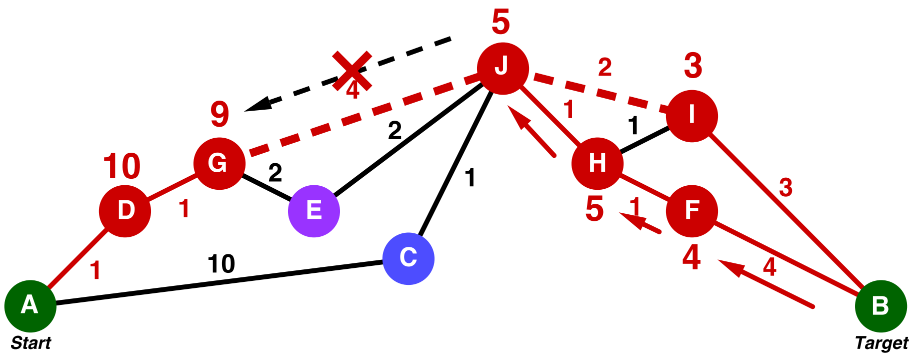
- Partial CH-CPD:
  - Construction:
    - Select top n% of CH nodes to construct a CH-CPD.
  - Bi-directional CPD search:
    - Incremental exploration:
      - A* search:
        - » $Search_s$ & $Search_t$.
        - » Landmark heuristics [4].
      - V_s & V_t .
      - Shortest path (sp).
    - Pruning:
      - Distance-based pruning:
        - » $g(s, v_s) + landmark(v_s, v_t) + g(v_t, t) \geq |sp|$.
      - Cost caching.



Bi-directional CPD search caches distance on each node when extracts the path from I to D.

Our Approach: Partial CH-CPD

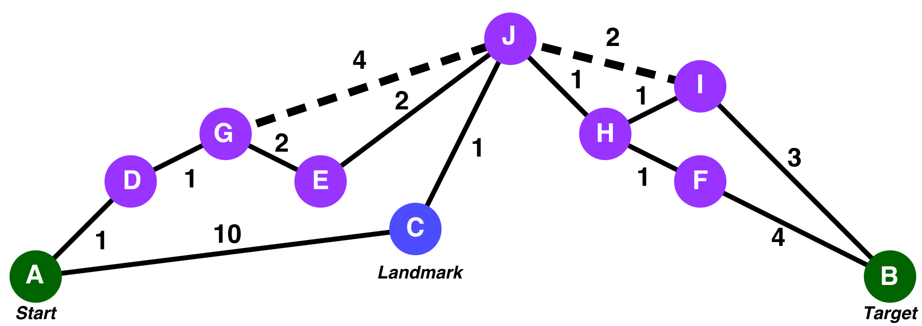
- Partial CH-CPD:
  - Construction:
    - Select top n% of CH nodes to construct a CH-CPD.
  - Bi-directional CPD search:
    - Incremental exploration:
      - A* search:
        - » $Search_s$ & $Search_t$.
        - » Landmark heuristics [4].
      - V_s & V_t .
      - Shortest path (sp).
    - Pruning:
      - Distance-based pruning:
        - » $g(s, v_s) + landmark(v_s, v_t) + g(v_t, t) \geq |sp|$.
      - Cost caching.



The path extraction from F to D terminates at J, because the cached distance g(B, J) < g(B, H) + d(H, J).

Our Approach:
Partial CH-CPD

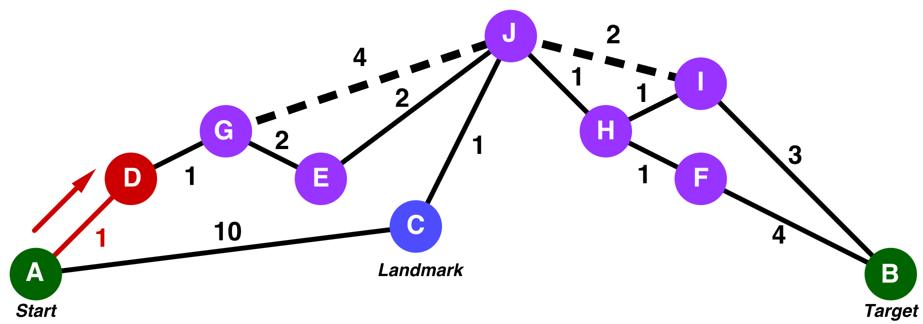
- Partial CH-CPD:
  - Construction:
    - Select top n% of CH nodes to construct a CH-CPD.
  - Bi-directional CPD search:
    - Incremental exploration:
      - A* search:
        - » $Search_s$ & $Search_t$.
        - » Landmark heuristics [4].
      - V_s & V_t .
      - Shortest path (sp).
    - Pruning:
      - Distance-based pruning:
        - » $g(s, v_s) + landmark(v_s, v_t) + g(v_t, t) \geq |sp|$.
      - Cost caching.
    - Running example:



The partial CH-CPD nodes are D-J which shown in purple color.

Our Approach: Partial CH-CPD

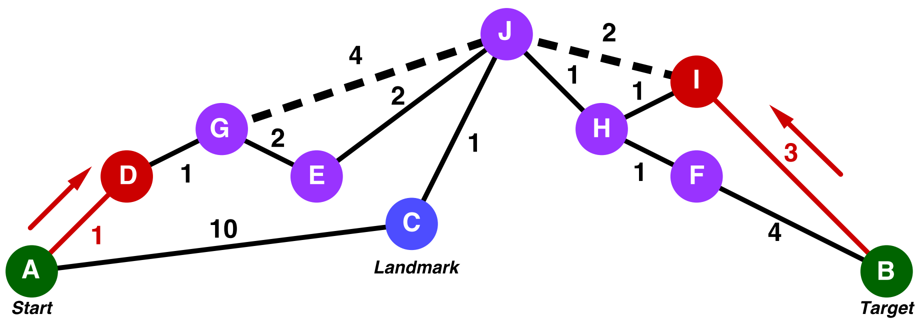
- Partial CH-CPD:
  - Construction:
    - Select top n% of CH nodes to construct a CH-CPD.
  - Bi-directional CPD search:
    - Incremental exploration:
      - A* search:
        - » $Search_s$ & $Search_t$.
        - » Landmark heuristics [4].
      - V_s & V_t .
      - Shortest path (sp).
    - Pruning:
      - Distance-based pruning:
        - » $g(s, v_s) + landmark(v_s, v_t) + g(v_t, t) \geq |sp|$.
      - Cost caching.
    - Running example:



The partial CH-CPD nodes are D-J which shown in purple color.

Our Approach: Partial CH-CPD

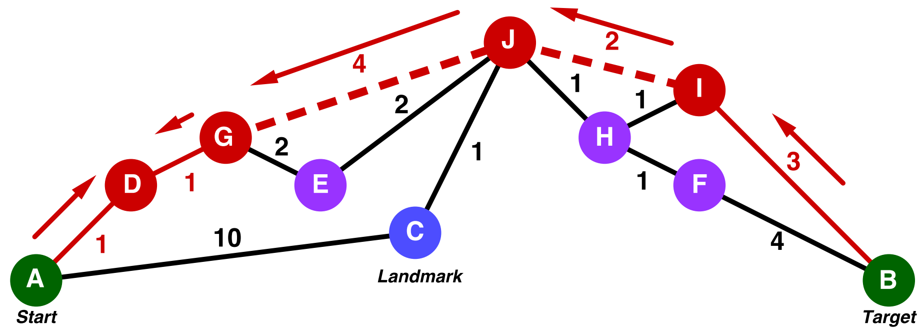
- Partial CH-CPD:
  - Construction:
    - Select top n% of CH nodes to construct a CH-CPD.
  - Bi-directional CPD search:
    - Incremental exploration:
      - A* search:
        - » $Search_s$ & $Search_t$.
        - » Landmark heuristics [4].
      - V_s & V_t .
      - Shortest path (sp).
    - Pruning:
      - Distance-based pruning:
        - » $g(s, v_s) + landmark(v_s, v_t) + g(v_t, t) \geq |sp|$.
      - Cost caching.
    - Running example:



The partial CH-CPD nodes are D-J which shown in purple color.

Our Approach:
Partial CH-CPD

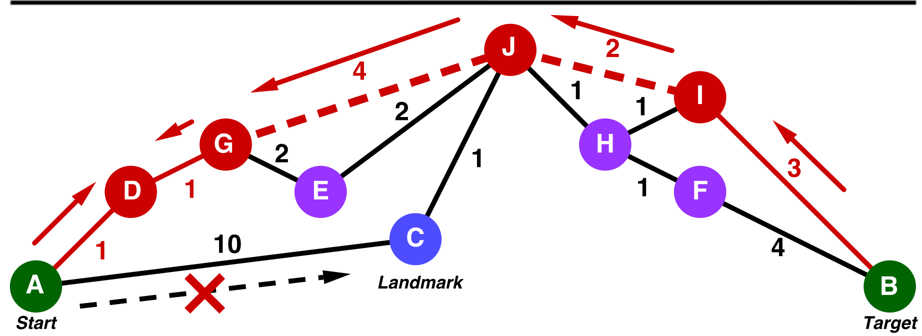
- Partial CH-CPD:
  - Construction:
    - Select top n% of CH nodes to construct a CH-CPD.
  - Bi-directional CPD search:
    - Incremental exploration:
      - A* search:
        - » $Search_s$ & $Search_t$.
        - » Landmark heuristics [4].
      - V_s & V_t .
      - Shortest path (sp).
    - Pruning:
      - Distance-based pruning:
        - » $g(s, v_s) + landmark(v_s, v_t) + g(v_t, t) \geq |sp|$.
      - Cost caching.
    - Running example:



The partial CH-CPD nodes are D-J which shown in purple color.

- Partial CH-CPD:
  - Construction:
    - Select top n% of CH nodes to construct a CH-CPD.
  - Bi-directional CPD search:
    - Incremental exploration:
      - A* search:
        - » $Search_s$ & $Search_t$.
        - » Landmark heuristics [4].
      - V_s & V_t .
      - Shortest path (sp).
    - Pruning:
      - Distance-based pruning:
        - » $g(s, v_s) + landmark(v_s, v_t) + g(v_t, t) \geq |sp|$.
      - Cost caching.
    - Running example:

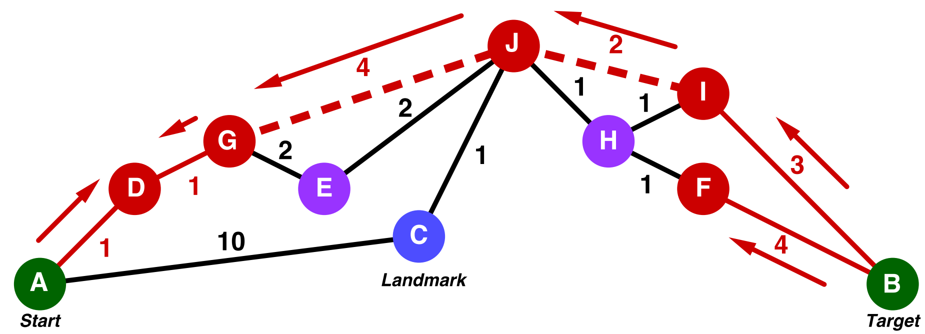
Ordering	G	D	A	C	J	E	H	F	B	I
$d(C, \_)$	5	6	7	0	1	3	2	3	6	3



The partial CH-CPD nodes are D-J which shown in purple color.

Our Approach: Partial CH-CPD

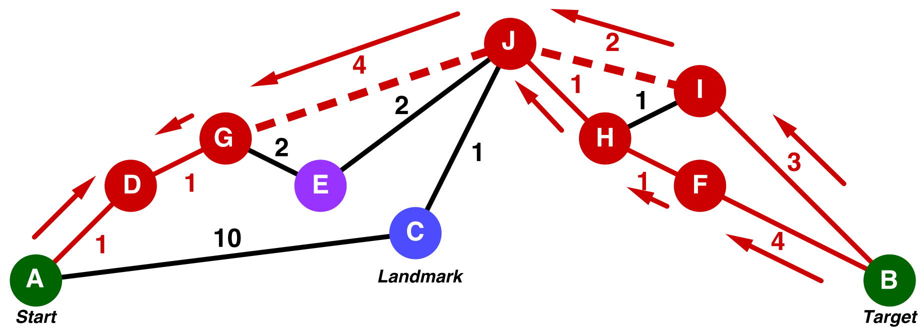
- Partial CH-CPD:
  - Construction:
    - Select top n% of CH nodes to construct a CH-CPD.
  - Bi-directional CPD search:
    - Incremental exploration:
      - A* search:
        - » $Search_s$ & $Search_t$.
        - » Landmark heuristics [4].
      - V_s & V_t .
      - Shortest path (sp).
    - Pruning:
      - Distance-based pruning:
        - » $g(s, v_s) + landmark(v_s, v_t) + g(v_t, t) \geq |sp|$.
      - Cost caching.
    - Running example:



The partial CH-CPD nodes are D-J which shown in purple color.

Our Approach: Partial CH-CPD

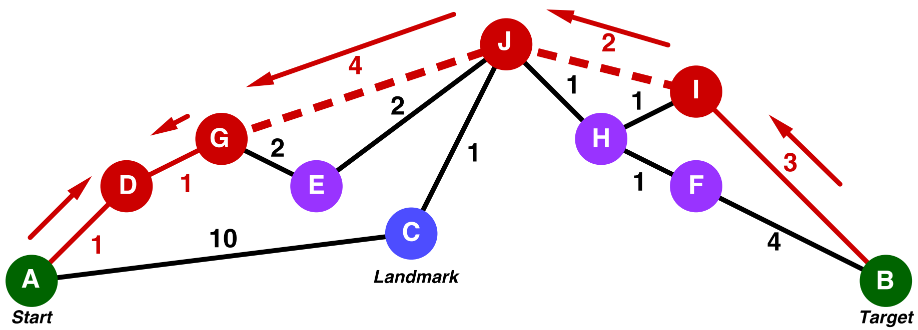
- Partial CH-CPD:
  - Construction:
    - Select top n% of CH nodes to construct a CH-CPD.
  - Bi-directional CPD search:
    - Incremental exploration:
      - A* search:
        - » $Search_s$ & $Search_t$.
        - » Landmark heuristics [4].
      - V_s & V_t .
      - Shortest path (sp).
    - Pruning:
      - Distance-based pruning:
        - » $g(s, v_s) + landmark(v_s, v_t) + g(v_t, t) \geq |sp|$.
      - Cost caching.
    - Running example:



The partial CH-CPD nodes are D-J which shown in purple color.

Our Approach:
Partial CH-CPD

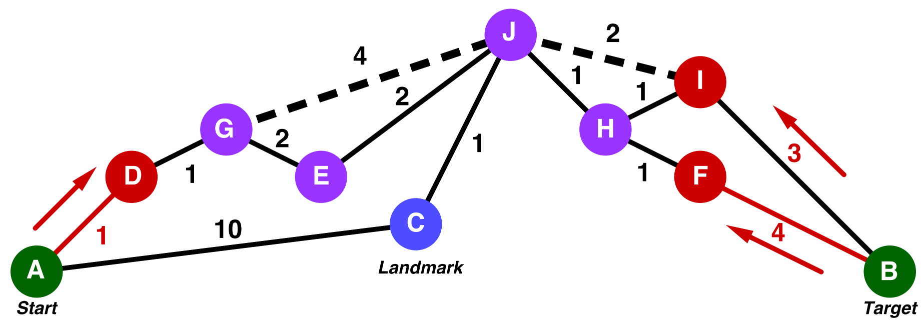
- Partial CH-CPD:
  - Construction:
    - Select top n% of CH nodes to construct a CH-CPD.
  - Bi-directional CPD search:
    - Incremental exploration:
      - A* search:
        - » $Search_s$ & $Search_t$.
        - » Landmark heuristics [4].
      - V_s & V_t .
      - Shortest path (sp).
    - Pruning:
      - Distance-based pruning:
        - » $g(s, v_s) + landmark(v_s, v_t) + g(v_t, t) \geq |sp|$.
      - Cost caching.
    - Running example:



The partial CH-CPD nodes are D-J which shown in purple color.

Our Approach: Partial CH-CPD

- Partial CH-CPD:
  - Construction:
    - Select top n% of CH nodes to construct a CH-CPD.
  - Bi-directional CPD search:
    - Incremental exploration:
      - A* search:
        - » $Search_s$ & $Search_t$.
        - » Landmark heuristics [4].
      - V_s & V_t .
      - Shortest path (sp).
    - Pruning:
      - Distance-based pruning:
        - » $g(s, v_s) + landmark(v_s, v_t) + g(v_t, t) \geq |sp|$.
      - Cost caching.
    - Running example:



The partial CH-CPD nodes are D-J which shown in purple color.

- Experimental Results:

Our Approach: Experimental Results

- Experimental Results:
  - Benchmarks:
    - Real-world road network from DIMACS challenge [5].

- Experimental Results:
  - Benchmarks:
    - Real-world road network from DIMACS challenge [5].
  - Preprocessing cost:

Map Type	Build Time (Mins)							Memory (MB)						
	CH-CPD					Competitors		CH-CPD					Competitors	
	20%	40%	60%	80%	100%	CPD	CH	20%	40%	60%	80%	100%	CPD	CH
NY Distance	0.36	0.73	1.18	1.87	2.95	8.76	0.24	70	104	183	271	338	219	29
NY Travel Time	0.27	0.96	1.79	2.24	3.00	11.03	0.16	63	88	156	222	277	188	28

- Experimental Results:
  - Benchmarks:
    - Real-world road network from DIMACS challenge [5].
  - Preprocessing cost:

Map Type	Build Time (Mins)							Memory (MB)						
	CH-CPD					Competitors		CH-CPD					Competitors	
	20%	40%	60%	80%	100%	CPD	CH	20%	40%	60%	80%	100%	CPD	CH
NY Distance	0.36	0.73	1.18	1.87	2.95	8.76	0.24	70	104	183	271	338	219	29
NY Travel Time	0.27	0.96	1.79	2.24	3.00	11.03	0.16	63	88	156	222	277	188	28

MONASH University

- Experimental Results:
  - Benchmarks:
    - Real-world road network from DIMACS challenge [5].
  - Preprocessing cost:

Map Type	Build Time (Mins)							Memory (MB)						
	CH-CPD					Competitors		CH-CPD					Competitors	
	20%	40%	60%	80%	100%	CPD	CH	20%	40%	60%	80%	100%	CPD	CH
NY Distance	0.36	0.73	1.18	1.87	2.95	8.76	0.24	70	104	183	271	338	219	29
NY Travel Time	0.27	0.96	1.79	2.24	3.00	11.03	0.16	63	88	156	222	277	188	28

  - Query performance:

Map Type	Average Runtime (µs)									
	CH-CPD					Competitors				
	20%	40%	60%	80%	100%	CPD	CH	CH+L	PHL	SHP
NY Distance	23.17	20.83	17.19	13.67	11.42	26.38	38.64	25.58	25.36	26.76
NY Travel Time	19.16	18.82	14.83	12.85	9.53	18.23	25.27	20.06	18.32	14.93

Our Approach: Experimental Results

- Experimental Results:
  - Benchmarks:
    - Real-world road network from DIMACS challenge [5].
  - Preprocessing cost:

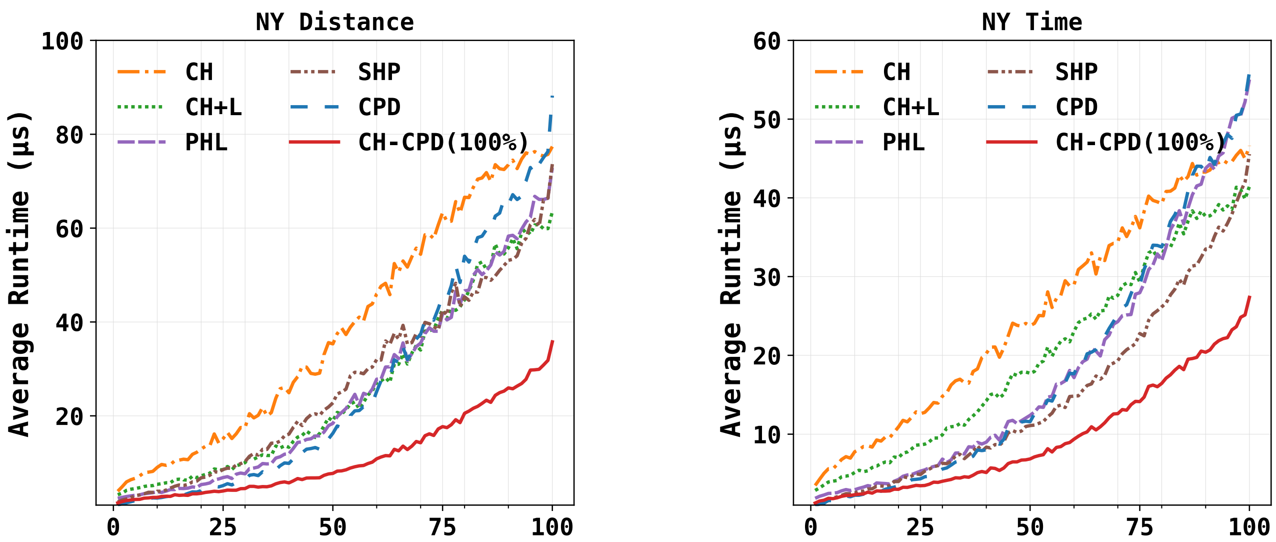
Map Type	Build Time (Mins)							Memory (MB)						
	CH-CPD					Competitors		CH-CPD					Competitors	
	20%	40%	60%	80%	100%	CPD	CH	20%	40%	60%	80%	100%	CPD	CH
NY Distance	0.36	0.73	1.18	1.87	2.95	8.76	0.24	70	104	183	271	338	219	29
NY Travel Time	0.27	0.96	1.79	2.24	3.00	11.03	0.16	63	88	156	222	277	188	28

  - Query performance:

Map Type	Average Runtime (μs)									
	CH-CPD					Competitors				
	20%	40%	60%	80%	100%	CPD	CH	CH+L	PHL	SHP
NY Distance	23.17	20.83	17.19	13.67	11.42	26.38	38.64	25.58	25.36	26.76
NY Travel Time	19.16	18.82	14.83	12.85	9.53	18.23	25.27	20.06	18.32	14.93

Our Approach: Experimental Results

- Experimental Results:
  - Query performance:



Runtime comparison. The x-axis shows the percentile ranks of path queries sorted based on actual distances between start and target.

Reference

[1] 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.

[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] Geisberger, R.; Sanders, P.; Schultes, D.; and Delling, D. 2008. Contraction Hierarchies: Faster and Simpler Hierarchical Routing in Road Networks. In *Experimental Algorithms, 7th International Workshop, WEA 2008, Province- town, MA, USA, May 30-June 1, 2008, Proceedings*, volume 5038, 319–333

[4] Goldberg, A. V.; and Harrelson, C. 2005. Computing the shortest path: A* search meets graph theory. In *Proceedings of the Sixteenth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2005, Vancouver, British Columbia, Canada, January 23-25, 2005*, 156–165

[5] http://www.diag.uniroma1.it//challenge9/download.shtml

Thank you for listening