Faster Transit Routing by Hyper Partitioning

Daniel Delling Julian Dibbelt
Thomas Pajor Tobias Zündorf

ATMOS · September 7th, 2017

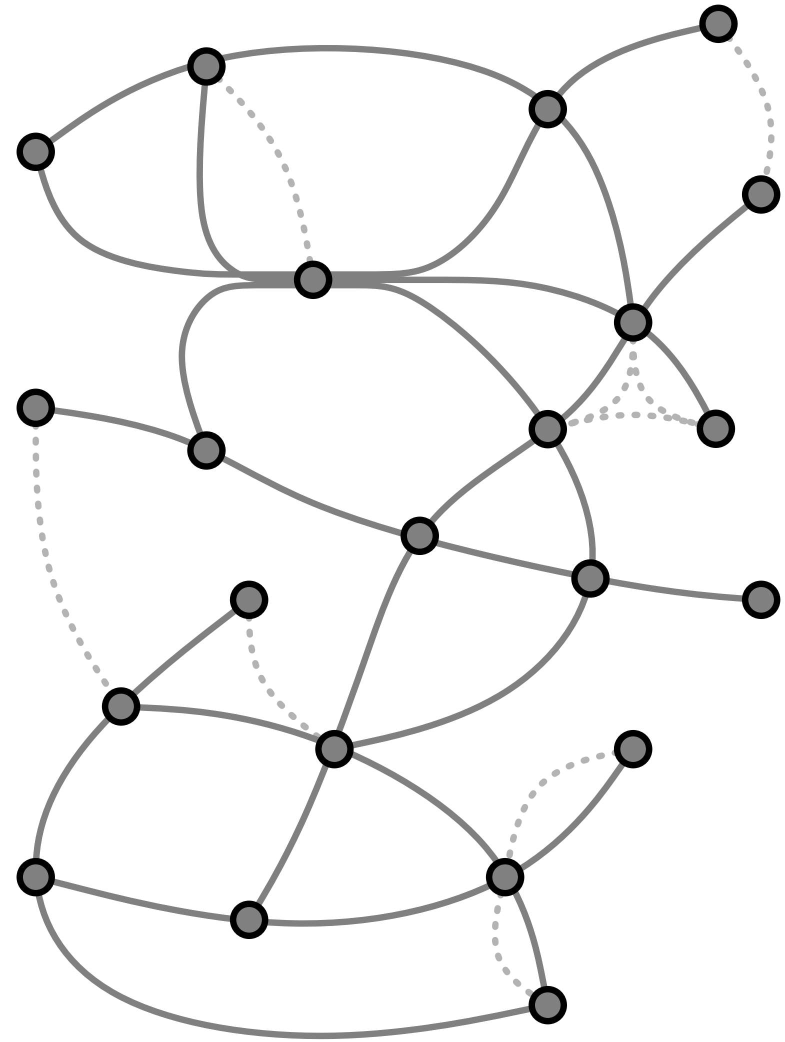
Public Transit Routing

Given:

- Public transportation network
  - Stops
  - Routes / Trips
  - Footpaths

Goal:

- Find optimal s-t-journeys
  - Travel time
  - Number of transfers



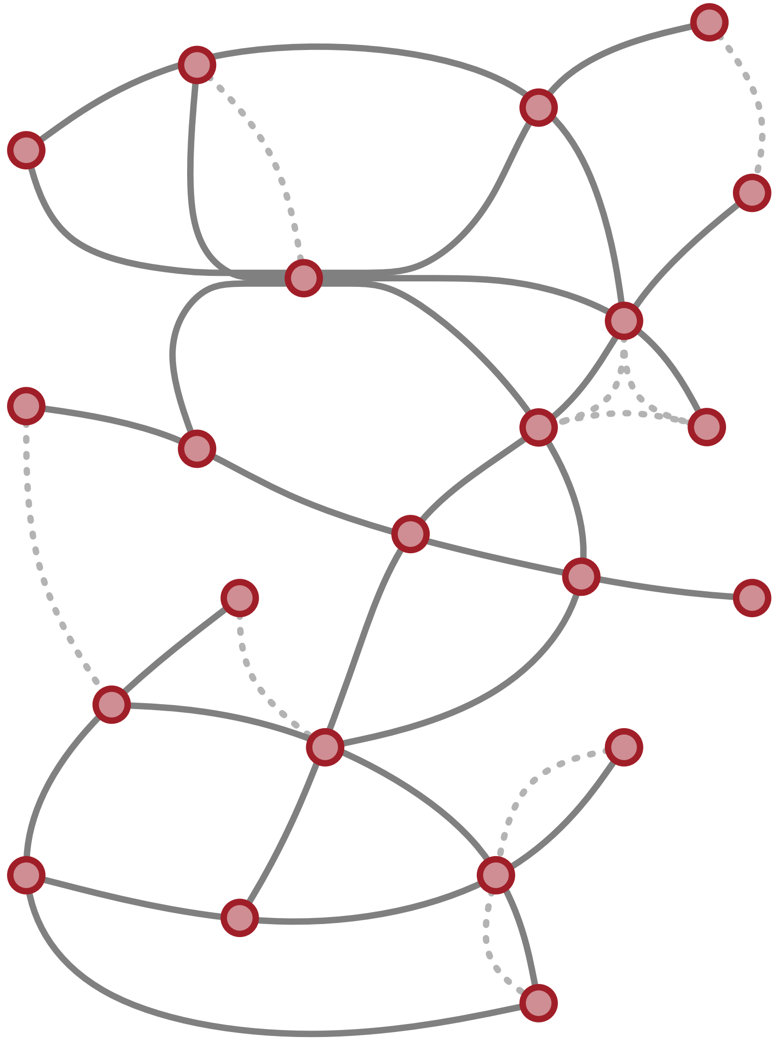
Public Transit Routing

Given:

- Public transportation network
  - Stops
  - Routes / Trips
  - Footpaths

Goal:

- Find optimal s-t-journeys
  - Travel time
  - Number of transfers



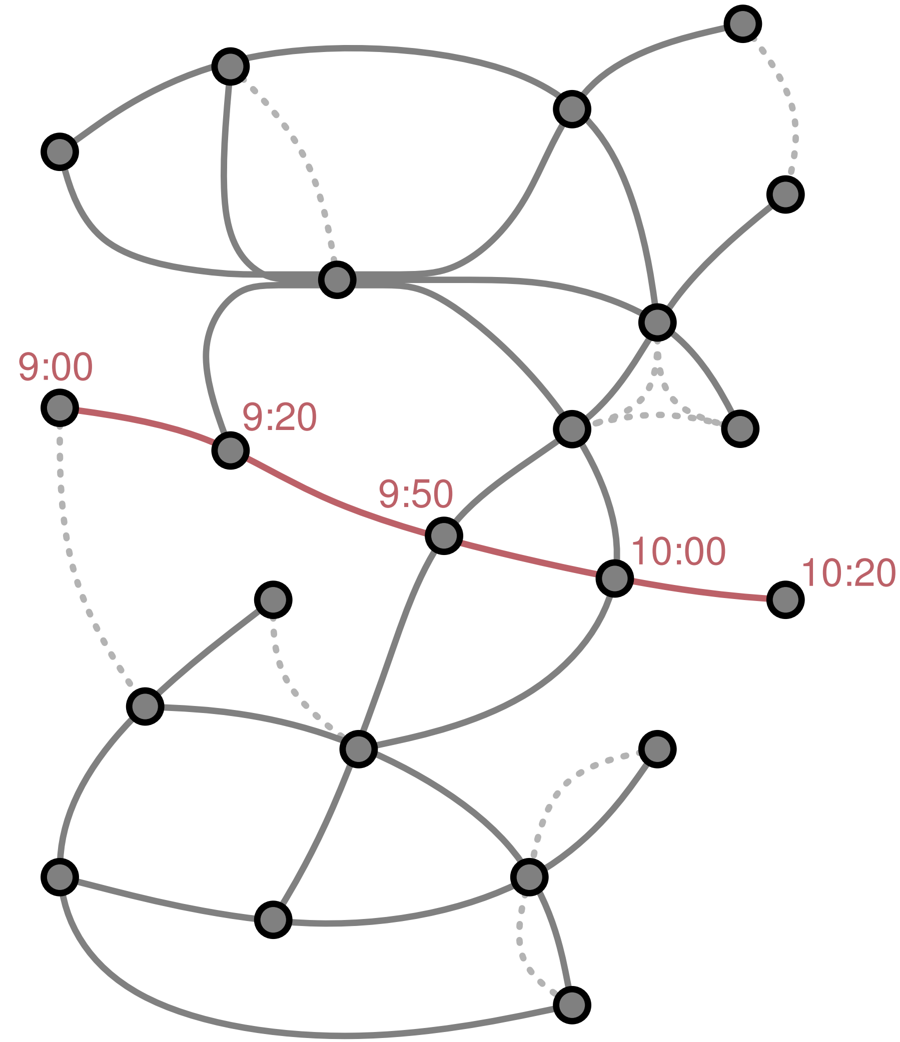
Public Transit Routing

Given:

- Public transportation network
  - Stops
  - Routes / Trips
  - Footpaths

Goal:

- Find optimal s-t-journeys
  - Travel time
  - Number of transfers



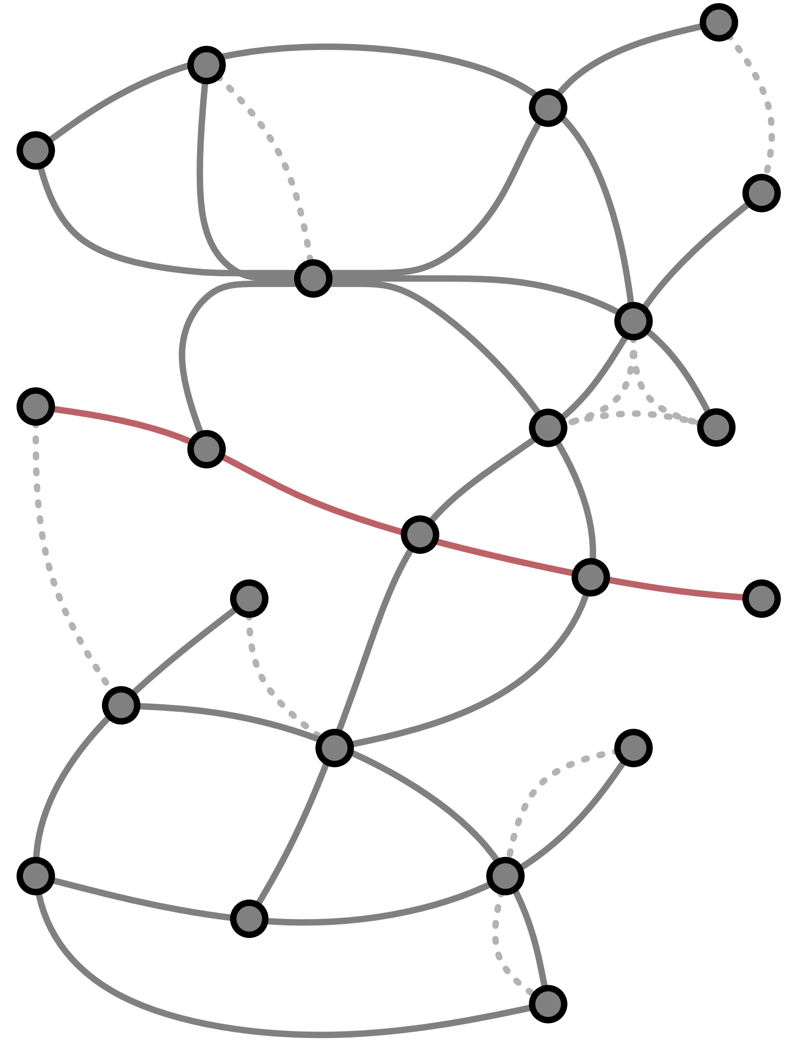
Public Transit Routing

Given:

- Public transportation network
  - Stops
  - Routes / Trips
  - Footpaths

Goal:

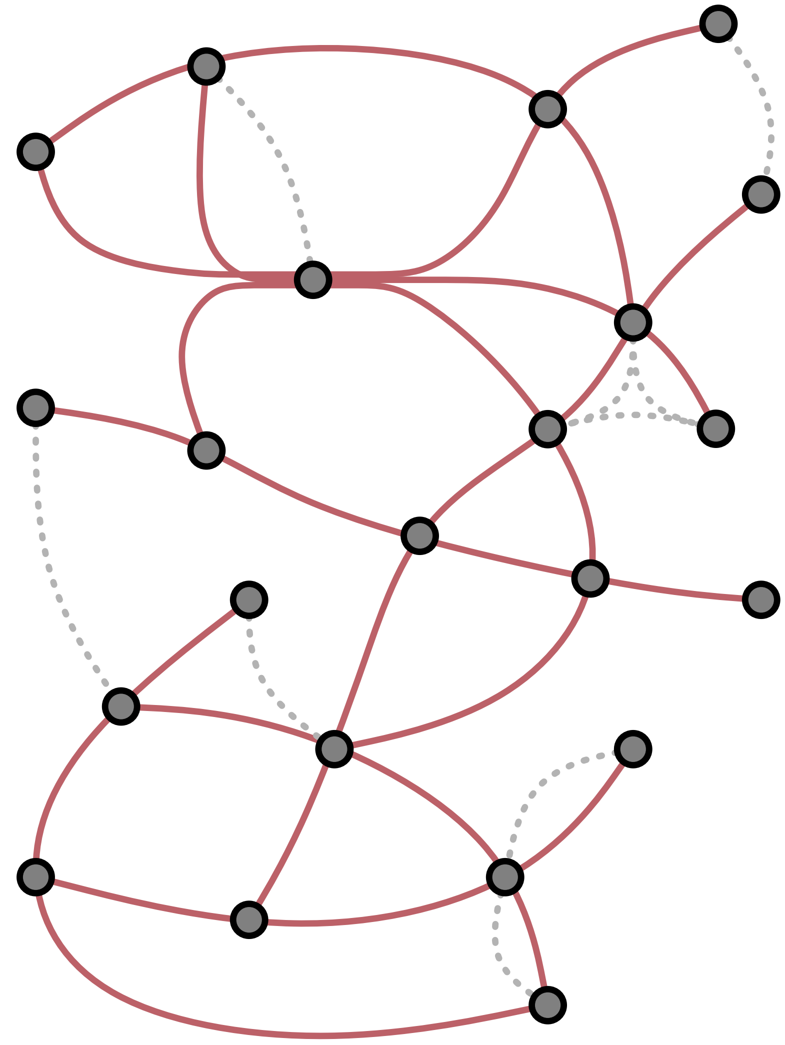
- Find optimal s-t-journeys
  - Travel time
  - Number of transfers



Public Transit Routing

Given:

- Public transportation network
  - Stops
  - Routes / Trips
  - Footpaths



Goal:

- Find optimal s-t-journeys
  - Travel time
  - Number of transfers

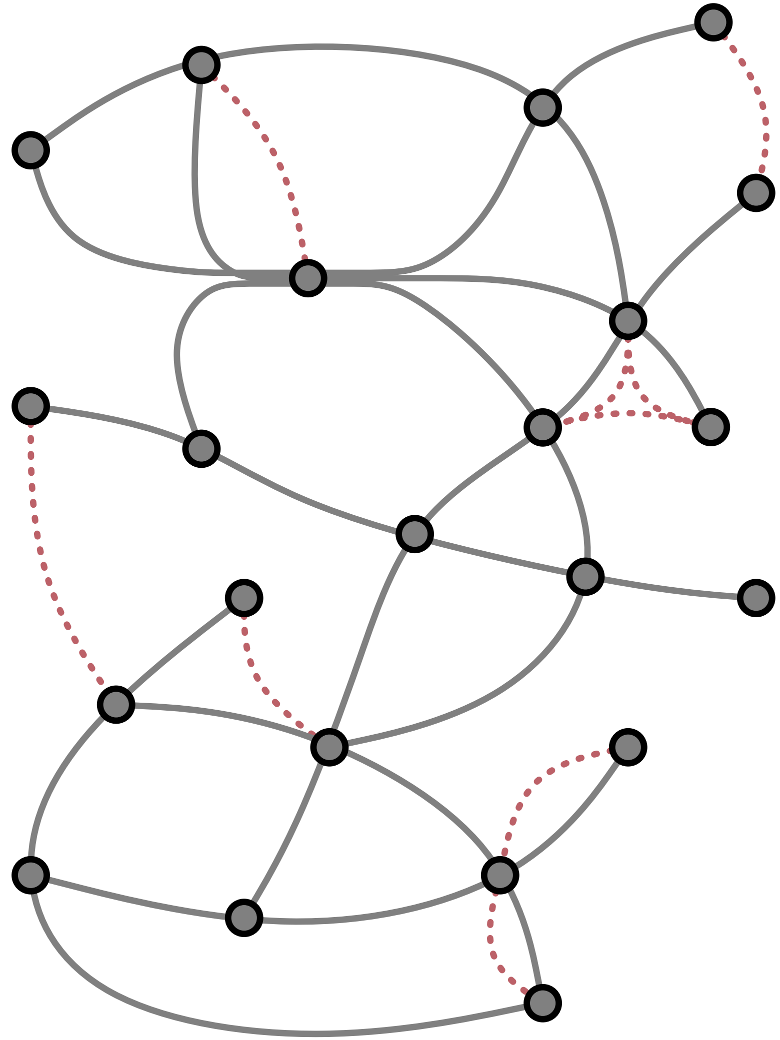
Public Transit Routing

Given:

- Public transportation network
  - Stops
  - Routes / Trips
  - Footpaths

Goal:

- Find optimal s-t-journeys
  - Travel time
  - Number of transfers



Related Work

Partitioning-based Approaches:

- Very successful on road networks
- Have already been adapted for common public transit algorithms

Connection Scan Accelerated [Strasser et al. ’14]

- Partition of the stops
- Has not been evaluated for full multi-criteria optimization

Scalable Transfer Patterns [Bast et al. ’16]

- Partition of the stops
- Preprocessing takes several hours

Related Work – RAPTOR [Delling et al. ’12/’14]

Overview:

- Round based algorithm
- Operates on routes as fundamental object
- One round $\hat{=}$ Using one vehicle

rRAPTOR: **(for profile queries)**

- Collect all possible departures at the source
- Run RAPTOR once for each departure

Properties:

- Easily adapted to additional criteria

- Has not yet been accelerated through preprocessing

Related Work – RAPTOR [Delling et al. ’12/’14]

Overview:

- Round based algorithm
- Operates on routes as fundamental object
- One round $\widehat{=}$ Using one vehicle

rRAPTOR: **(for profile queries)**

- Collect all possible departures at the source
- Run RAPTOR once for each departure

Properties:

- Easily adapted to additional criteria
- Does not require transitively closed footpath graph
- Has not yet been accelerated through preprocessing

Our Approach

Basic Idea:

- Restrict RAPTOR to a subset of the routes
- Therefore, use a partition of the routes
- For every cell of the partition:
  - Identify routes required for traversing the cell (fill-in)

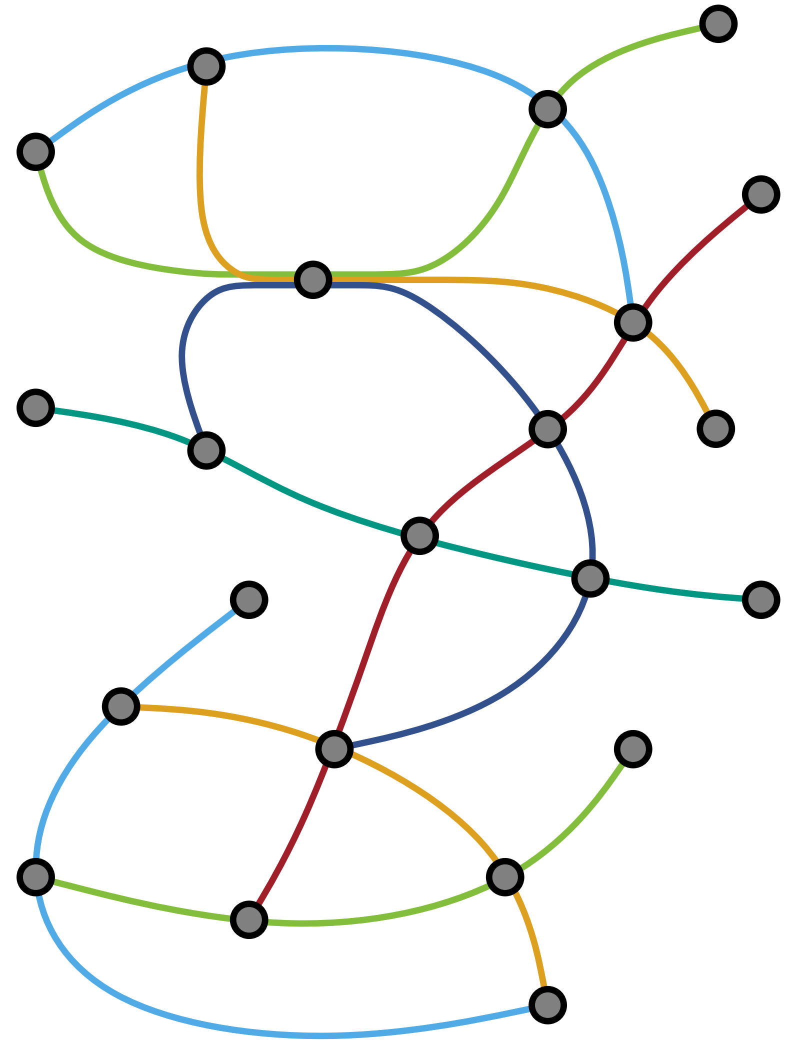
Required Steps:

- Construct the route graph
- Partition the graph
- Compute the fill-in
- Use partition + fill-in to accelerate query

Route graph and Partitioning

Construction:

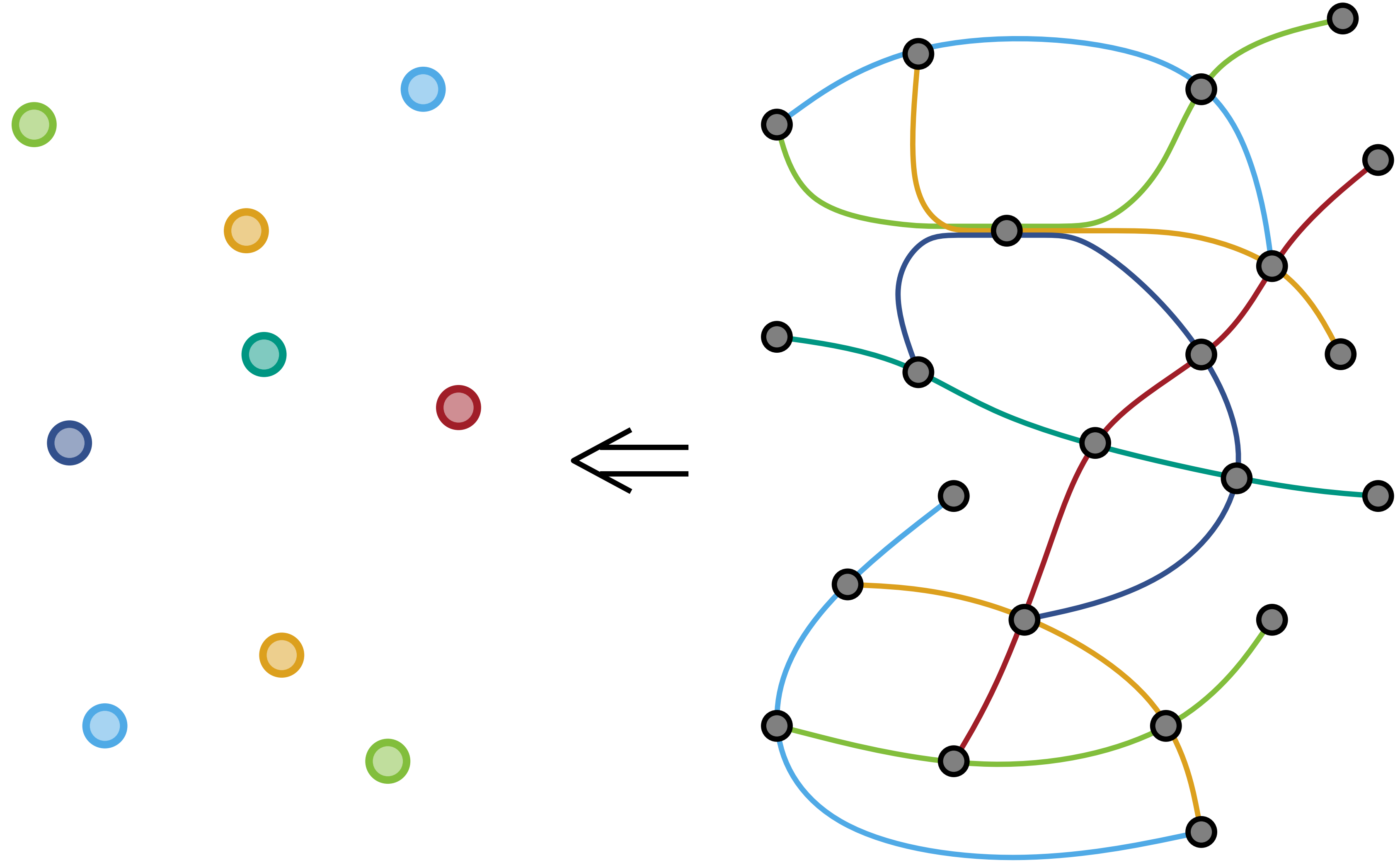
- Create a Vertex for every Route in the network



Route graph and Partitioning

Construction:

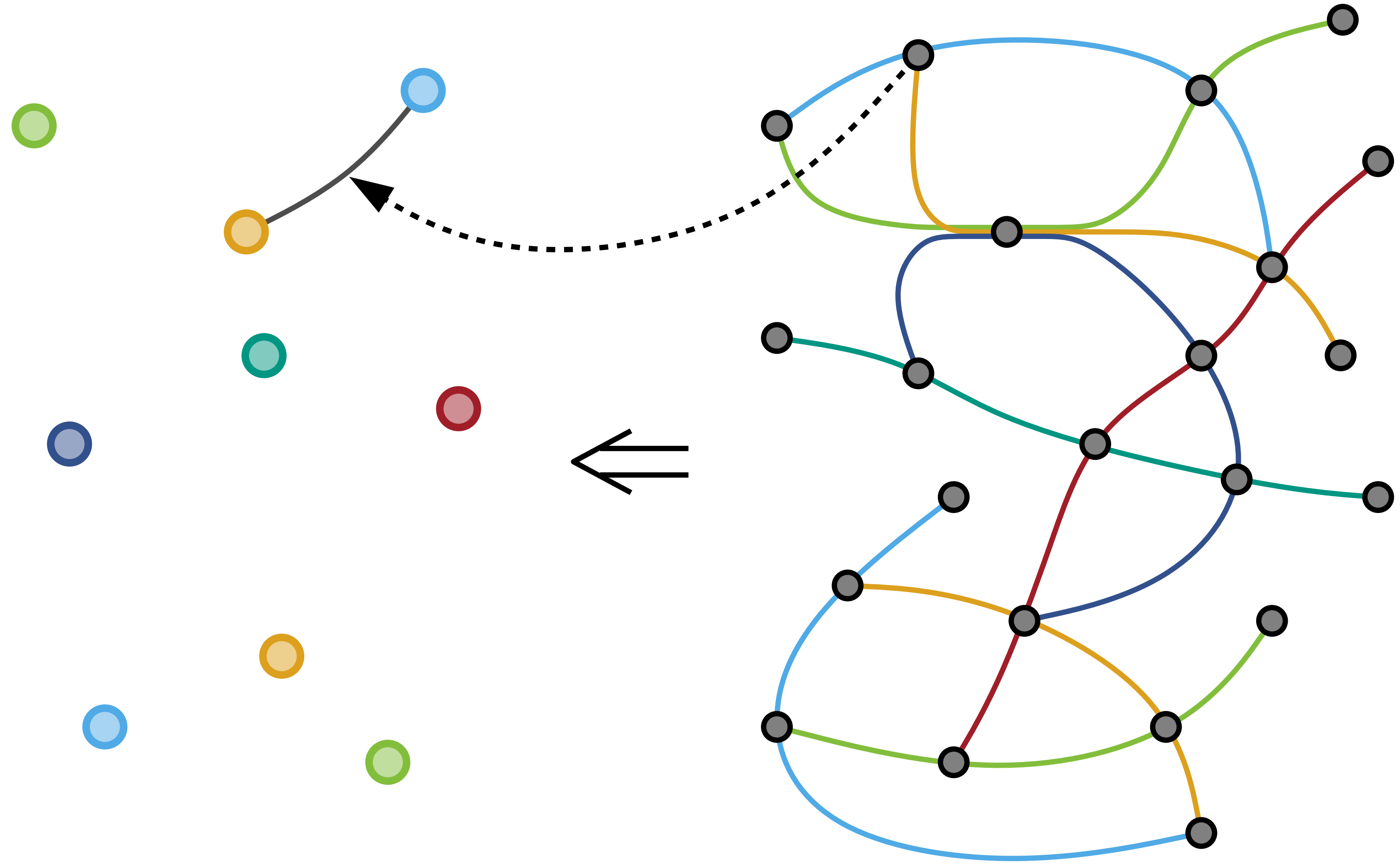
- Create a Vertex for every Route in the network



Route graph and Partitioning

Construction:

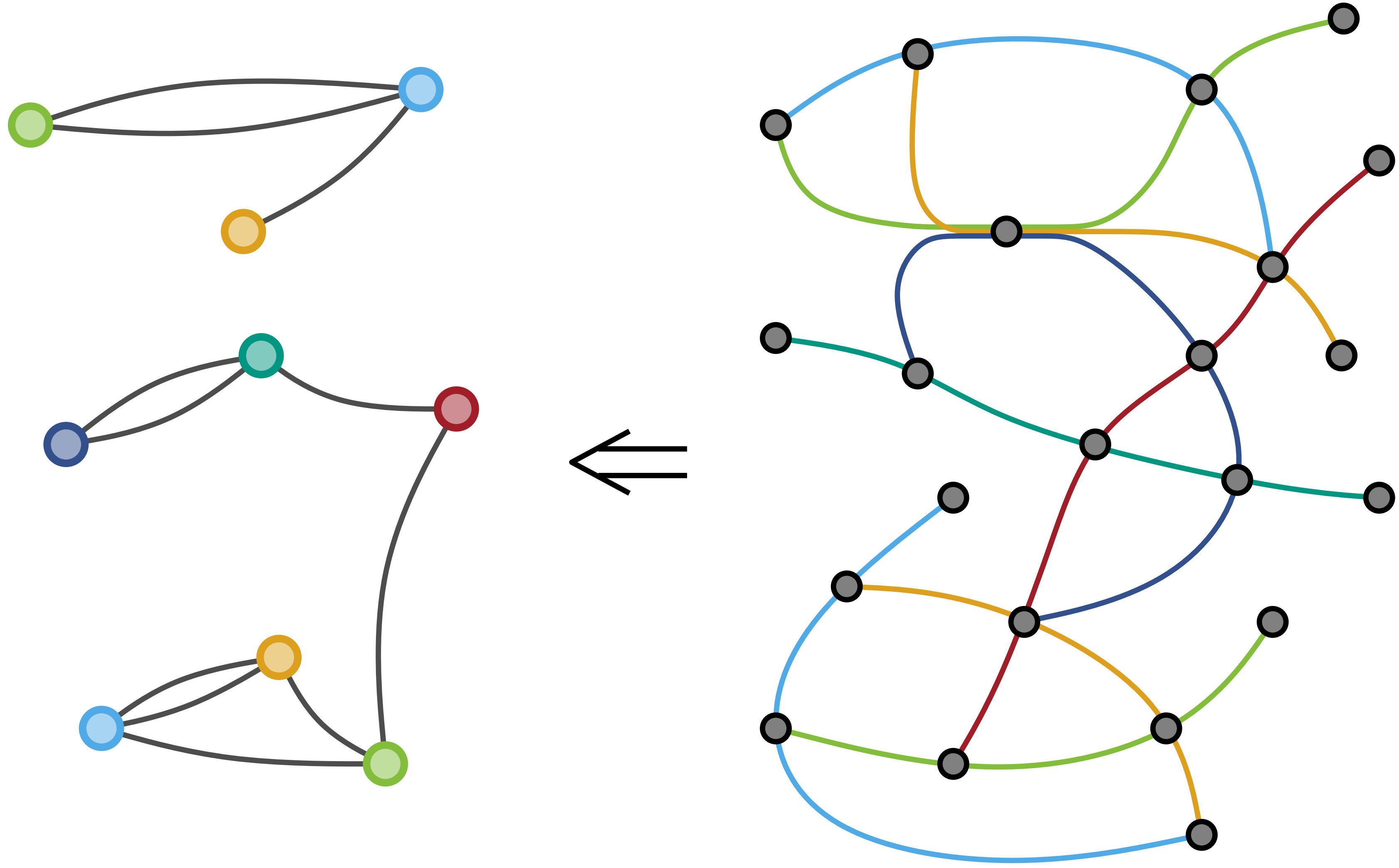
- Vertices are connected by an edge, if they share a stop



Route graph and Partitioning

Construction:

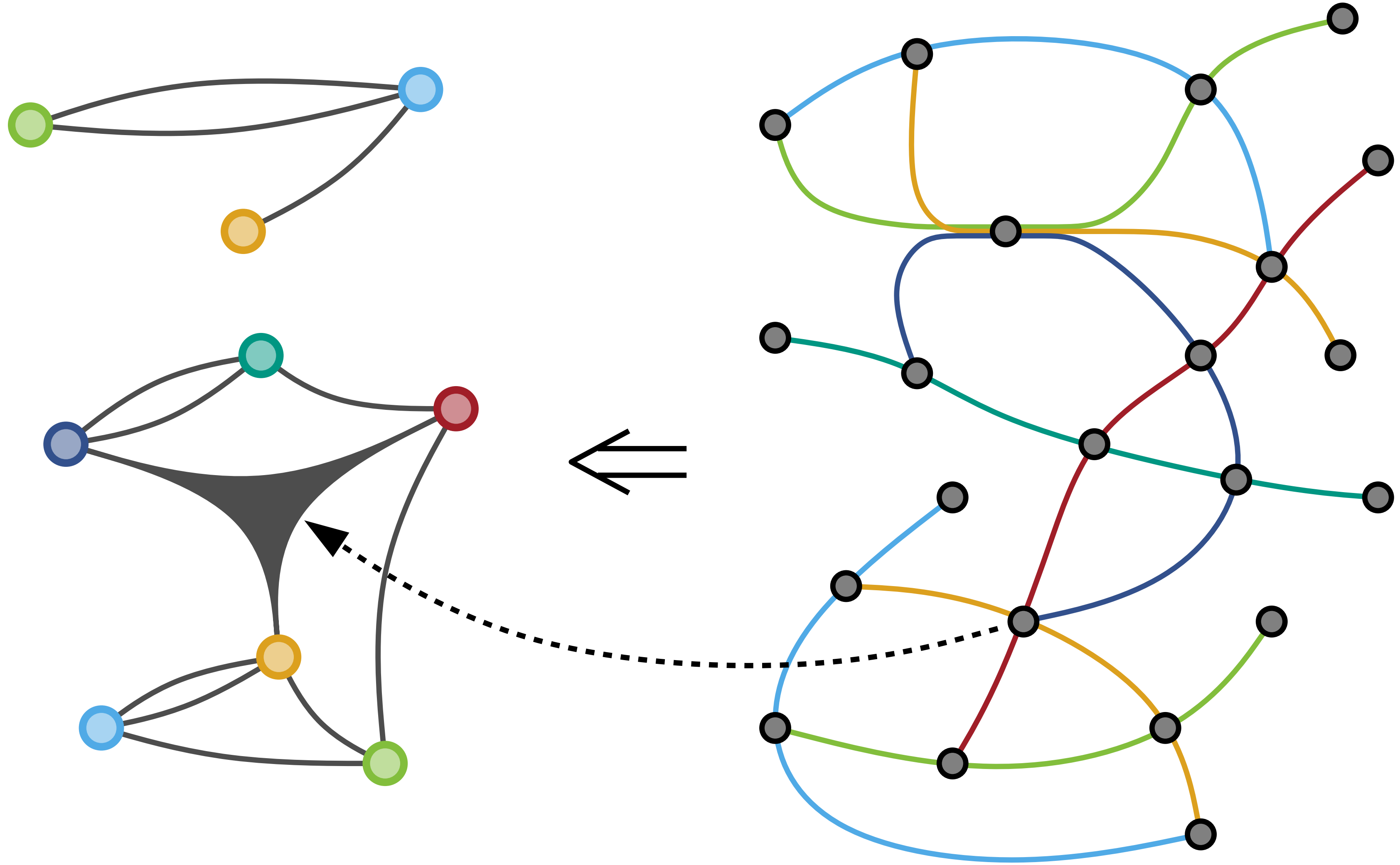
- Vertices are connected by an edge, if they share a stop



Route graph and Partitioning

Construction:

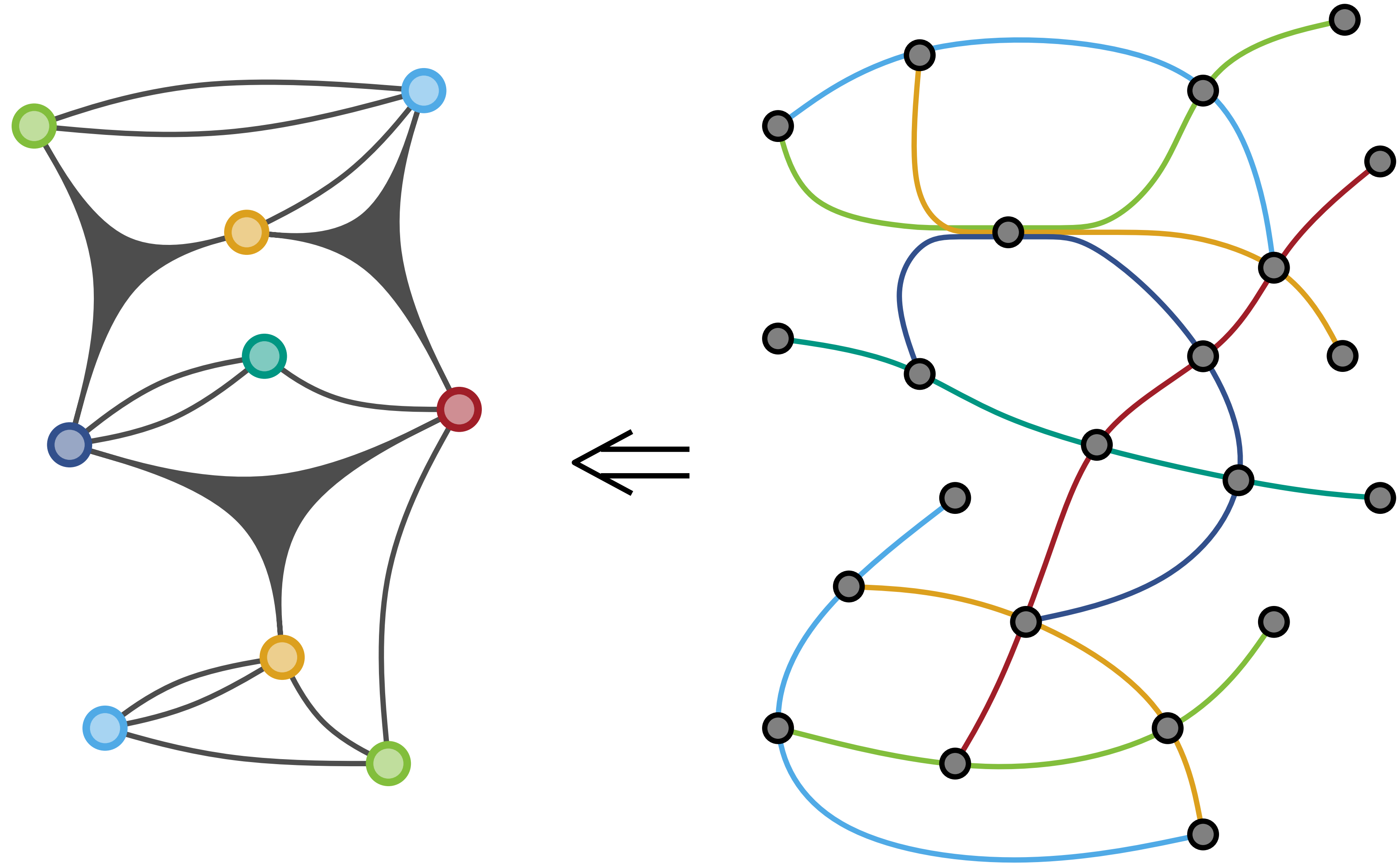
- Stops with more than two routes result in hyperedges



Route graph and Partitioning

Construction:

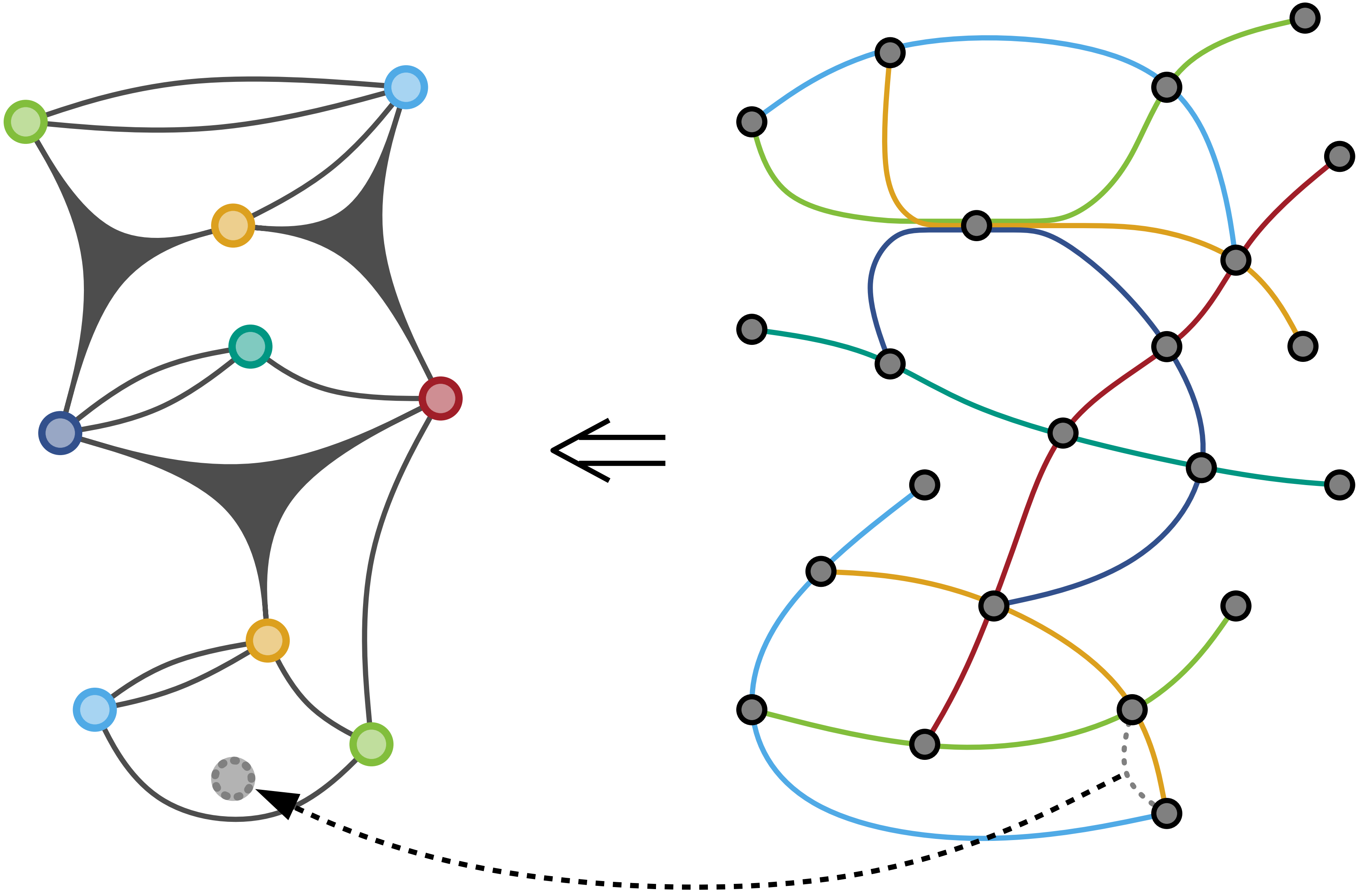
- Stops with more than two routes result in hyperedges



Route graph and Partitioning

Construction:

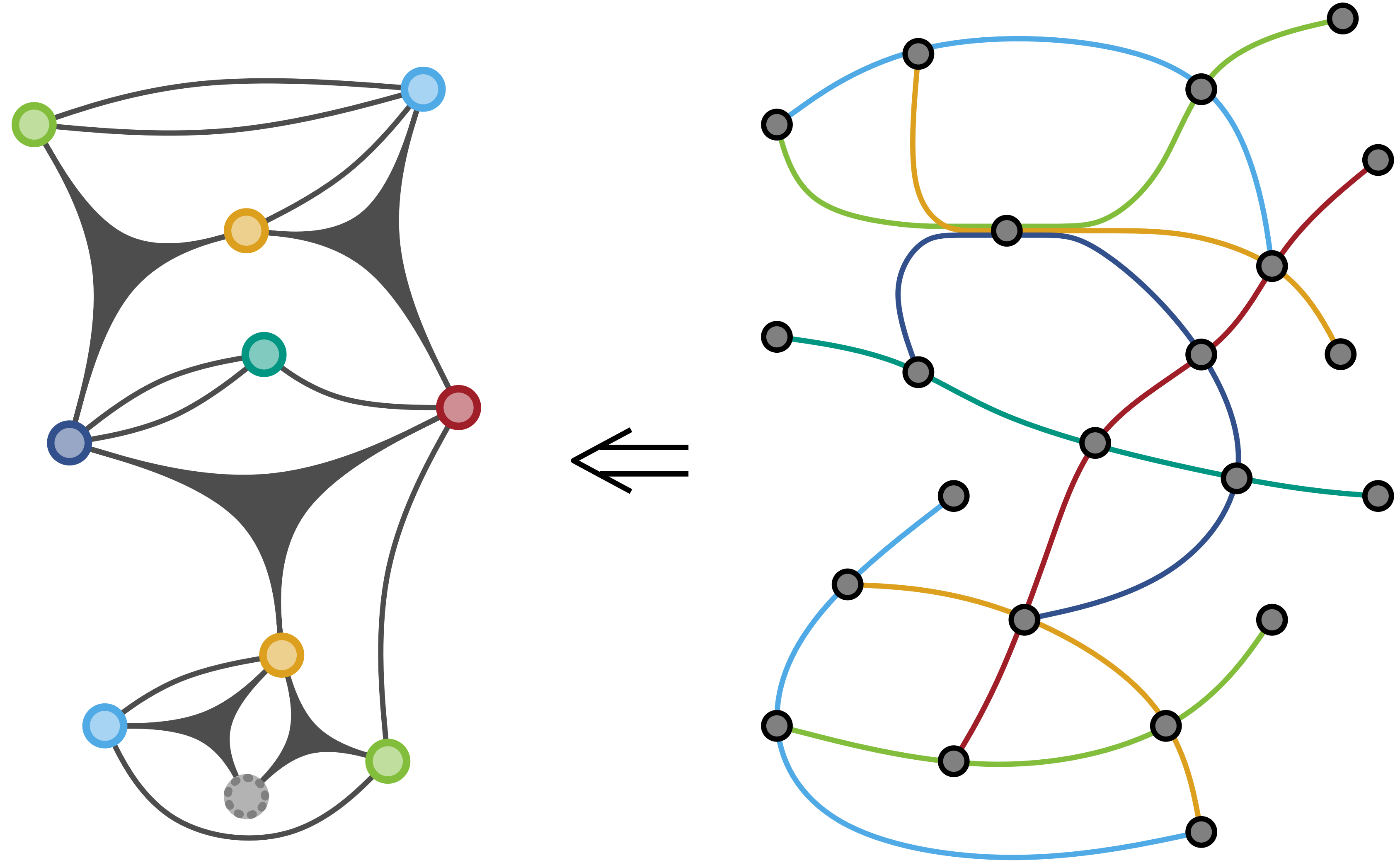
- Footpaths are treated like routes (and become vertices)



Route graph and Partitioning

Construction:

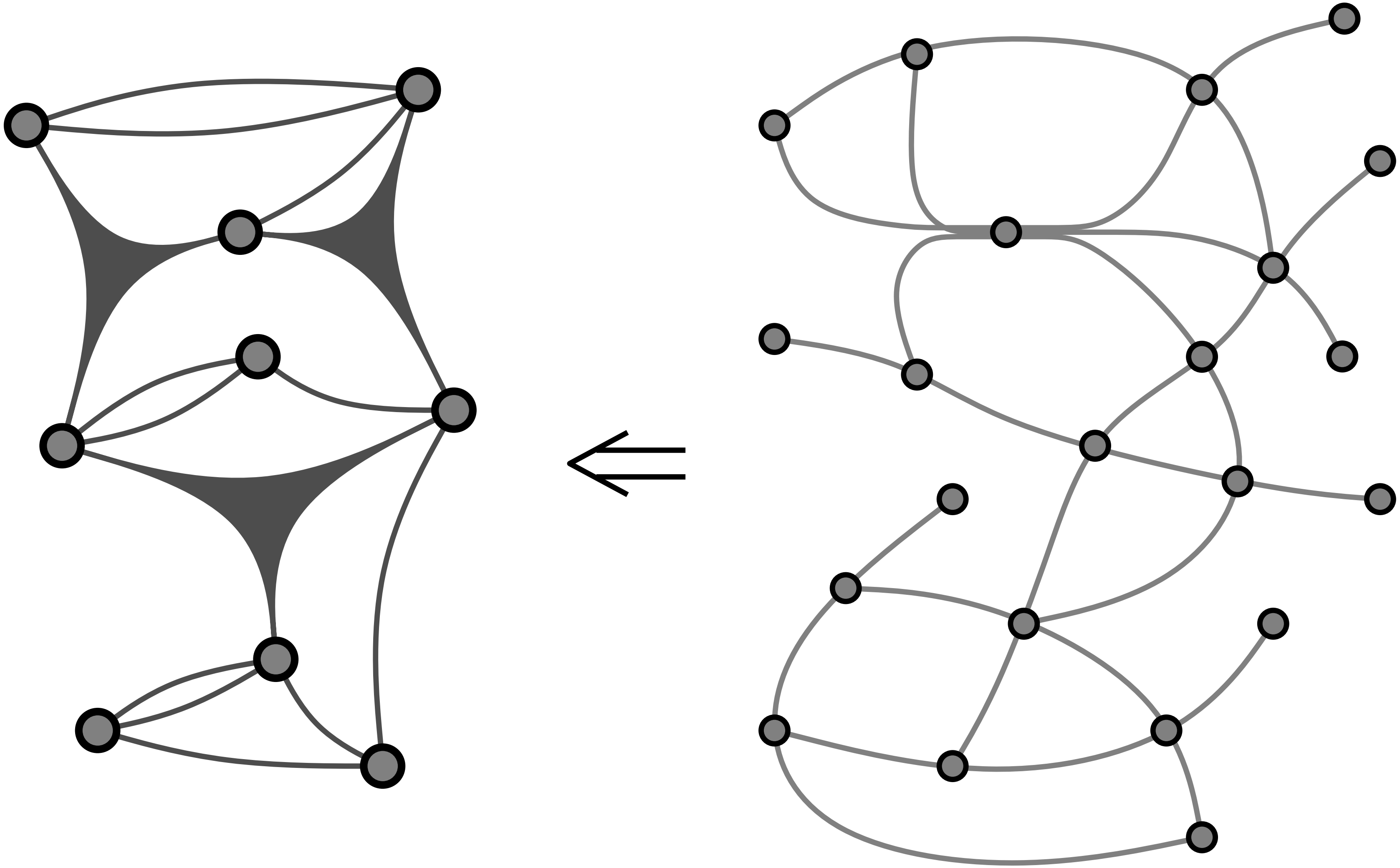
- Vertices are connected by an edge, if they share a stop



Route graph and Partitioning

Construction:

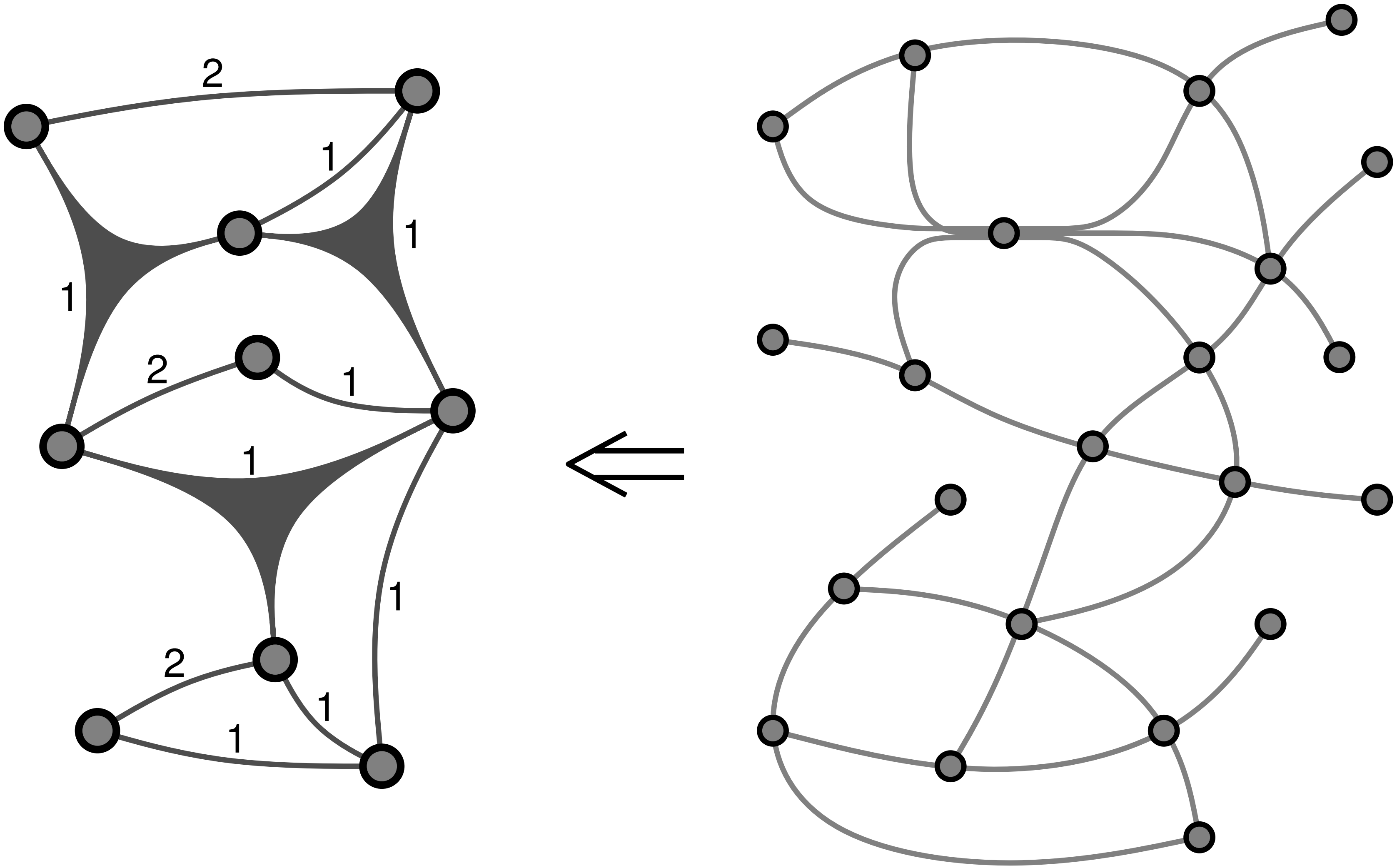
- Vertices are connected by an edge, if they share a stop



Route graph and Partitioning

Construction:

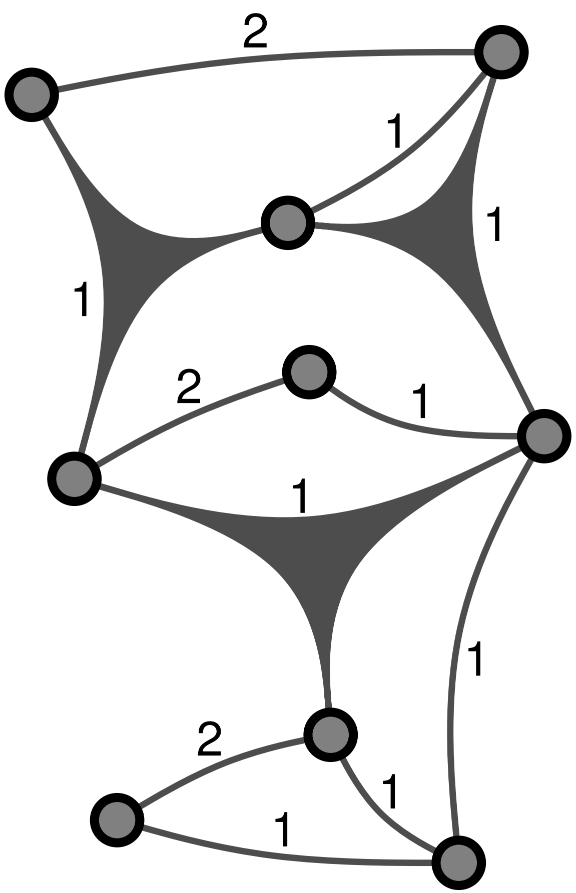
- Finally, multi-edges can be replaced by weighted edges

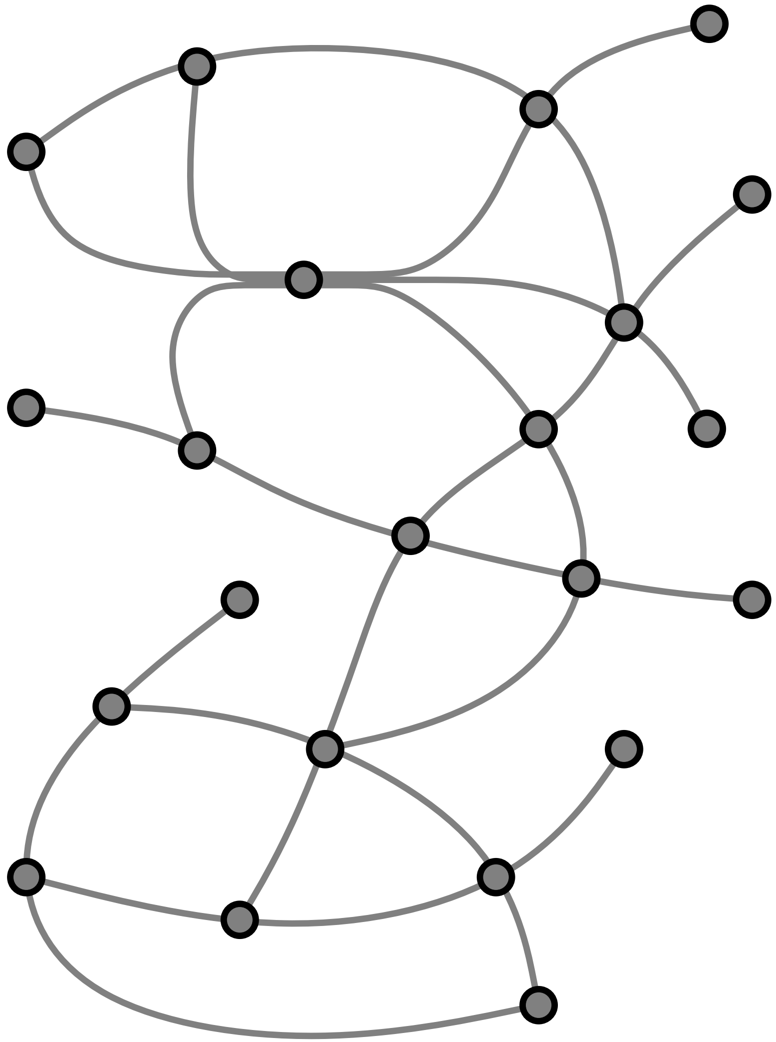


Route graph and Partitioning

Partitioning:

- Find a minimal edge cut with balanced cells

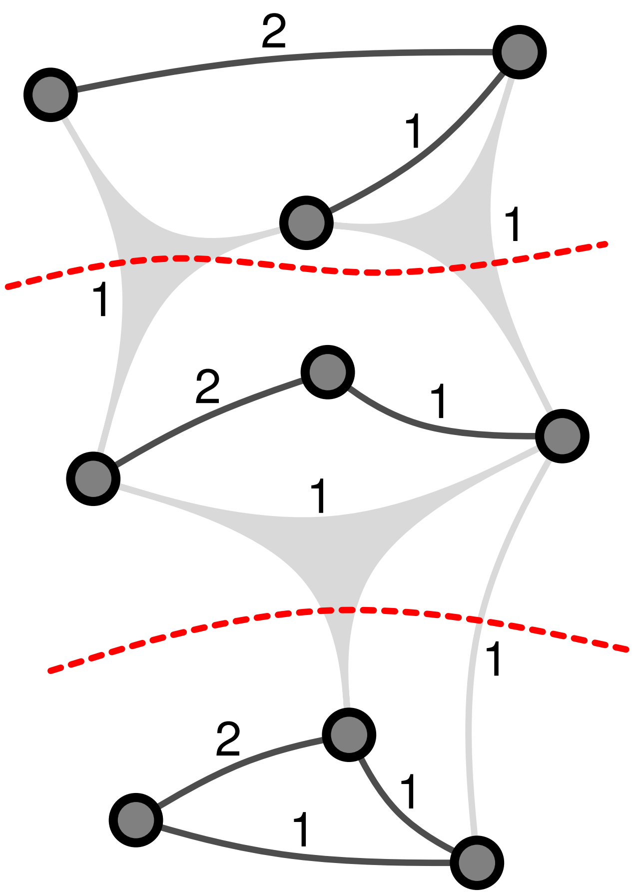


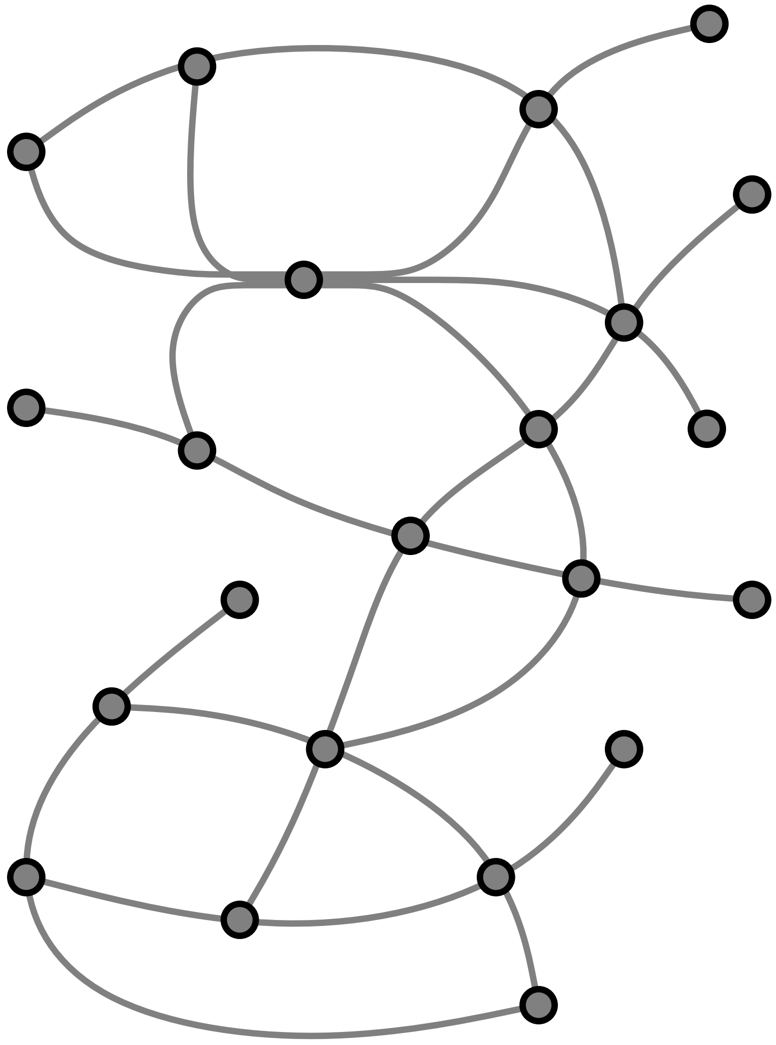


Route graph and Partitioning

Partitioning:

- Find a minimal edge cut with balanced cells

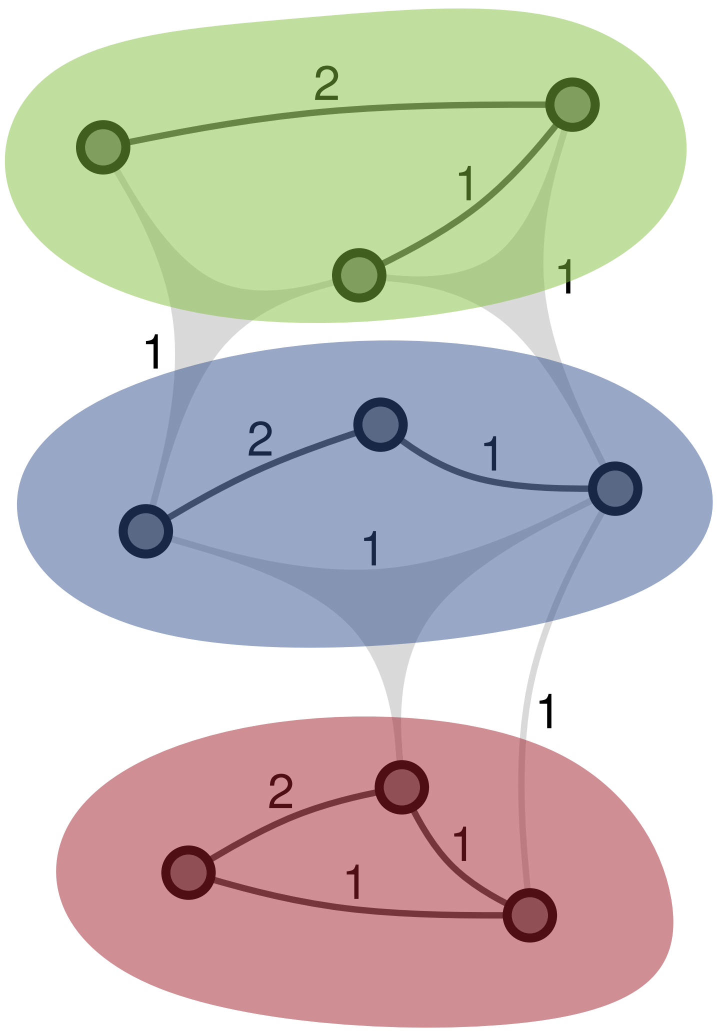


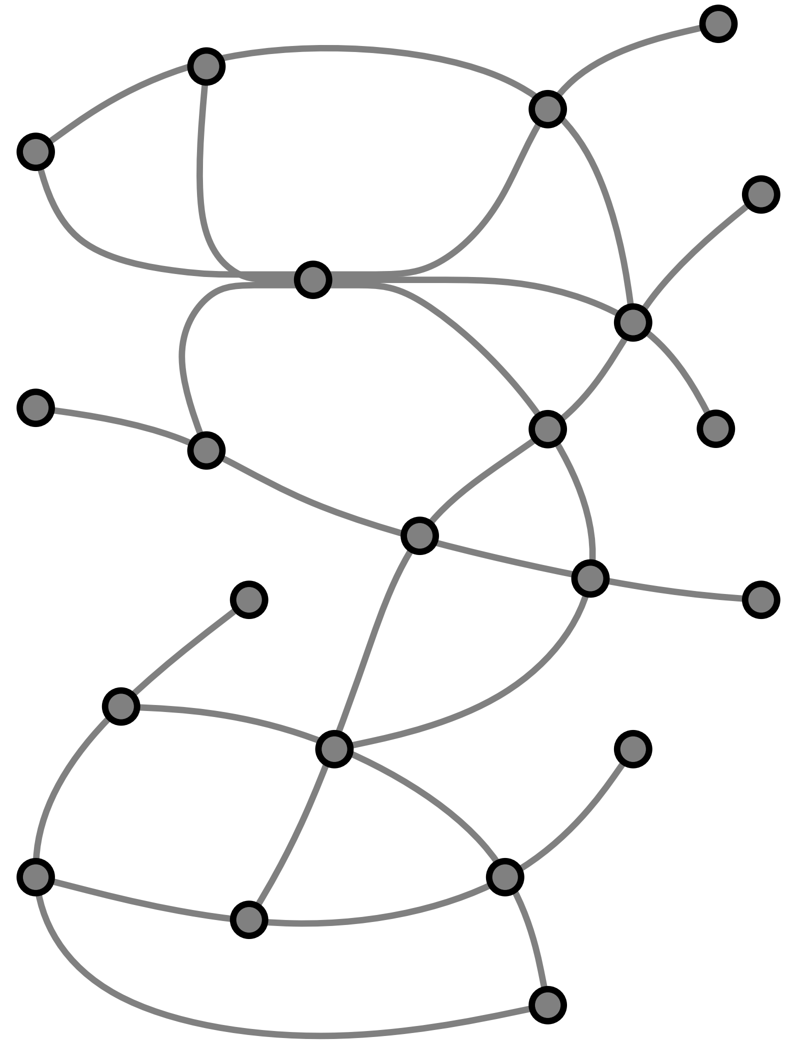


Route graph and Partitioning

Partitioning:

- Find a minimal edge cut with balanced cells

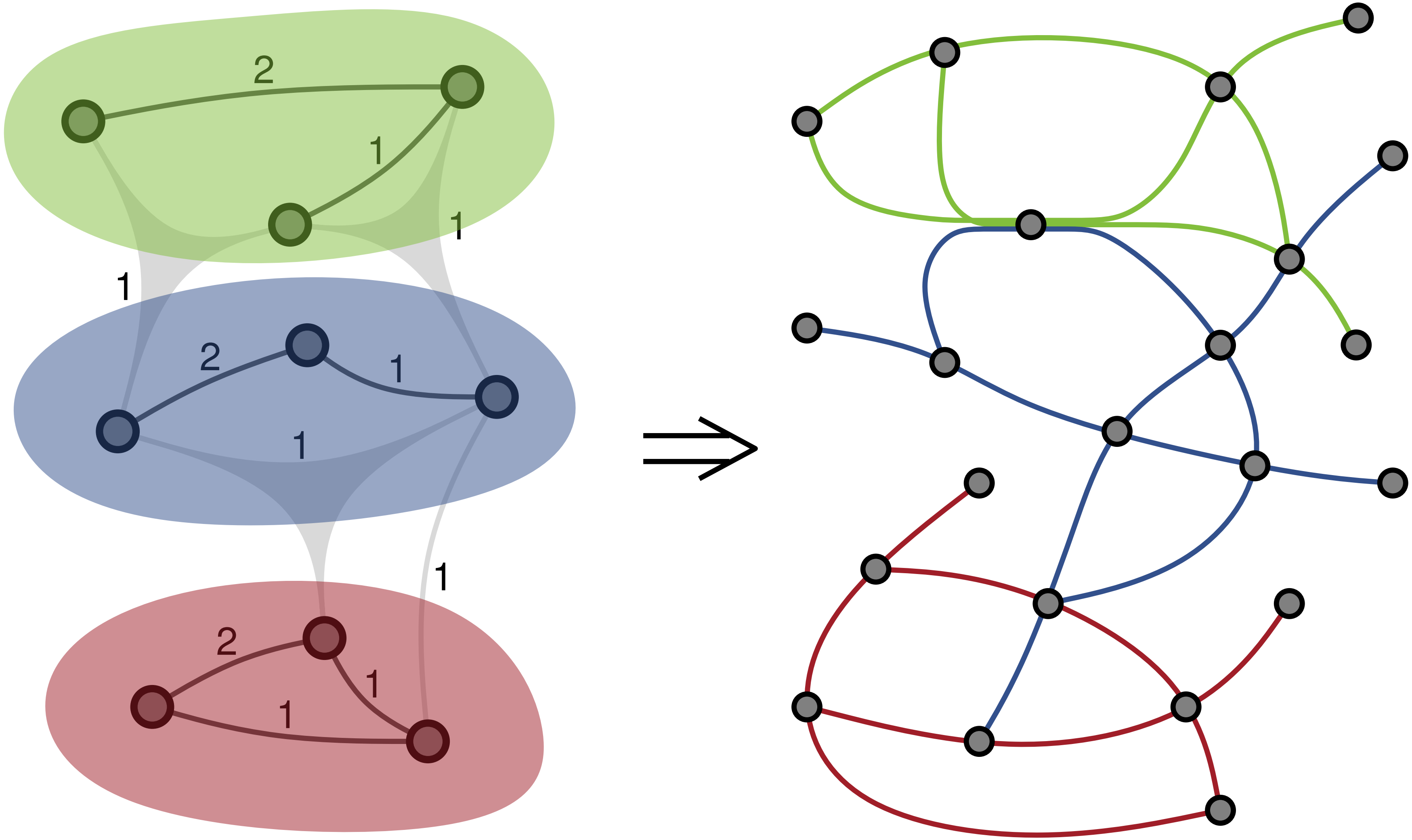




Route graph and Partitioning

Partitioning:

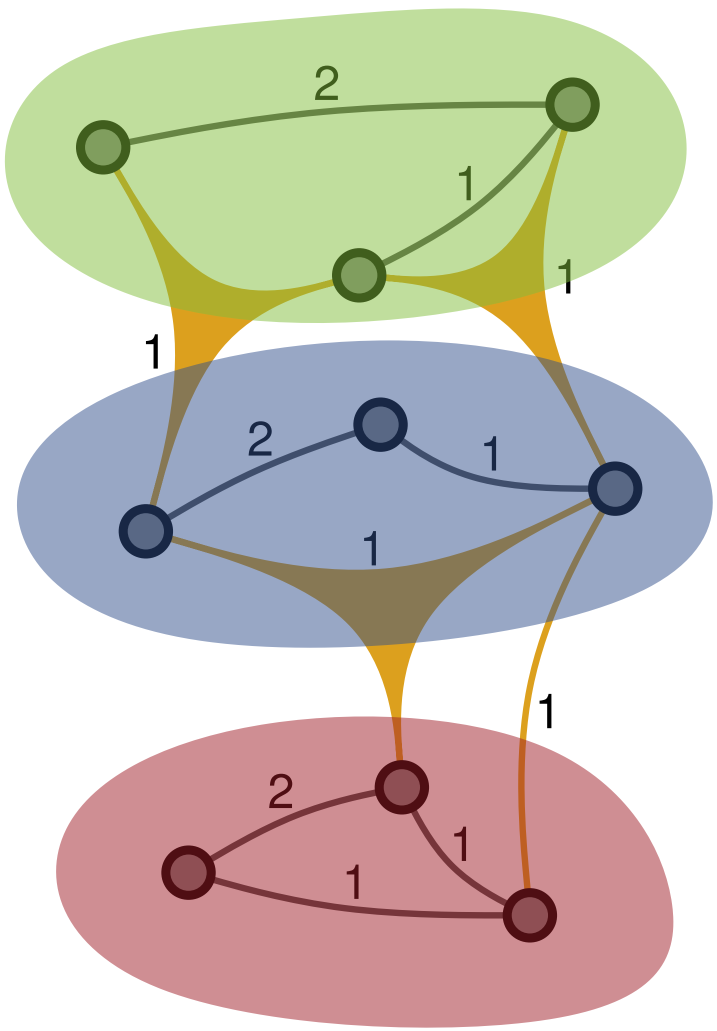
- Cells of the partition correspond to sets of routes



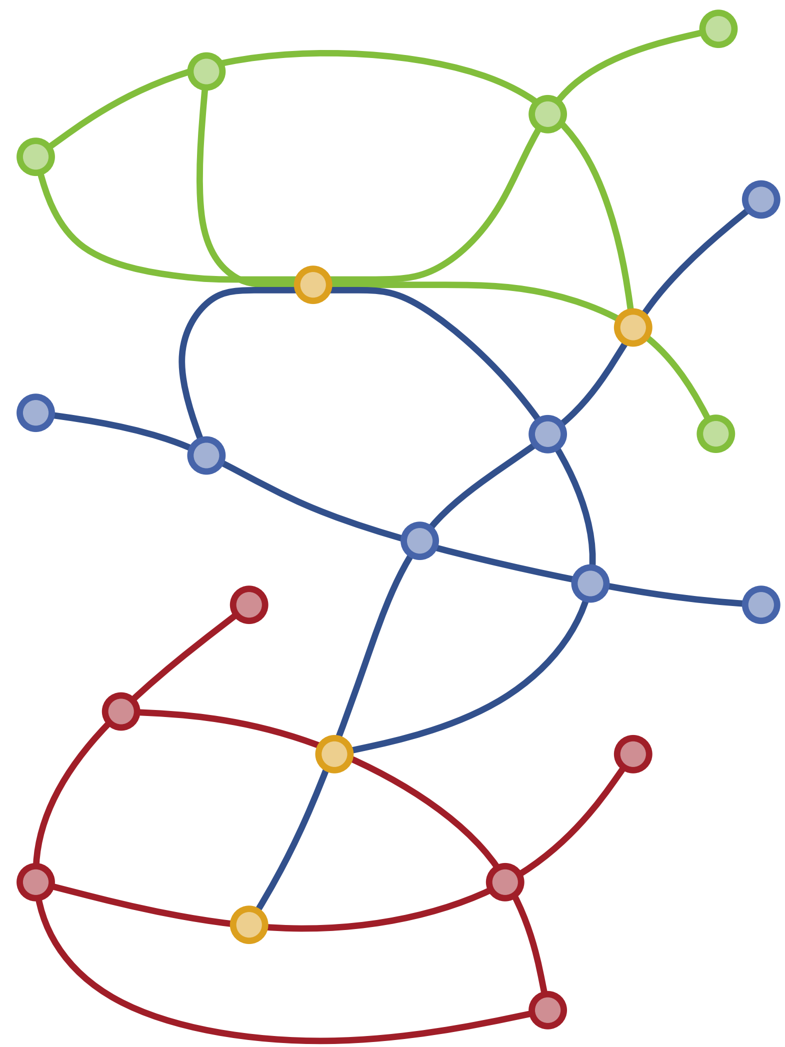
Route graph and Partitioning

Partitioning:

- Cut edges correspond to cut stops



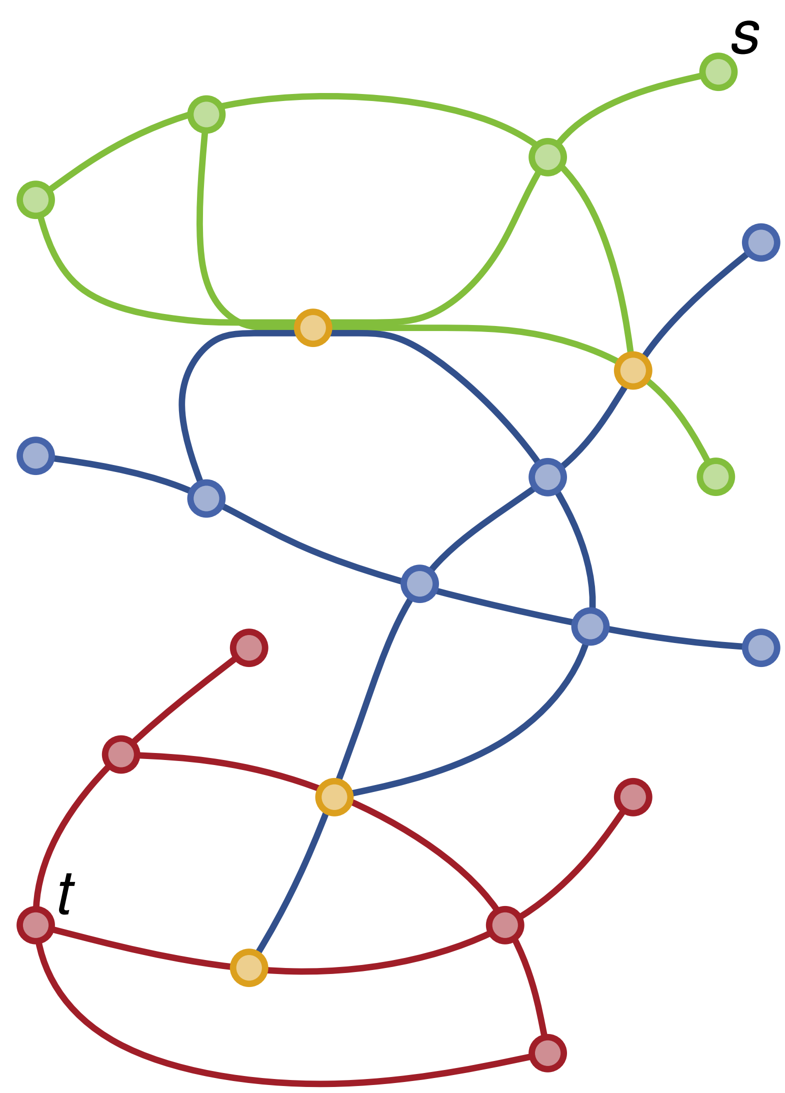
⇒



Query Algorithm

Idea:

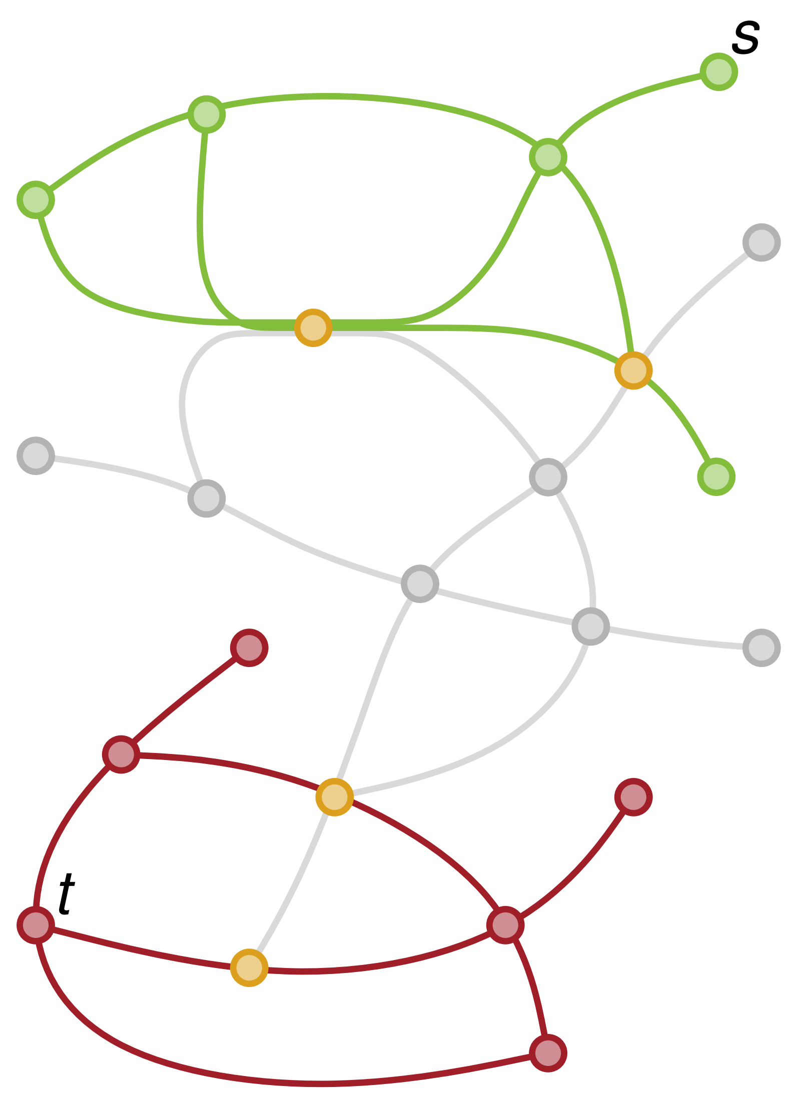
- RAPTOR restricted to:
  - Source cell
  - Target cell



Query Algorithm

Idea:

- RAPTOR restricted to:
  - Source cell
  - Target cell



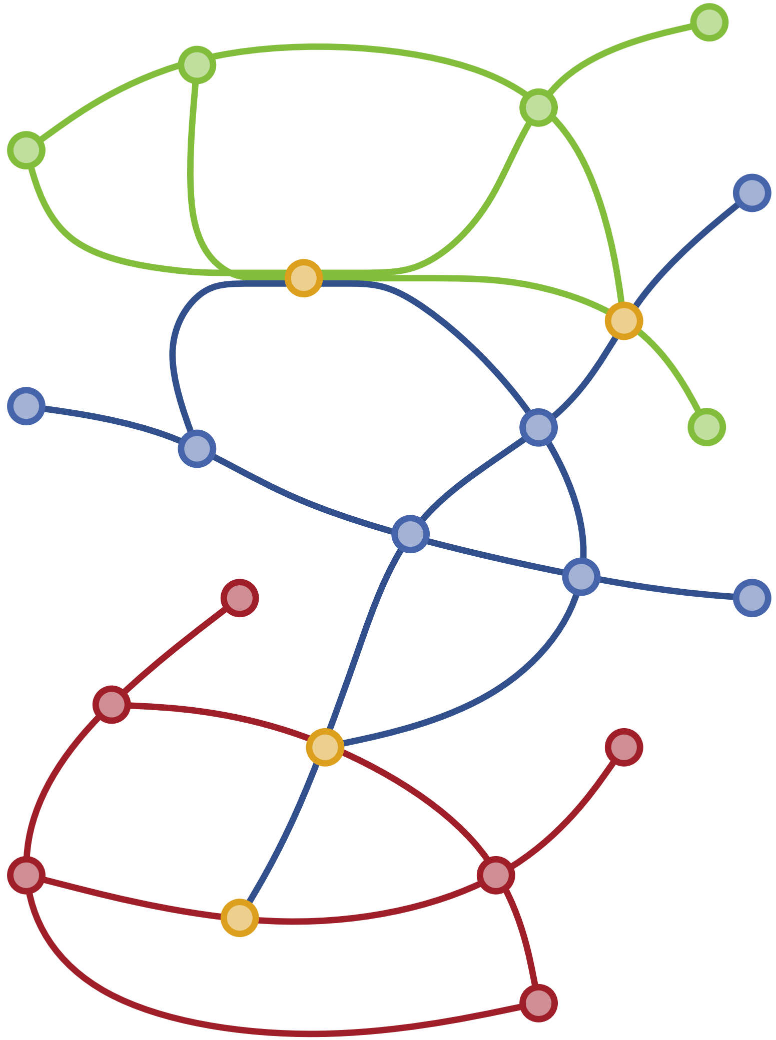
Problem:

- Other cells have to be traversed

Fill-In Computation

Goal:

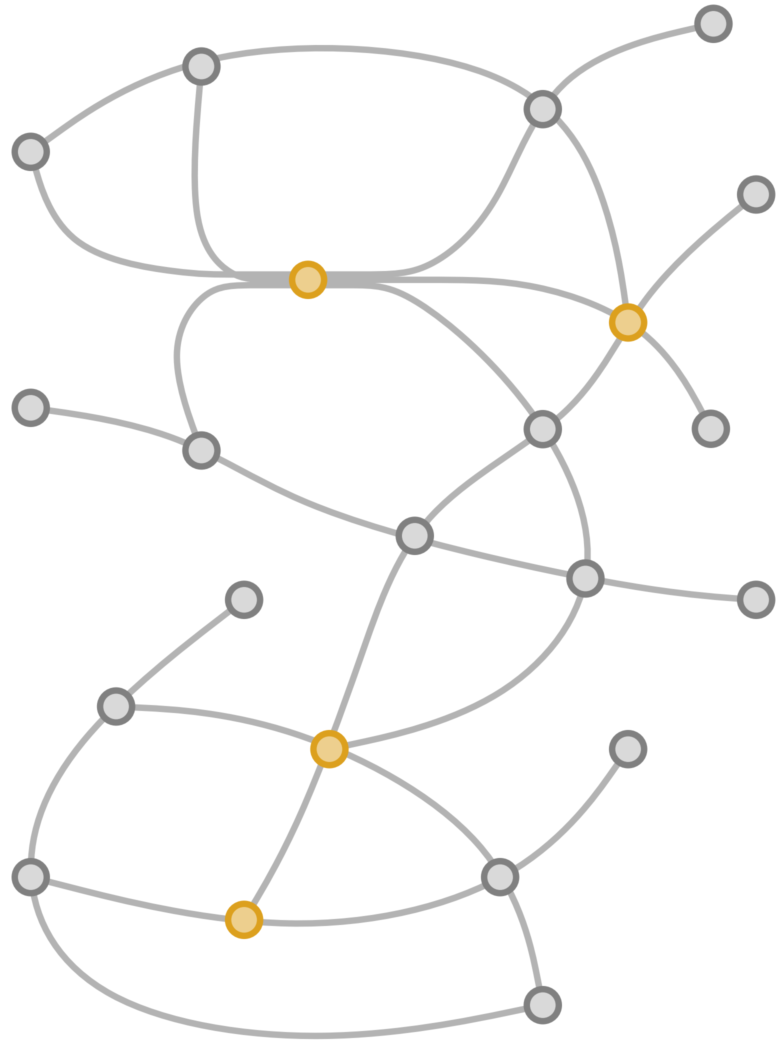
- Find routes required for traveling between cut stops



Fill-In Computation

Goal:

- Find routes required for traveling between cut stops



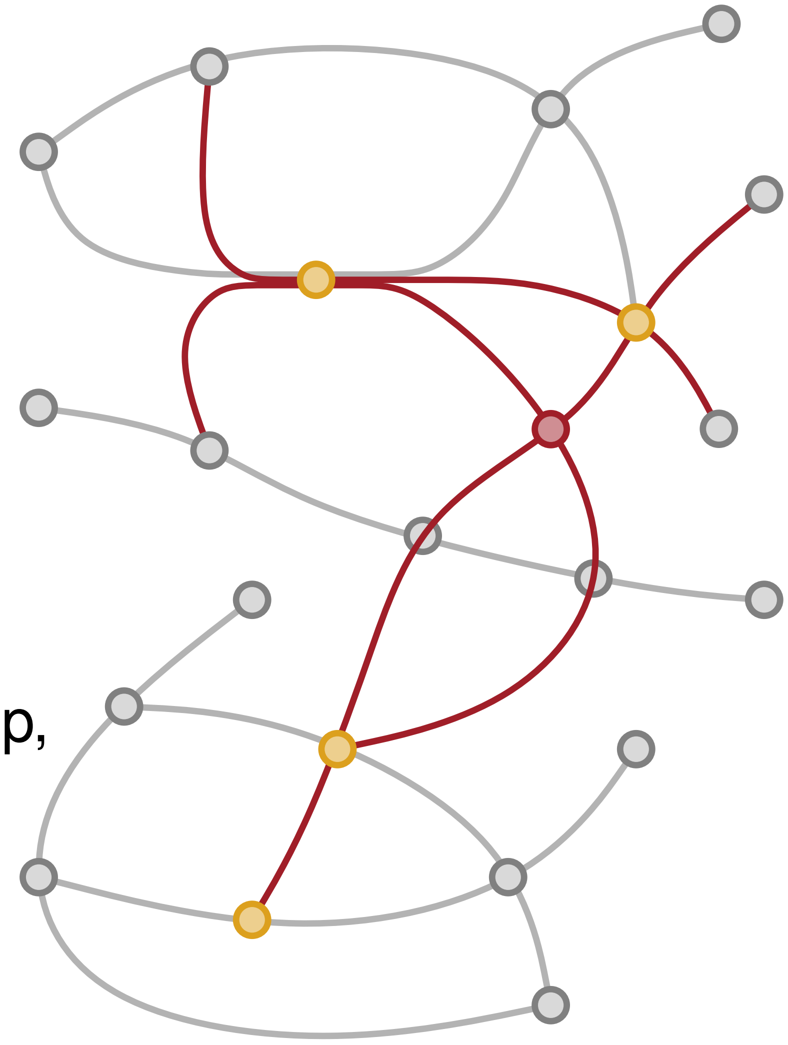
Fill-In Computation

Goal:

- Find routes required for traveling between cut stops

Approaches:

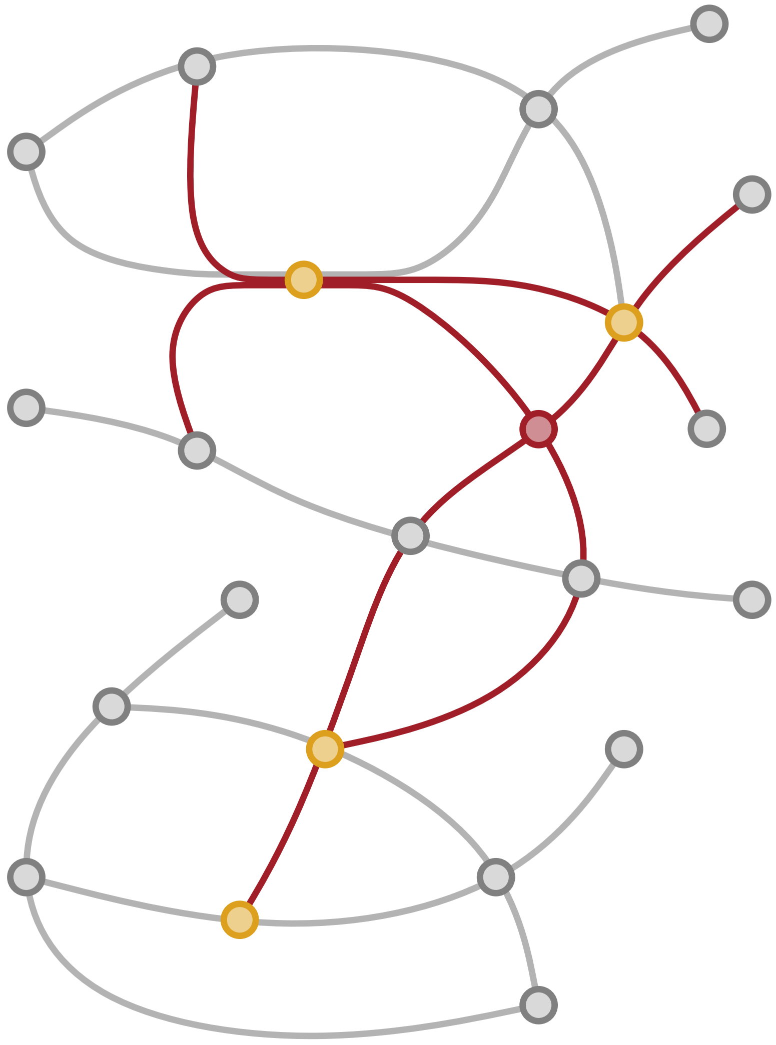
- Trade off between preprocessing time and fill-in size:
  1. Run rRAPTOR once for every cut stop
  2. Run rRAPTOR for every cut stop, restricted to adjacent cells
  3. Run rRAPTOR for every pair of cell and cut stop



Fill-In Representation

Problem:

- Not all trips of a route have to be part of the fill-in
- Not all stops of a route have to be part of the fill-in
- How can the essential parts of the route be represented?



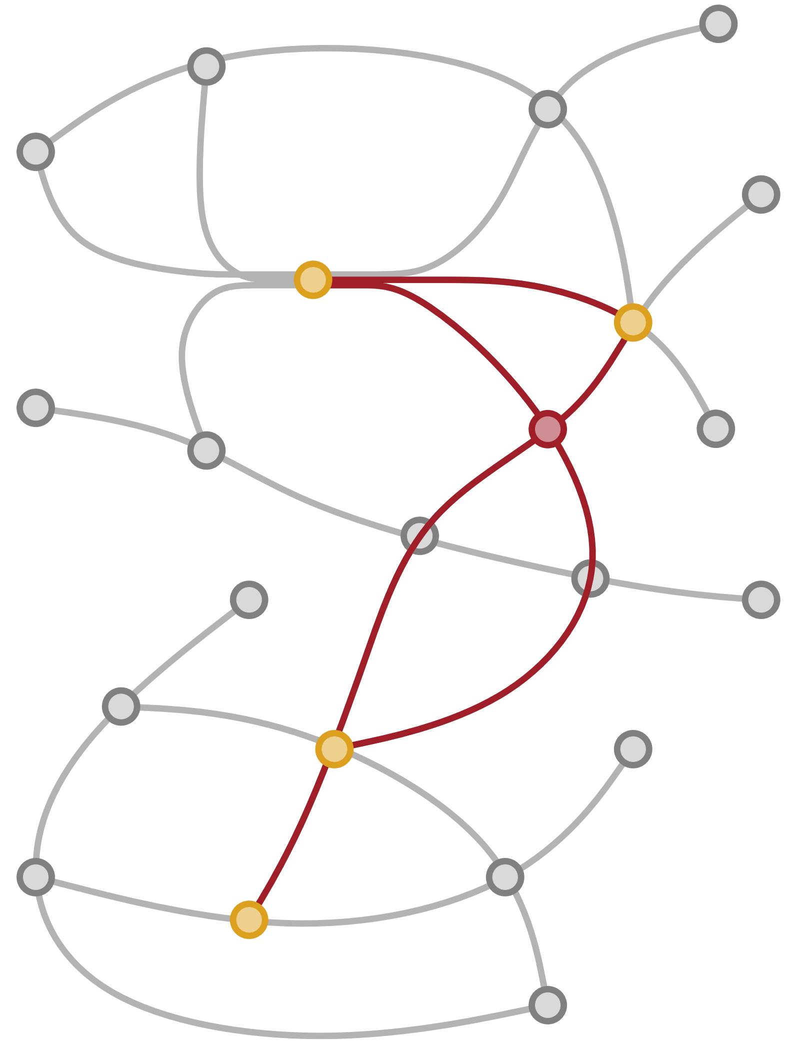
Fill-In Representation

Problem:

- Not all trips of a route have to be part of the fill-in
- Not all stops of a route have to be part of the fill-in
- How can the essential parts of the route be represented?

Approaches:

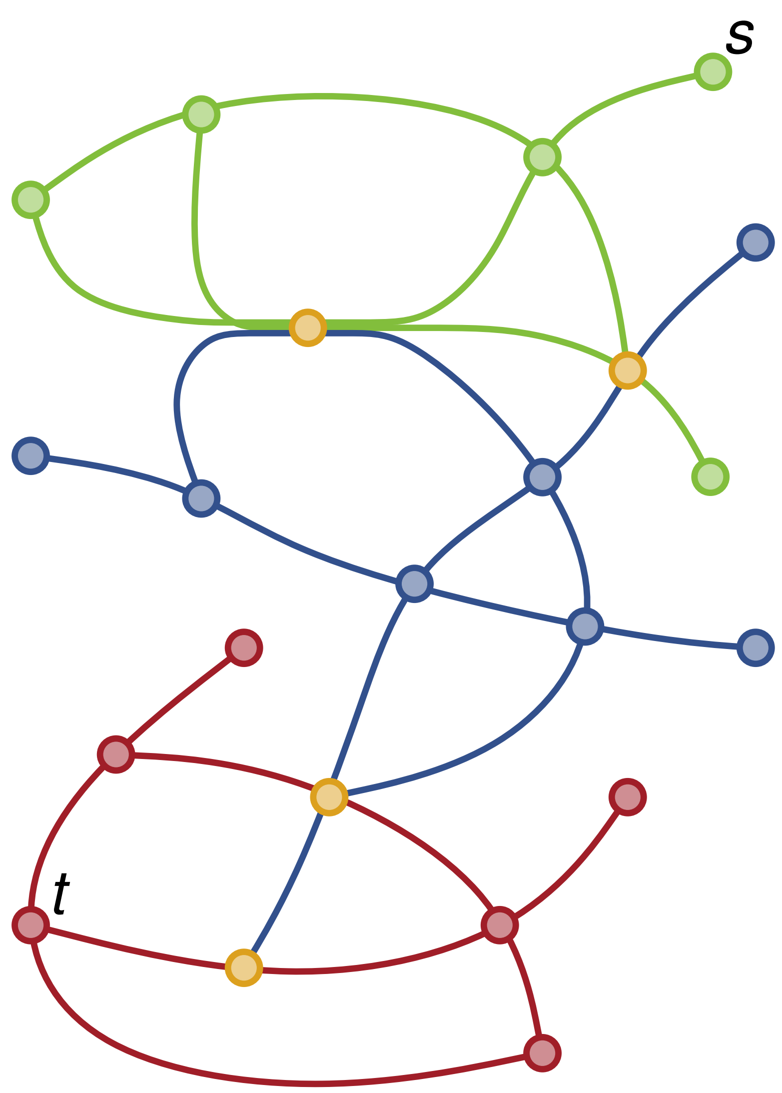
- Mark important events with flags
- Precompute offsets between important events
- Create compressed fill-in routes



Query Algorithm

Idea:

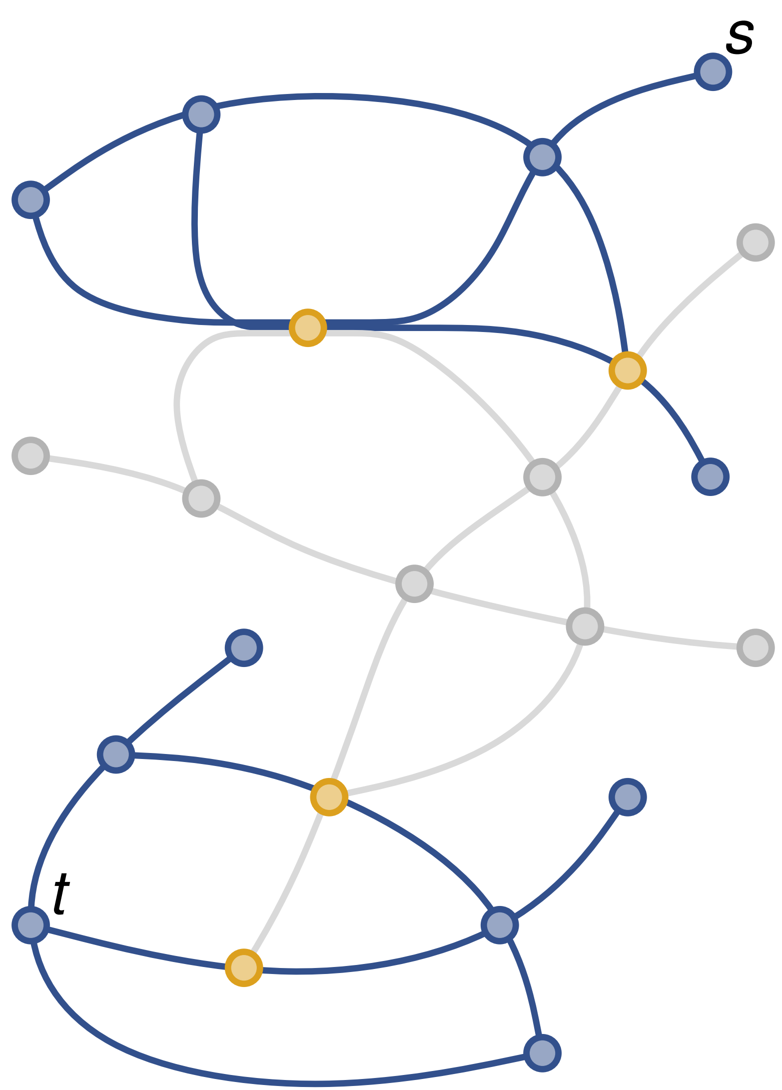
- RAPTOR restricted to:
  - Source cell
  - Target cell



Query Algorithm

Idea:

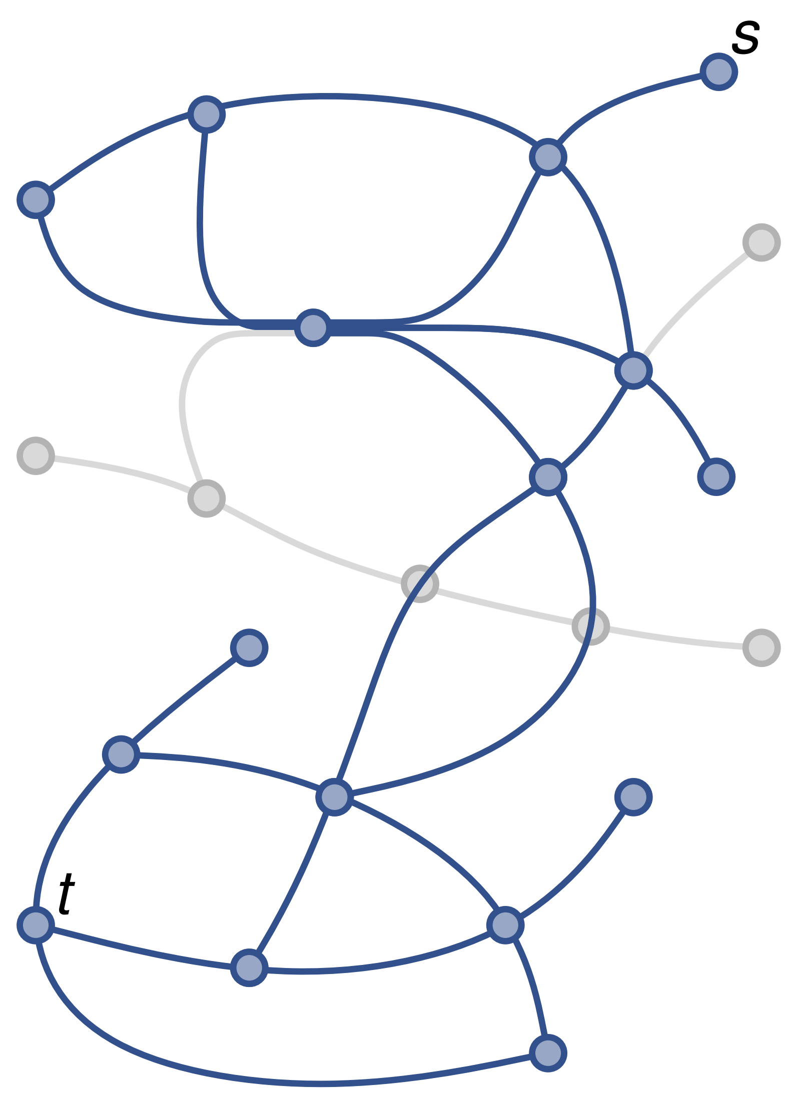
- RAPTOR restricted to:
  - Source cell
  - Target cell



Query Algorithm

Idea:

- RAPTOR restricted to:
  - Source cell
  - Target cell
  - Fill-in



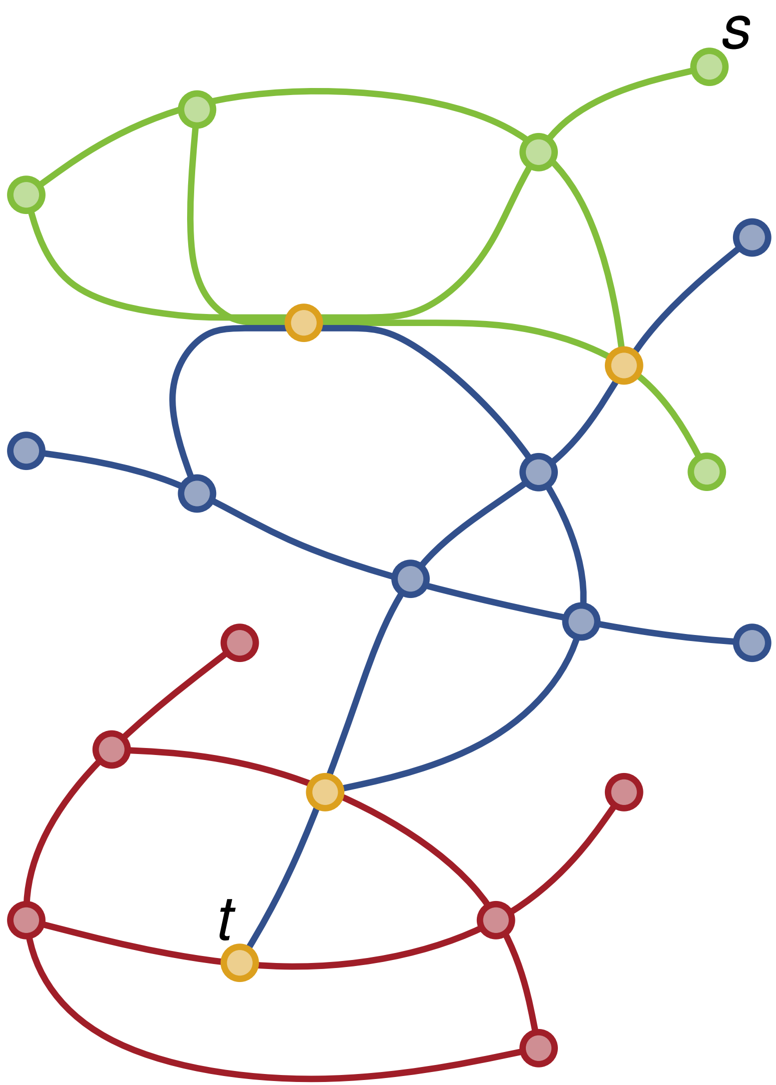
Query Algorithm

Idea:

- RAPTOR restricted to:
  - Source cell
  - Target cell
  - Fill-in

Special Case:

- Source or target is a cut stop



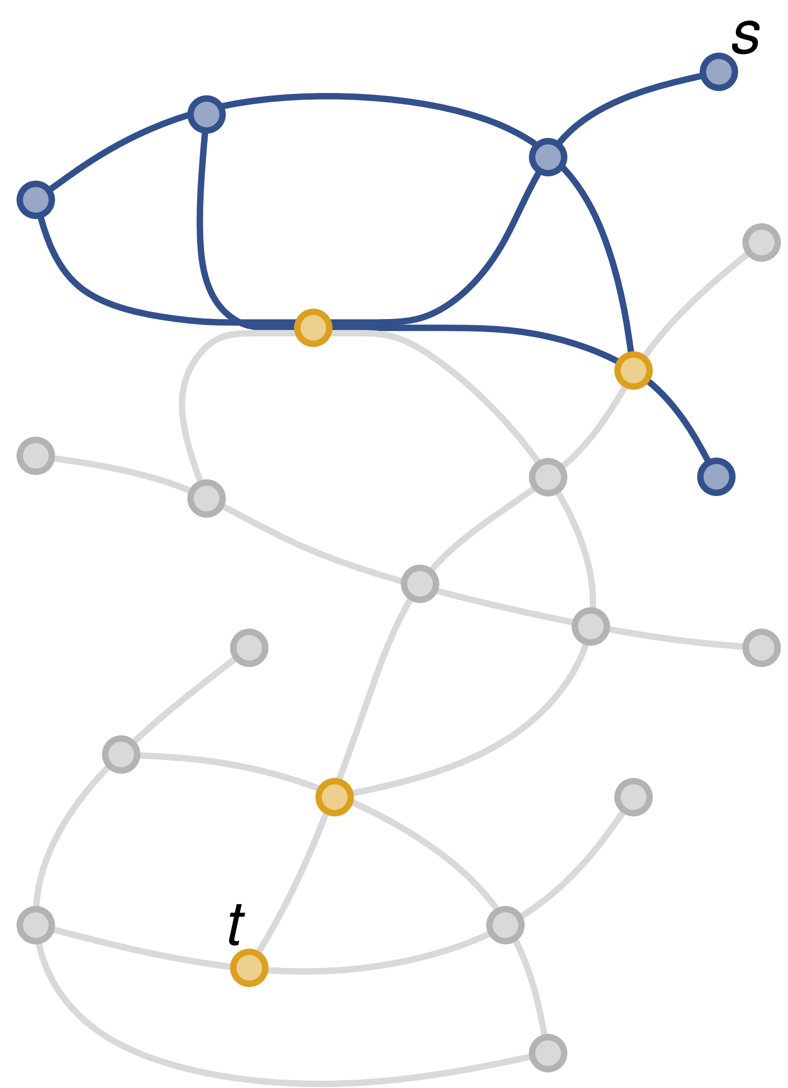
Query Algorithm

Idea:

- RAPTOR restricted to:
  - Source cell
  - Target cell
  - Fill-in

Special Case:

- Source or target is a cut stop
- Cut stops are not part of any cell



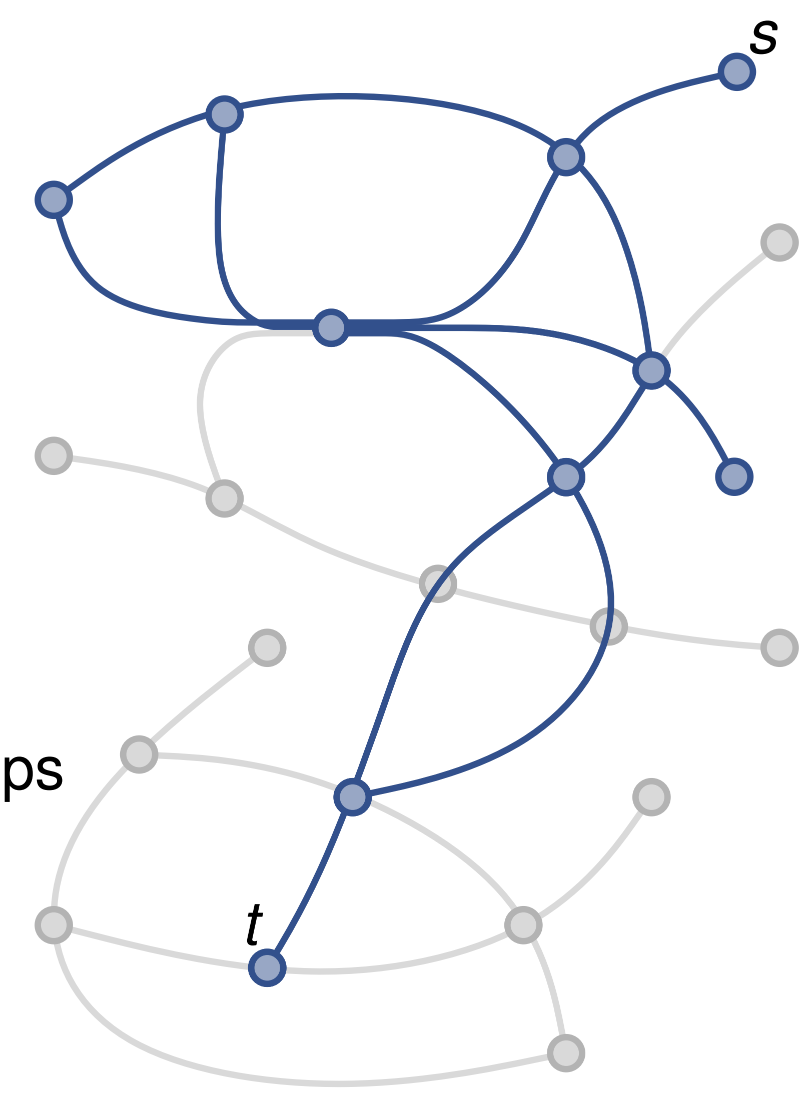
Query Algorithm

Idea:

- RAPTOR restricted to:
  - Source cell
  - Target cell
  - Fill-in

Special Case:

- Source or target is a cut stop
- Cut stops are not part of any cell
- Fill-in is sufficient to reach cut stops



Experiments – Instances

Networks:

- Netherlands and Switzerland
  [datahub.io/dataset/gtfs-nl]
  [gtfs.geops.ch]
- Footpaths up to 200 meters

Instance	Netherlands	Switzerland
Stops	54 500	25 607
Routes	12 989	16 122
Trips	618 961	1 076 662
Stop events	13 231 954	12 733 856
Footpaths	65 018	14 717

Structure:

Experiments – Preprocessing

Preprocessing (partition by hmetis):

	Netherlands				Switzerland			
# cells	# cut	% fn. rts	% fn. se	[m:s]	# cut	% fn. rts	% fn. se	[m:s]
2	365	31.5	5.3	67:32	155	19.1	1.5	13:02
4	589	40.7	7.3	82:53	345	32.0	3.5	20:58
8	1,072	54.7	13.0	113:45	545	42.6	6.1	27:19
16	1,980	68.2	22.1	203:34	907	52.5	14.4	36:51

Partition with 8 cells:

Experiments – Queries

Query Performance:

- Average over 10,000 random queries
- Number of rounds (#rnd)
- Number of scanned routes (#rts)
- Percentage of scanned fill-in routes (#fn.rts)

		Netherlands				Switzerland			
Algorithm	# cells	# rnd	# rts	% fn. rts	[ms]	# rnd	# rts	# fn. rts	[ms]
RAPTOR	—	10.0	28,021	—	29.3	9.1	29,090	—	19.3
HypRAPTOR	2	9.8	24,666	7.8	25.0	9.1	25,306	4.4	16.8
HypRAPTOR	4	9.6	21,313	30.4	19.3	8.9	19,654	24.1	11.8
HypRAPTOR	8	9.7	20,278	57.3	17.5	8.8	17,405	49.1	9.3
HypRAPTOR	16	9.9	21,085	77.3	18.2	8.7	17,799	73.0	10.1

[C++ using LLVM 8.1, on a 2015 15-inch MacBook Pro, Core i7, 16 GiB of 1600 MHz DDR-3 RAM]

Conclusion

Our Algorithm:

- First partition based speedup technique for RAPTOR
- Based on route-partition instead of stop-partition
- Based on route-partition instead of stop-partition

Future work:

- Find better partitions
- Use multi-level partitions
- Optimize more criteria
- Evaluate for unrestricted walking

Conclusion

Our Algorithm:

- First partition based speedup technique for RAPTOR
- Based on route-partition instead of stop-partition
- Based on route-partition instead of stop-partition

Future work:

- Find better partitions
- Use multi-level partitions
- Optimize more criteria
- Evaluate for unrestricted walking

Thank you for your attention!