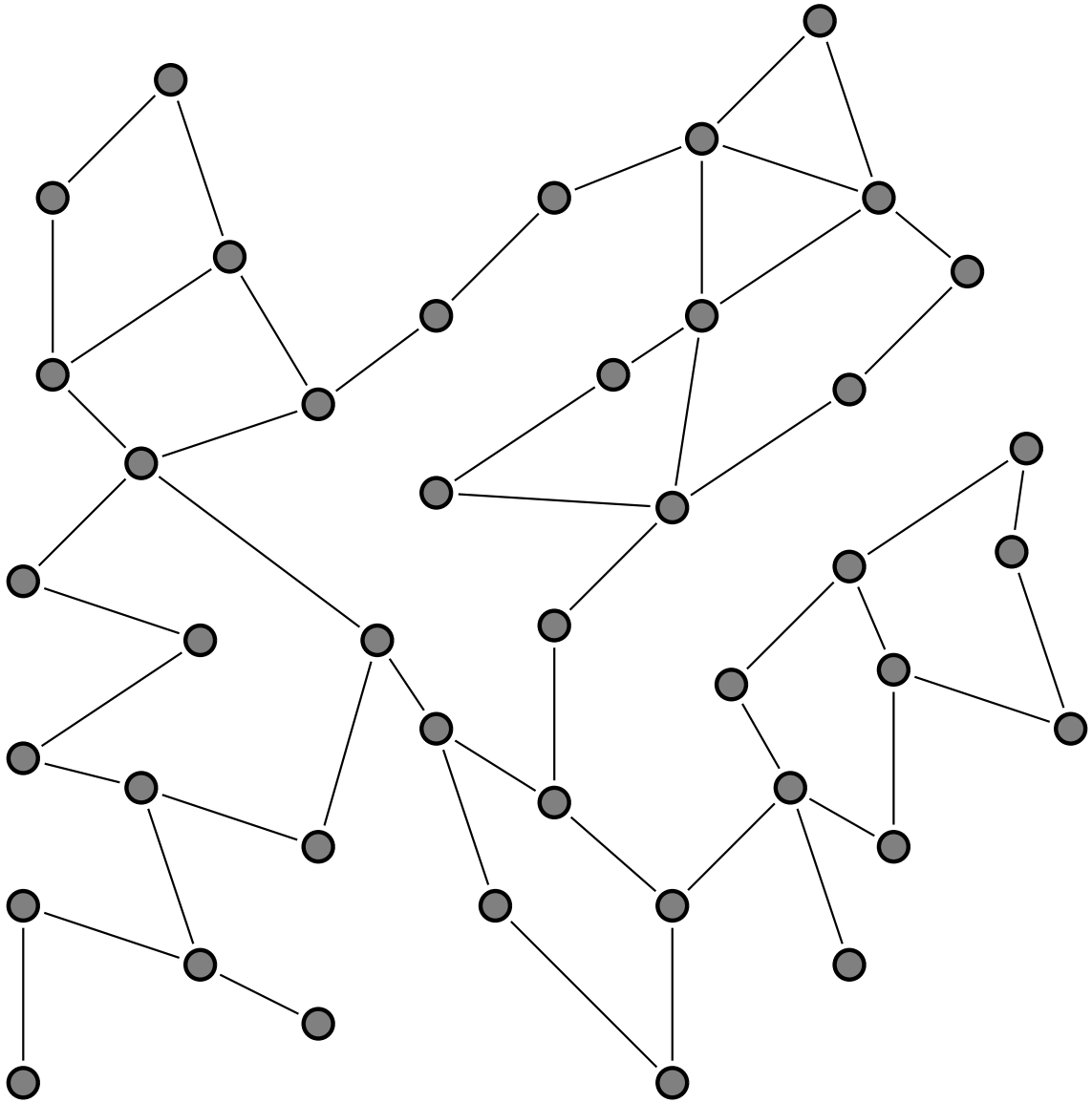
Fast and Stable Repartitioning of Road Networks

18th International Symposium on Experimental Algorithms

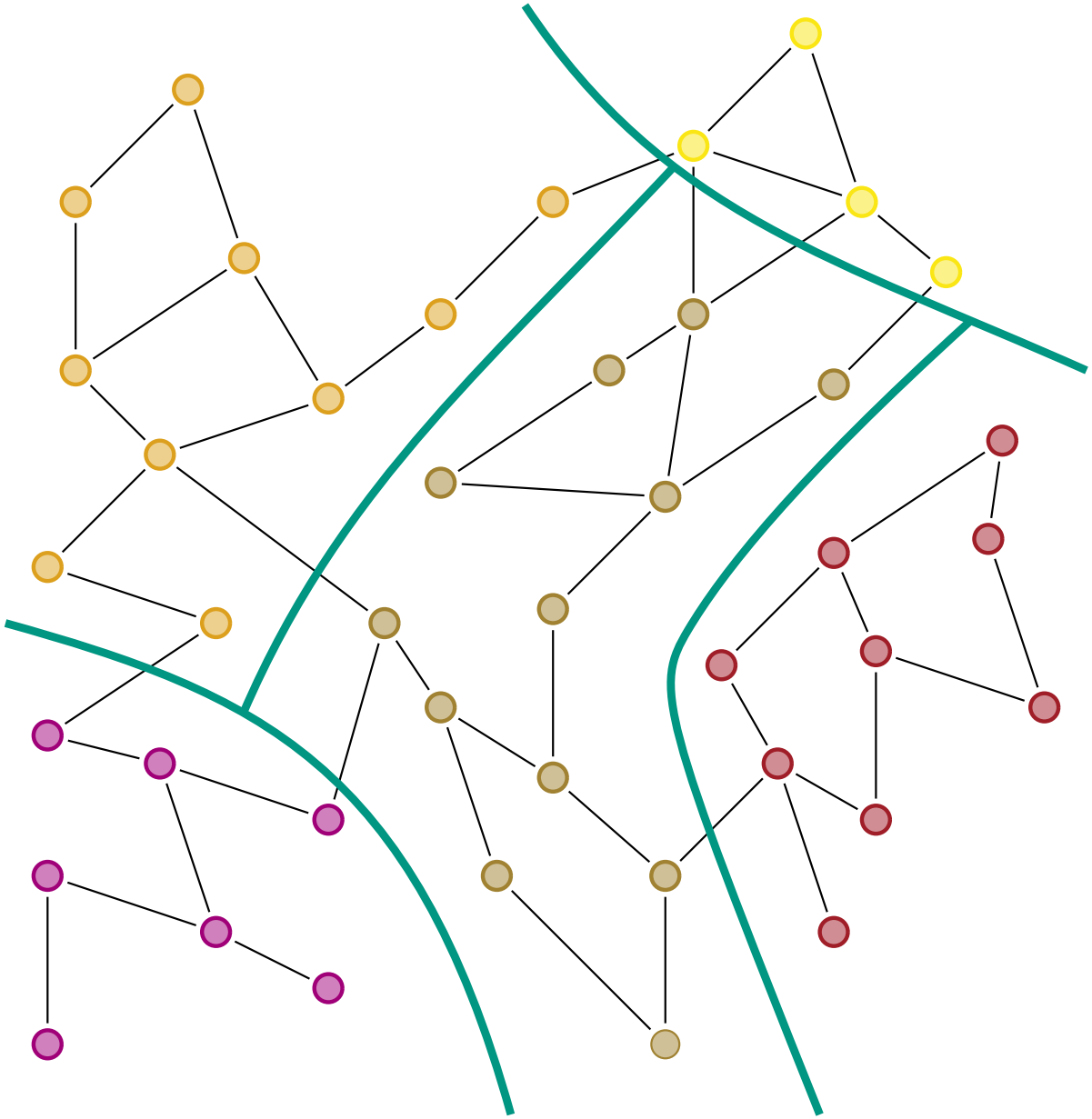
Valentin Buchhold Daniel Delling
Dennis Schieferdecker Michael Wegner

June 18, 2020

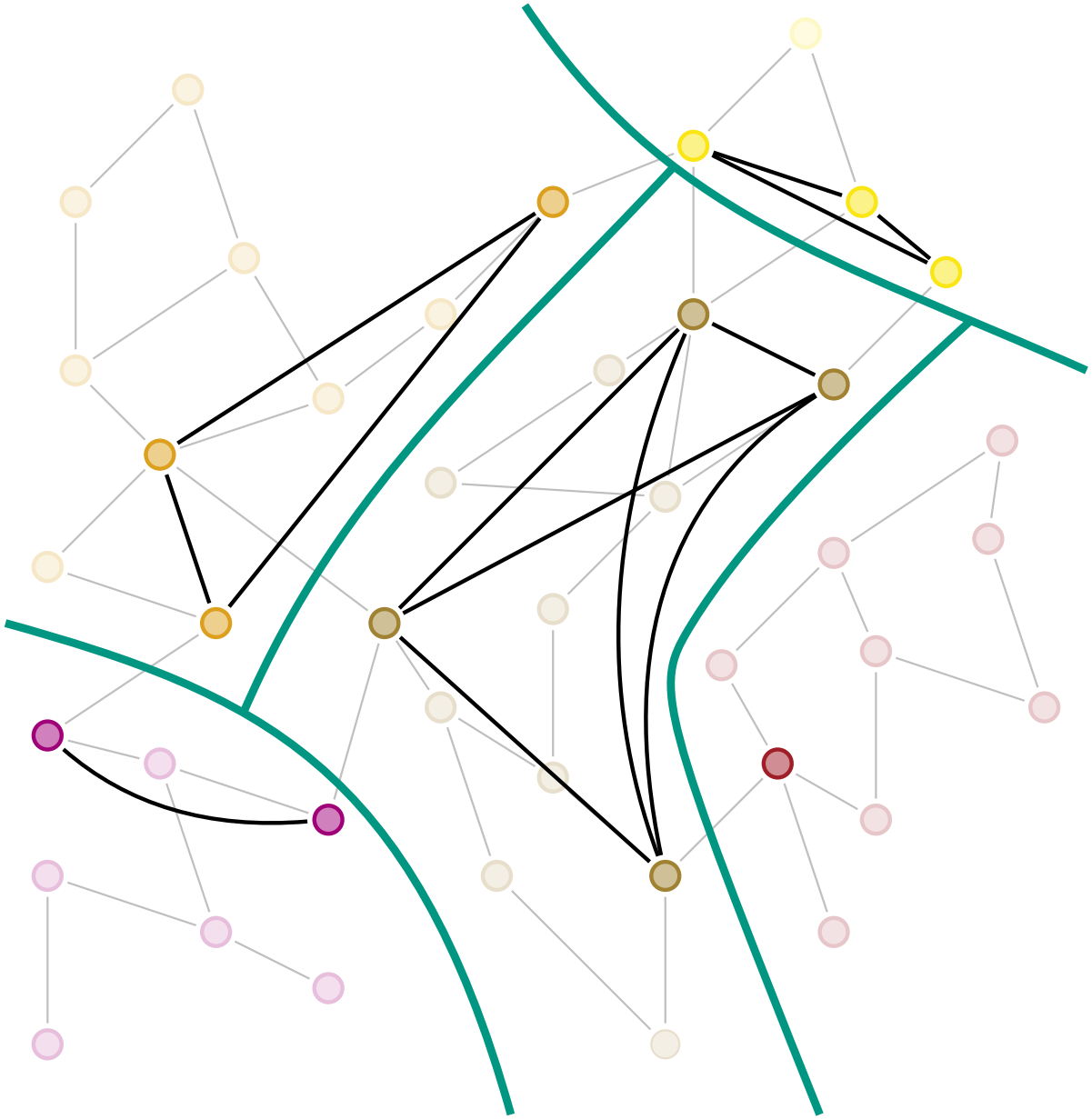
Partition-Based Route Planning



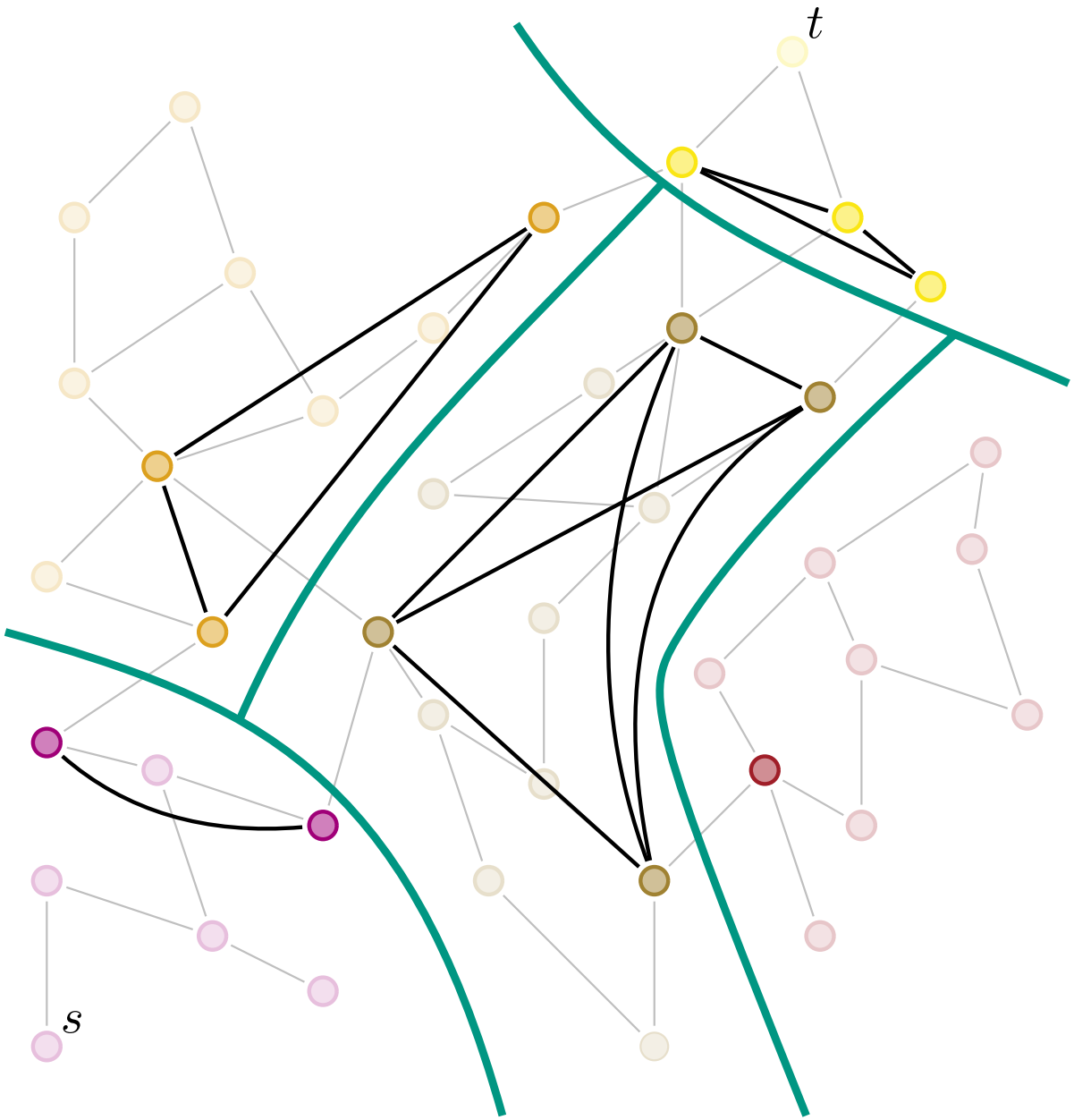
Partition-Based Route Planning



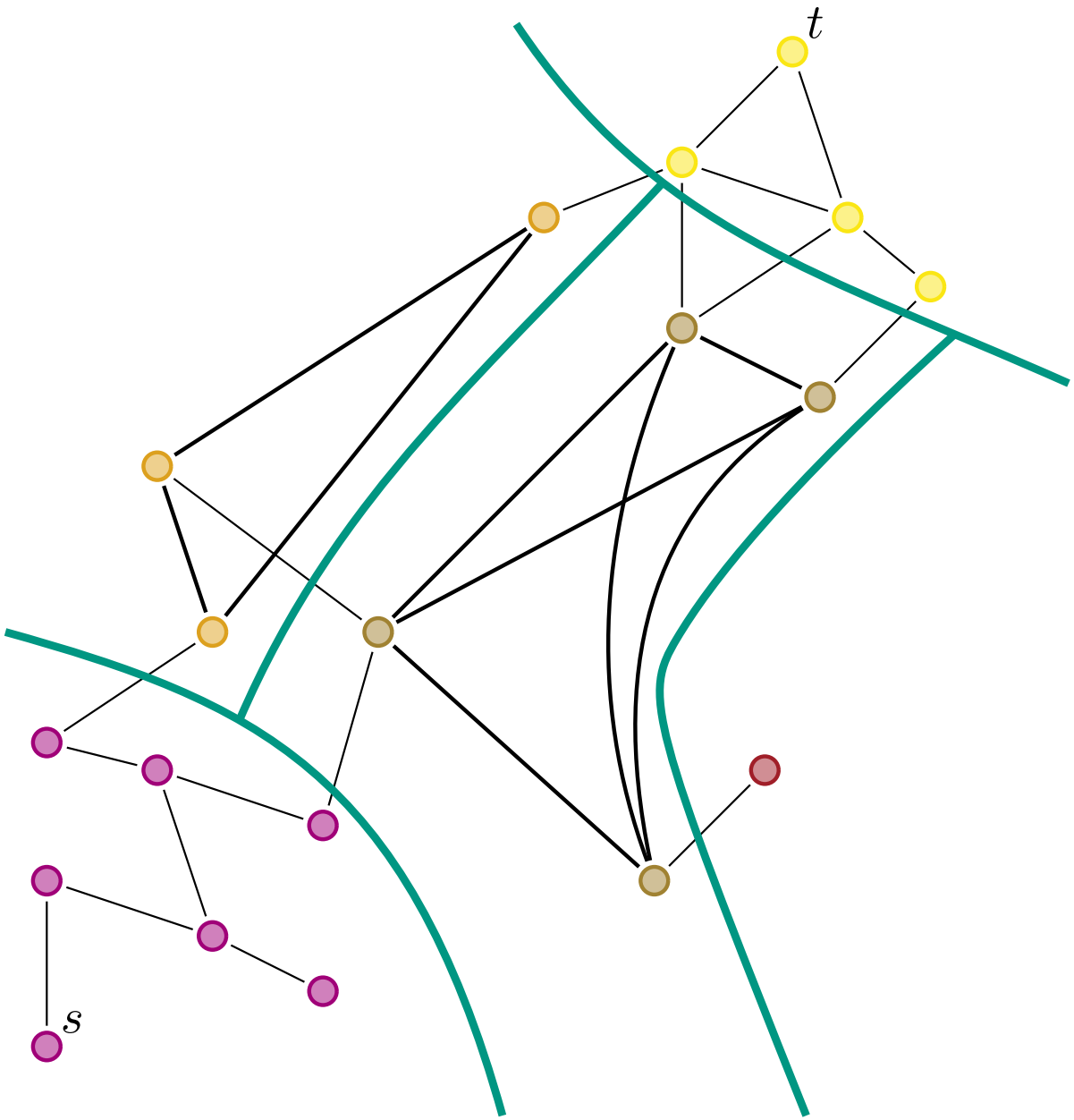
Partition-Based Route Planning



Partition-Based Route Planning



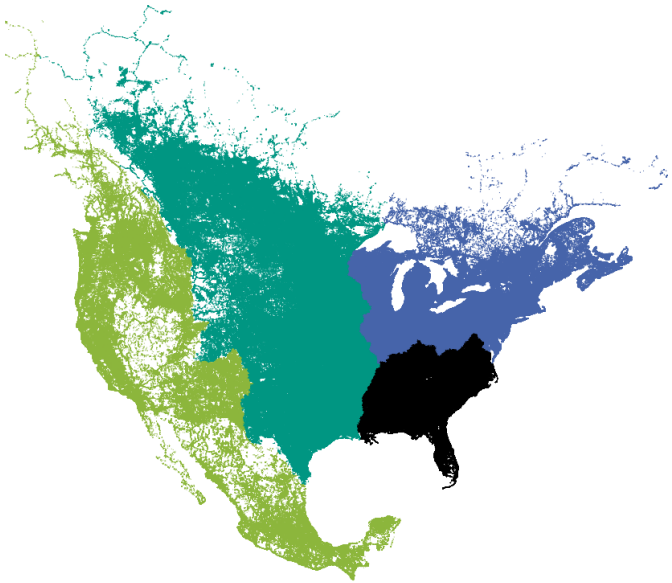
Partition-Based Route Planning



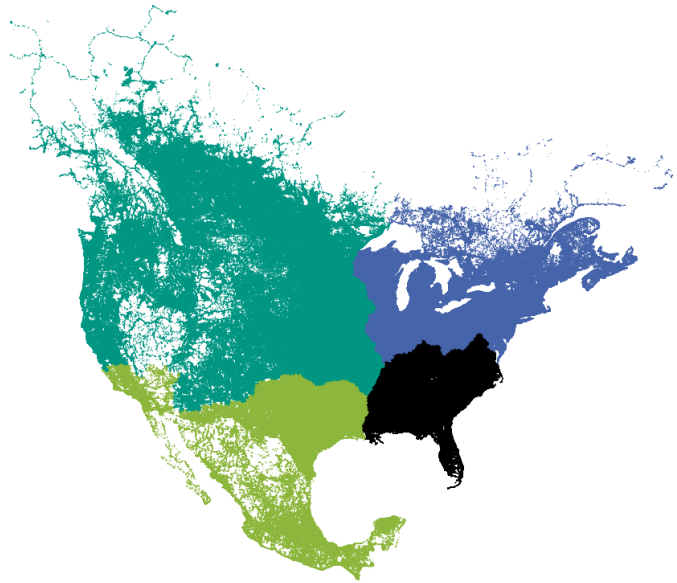
Real-World Production Systems

observations:

- data changes continuously
- OpenStreetMap: several million changes each day
- partitioning from scratch is highly sensitive to small changes



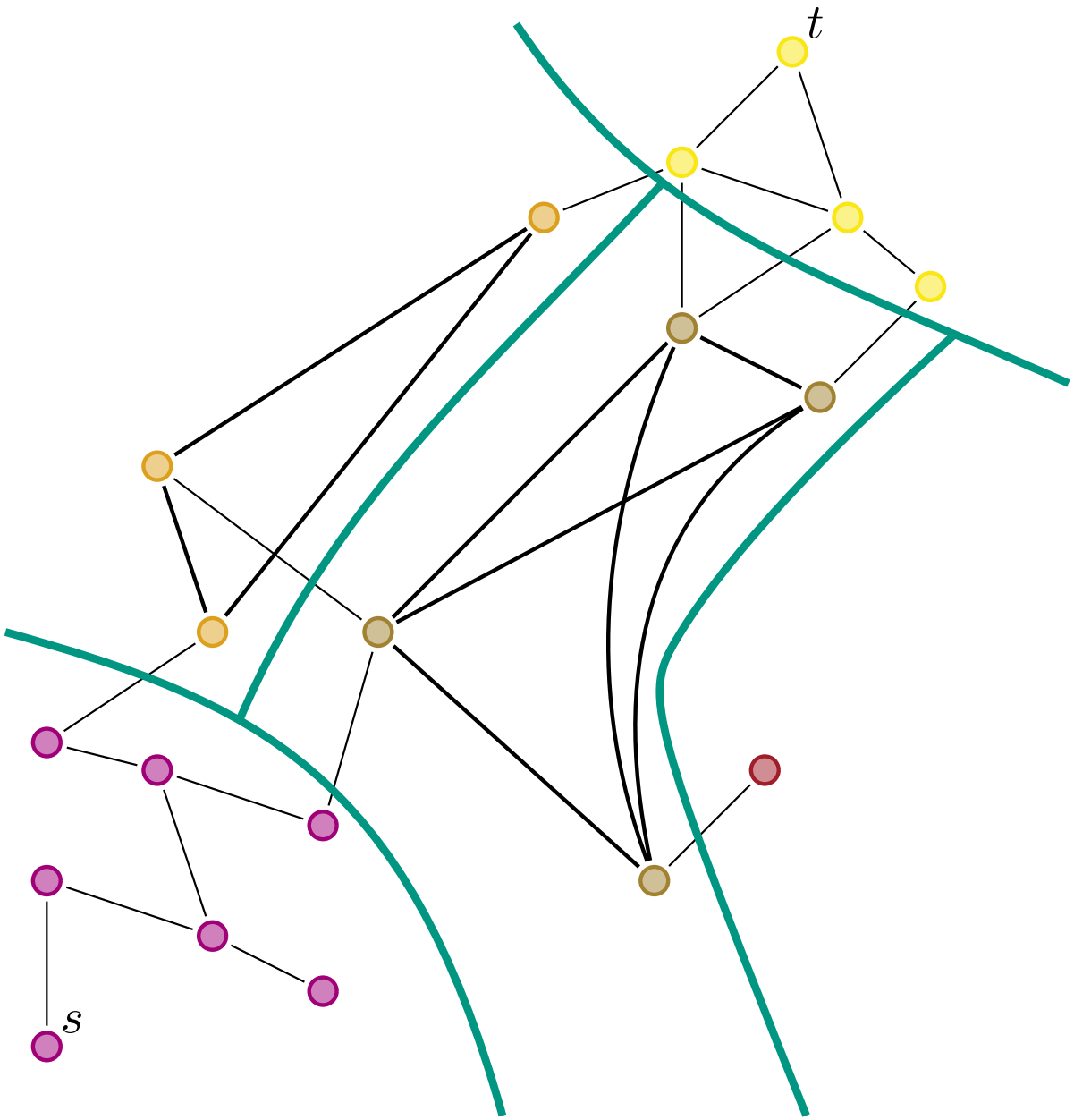
August 2018



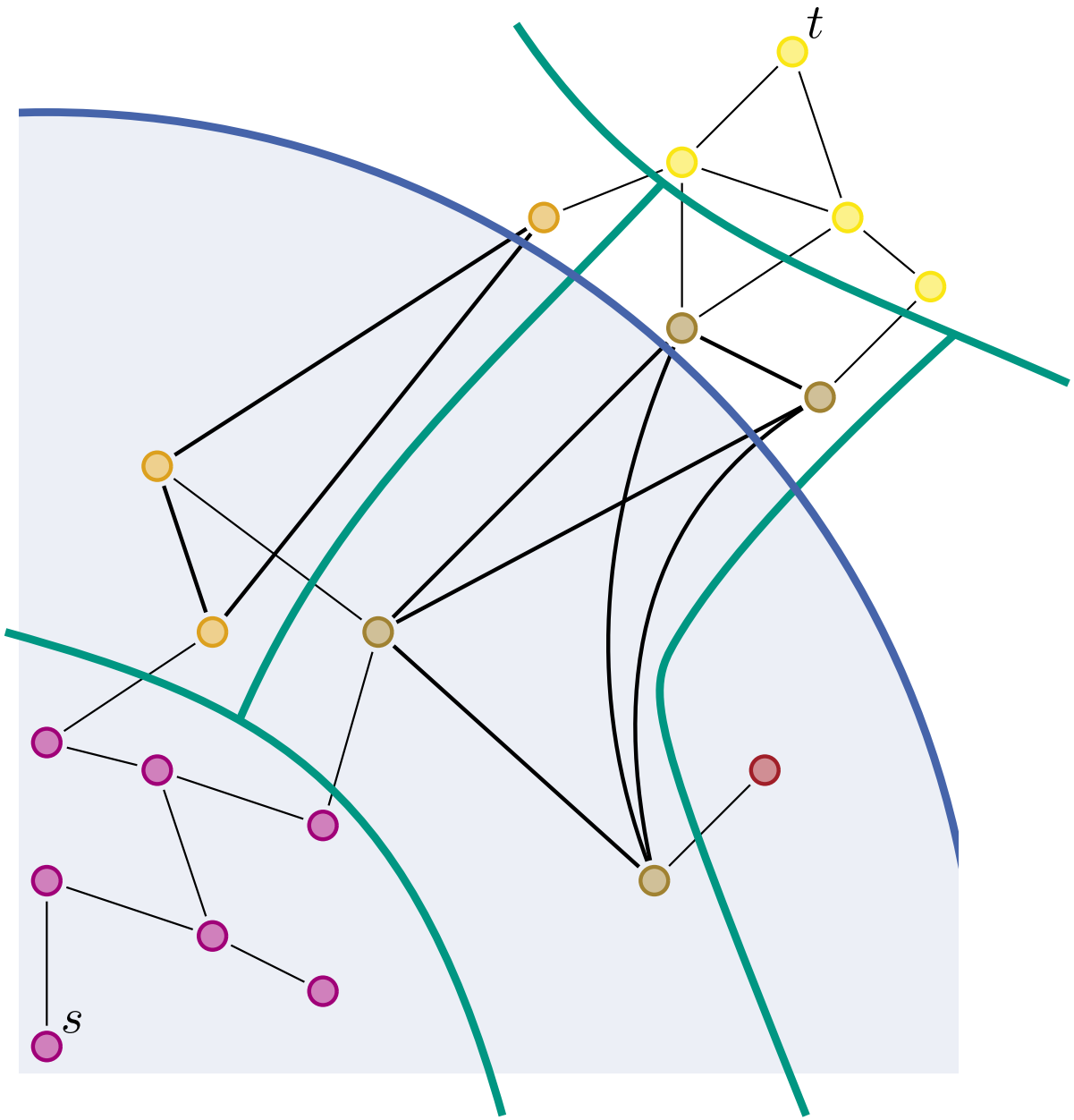
September 2018

change in vertices: 0.36 % are added, 0.21 % are removed

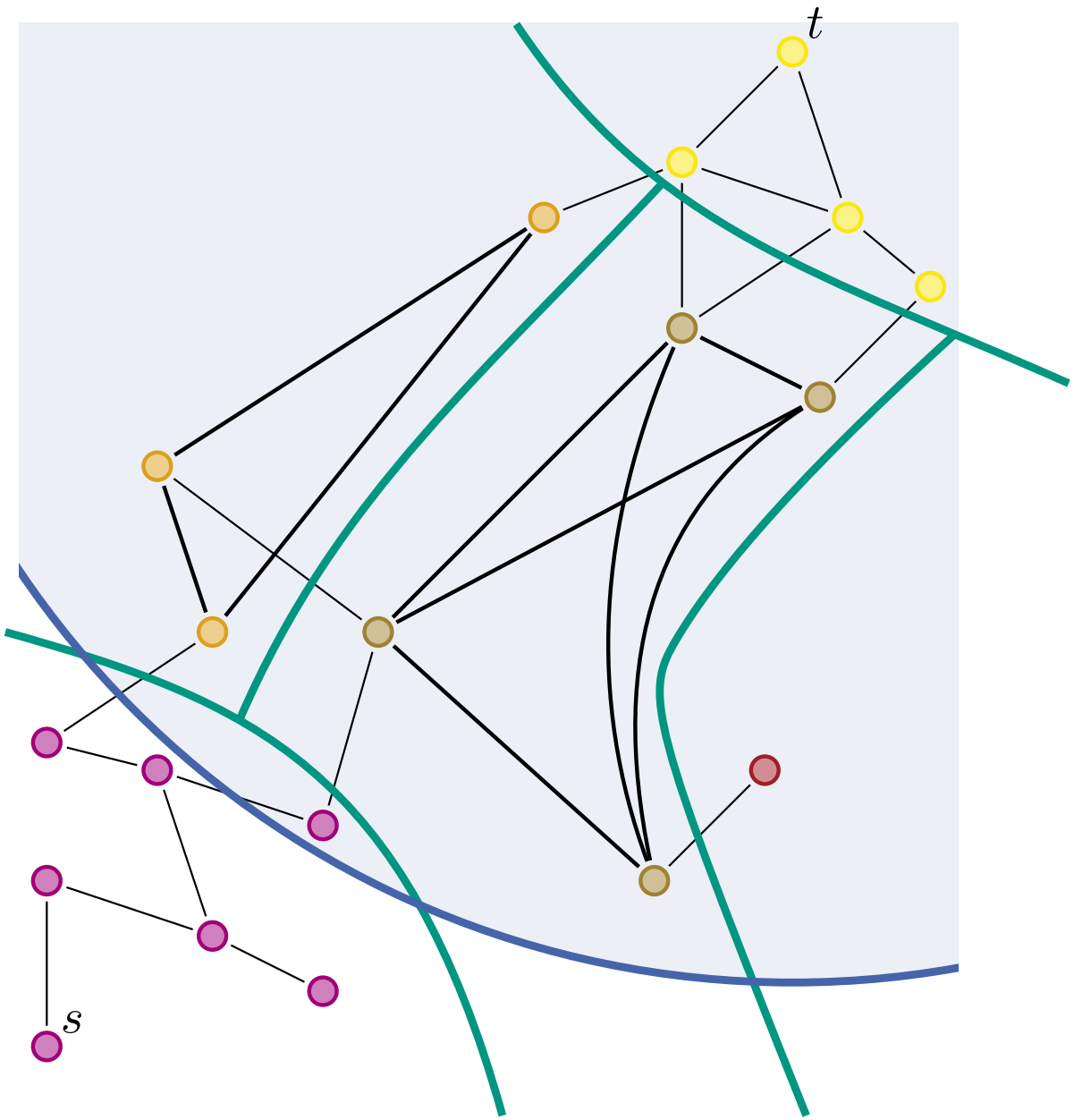
Computing Alternative Routes



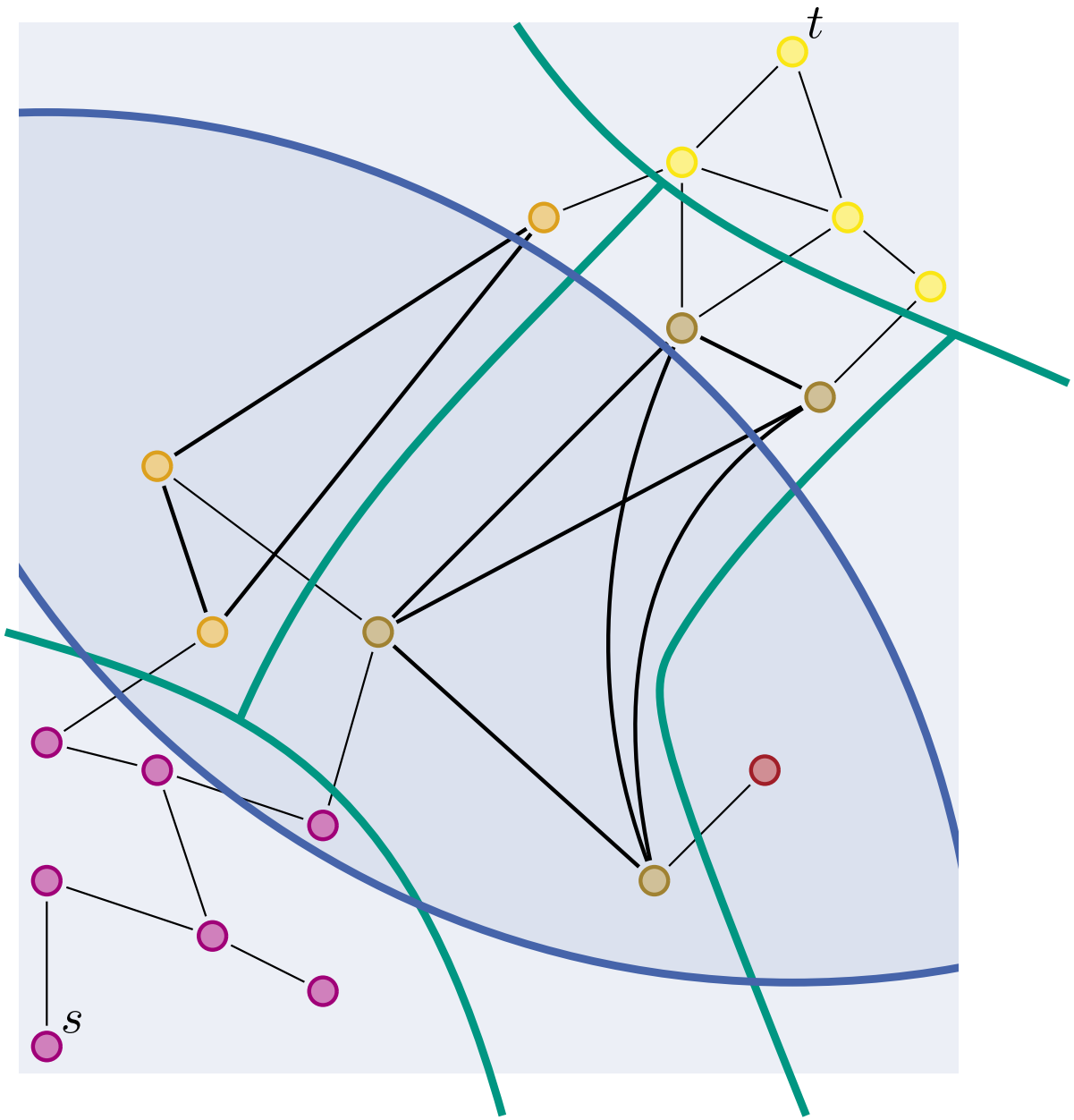
Computing Alternative Routes



Computing Alternative Routes



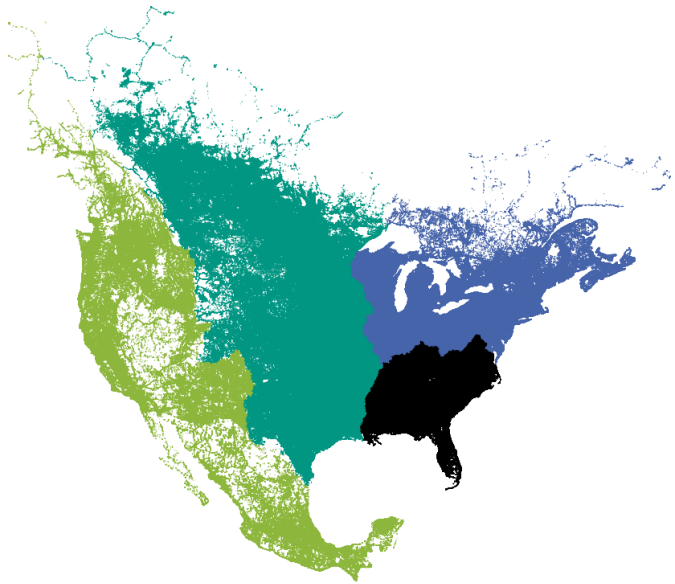
Computing Alternative Routes



Stable Graph Repartitioning

input:

- old graph
- new graph obtained from old one
  by inserting and removing vertices and edges
- partition of old graph
  into disjoint cells of roughly equal size



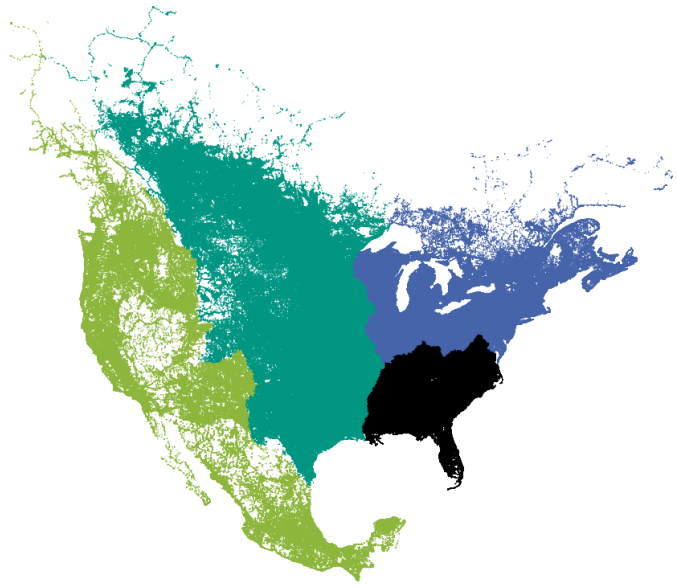
problem:

- cut new graph into disjoint cells
- of roughly equal size bounded by U
- while minimizing number of cut edges
- and maximizing similarity to old partition

Stable Graph Repartitioning

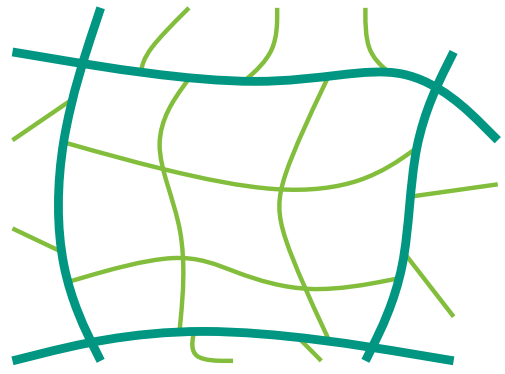
input:

- old graph
- new graph obtained from old one
  by inserting and removing vertices and edges
- partition of old graph
  into disjoint cells of roughly equal size



problem:

- cut new graph into disjoint cells
- of roughly equal size bounded by U
- while minimizing number of cut edges
- and maximizing similarity to old partition



$\Rightarrow$ **we consider nested multilevel partitions**

Related Work

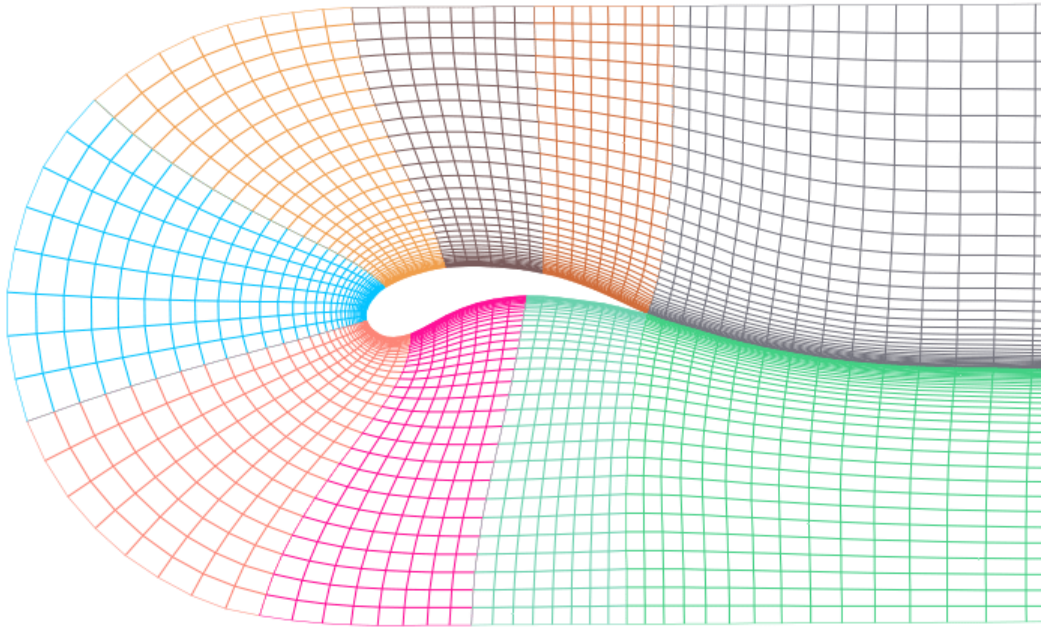
road network partitioning:

- PUNCH (2011)
- Buffoon (2012)
- Inertial Flow (2015)
- FlowCutter (2018)
- InertialFlowCutter (2019)

Related Work

road network partitioning:

- PUNCH (2011)
- Buffoon (2012)
- Inertial Flow (2015)
- FlowCutter (2018)
- InertialFlowCutter (2019)



adaptive mesh repartitioning:

- scratch-remap repartitioners (1994, 1995, 1998, 2001)
- diffusion-based repartitioners (1997, 1997, 2001)
- unified repartitioning algorithm (2000)

$\Rightarrow$ new mesh obtained by splitting or merging vertices

$\Rightarrow$ most work considers a fixed topology

mesh refinements handled as vertex weight increases

Our Approach to Graph Repartitioning

idea:

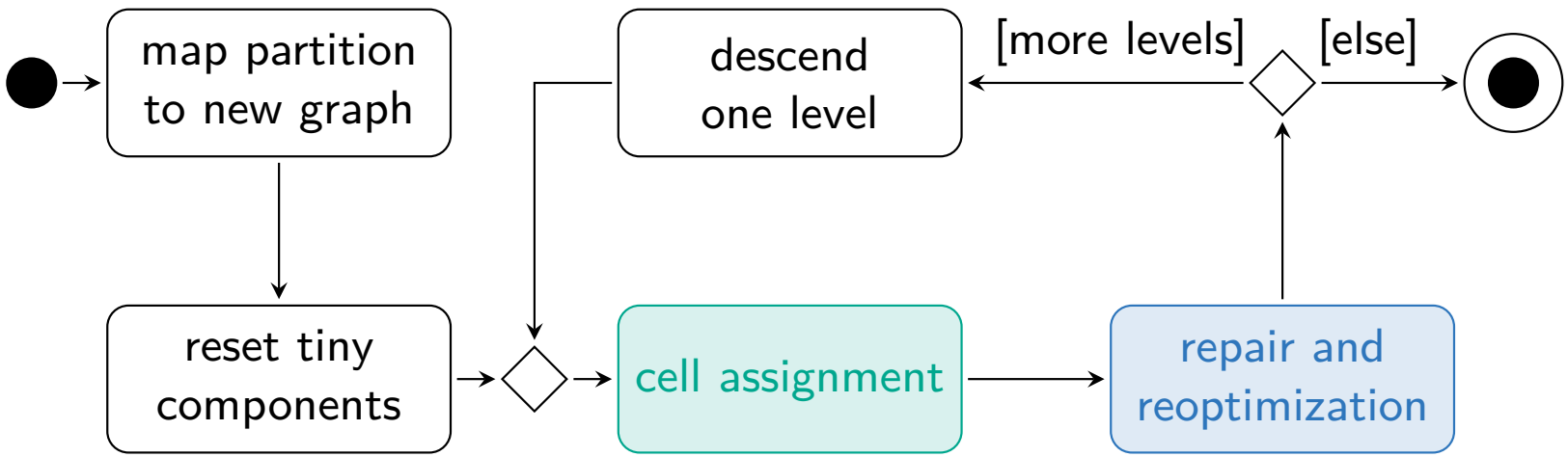
- start with previous partition
- incorporate new vertices (cell assignment)
- repair and reoptimize partition

Our Approach to Graph Repartitioning

idea:

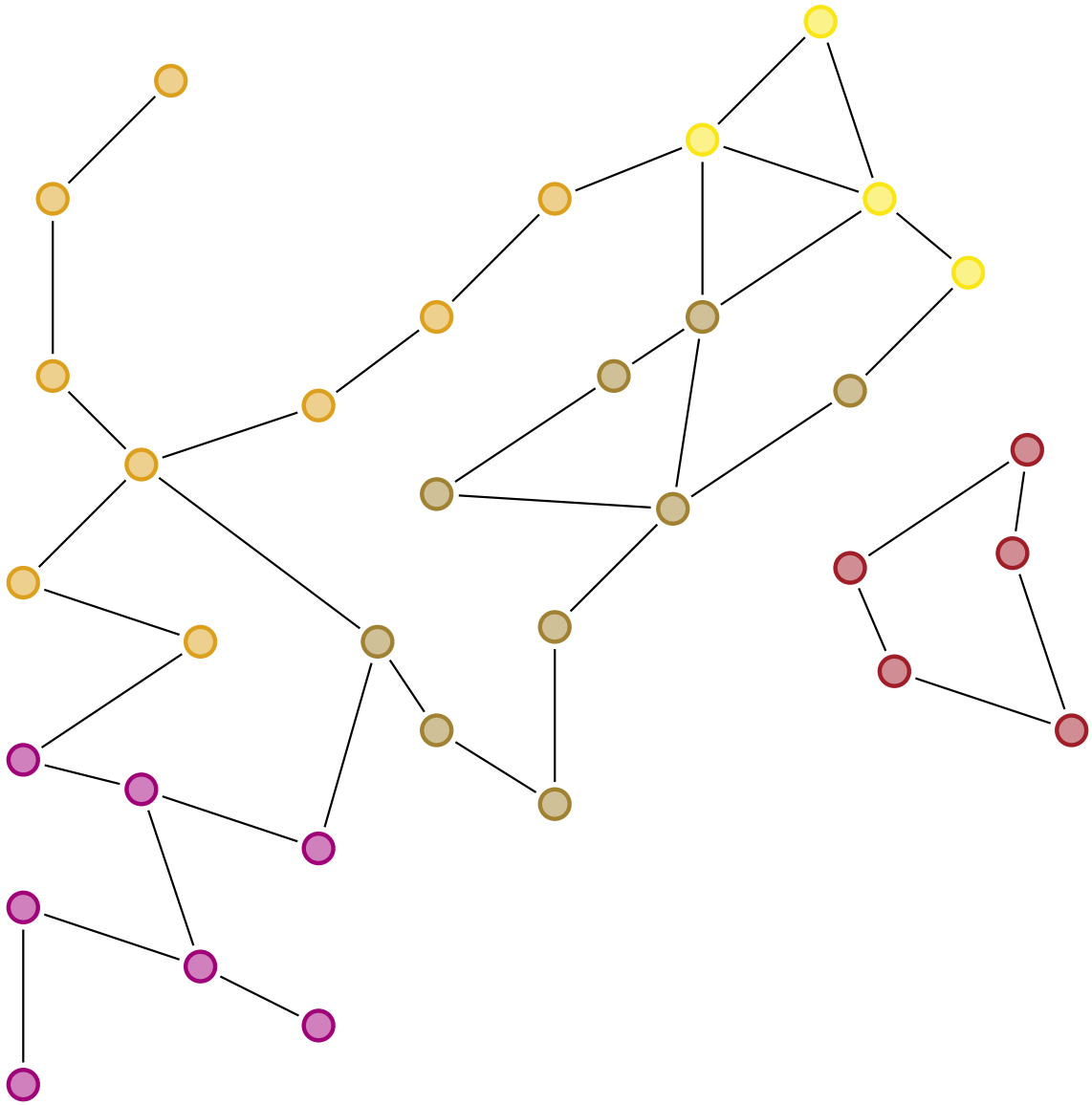
- start with previous partition
- incorporate new vertices (cell assignment)
- repair and reoptimize partition

process levels in top-down fashion:



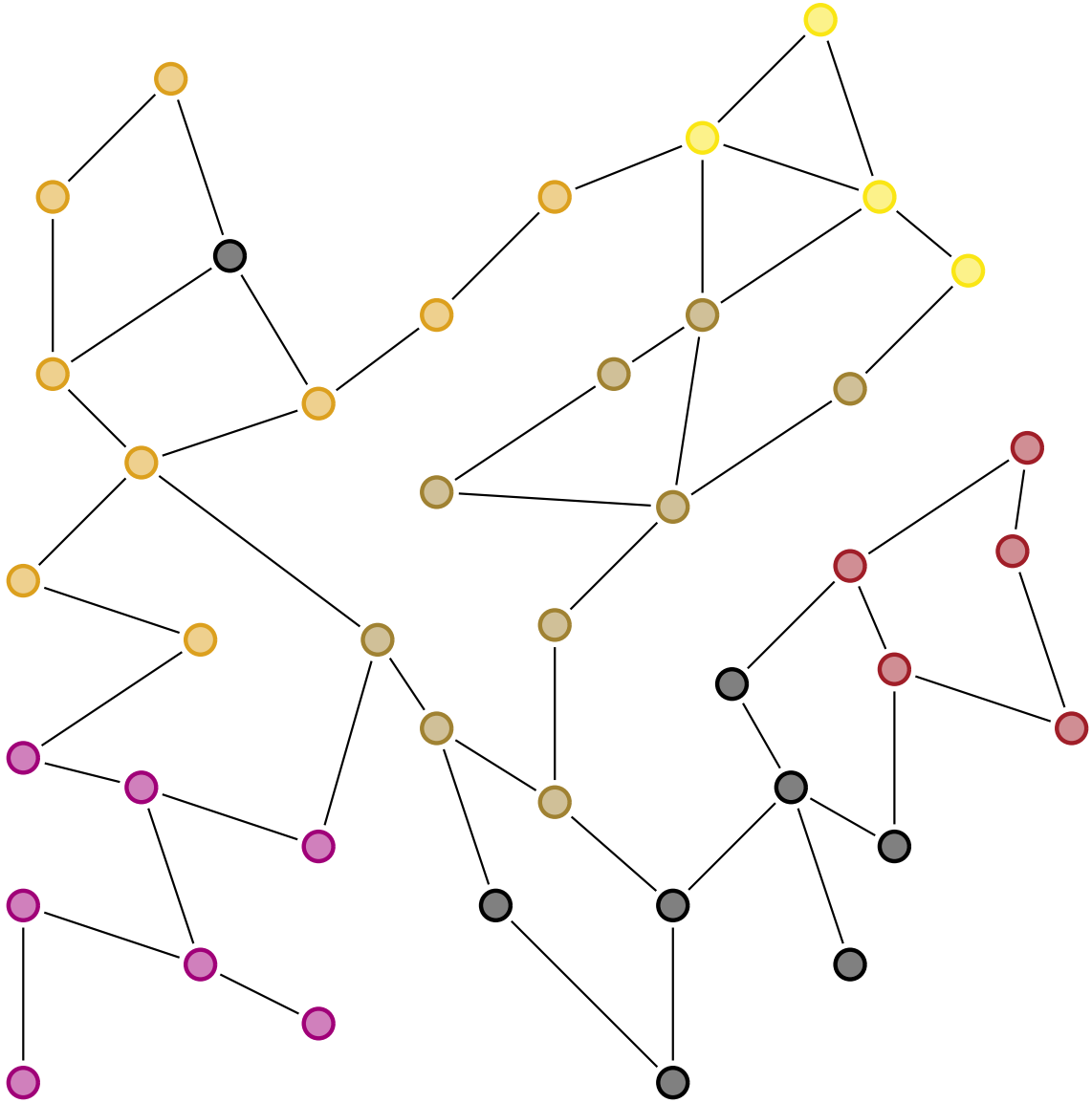
Cell Assignment

each new vertex chooses cell to which majority of its neighbors belong:



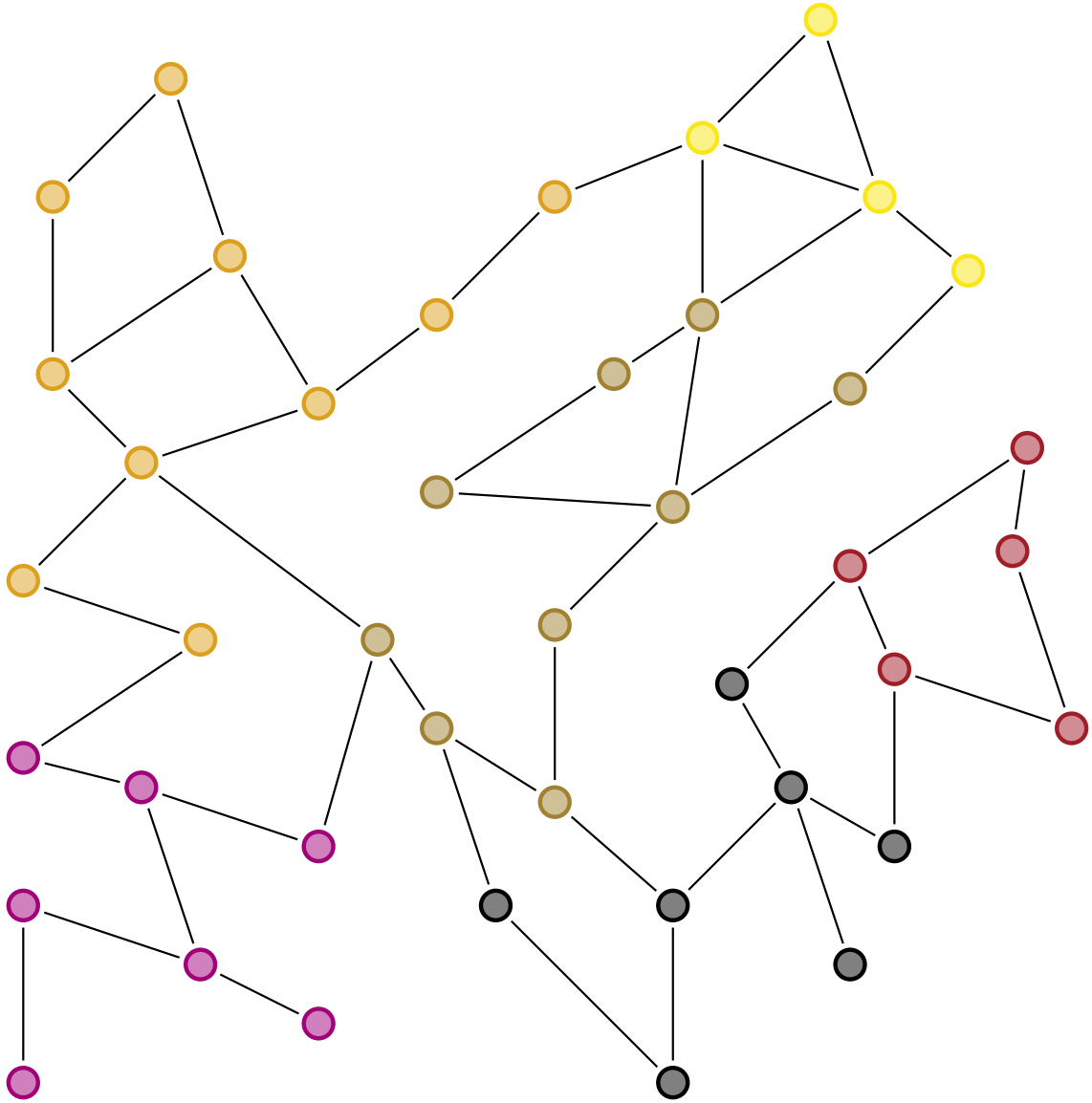
Cell Assignment

each new vertex chooses cell to which majority of its neighbors belong:



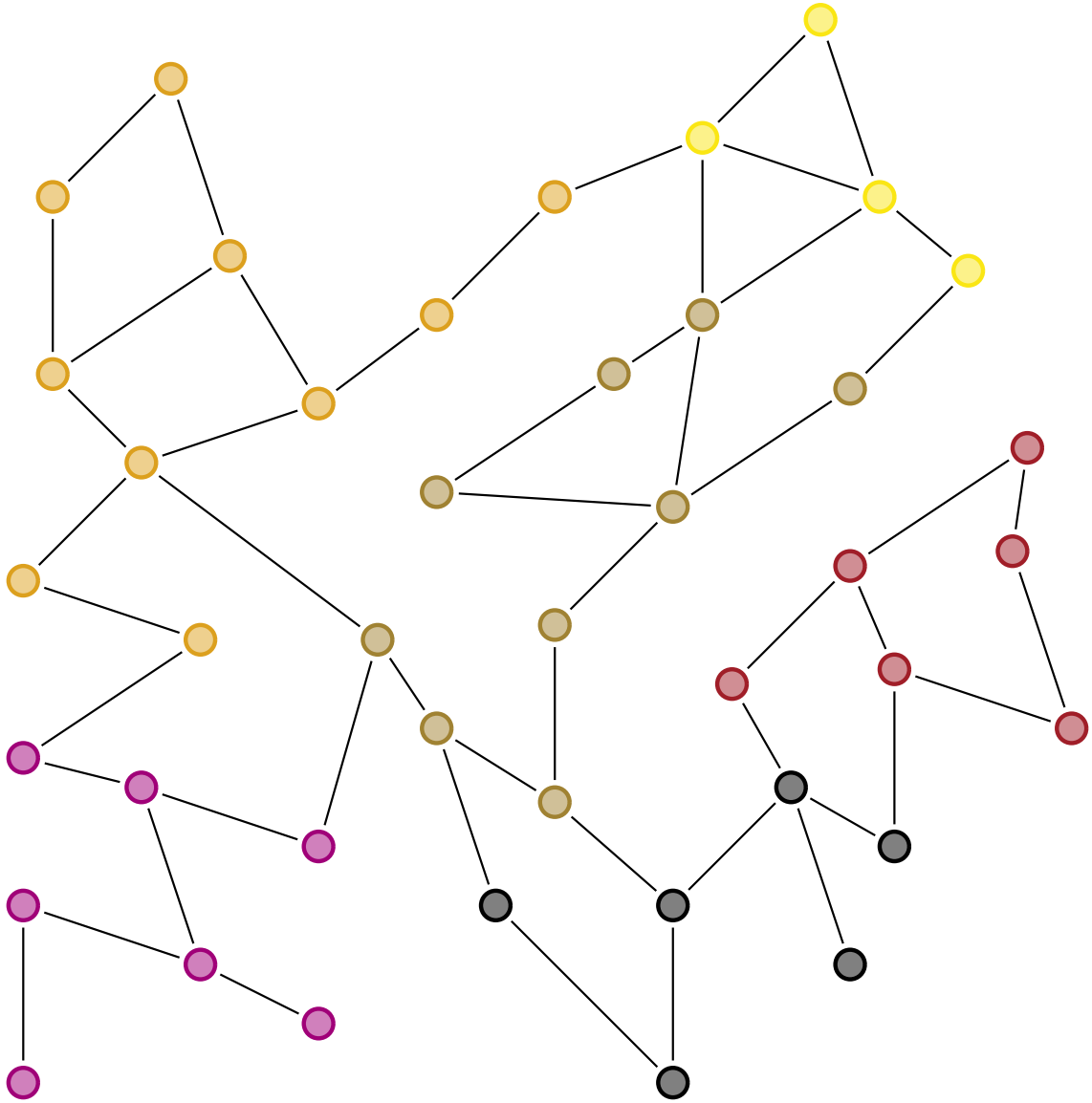
Cell Assignment

each new vertex chooses cell to which majority of its neighbors belong:



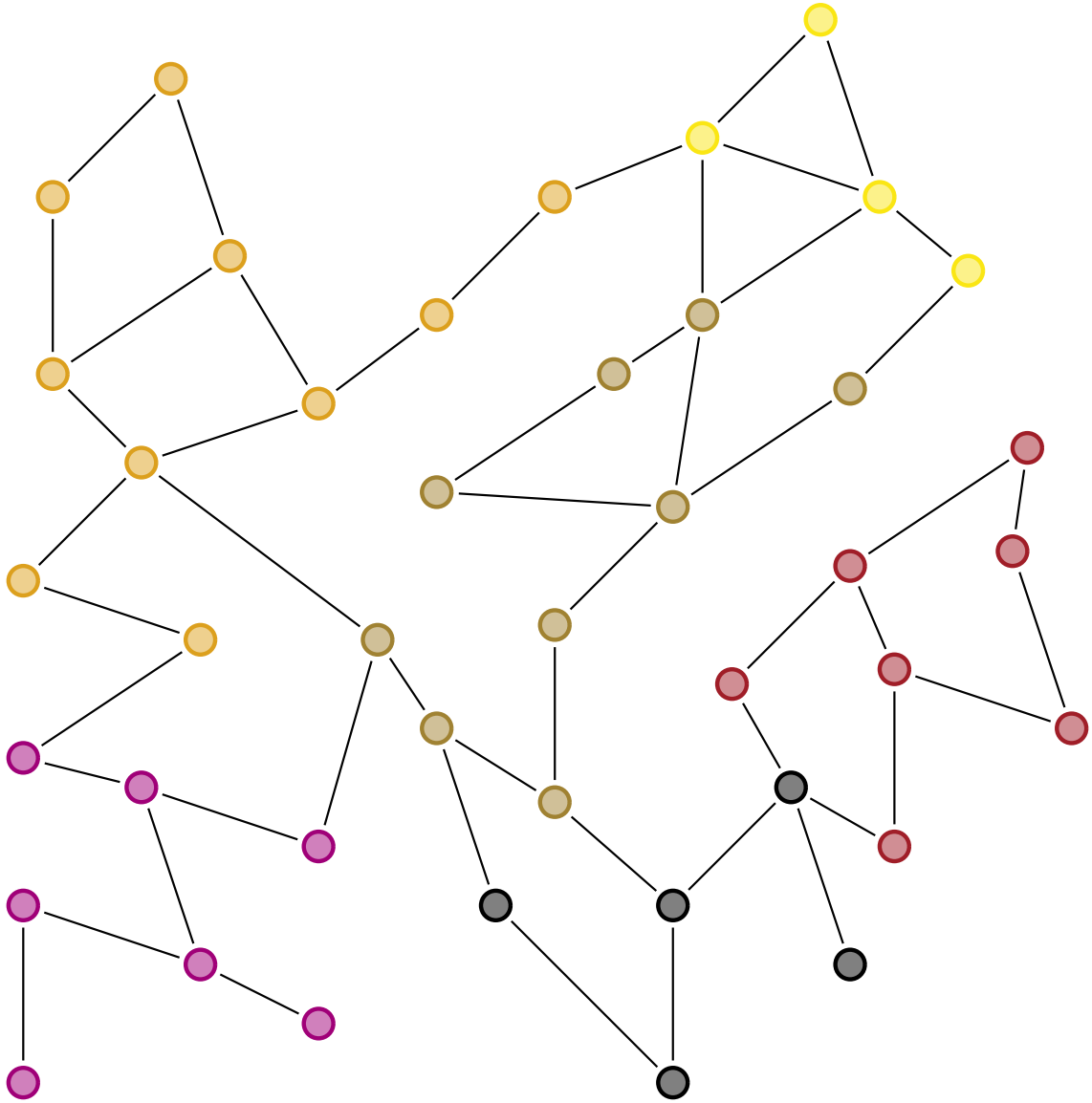
Cell Assignment

each new vertex chooses cell to which majority of its neighbors belong:



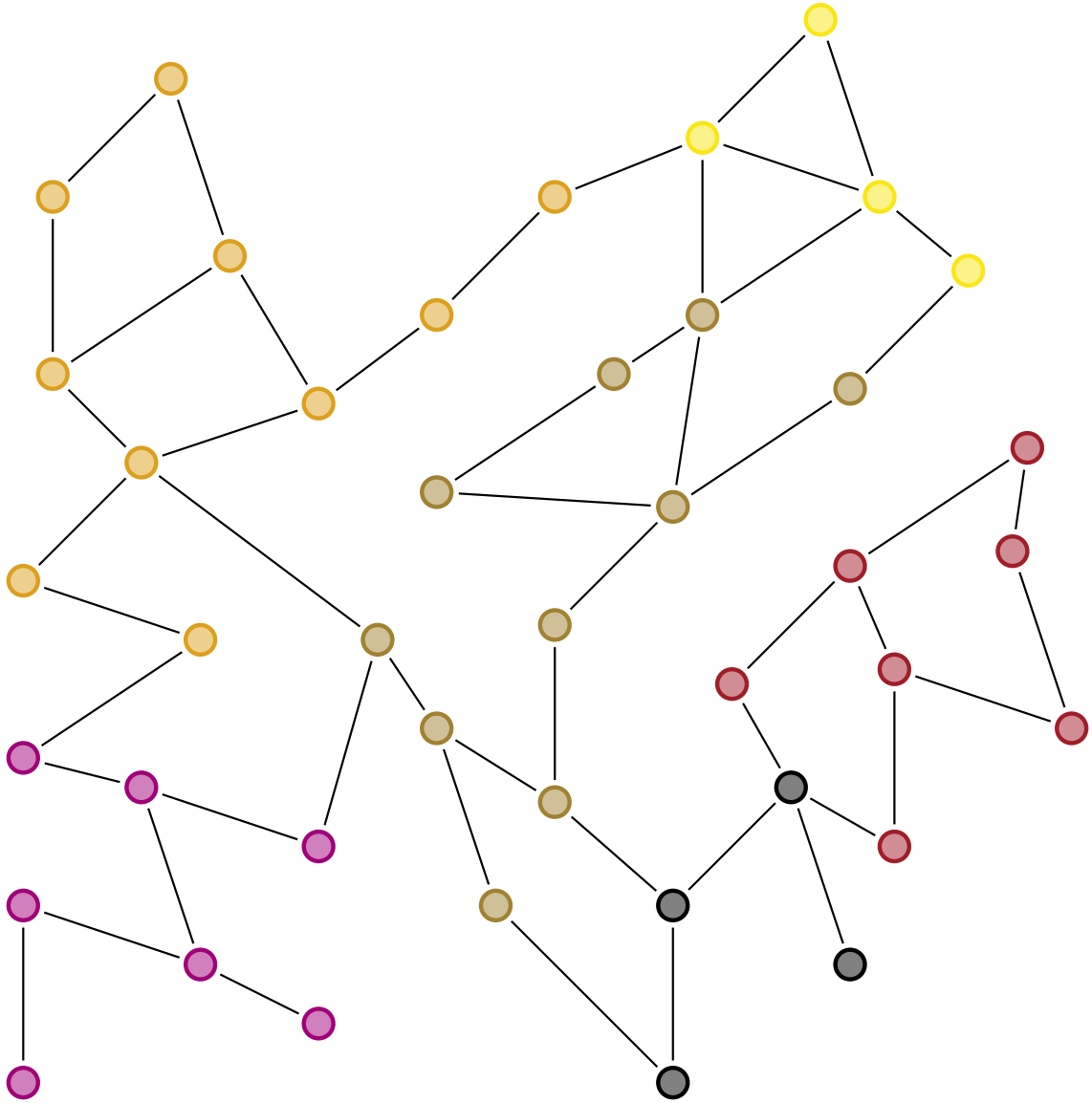
Cell Assignment

each new vertex chooses cell to which majority of its neighbors belong:



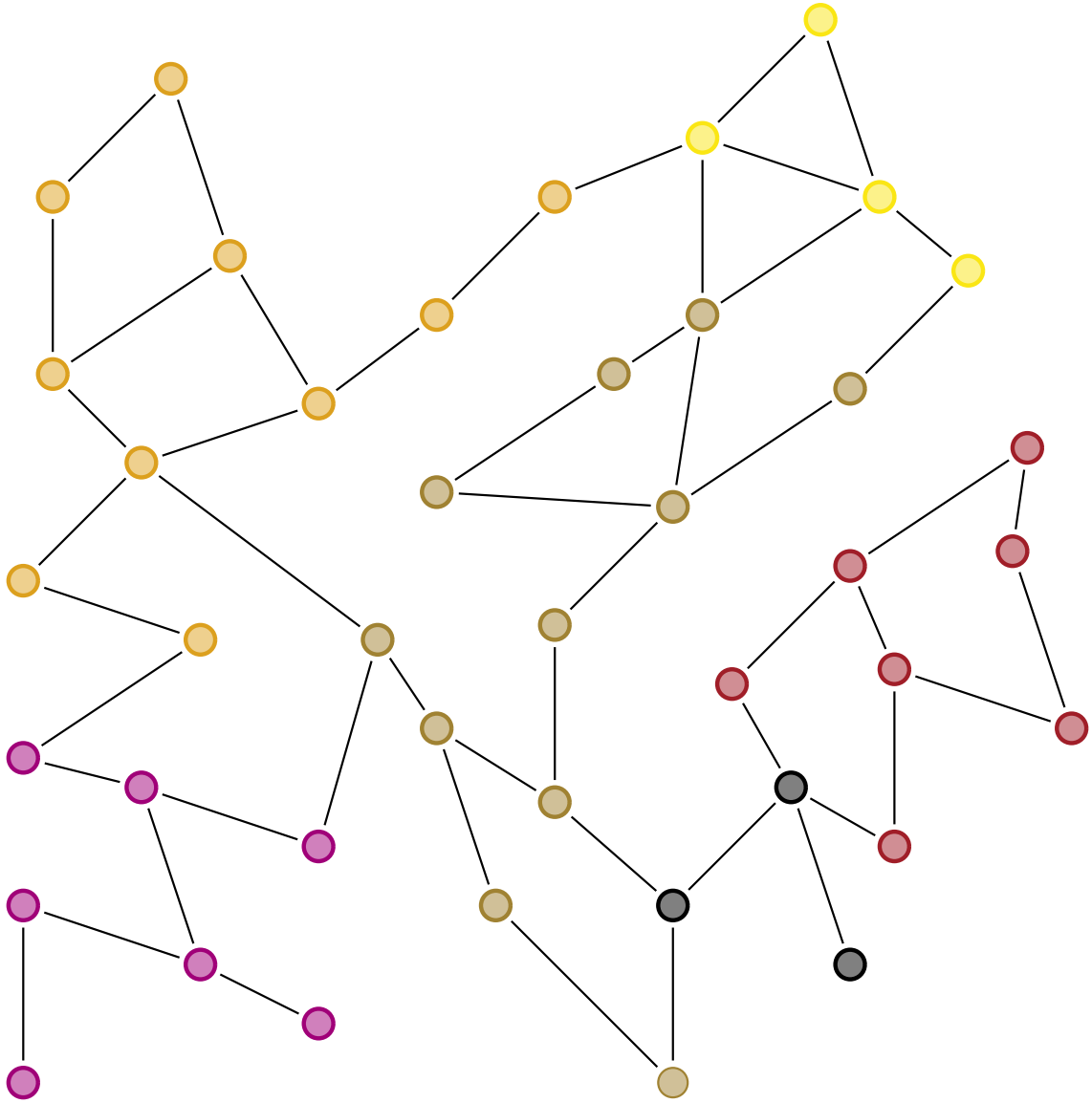
Cell Assignment

each new vertex chooses cell to which majority of its neighbors belong:



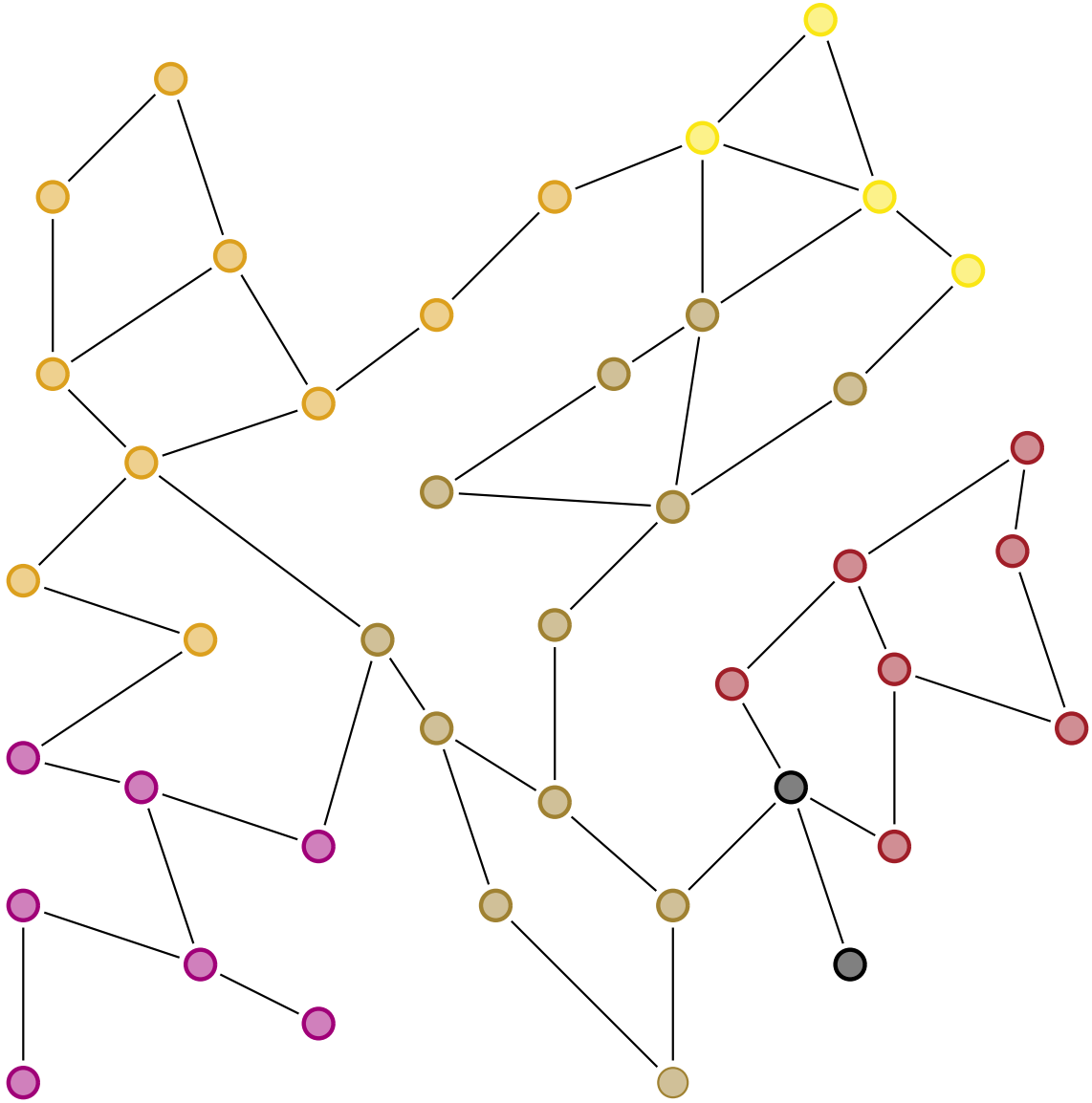
Cell Assignment

each new vertex chooses cell to which majority of its neighbors belong:



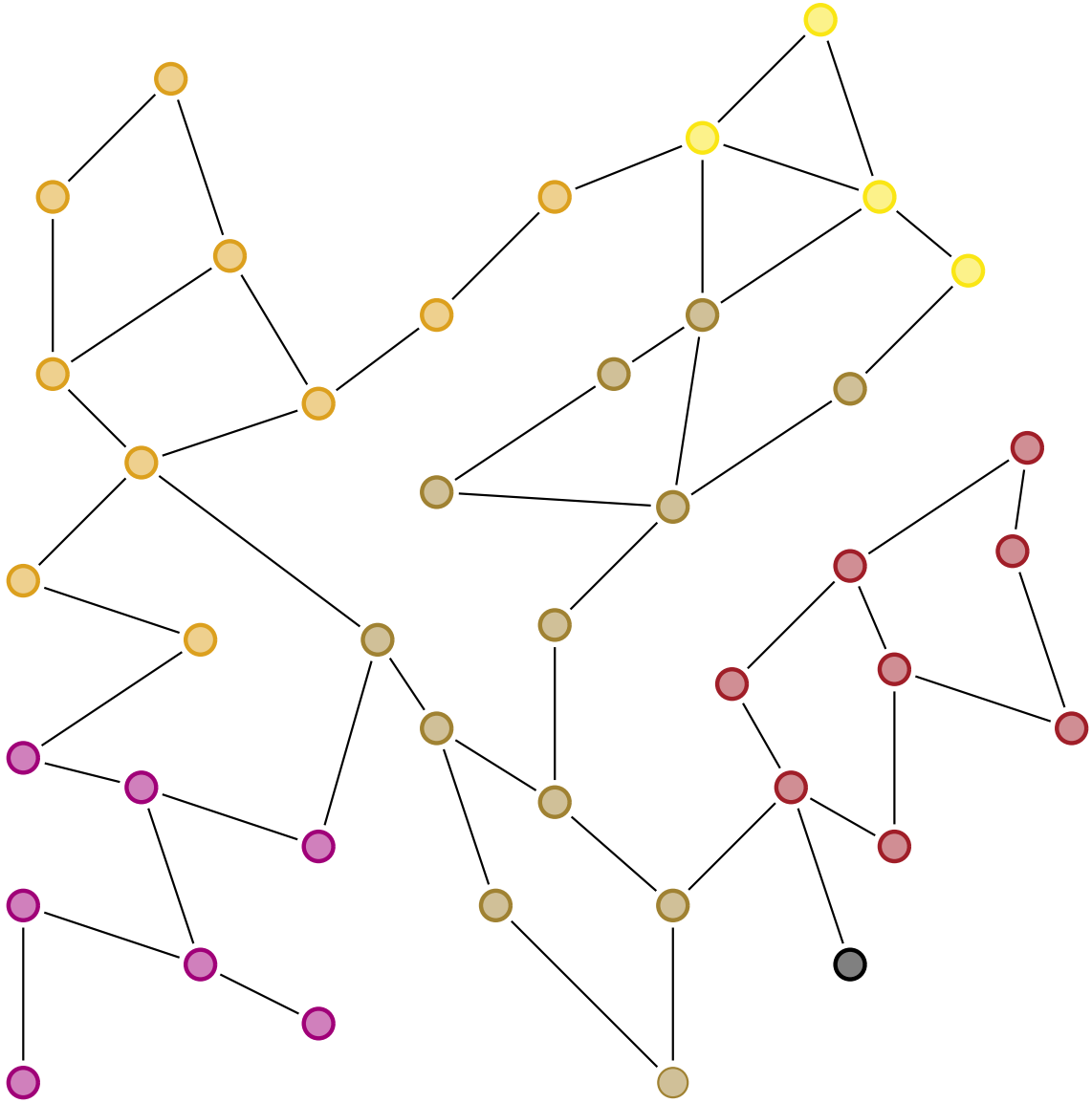
Cell Assignment

each new vertex chooses cell to which majority of its neighbors belong:



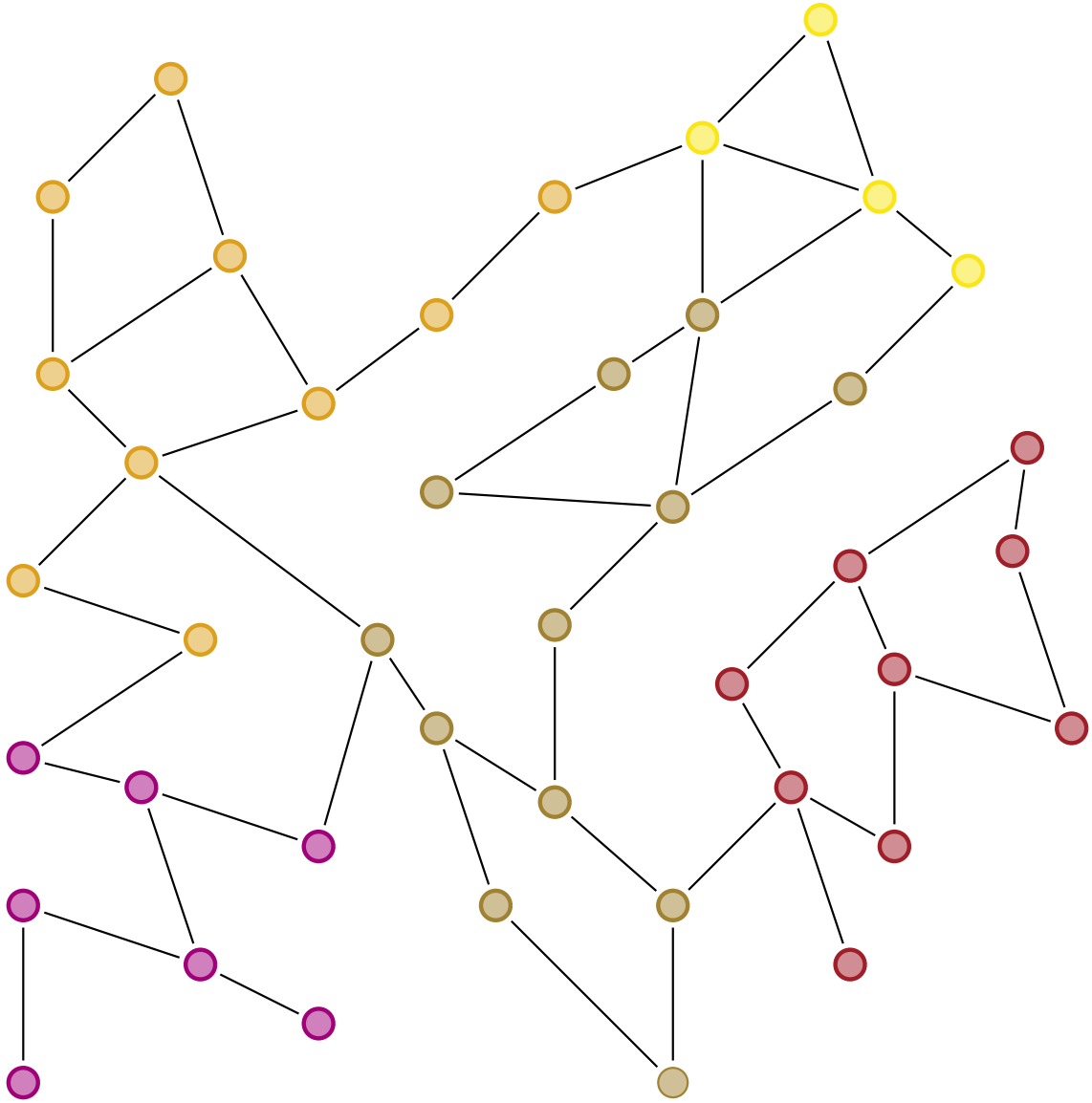
Cell Assignment

each new vertex chooses cell to which majority of its neighbors belong:



Cell Assignment

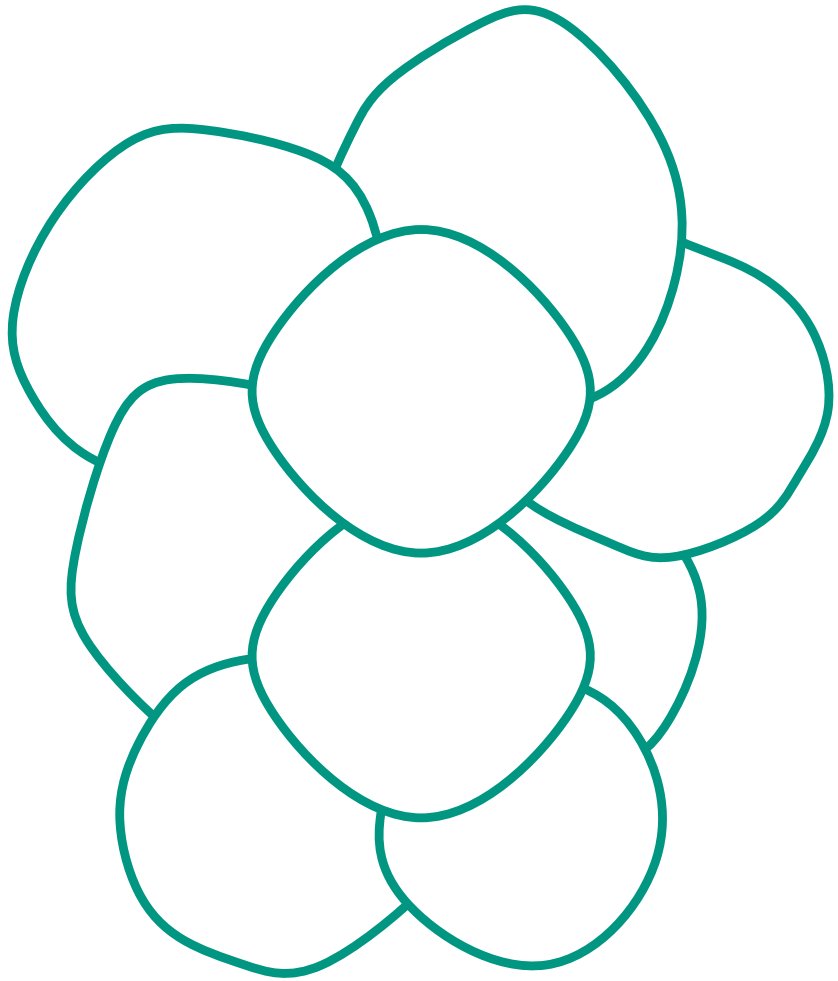
each new vertex chooses cell to which majority of its neighbors belong:



Repair and Reoptimization

observations:

- some cells are too large
- some cells are very small

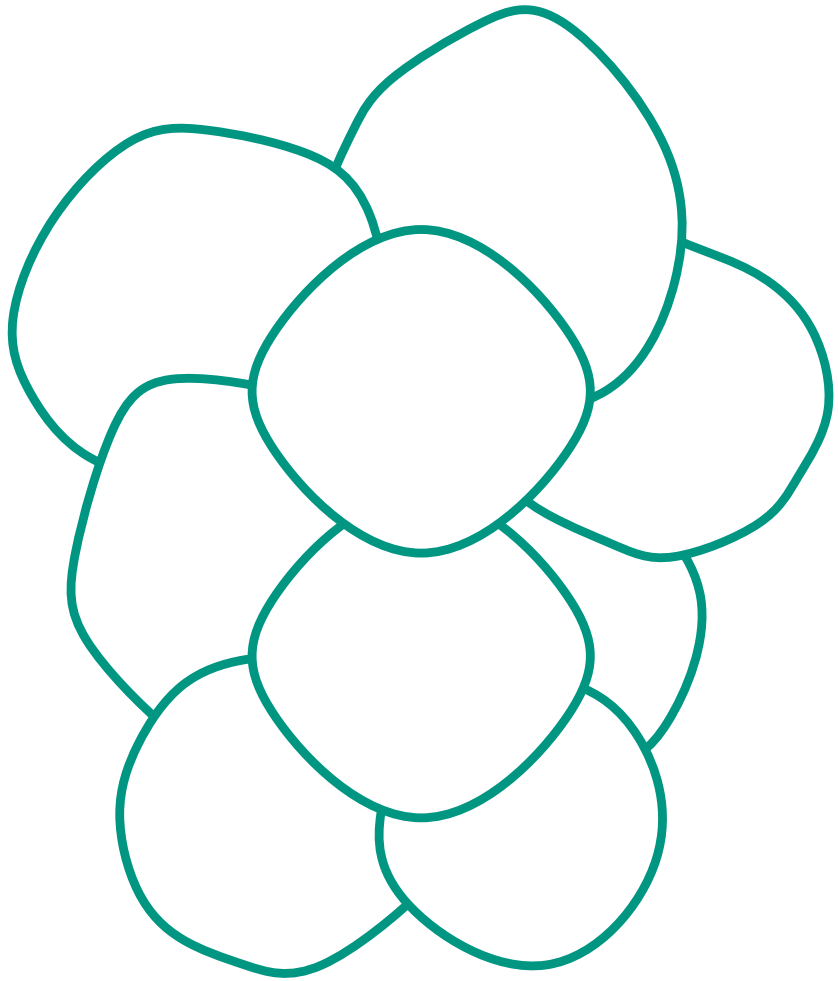


Repair and Reoptimization

observations:

- some cells are too large
- some cells are very small

repair and reoptimize partition:



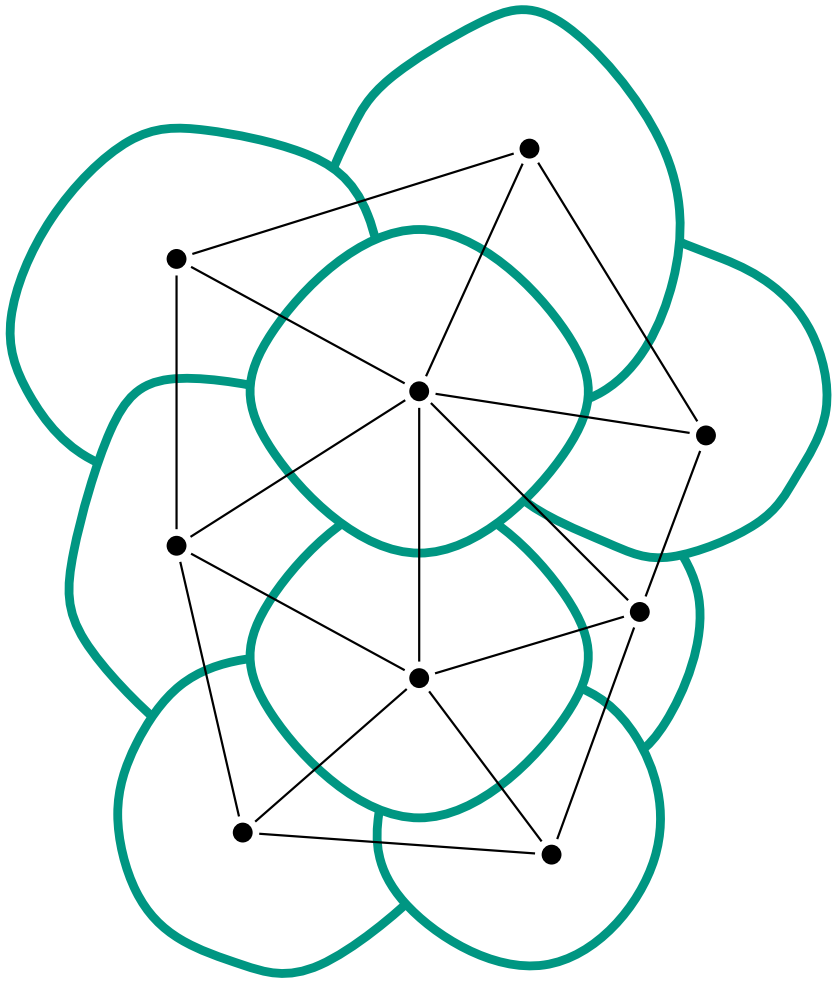
Repair and Reoptimization

observations:

- some cells are too large
- some cells are very small

repair and reoptimize partition:

1. build graph whose vertices are the cells in the partition



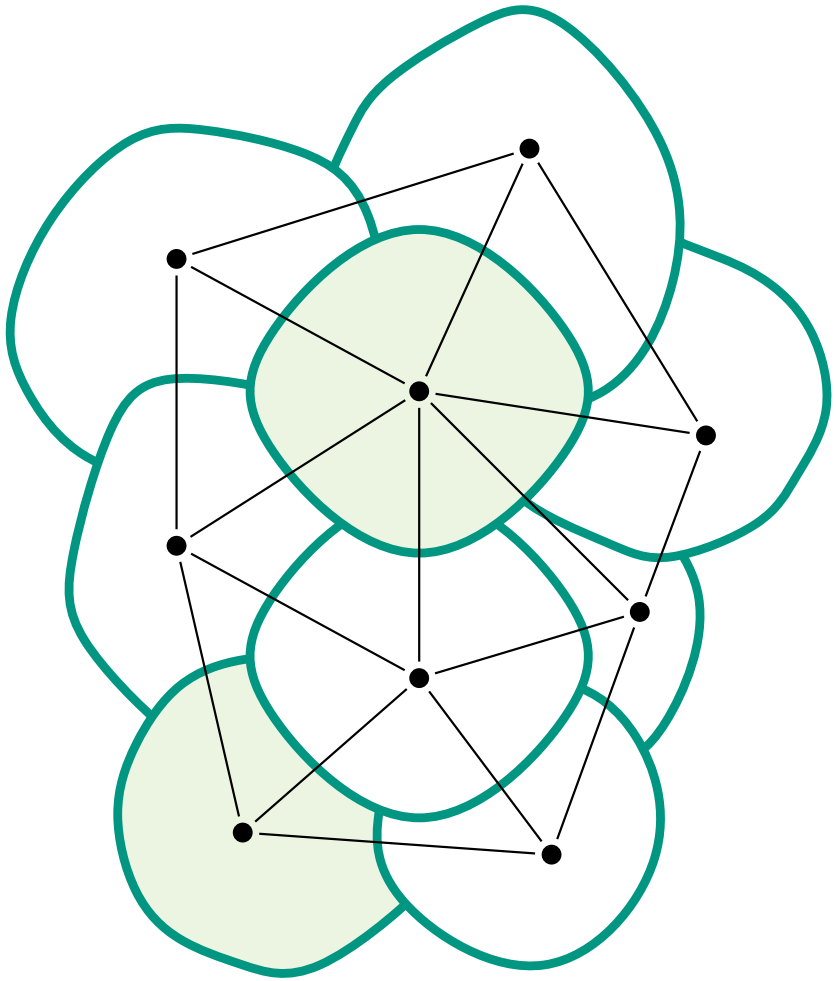
Repair and Reoptimization

observations:

- some cells are too large
- some cells are very small

repair and reoptimize partition:

1. build graph whose vertices are the cells in the partition



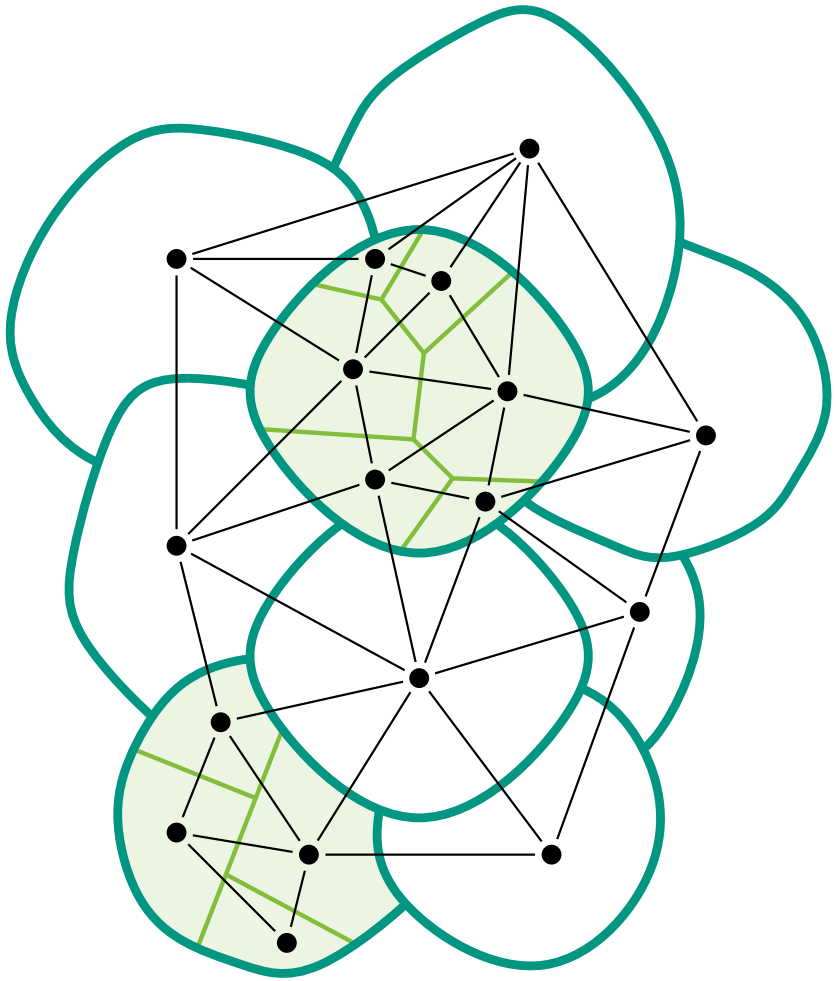
Repair and Reoptimization

observations:

- some cells are too large
- some cells are very small

repair and reoptimize partition:

1. build graph whose vertices are the cells in the partition
2. unpack oversized cells



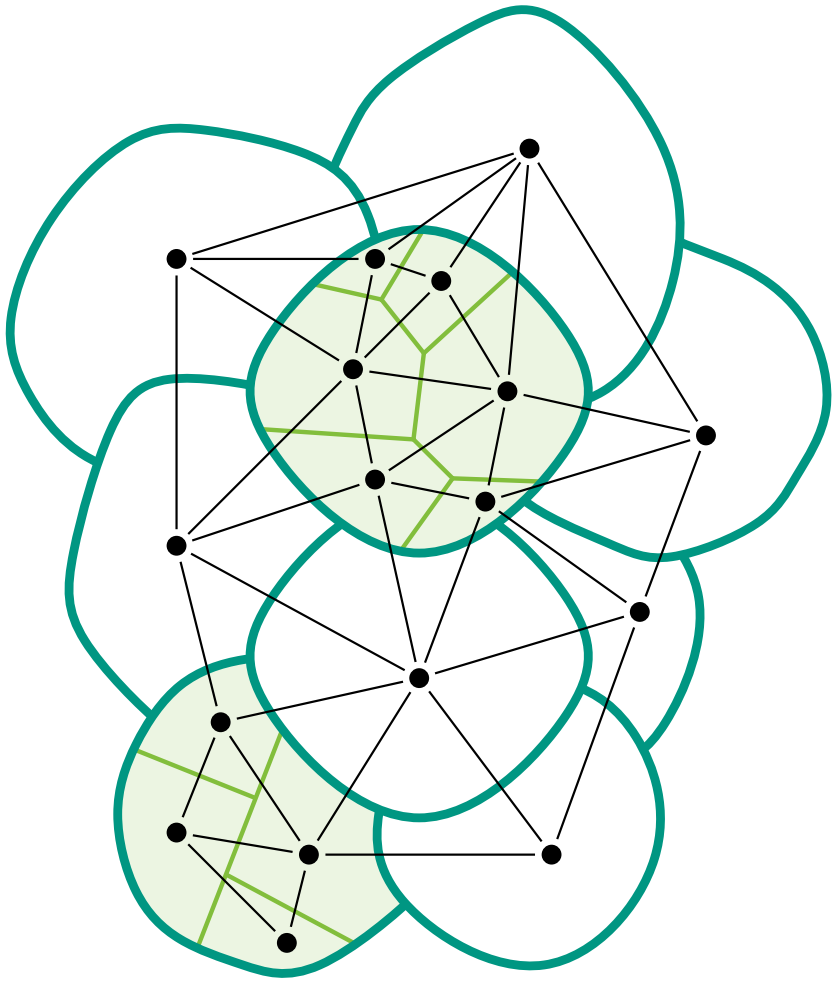
Repair and Reoptimization

observations:

- some cells are too large
- some cells are very small

repair and reoptimize partition:

1. build graph whose vertices are the cells in the partition
2. unpack oversized cells
3. greedily merge pairs of adjacent vertices with combined size $\leq U$



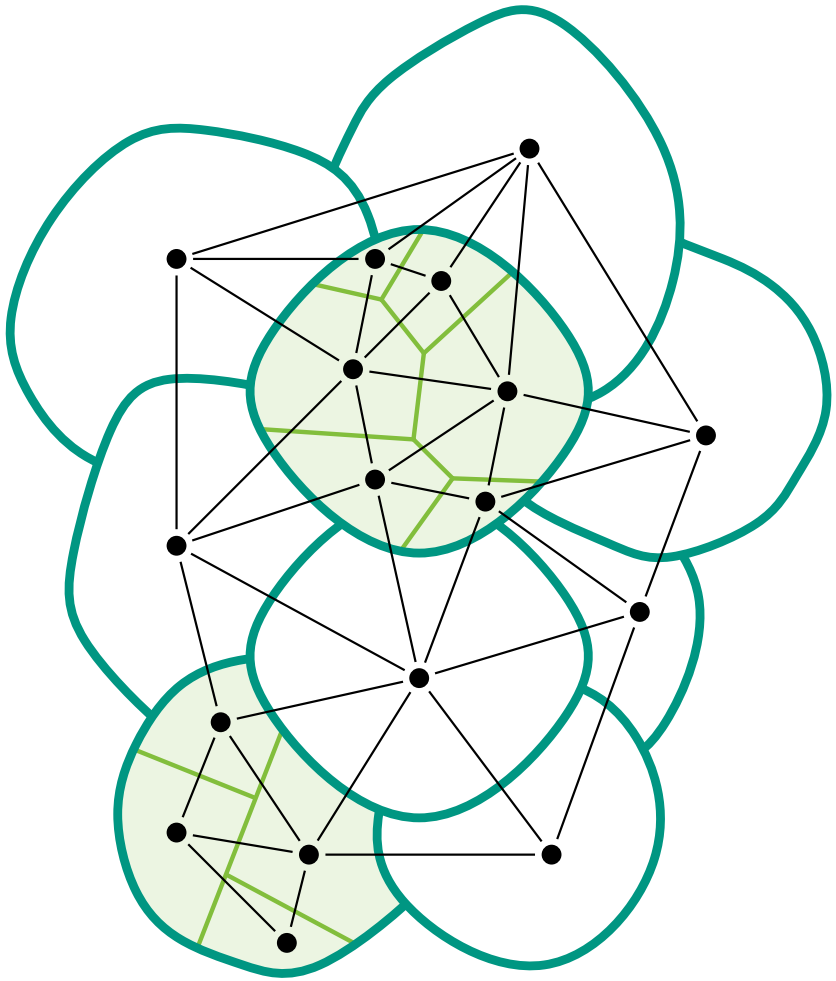
Repair and Reoptimization

observations:

- some cells are too large
- some cells are very small

repair and reoptimize partition:

1. build graph whose vertices are the cells in the partition
2. unpack oversized cells
3. greedily merge pairs of adjacent vertices with combined size $\leq U$
4. locally optimize boundary between pairs of adjacent cells



Repair and Reoptimization

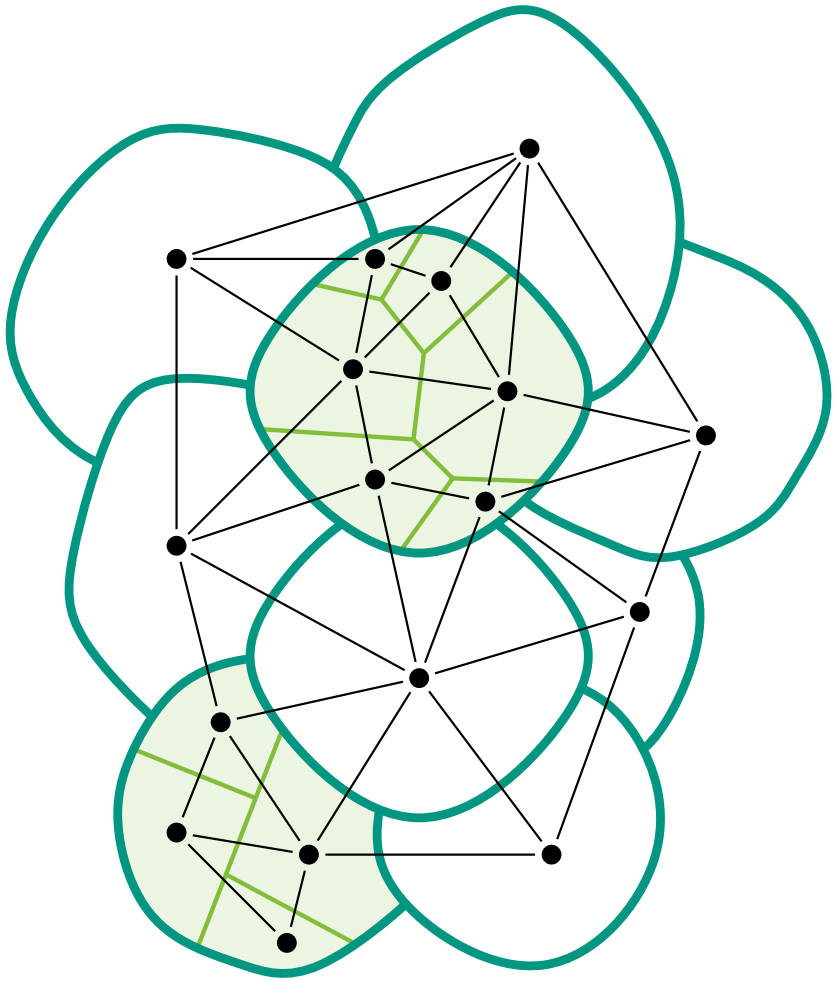
observations:

- some cells are too large
- some cells are very small

repair and reoptimize partition:

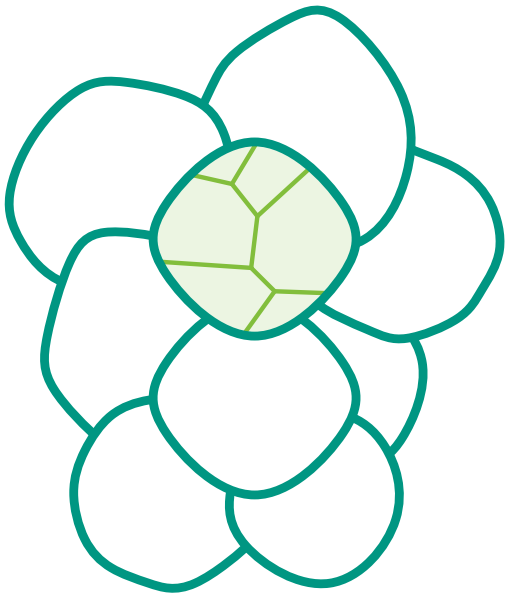
1. build graph whose vertices are the cells in the partition
2. unpack oversized cells
3. greedily merge pairs of adjacent vertices with combined size $\leq U$
4. locally optimize boundary between pairs of adjacent cells

$\Rightarrow$ **greedy algorithm and local search are subroutines from PUNCH**



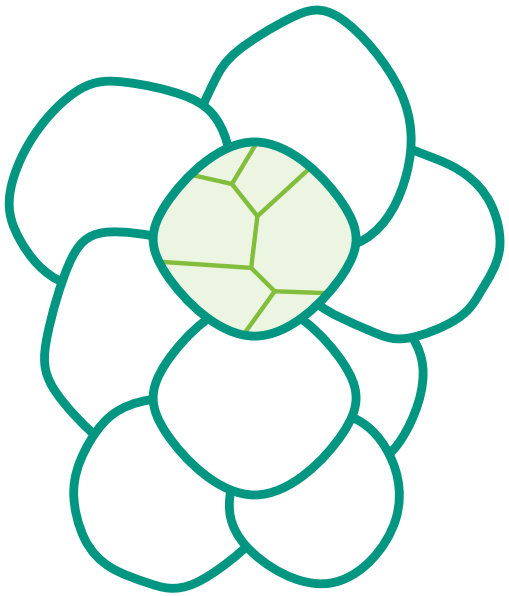
Cell Unpacking

simple unpacking:

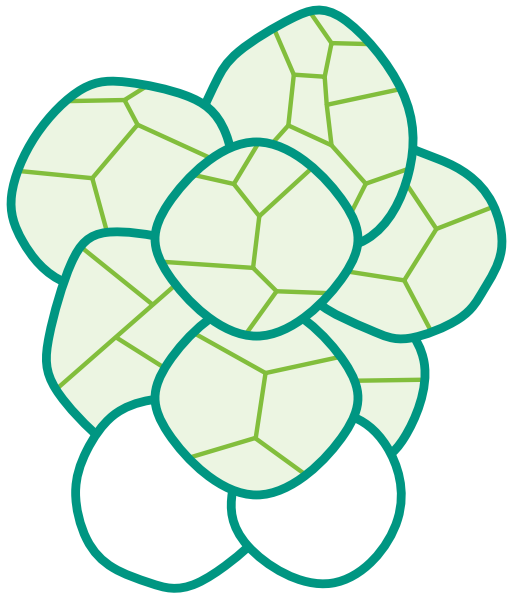


Cell Unpacking

simple unpacking:

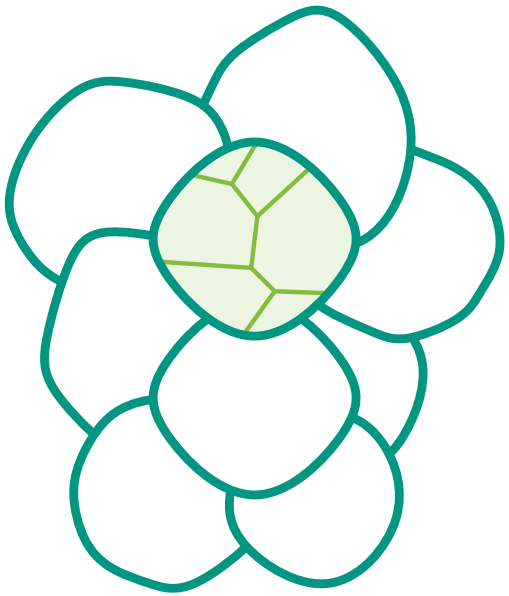


neighbor unpacking:

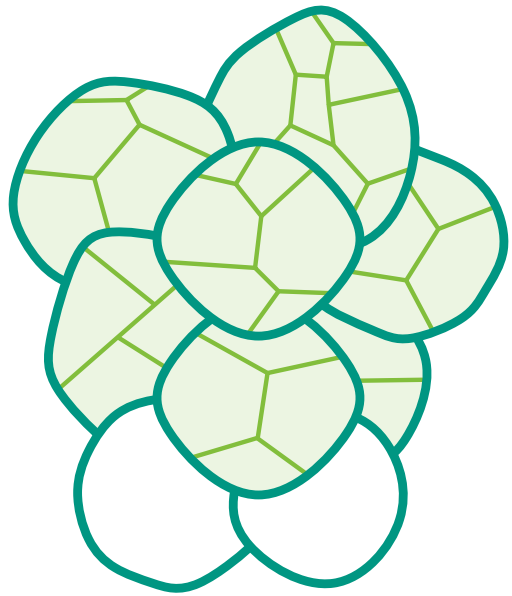


Cell Unpacking

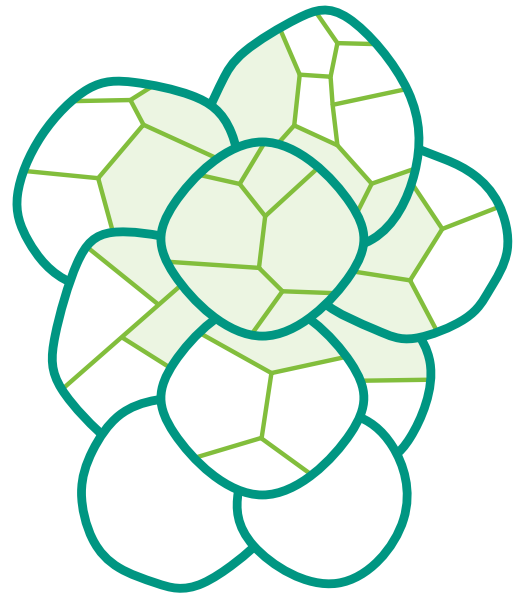
simple unpacking:



neighbor unpacking:

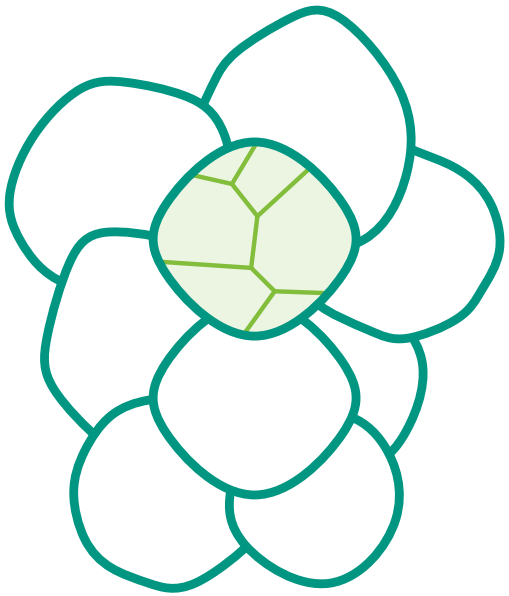


partial unpacking:

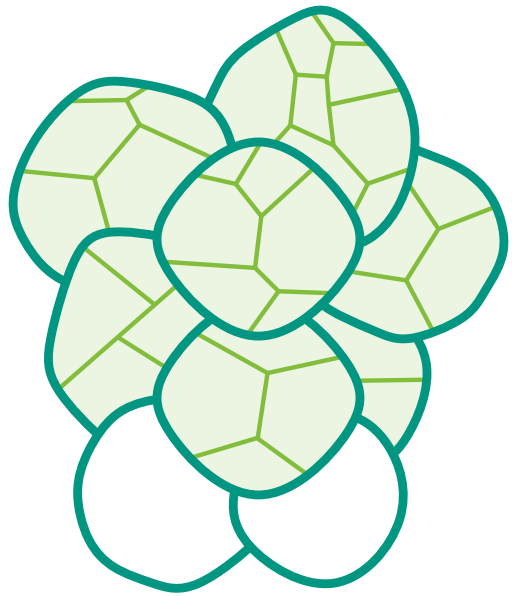


Cell Unpacking

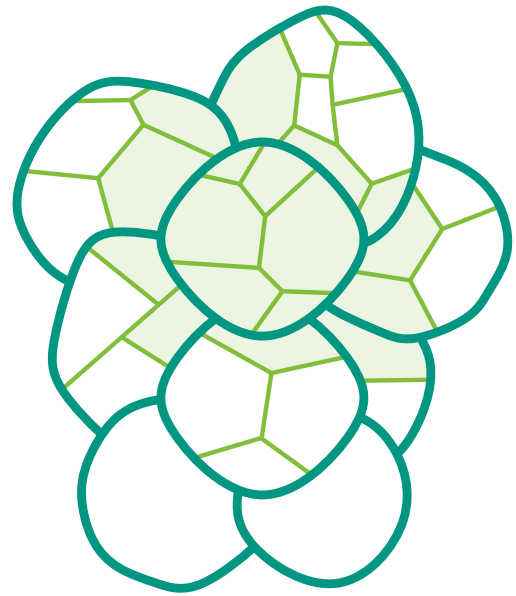
simple unpacking:



neighbor unpacking:



partial unpacking:

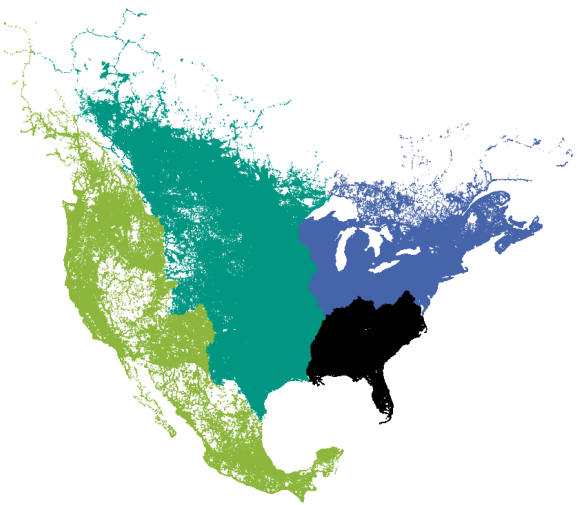


$\Rightarrow$ **simple unpacking yields good tradeoff between quality and similarity**

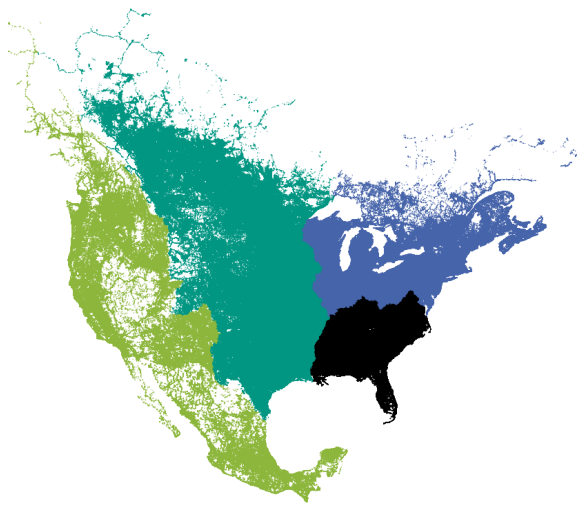
Experimental Evaluation

OSM North America:

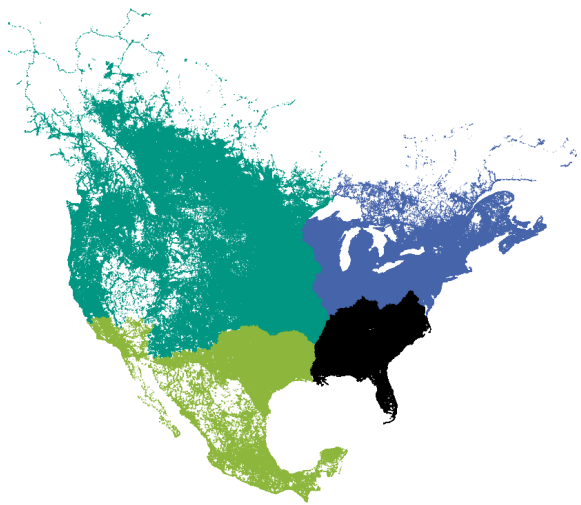
- 25 million vertices
- monthly snapshots
- change in vertices: 0.36 % are added, 0.21 % are removed



August 2018
from scratch



September 2018
repartitioned

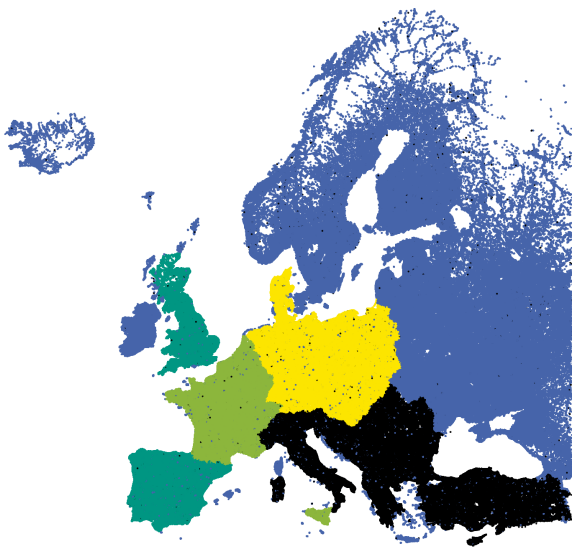


September 2018
from scratch

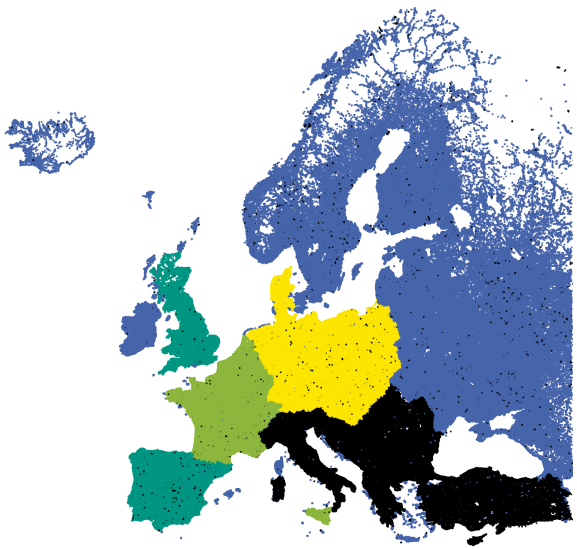
Experimental Evaluation

OSM Europe:

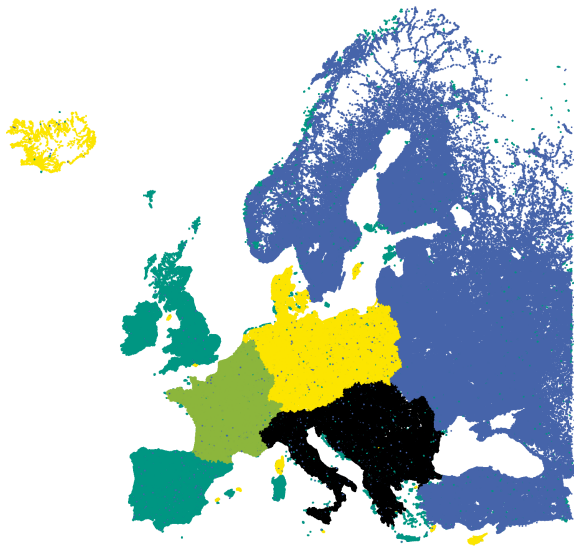
- 31 million vertices
- yearly snapshots
- change in vertices: 7.12 % are modified



January 2018
from scratch



January 2019
repartitioned



January 2019
from scratch

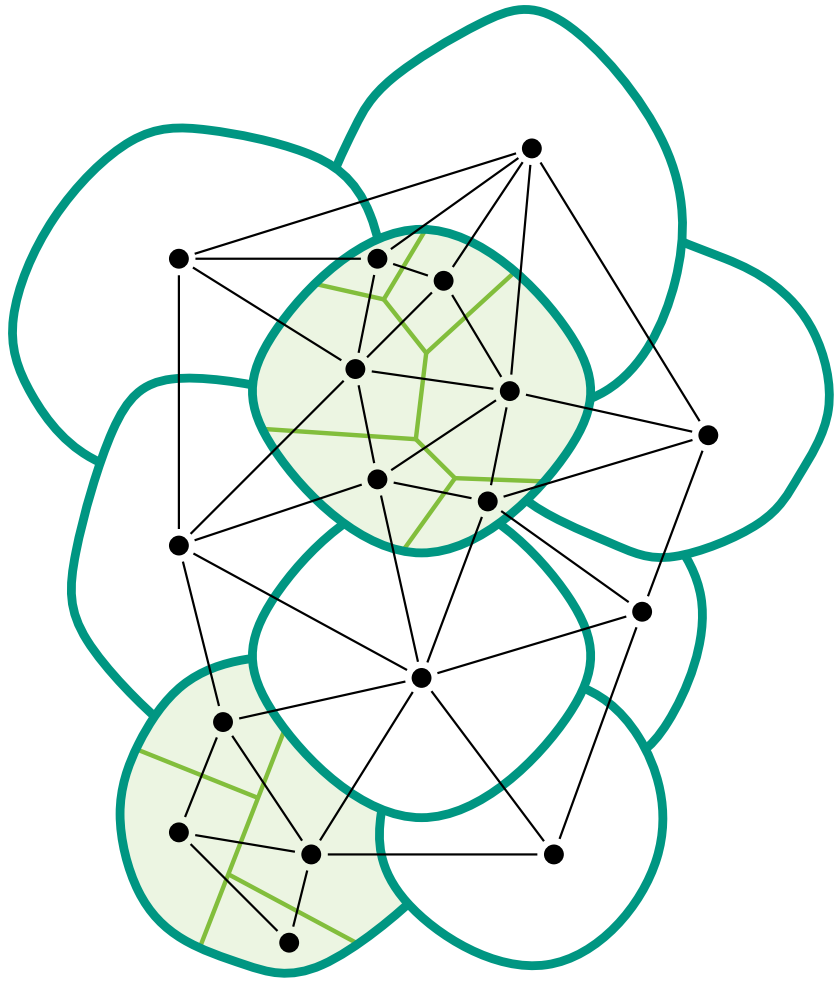
Experimental Evaluation

			similarity [%]				
		cut	lowest level		highest level		
period	instance	[%]	repair	full	repair	full	speedup
mo	Australia	3.09	98.87	64.57	97.60	39.27	12.07
	N America	3.11	98.64	64.01	87.16	31.07	9.64
	Europe	3.24	98.35	62.70	94.70	50.73	9.35
yr	Australia	10.91	85.16	49.60	75.49	23.33	6.31
	N America	13.14	91.04	54.55	62.46	32.53	4.09
	Europe	12.83	90.41	52.98	73.64	46.03	7.48

Conclusion

summary:

- new repartitioning algorithm
- improves fraction of unchanged overlay vertices to more than 90 %
- accelerates network repartitioning by an order of magnitude
- cut size is only about 3 % higher



Conclusion

summary:

- new repartitioning algorithm
- improves fraction of unchanged overlay vertices to more than 90 %
- accelerates network repartitioning by an order of magnitude
- cut size is only about 3 % higher

future work:

- handle larger changes
- other classes of evolving networks