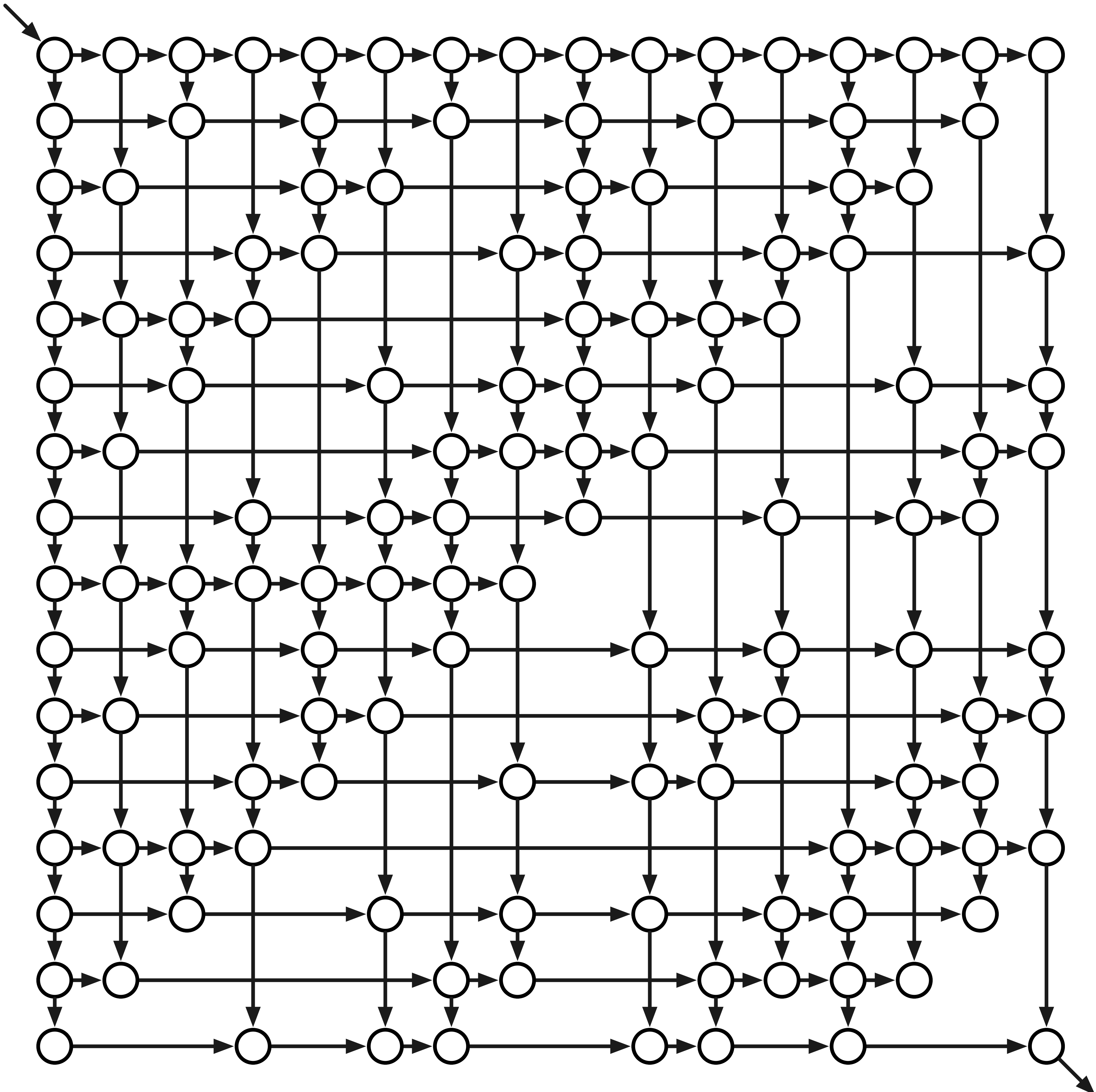
Algorithms for Planar Graphs

Jeff Erickson
University of Illinois, Urbana-Champaign

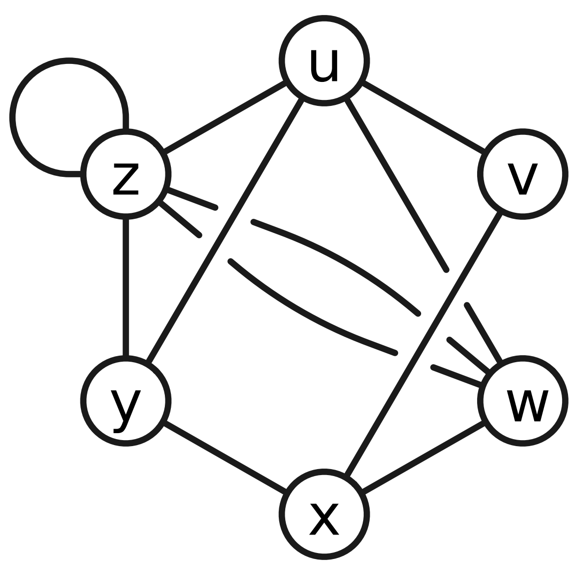
CIMAT, Guanajuato, Mexico
September 10, 2018

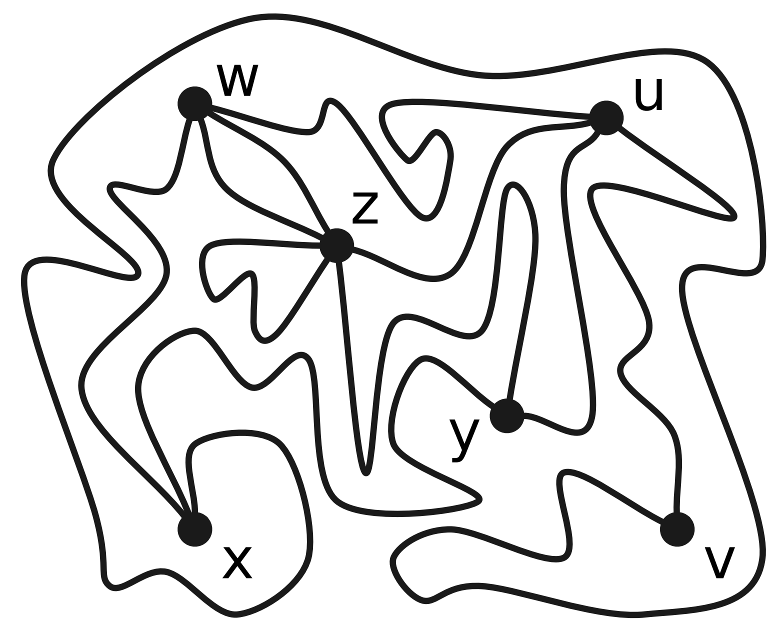
Please ask questions!

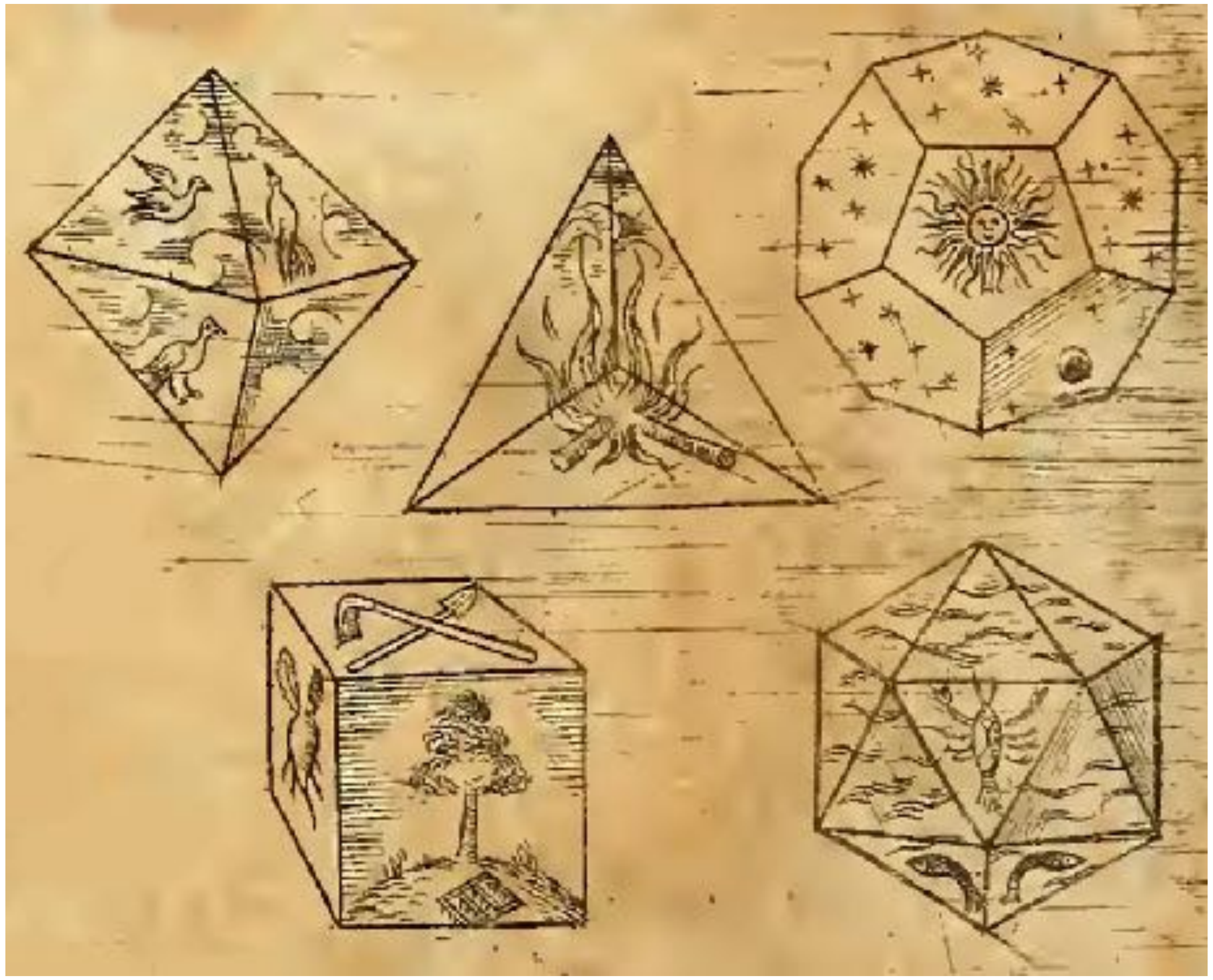


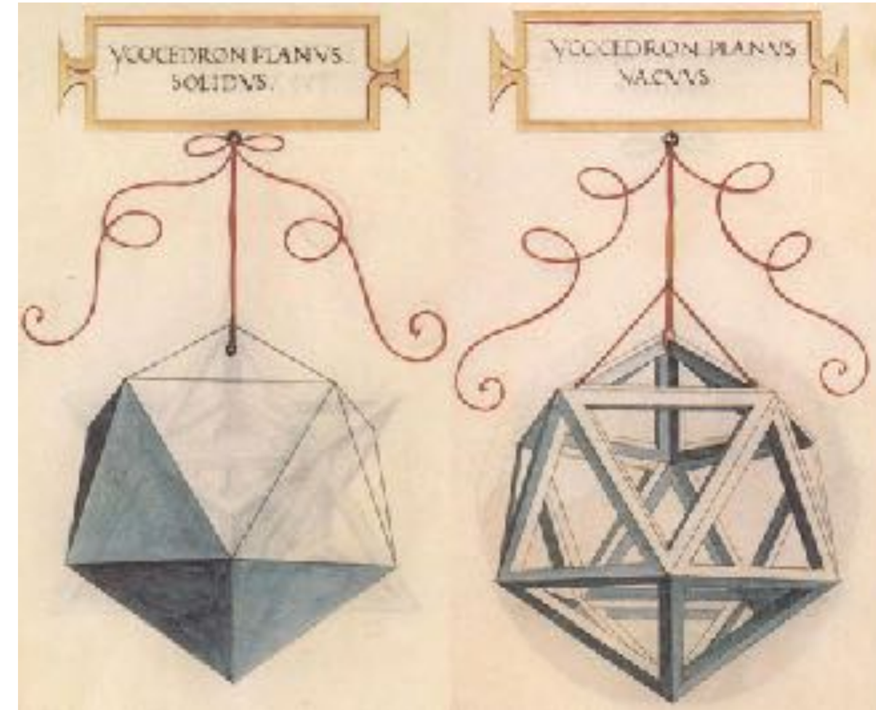
Planar Graphs

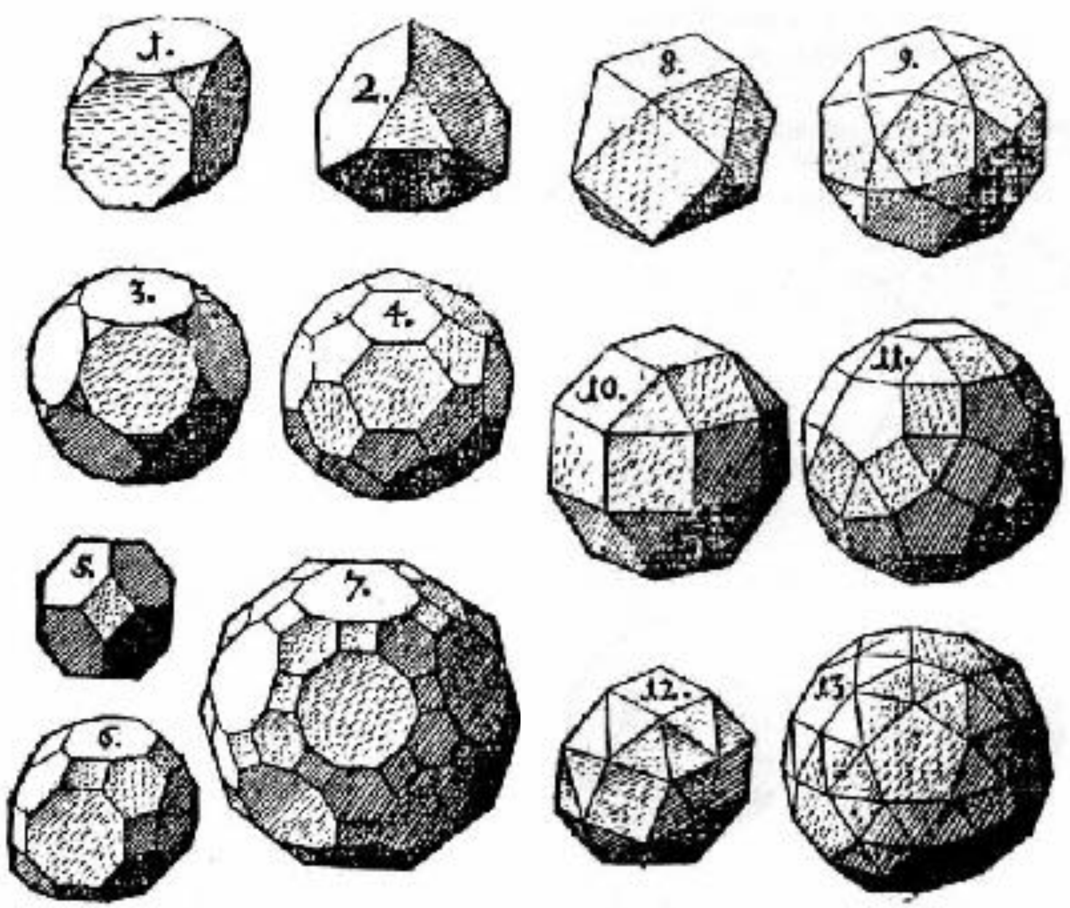
- A graph is *planar* if it can be drawn in the plane without edge crossings
  - Vertices = points
  - Edges = simple disjoint curves





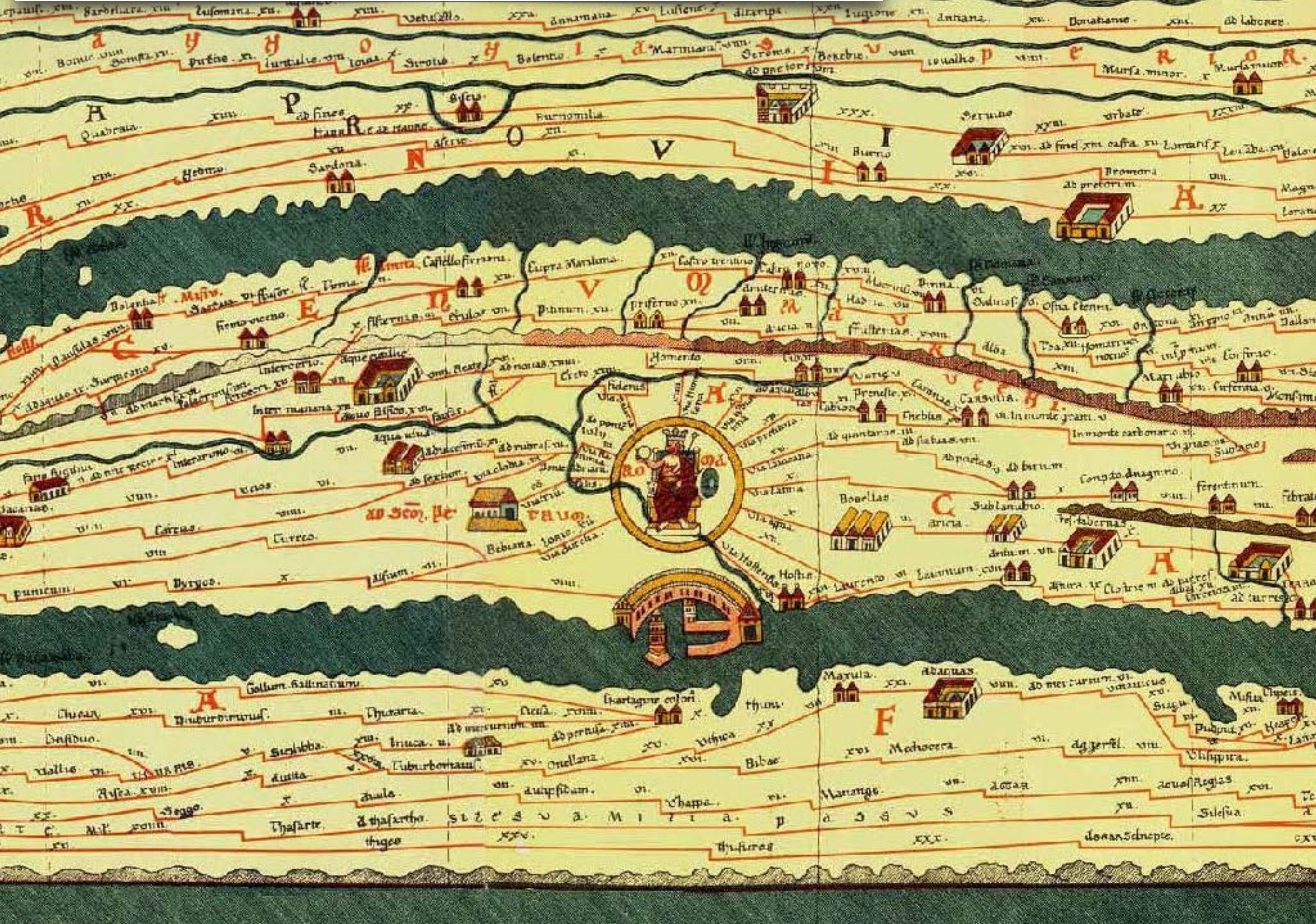



YCOCEDRON PLANVS.
SOLIDVS.
YCOCEDRON PLANVS
VACVVS

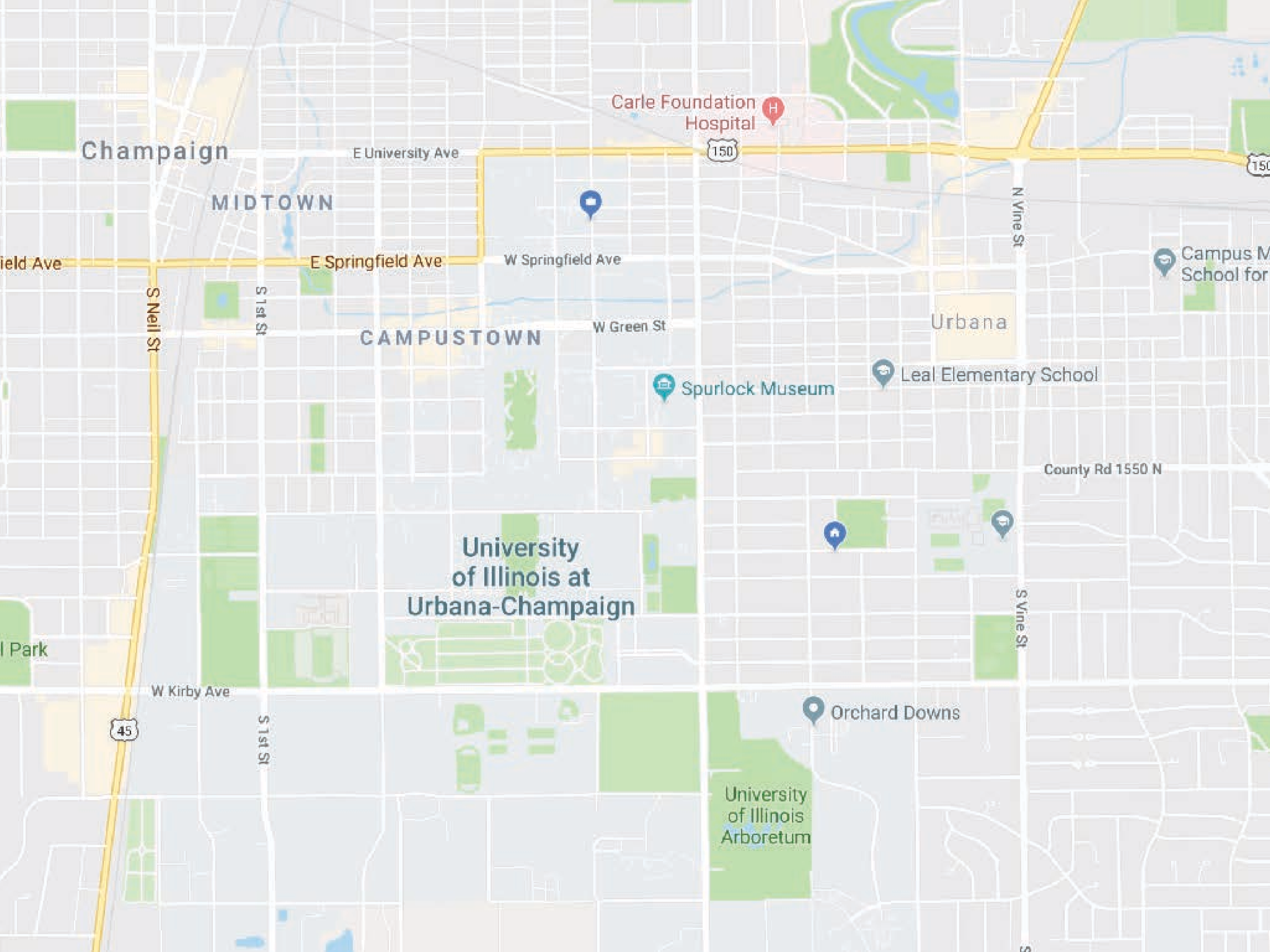

1.
2.
8.
9.
3.
4.
10.
11.
5.
7.
6.
12.
13.

Tabula Peutingeriana [1st c.→ 5th c.→ 9th c.→ 13th c.→ Miller 1887]

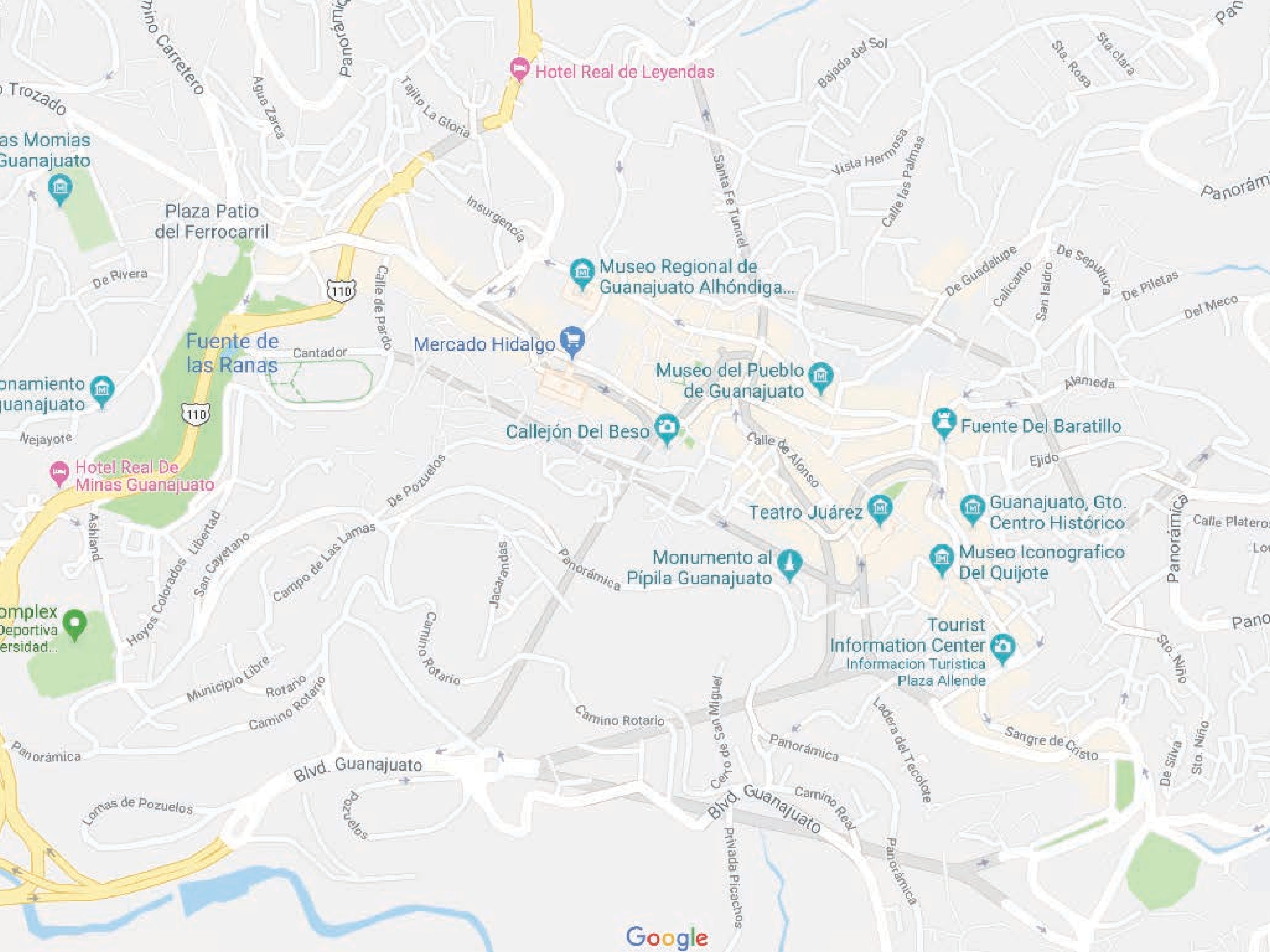
Tabula Peutingeriana [1st c.→ 5th c.→ 9th c.→ 13th c.→ Miller 1887]



Tabula Peutingeriana [1st c.→ 5th c.→ 9th c.→ 13th c.→ Miller 1887]

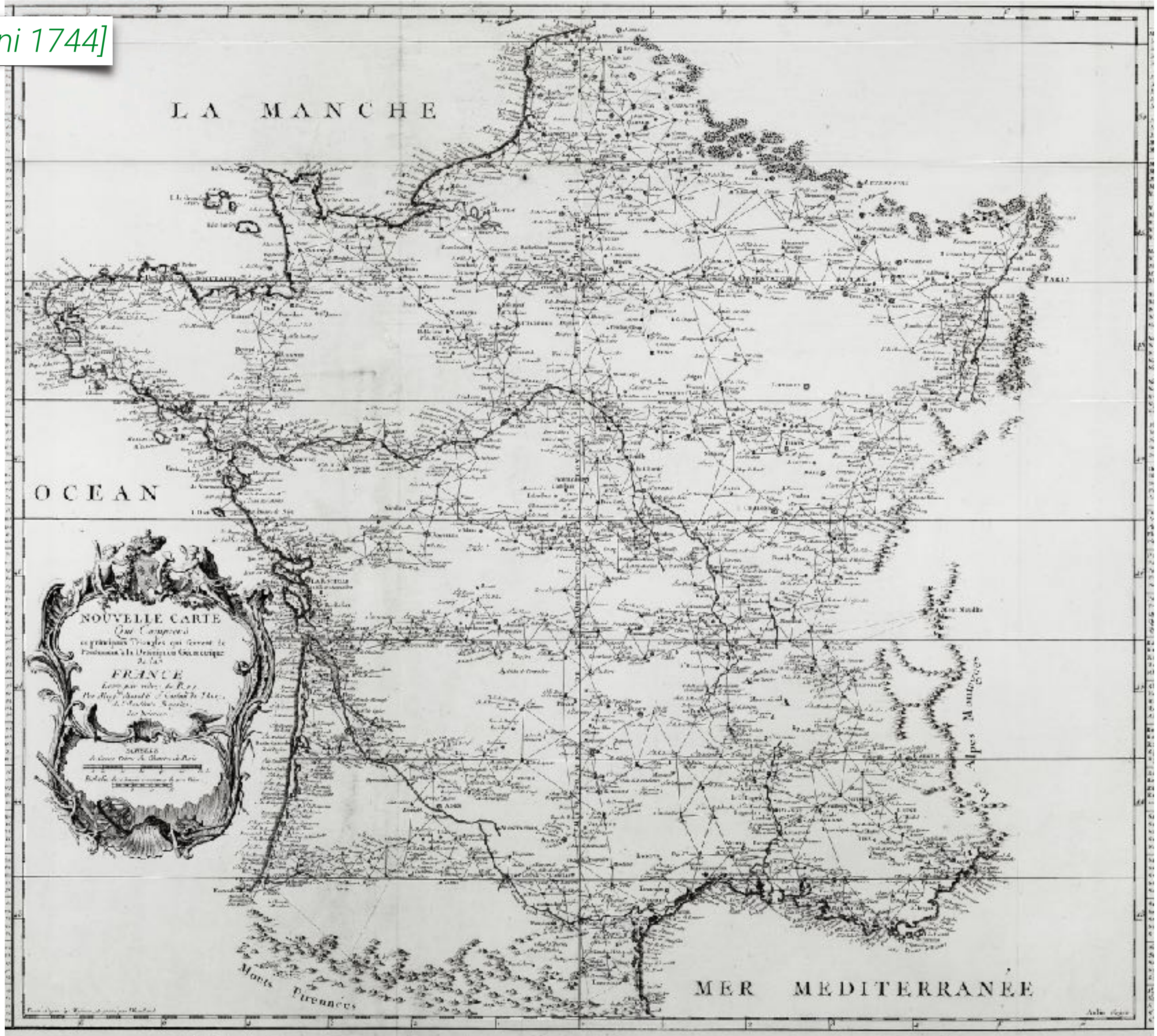


Carle Foundation
Hospital
Champaign
E University Ave
150
MIDTOWN
ield Ave
E Springfield Ave
W Springfield Ave
N Vine St
Campus M
School for
S Neil St
S 1st St
W Green St
CAMPUSTOWN
Urbana
Leal Elementary School
Spurlock Museum
County Rd 1550 N
University
of Illinois at
Urbana-Champaign
S Vine St
l Park
W Kirby Ave
Orchard Downs
45
S 1st St
University
of Illinois
Arboretum



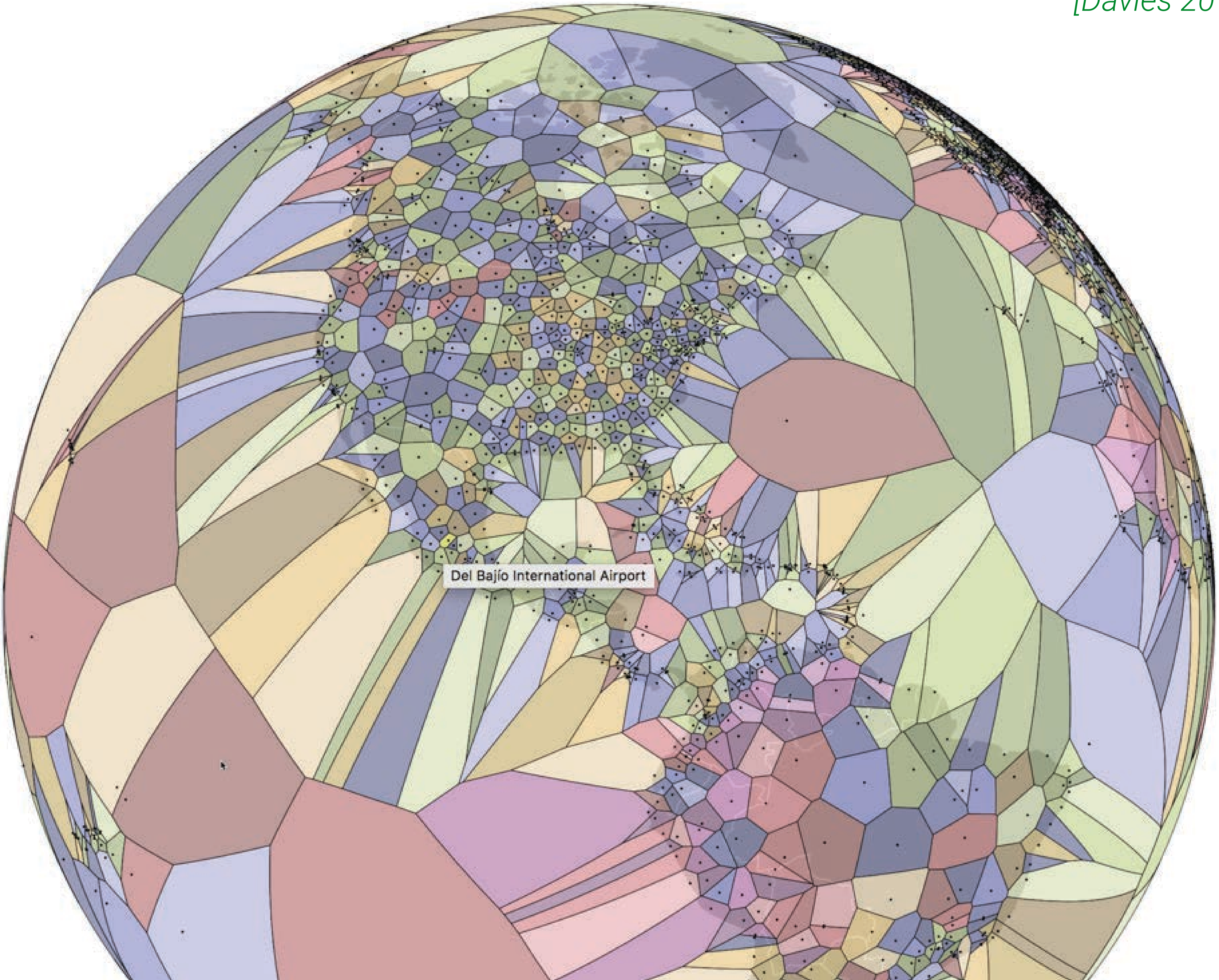
Hotel Real de Leyendas
Museo Regional de Guanajuato Alhóndiga...
Mercado Hidalgo
Museo del Pueblo de Guanajuato
Callejón Del Beso
Fuente Del Baratillo
Teatro Juárez
Guanajuato, Gto. Centro Histórico
Monumento al Pípila Guanajuato
Museo Iconografico Del Quijote
Tourist Information Center
Informacion Turistica Plaza Allende
Plaza Patio del Ferrocarril
Fuente de las Ranas
Hotel Real De Minas Guanajuato
110
Bajada del Sol
Santa Fe Tunnel
Vista Hermosa
Calle las Palmas
Insurgencia
Tajito La Gloria
Agua Zarca
Calle de Pardo
Cantador
De Rivera
Nejayote
De Guadalupe
Calicanto
San Isidro
De Sepultura
De Piletas
Del Meco
Alameda
Ejido
Calle de Alonso
Calle Plateros
De Pozuelos
Campo de Las Lamas
San Cayetano
Libertad
Hoyos Colorados
Ashland
Jacarandas
Panorámica
Camino Rotario
Municipio Libre
Rotario
Blvd. Guanajuato
Lomas de Pozuelos
Pozuelos
Cerro de San Miguel
Privada Picachos
Camino Real
Ladera del Tecolote
Sangre de Cristo
Sto. Niño
De Silva
Sta. Rosa
Sta. clara
Google

[Cassini 1744]

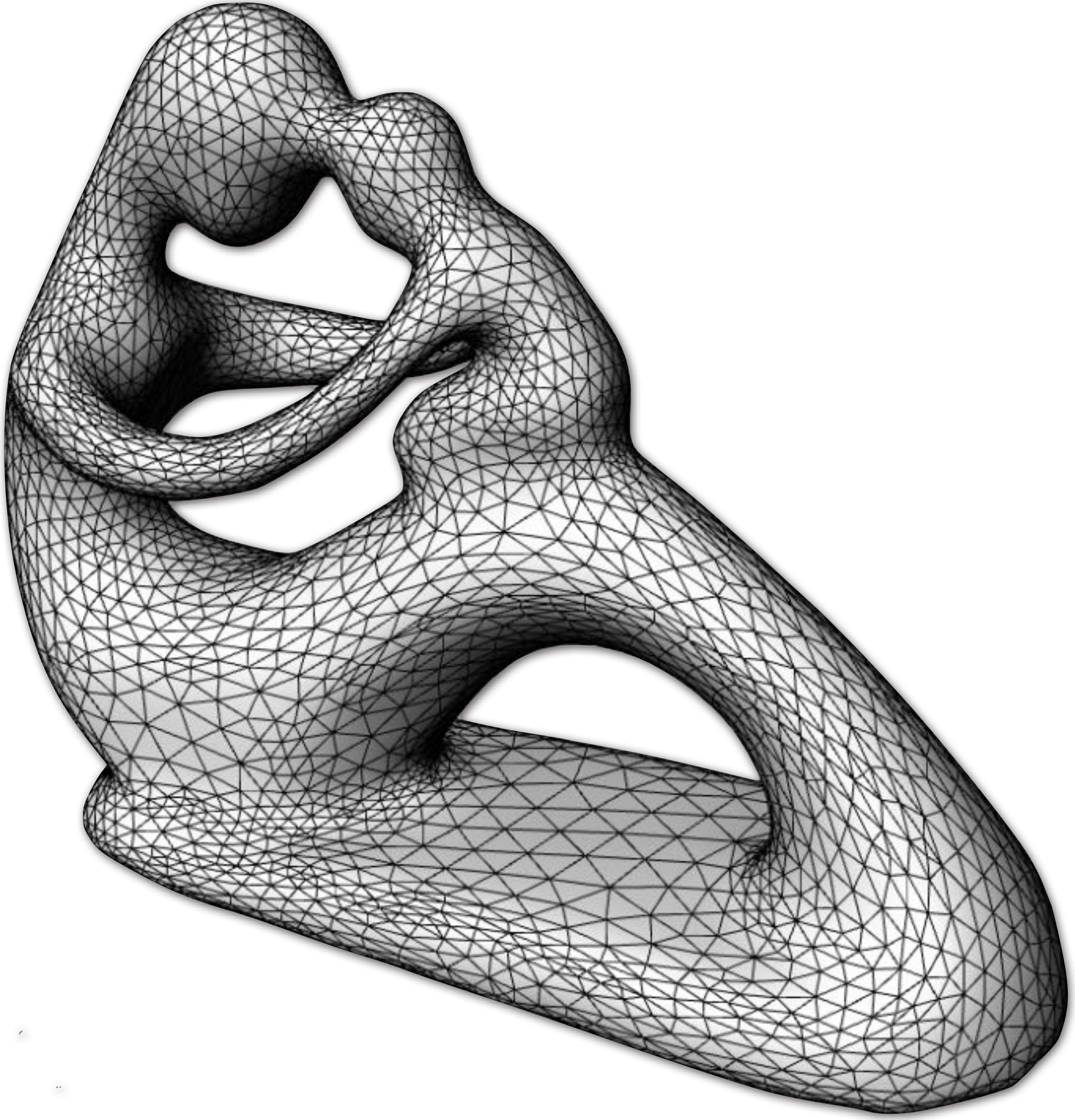


[Cassini 1744]

[Davies 2015]

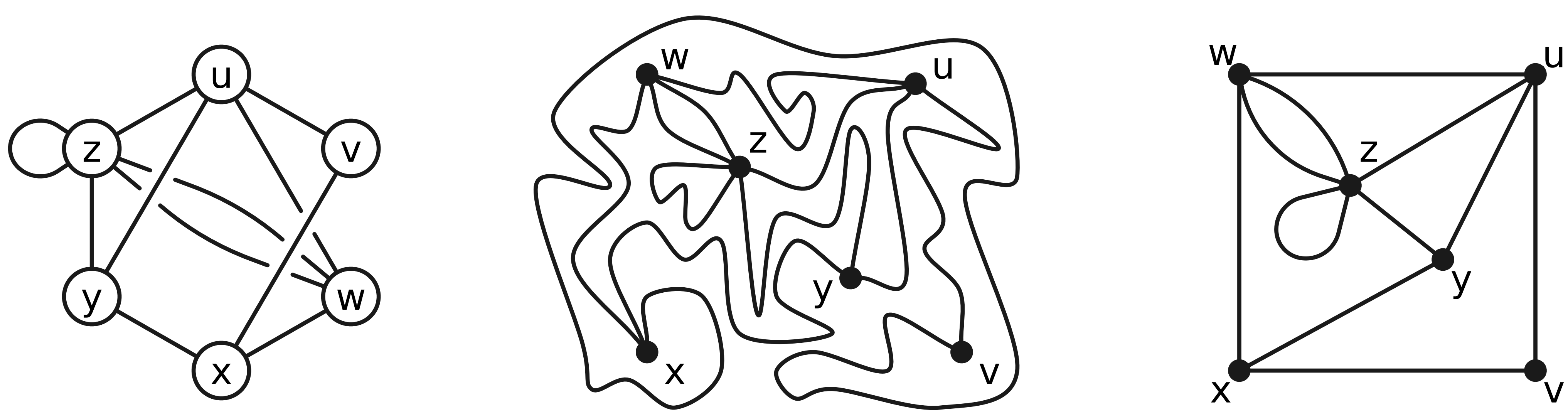


[Ohtake, Belyaev, Seidel 2005]



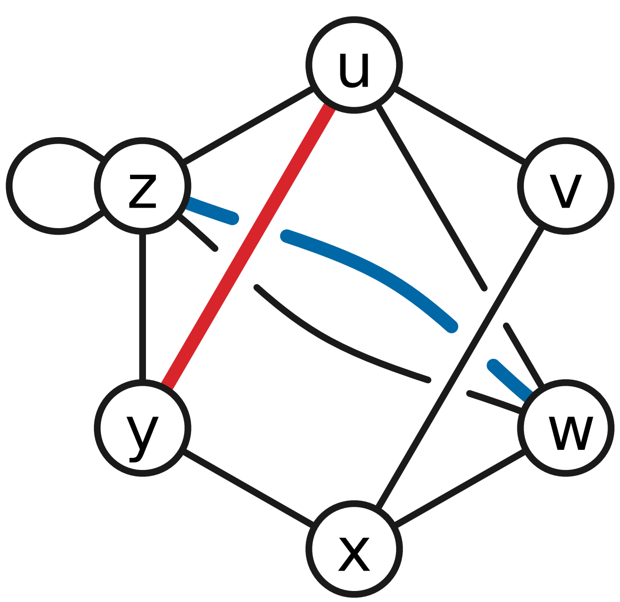
Planar embedding

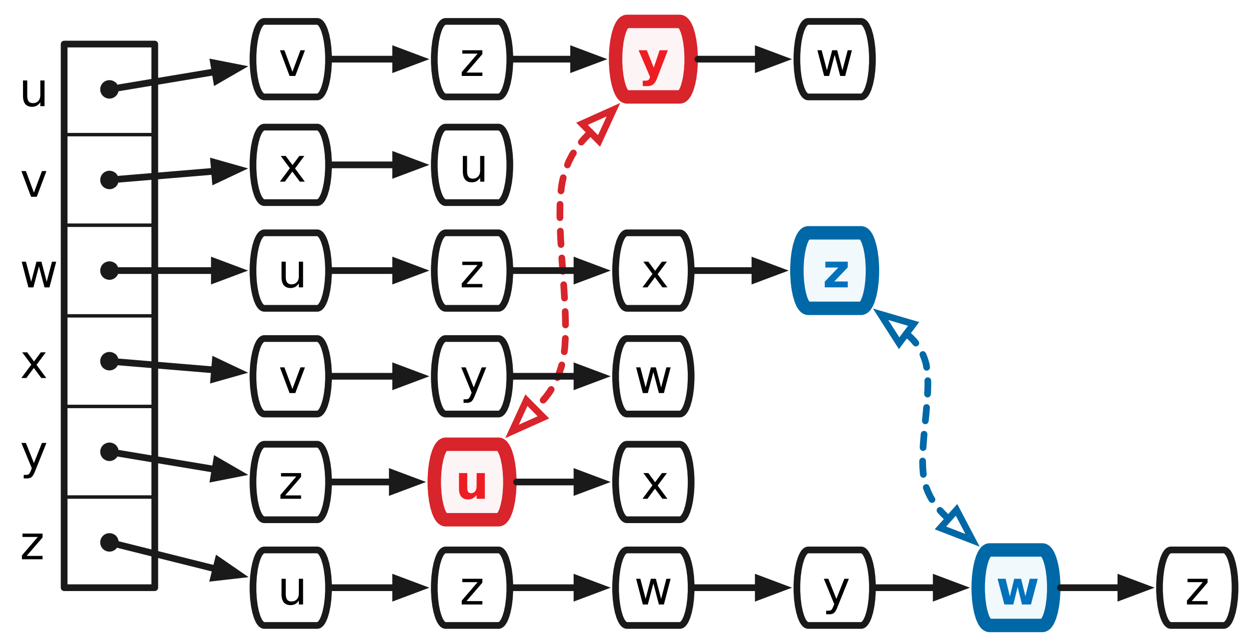
- Vertices = points
- Edges = simple interior-disjoint curves



Incidence list

- Each edge represented by two directed edges or *darts*
- Each vertex *v* stores the list of darts into *v*
- Each dart stores successor, reversal, and tail vertex

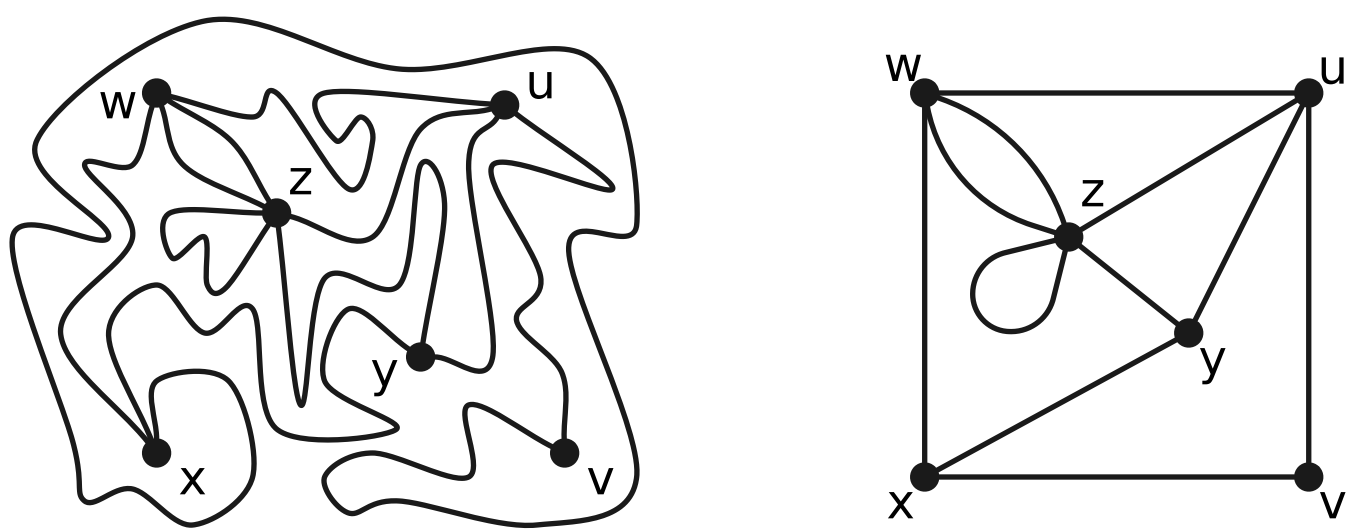




Rotation system

[Hamilton 1856] [Kirkman 1856] [Cayley 1857] [Heffter 1891] [Brückner 1900]....

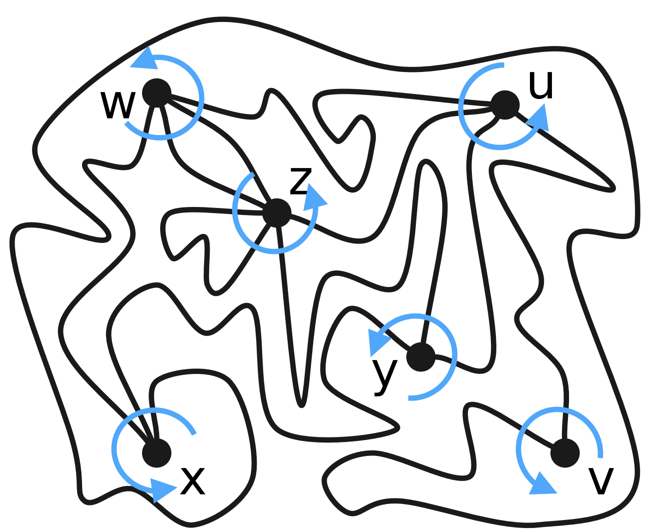
- Counterclockwise order of edges incident to each vertex
- Specifies the embedding of G on the sphere, up to (ambient) isotopy

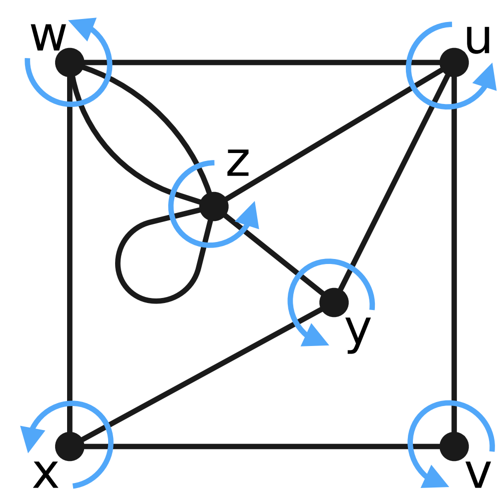


Rotation system

[Hamilton 1856] [Kirkman 1856] [Cayley 1857] [Heffter 1891] [Brückner 1900]....

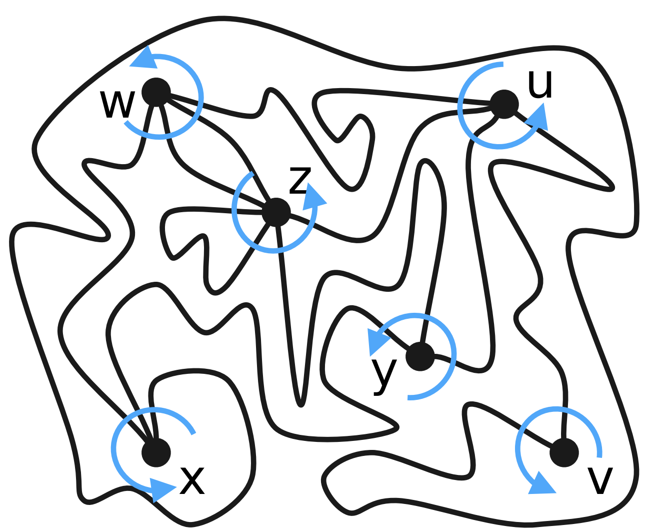
- Counterclockwise order of edges incident to each vertex
- Specifies the embedding of G on the sphere, up to (ambient) isotopy

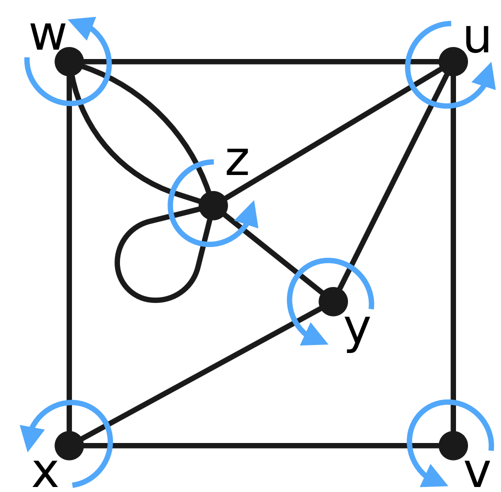


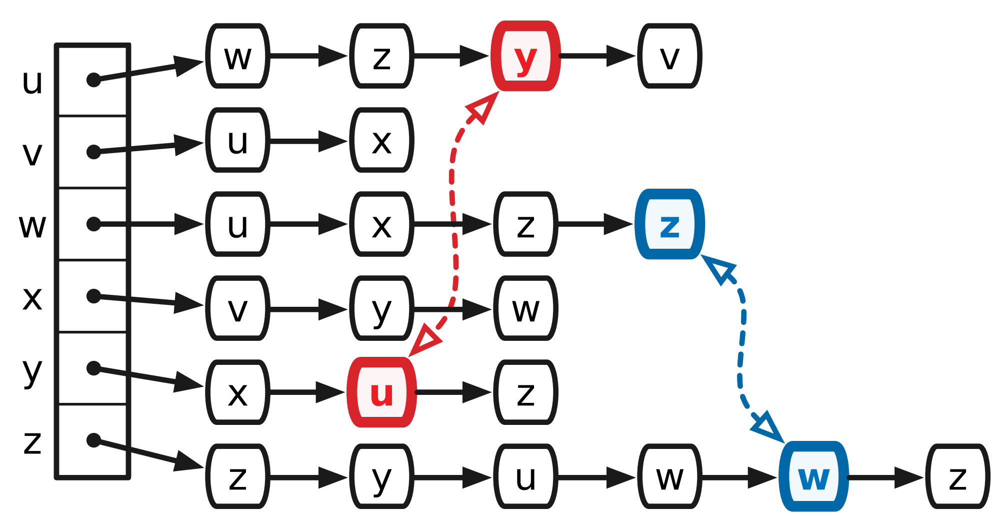


Ordered incidence list

- Record the rotation system in the incidence list



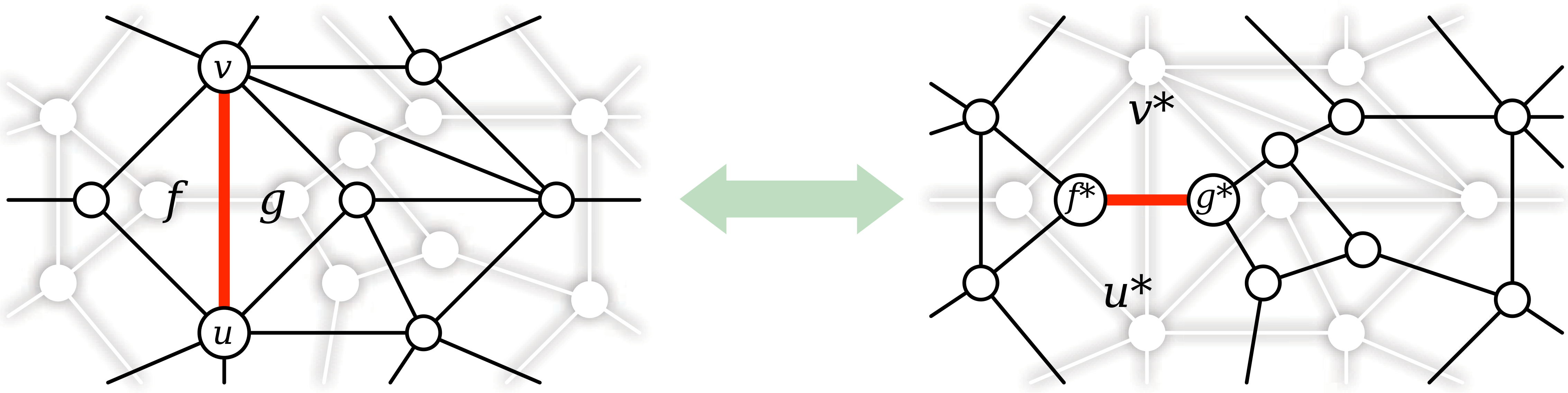




Duality

Every plane graph G has a natural *dual* plane graph G^*:

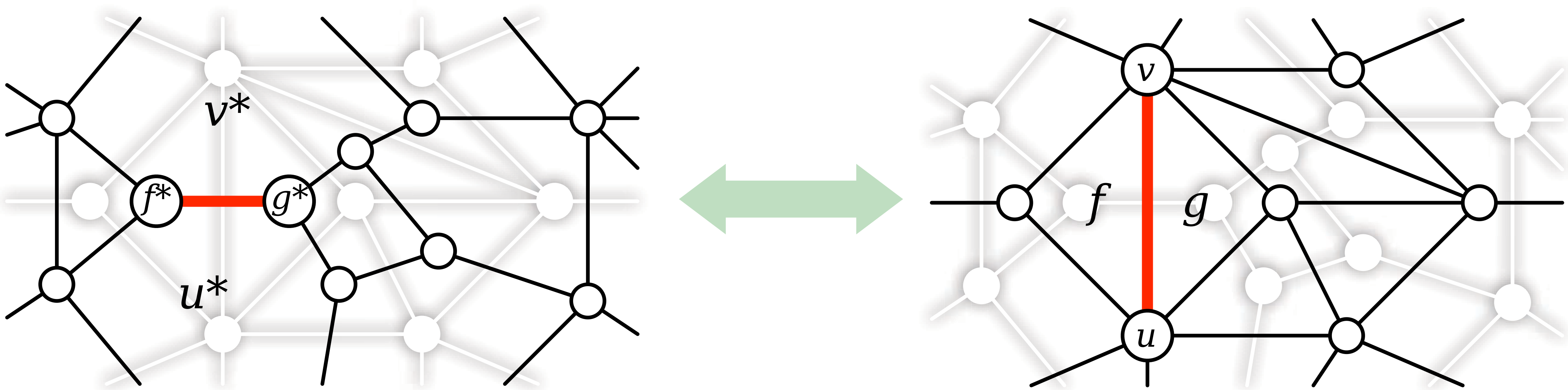
- vertices of G^* = faces of G
- edges of G^* = edges of G
- faces of G^* = vertices of G



Duality

Every plane graph *G* has a natural *dual* plane graph *G**:

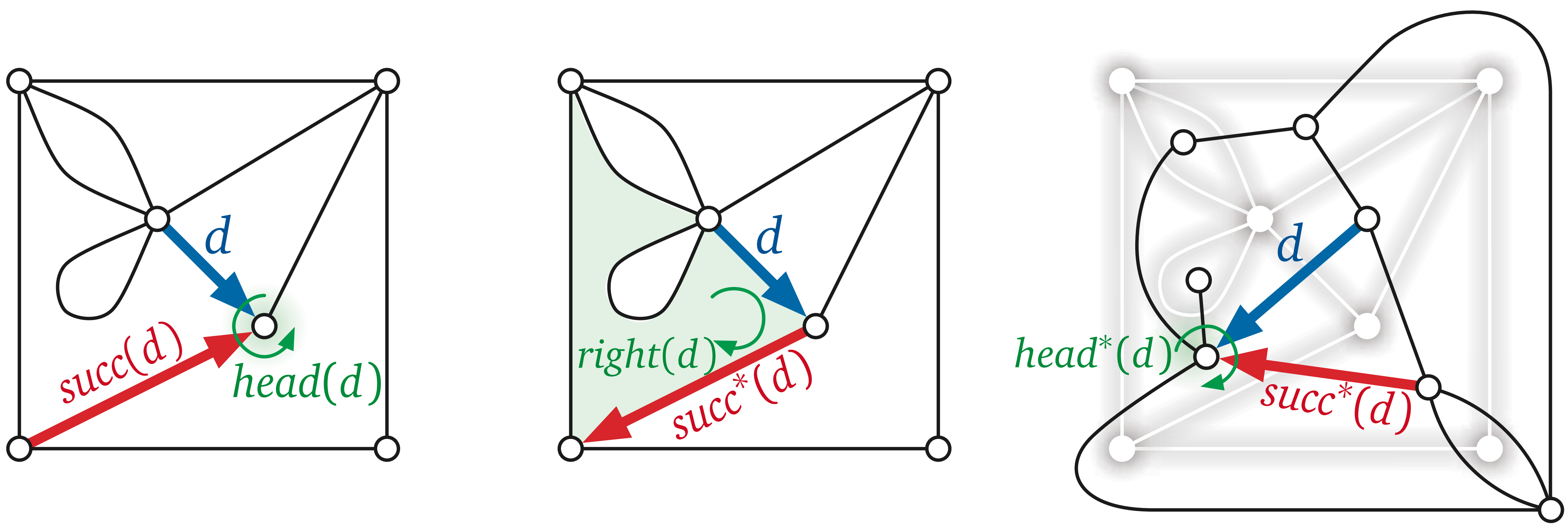
- vertices of *G** = faces of *G*
- edges of *G** = edges of *G*
- faces of *G** = vertices of *G*



No new data structure

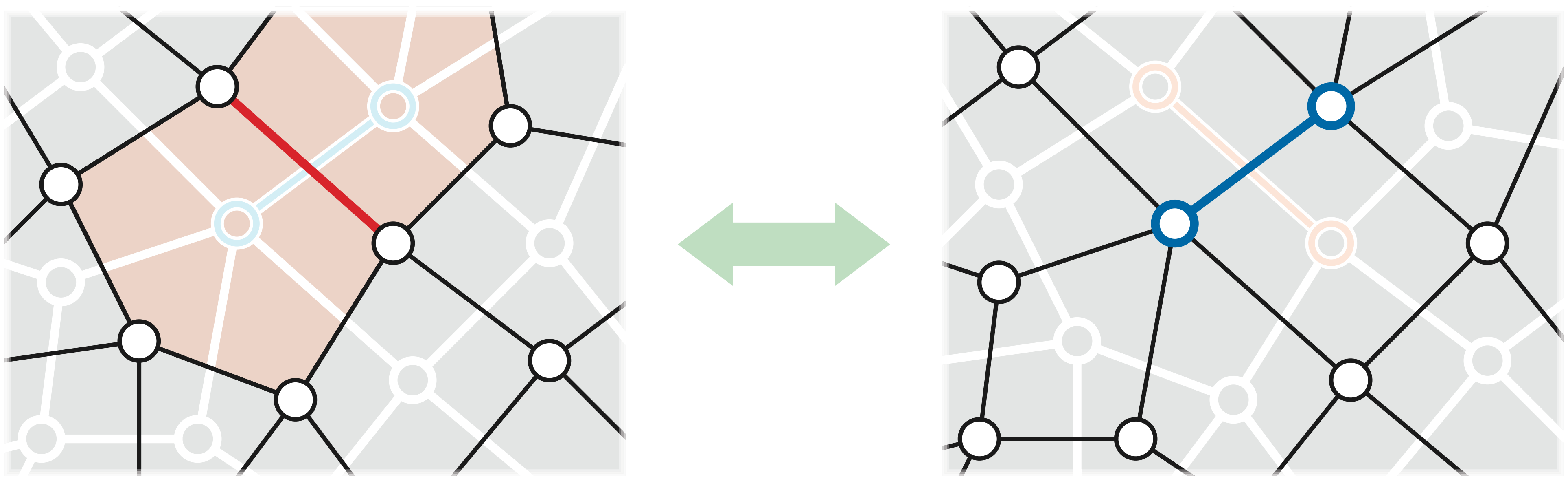
- The same ordered incidence list encodes *both G* and *G**.

$\boxed{succ^*(d) = rev(succ(d))}$



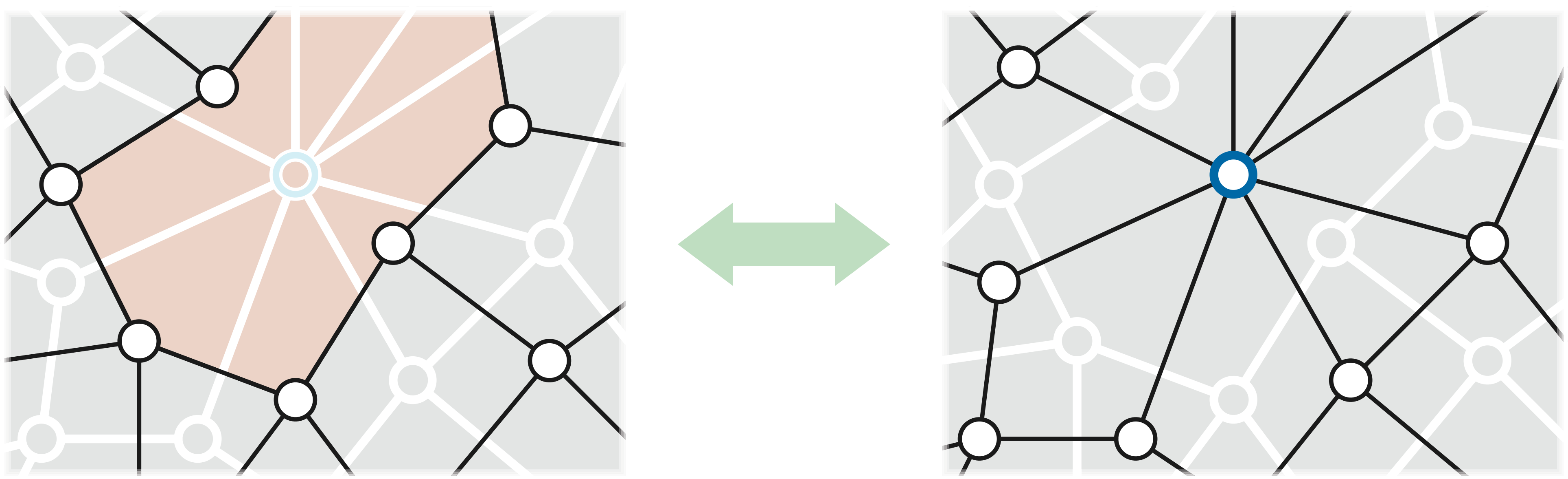
Contraction ⇔ Deletion

- *Deleting* an edge merges the *faces* on both *sides*
- *Contracting* an edge merges the *vertices* at both *ends*



Contraction ⇔ Deletion

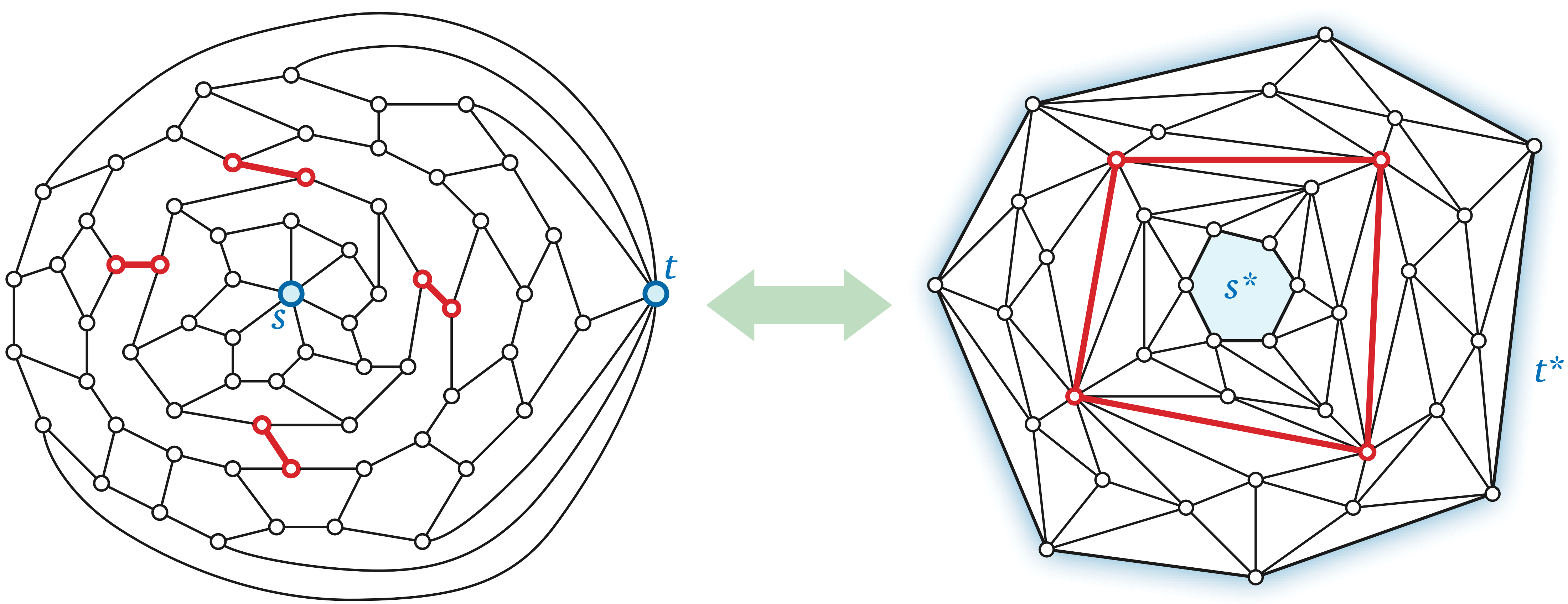
- *Deleting* an edge merges the *faces* on both *sides*
- *Contracting* an edge merges the *vertices* at both *ends*



Cycles ⇔ Cuts

[Whitney 1932]

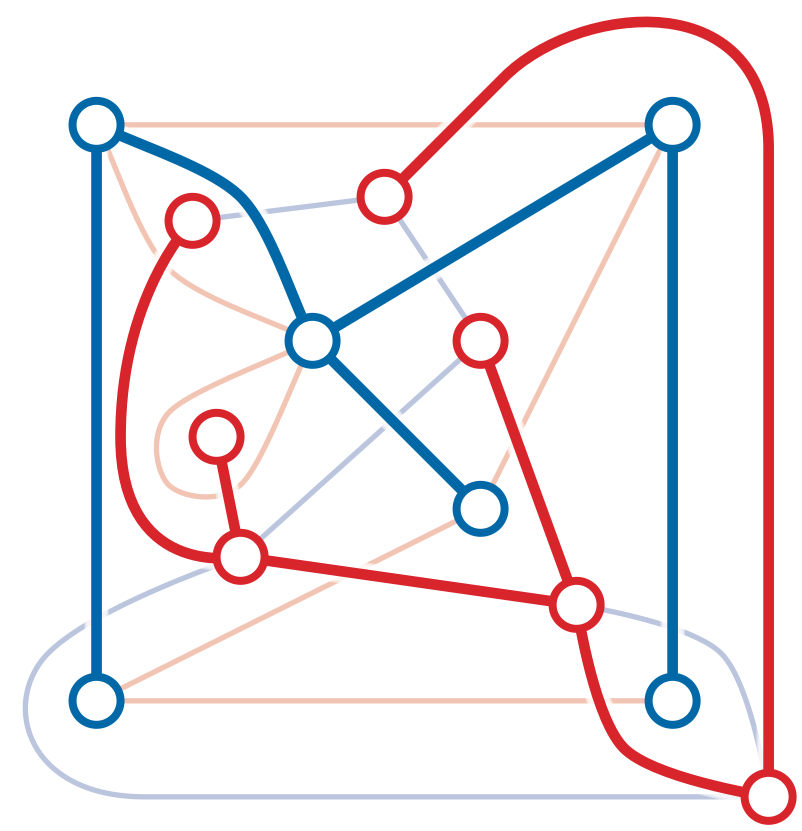
- A *cut* separates the *vertices* on one side from the other
- A *cycle* separates the *faces* on one side from the other
  - …by the Jordan curve theorem



Tree-cotree decomposition

[Von Staudt 1847]

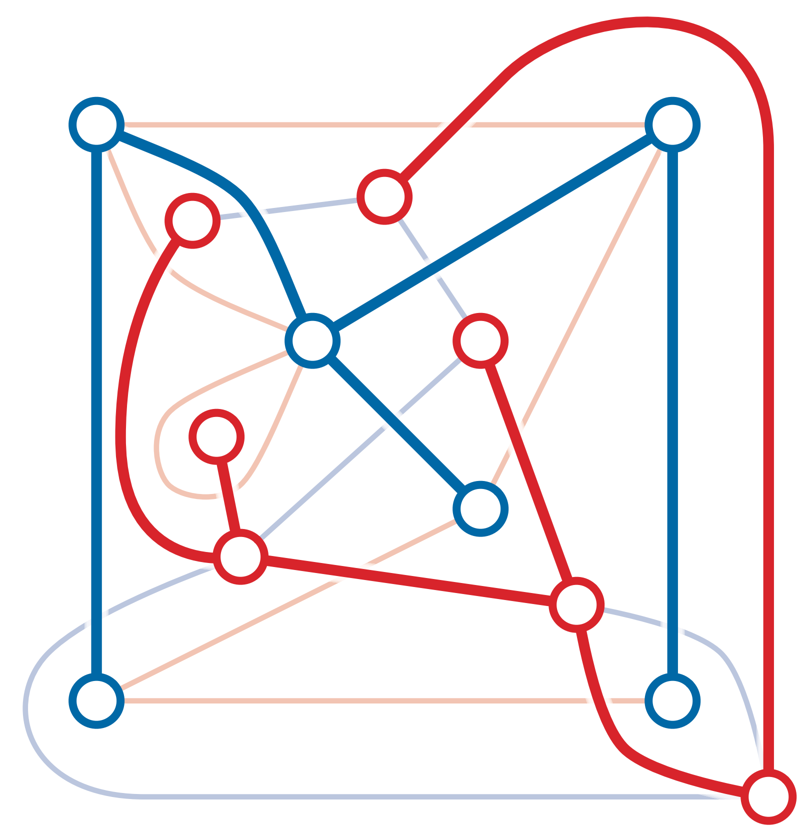
- Let T be an arbitrary spanning tree of G.
- Then $C^* = (G \setminus T)^*$ is a spanning tree of G^*.
  - T is connected $\Rightarrow$ C^* is acyclic
  - T is acyclic $\Rightarrow$ C^* is connected



Tree-cotree decomposition

[Von Staudt 1847]

- Let C^* be an arbitrary spanning tree of G^*.
- Then $T = (G^* \setminus C^*)^*$ is a spanning tree of G.
  - C^* is connected $\Rightarrow$ T is acyclic
  - C^* is acyclic $\Rightarrow$ T is connected



Euler's Formula

- For every planar graph with V vertices, E edges, and F faces:

$$\boxed{V - E + F = 2}$$

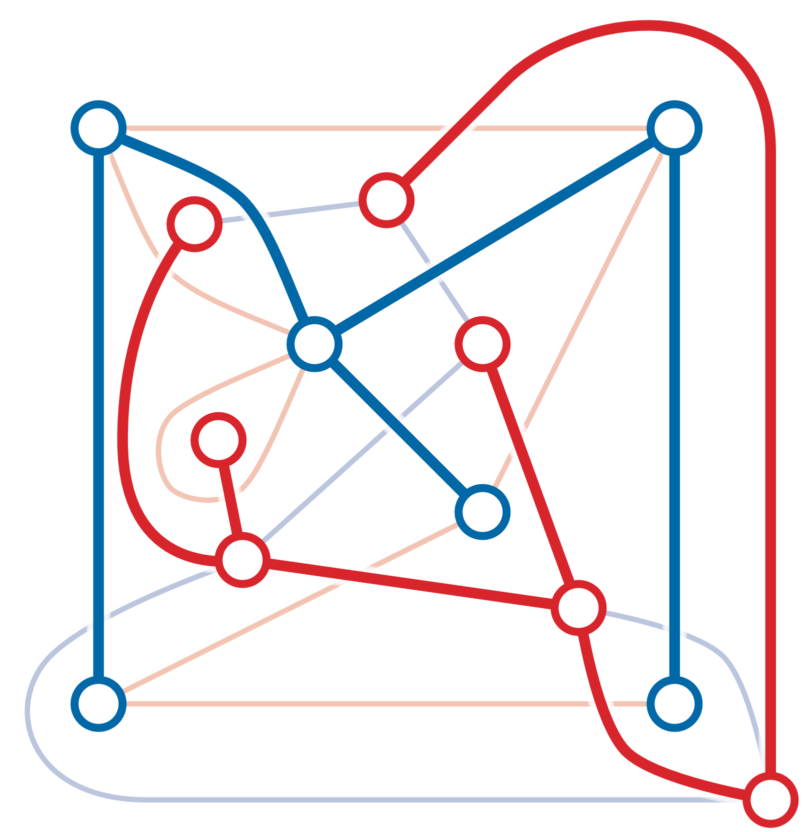
Euler's Formula

- For every planar graph with V vertices, E edges, and F faces:

$$\boxed{V - E + F = 2}$$

Proof:

- Fix a tree-cotree decomposition (T, C).
- T has $V-1$ edges.
- C^* has $F-1$ edges.
- Thus, $E = (V-1) + (F-1)$.



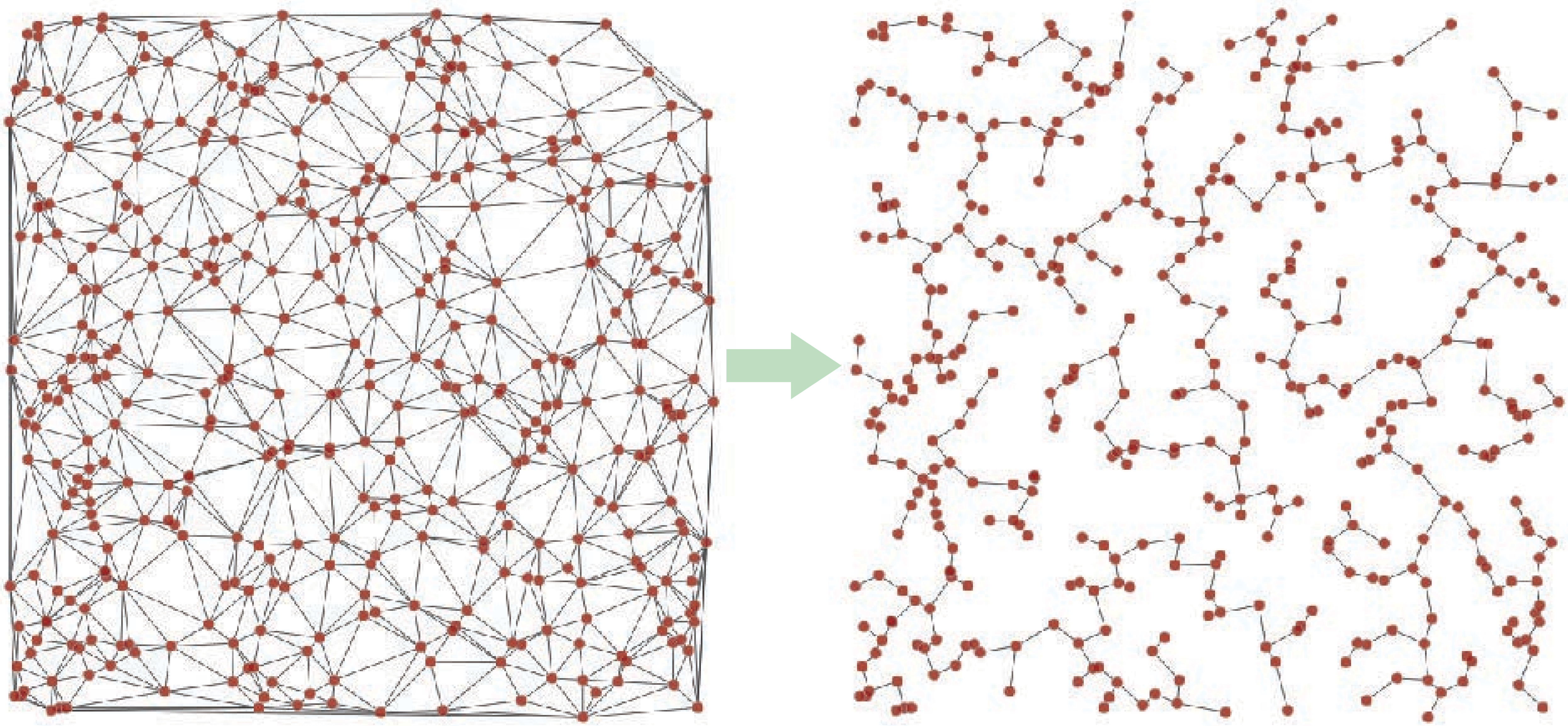
[Von Staudt 1847]

Easy Consequences

- For any planar *triangulation*, $E = 3V-6$ and $F = 2V-4$
  - Every face has exactly three sides
- For any *simple* planar graph, $E \leq 3V-6$ and $F \leq 2V-4$
  - No loops or parallel edges
  - Subgraph of a triangulation
- Every simple planar graph has a vertex of degree ≤ 5
  - ... *at least 4* vertices of degree ≤ 5
  - ... *either* a vertex of degree ≤ 3 *or* a face of degree ≤ 3

Please ask questions!

Minimum spanning trees



Tarjan's red-blue rule

[Tarjan 1983]

- For each *cycle* in G, color the heaviest edge *red*.
- For each *cut* in G, color the lightest edge *blue*.
- Every edge of G is either *red* or *blue* but not both.
- The *blue* edges define the *minimum* spanning tree of *G*.

Tarjan's red-blue algorithm

[Tarjan 1983]

- While G has at least one edge, either *delete* any *red* edge or *contract* any *blue* edge.
- *Contracted* edges = minimum spanning tree of G.

Textbook MST algorithms

[Tarjan 1983]

- ***Borůvka:*** *Contract* lightest edge incident to each vertex, *delete* heavy parallel edges, and (if any edges left) recurse.

- ***Jarník/Prim:*** Fix a vertex v. *Contract* lightest edge incident to v, *delete* heavy parallel edges, and (if edges left) recurse.

- ***Kruskal:*** For all edges in increasing weight order: If the current edge is a loop, *delete* it; otherwise, *contract* it.

...in planar graphs

- For each cycle in *G*, color the heaviest edge *red*.
- For each cut in *G*, color the lightest edge *blue*.
- The *blue* edges define the *minimum* spanning tree of *G*.

- For each cut in *G**, the heaviest edge is *red*.
- For each cycle in *G**, the lightest edge is *blue*.
- The *red* edges define the *maximum* spanning tree of *G**.

Tarjan's red-blue algorithm

- While G has at least one edge, either *delete* any *red* edge or *contract* any *blue* edge.
- *Contracted* edges = minimum spanning tree of G.
- *Deleted* edges = maximum spanning tree of G^*.

Borůvka's algorithm

[Borůvka 1928]

- While G has at least one edge
  - For each vertex v
    - *Contract* the lightest edge incident to v (which must be *blue*)
  - *Delete* any heavy parallel edges (which must be *red*)

- All *contractions* in the first iteration take $O(E) = O(V)$ time
- All *deletions* in the first iteration take $O(E) = O(V)$ time
- Each iteration removes at least half the vertices.
- So overall running time is $O(V)$

Mareš's algorithm

[Mareš 2006]

Any constant will work here

- While G has at least one edge
  - Pick any vertex *v* with degree at most 5
  - *Contract* the lightest edge incident to *v* (which must be *blue*)
  - *Delete* any heavy parallel edges (which must be *red*)

- Each iteration takes $O(1)$ time, so total running time is $O(V)$.

Matsui's algorithm

[Matsui 1995]

- While G has at least one edge
  - If G has a vertex v of degree at most 3
    - If v is incident to a loop, delete the loop
    - Else contract the lightest edge incident to v
  - If G has a face f with degree at most 3
    - If f is incident to a bridge, contract the bridge
    - Else delete the heaviest edge incident to f
- Each iteration takes $O(1)$ time, so total running time is $O(V)$.

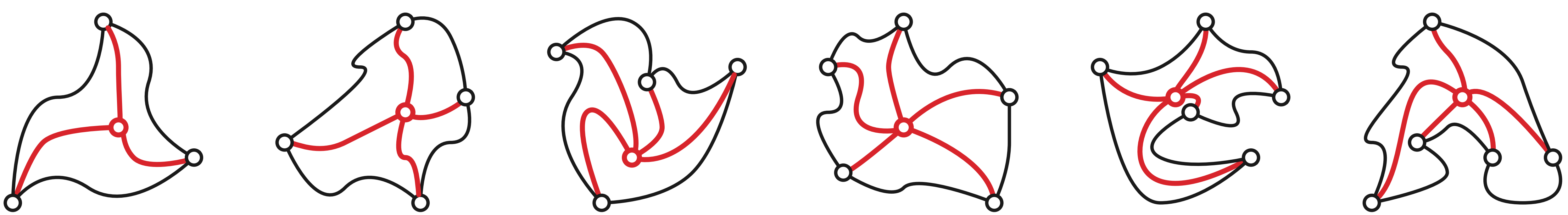
Please ask questions!

Straight-line embedding

- Every planar embedding of a *simple* planar graph G is equivalent to an embedding where every edge is a straight line segment.
  [Steinitz 1916, Wagner 1936, Fáry 1948, Stein 1951, Stojaković 1959]

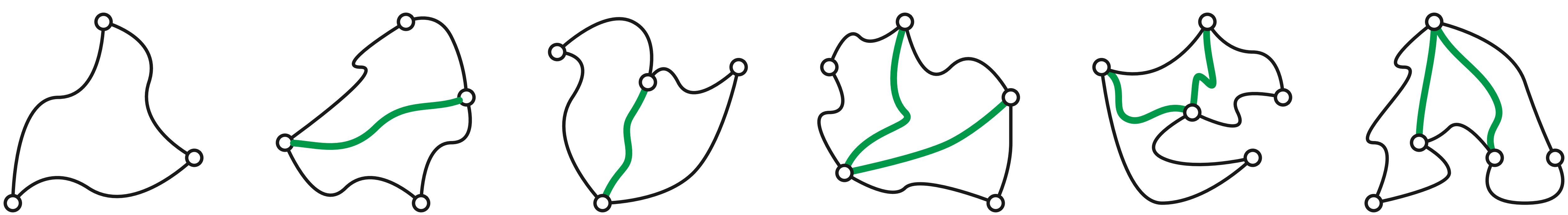
Straight-line embedding

- Suppose G is a triangulation.
- Delete any vertex of degree ≤ 5 and retriangulate the hole
- Recursively embed the remaining graph
  - Base case: Embed 3-cycle as a triangle
- Reinsert vertex in resulting polygonal hole



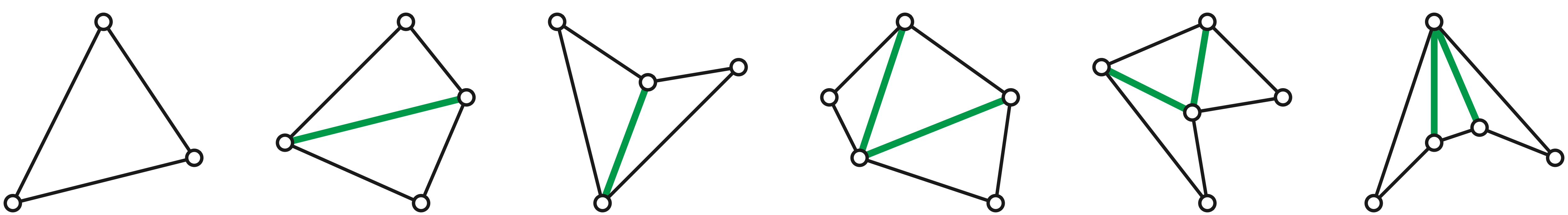
Straight-line embedding

- Suppose G is a triangulation.
- Delete any vertex of degree ≤ 5 and retriangulate the hole
- Recursively embed the remaining graph
  - Base case: Embed 3-cycle as a triangle
- Reinsert vertex in resulting polygonal hole



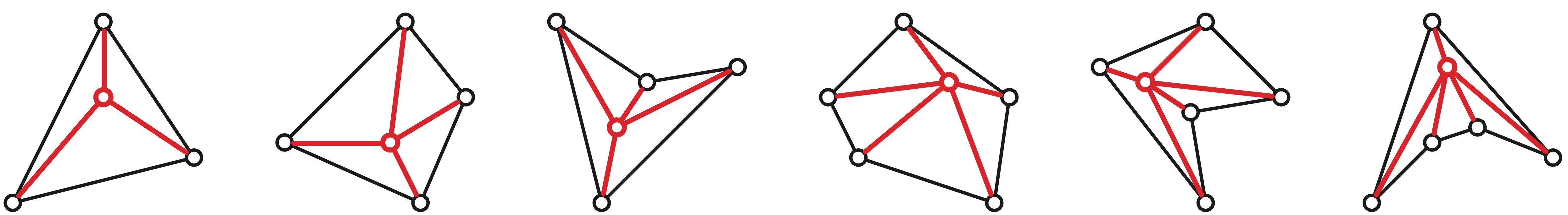
Straight-line embedding

- Suppose G is a triangulation.
- Delete any vertex of degree ≤ 5 and retriangulate the hole
- Recursively embed the remaining graph
  - Base case: Embed 3-cycle as a triangle
- Reinsert vertex in resulting polygonal hole



Straight-line embedding

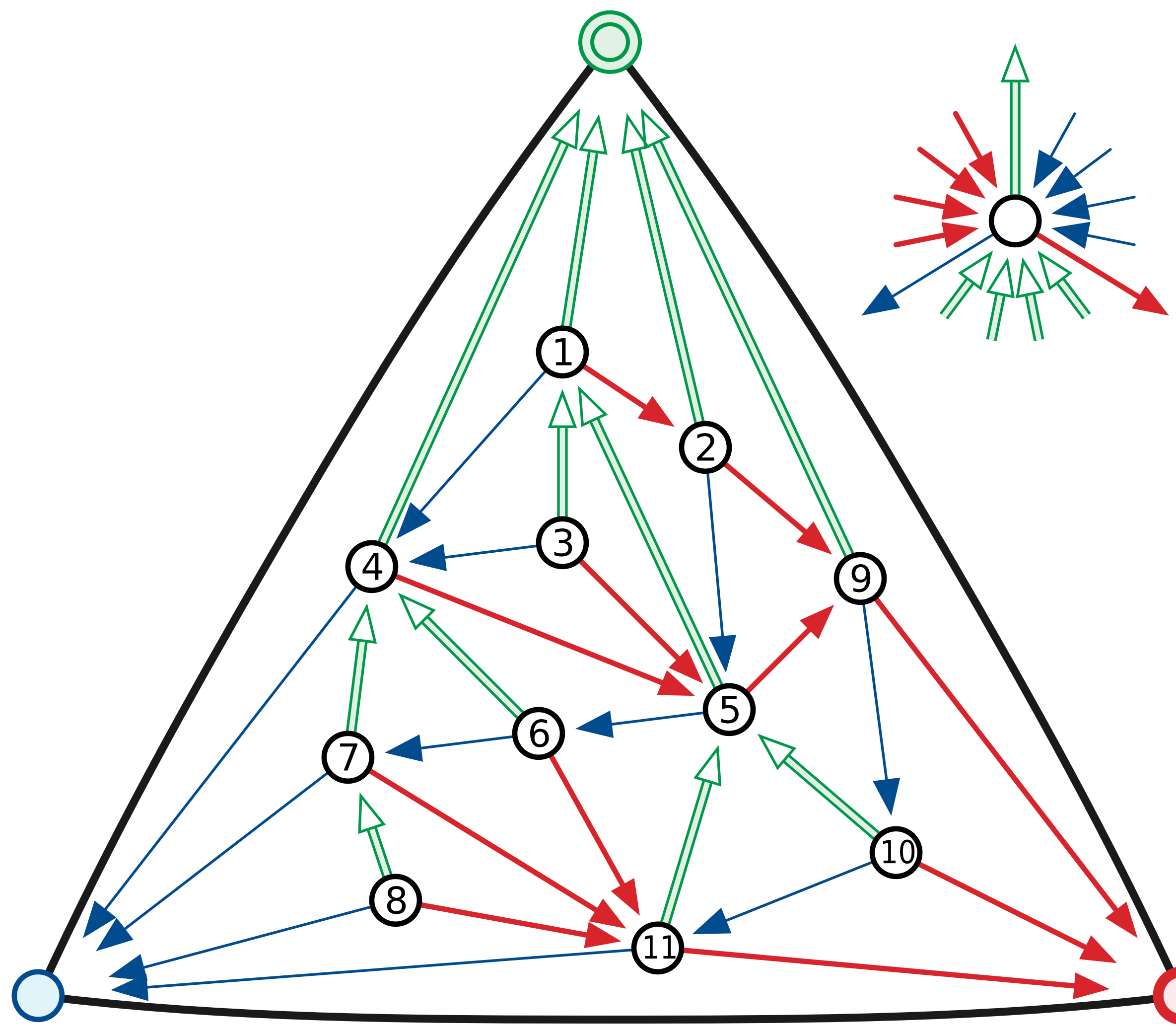
- Suppose G is a triangulation.
- Delete any vertex of degree ≤ 5 and retriangulate the hole
- Recursively embed the remaining graph
  - Base case: Embed 3-cycle as a triangle
- Reinsert vertex in resulting polygonal hole



Schnyder wood

[Schnyder 1990]

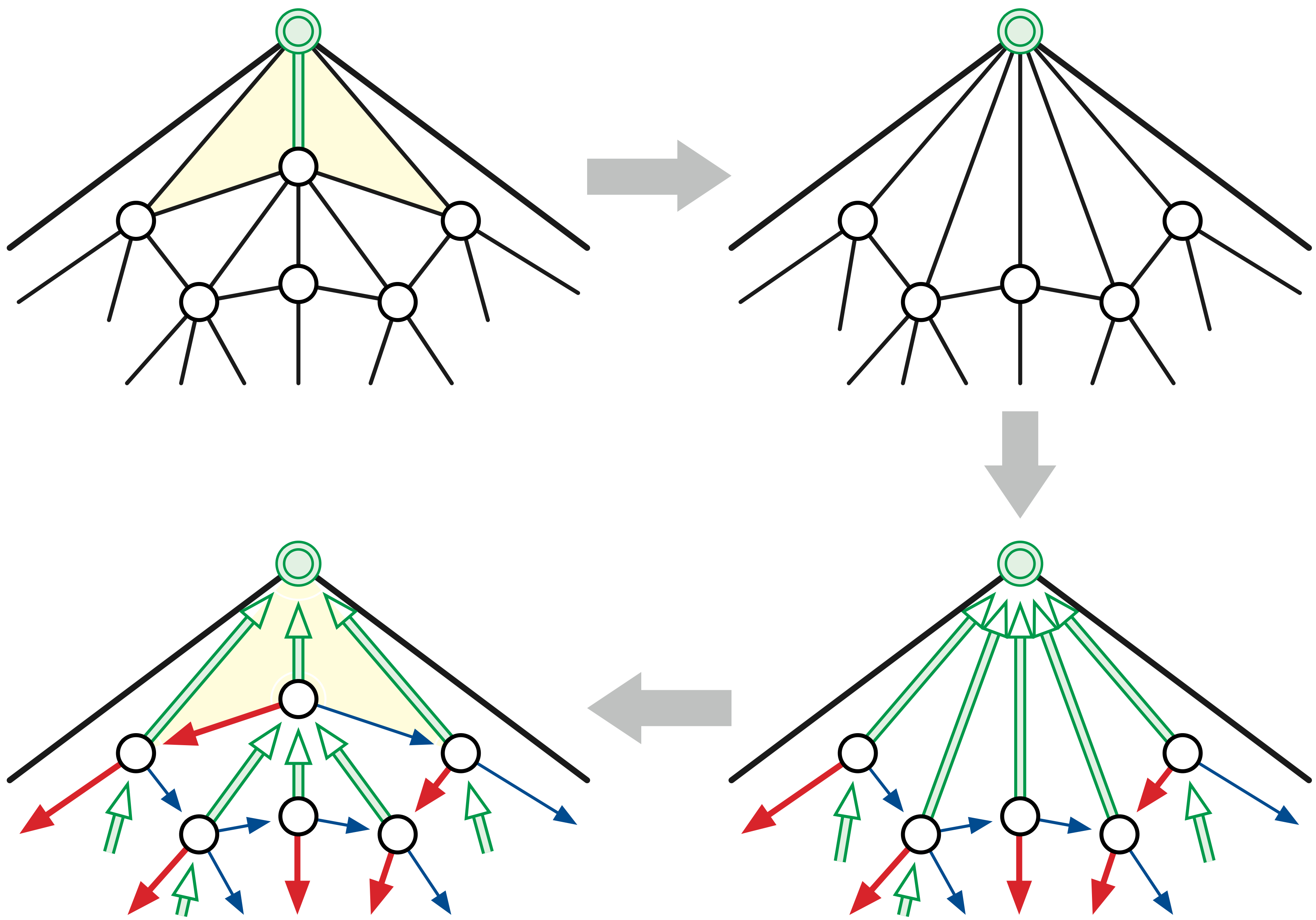
- WLOG suppose G is a *triangulation*
- Color the outer vertices red, green, and blue
- Direct and color the interior edges as follows:



Schnyder wood

[Schnyder 1990]

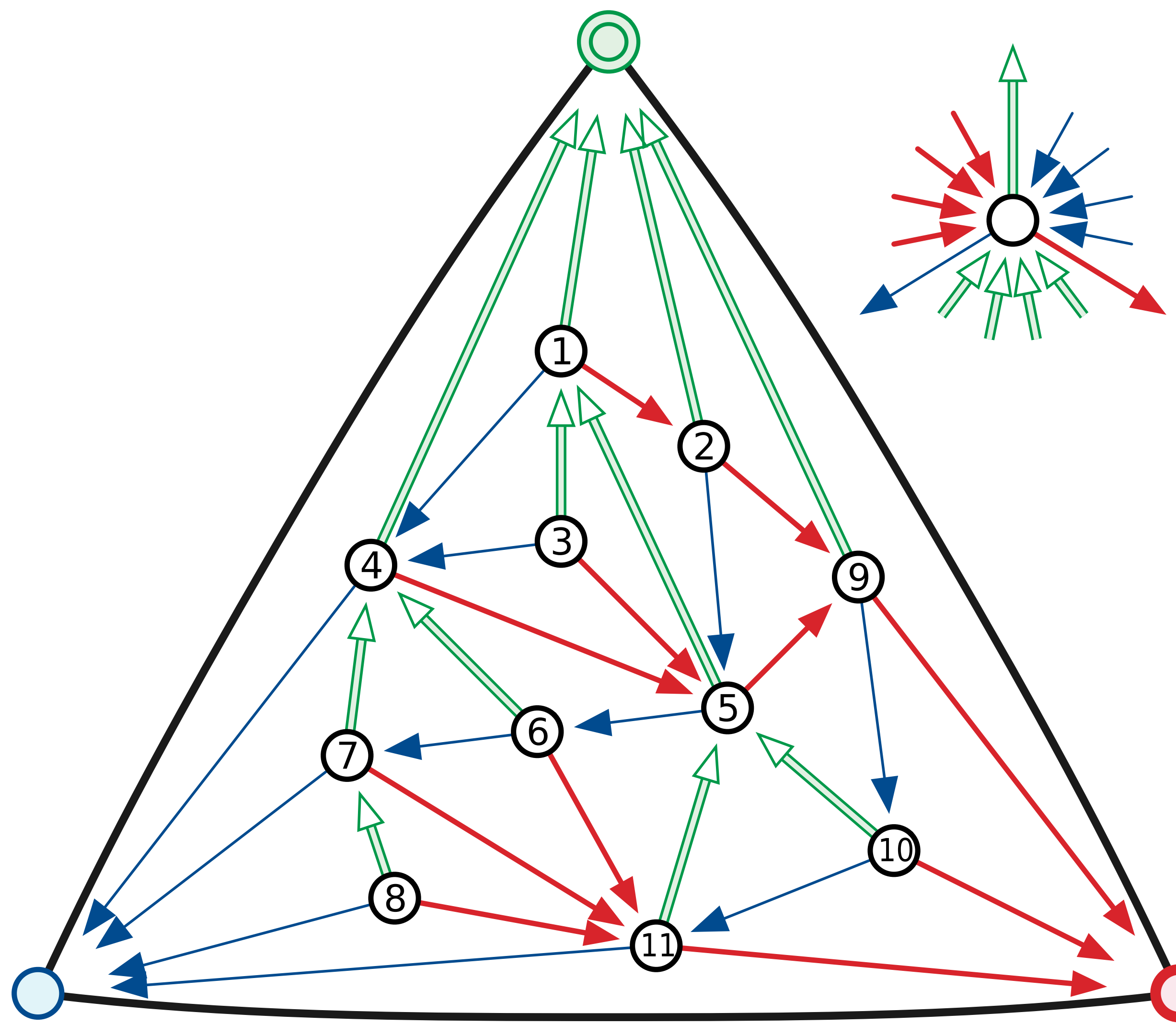
- Recursive construction:



Schnyder wood

[Schnyder 1990]

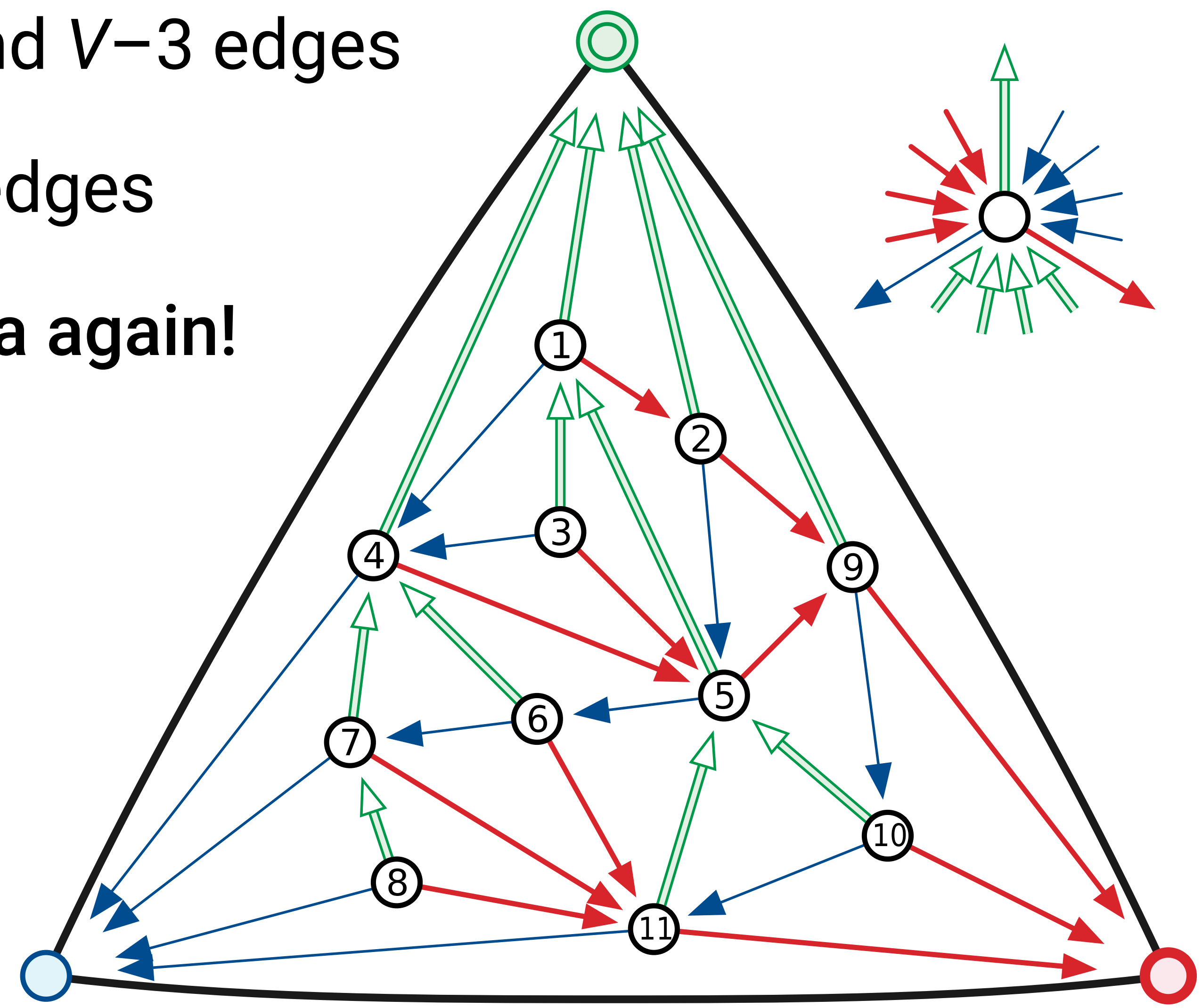
- The edges of each color induce a *directed spanning tree* of the interior vertices, rooted at the outer vertex of that color.



Schnyder wood

[Schnyder 1990]

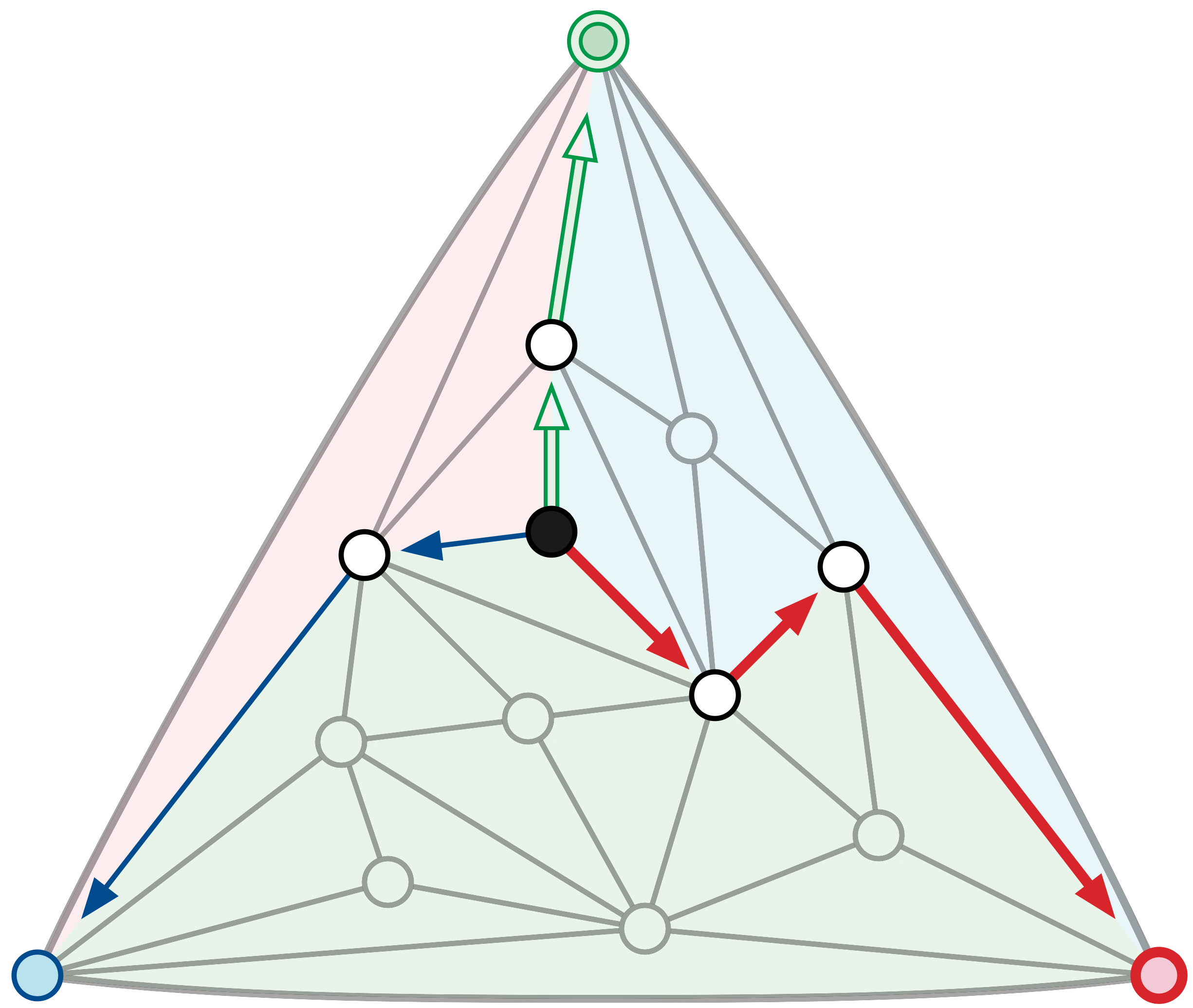
- The edges of each color induce a *directed spanning tree* of the interior vertices, rooted at the outer vertex of that color.
  - Each tree has $V-2$ vertices and $V-3$ edges
  - So G has $3(V-3) + 3 = 3V-6$ edges
  - **We just proved Euler's formula again!**



Schnyder coordinates

[Schnyder 1990]

- Directed paths from each vertex partition the faces of G

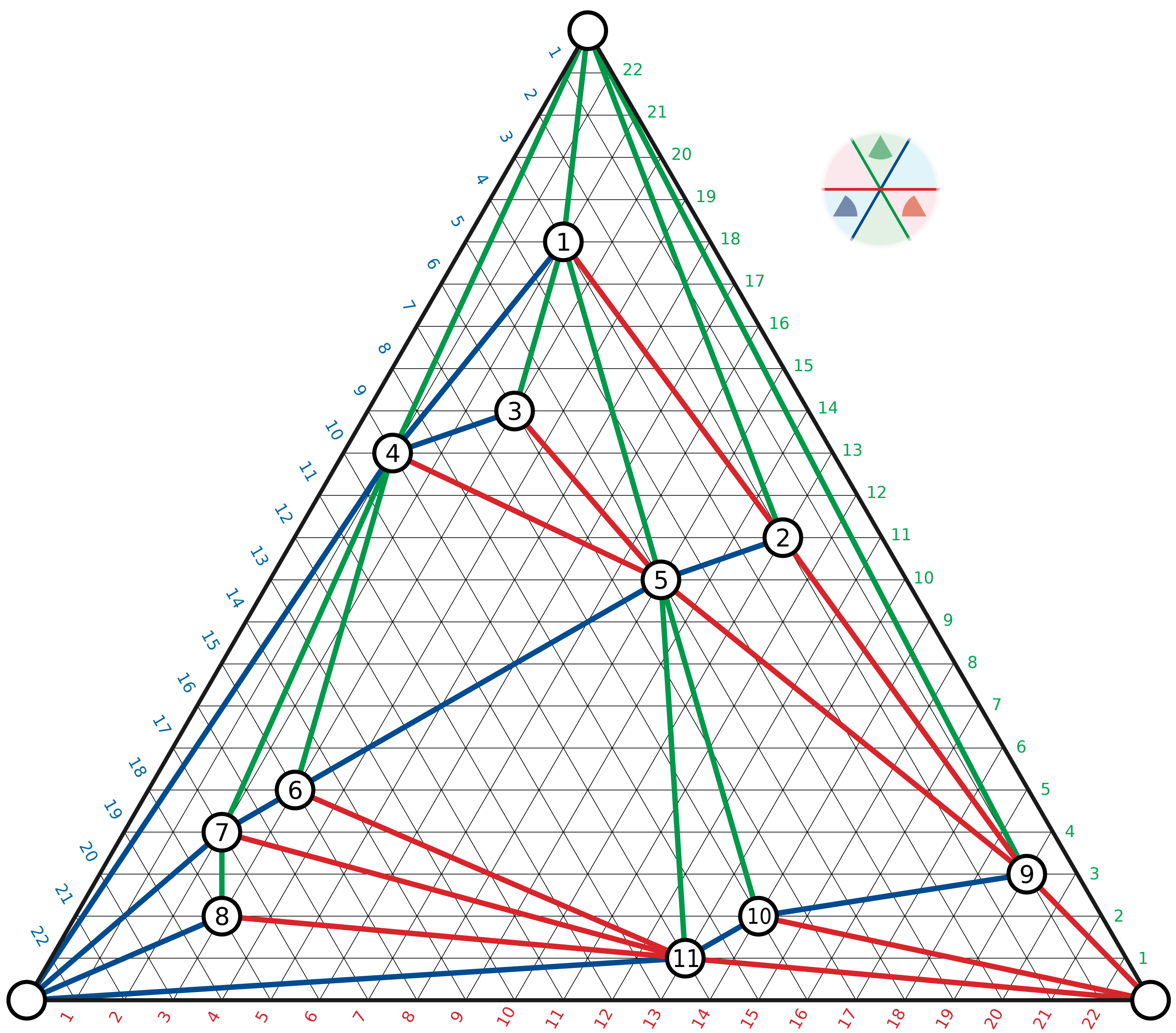


(3, 14, 6)

Grid embedding

[Schnyder 1990]

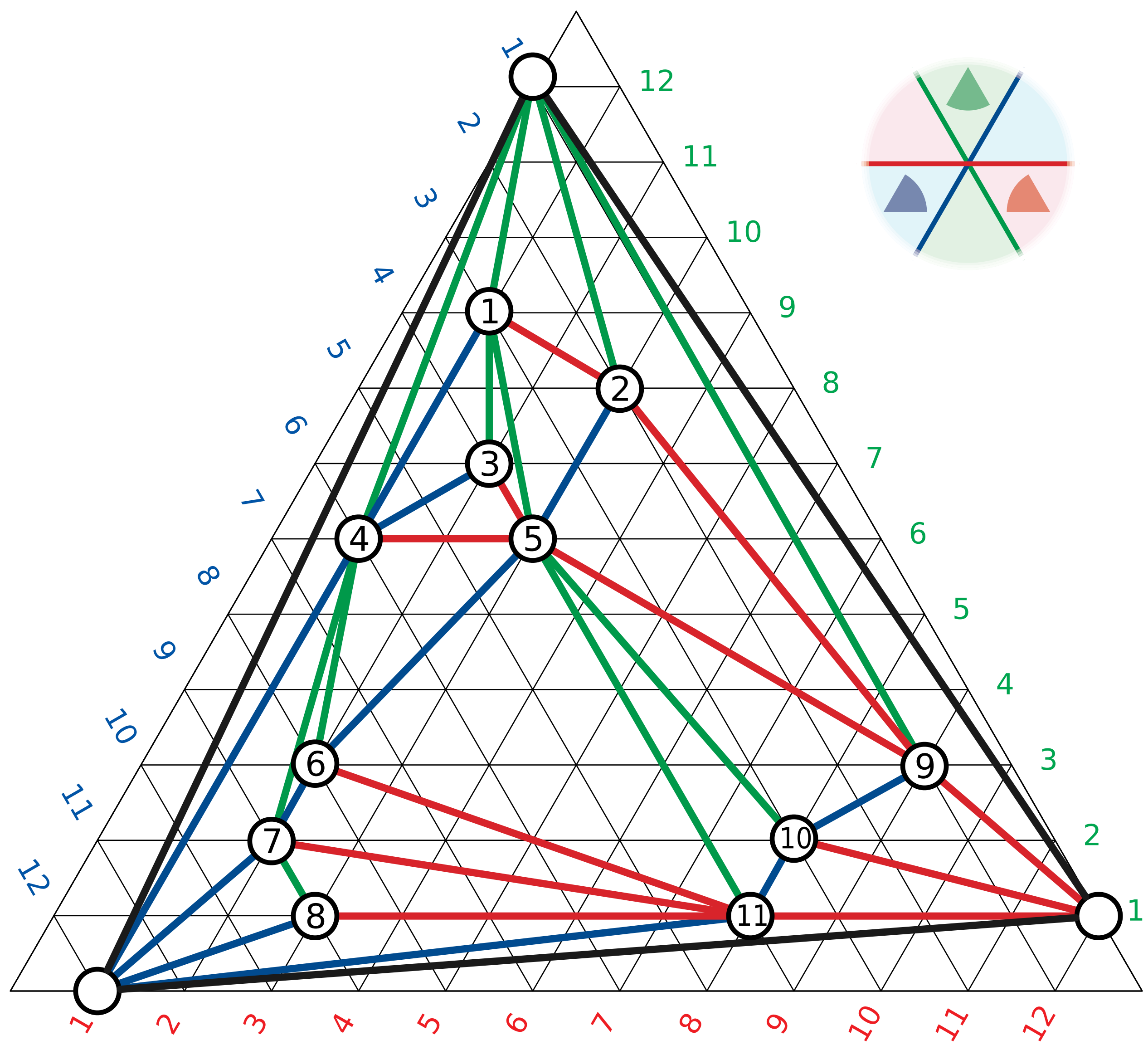
- Interpreting face counts as barycentric coordinates gives us a straight-line embedding of G on a $(2n-4)\times(2n-4)$ grid.



Grid embedding

[Schnyder 1990]

- Using a similar partitioning of the vertices, we can embed G on an $(n-1)\times(n-1)$ grid.



Please ask questions!

Separators

- A *separator* of an n-vertex graph G is a subset S of vertices where each component of $G \setminus S$ has at most $2n/3$ vertices.

- Used in ***lots*** of divide-and-conquer algorithms

$$T(n) \le T(n/3) + T(2n/3) + f(n)$$

$$\implies T(n) = \begin{cases} O(n) & \text{if } f(n) = O(n^{1-\varepsilon}) \\ O(n \log n) & \text{if } f(n) = \Theta(n) \\ O(f(n)) & \text{if } f(n) = \Omega(n^{1+\varepsilon}) \end{cases}$$

Easy separators

- *Paths:* One vertex

Easy separators

- ***Paths:*** One vertex
- ***Cycles:*** Two vertices

Easy separators

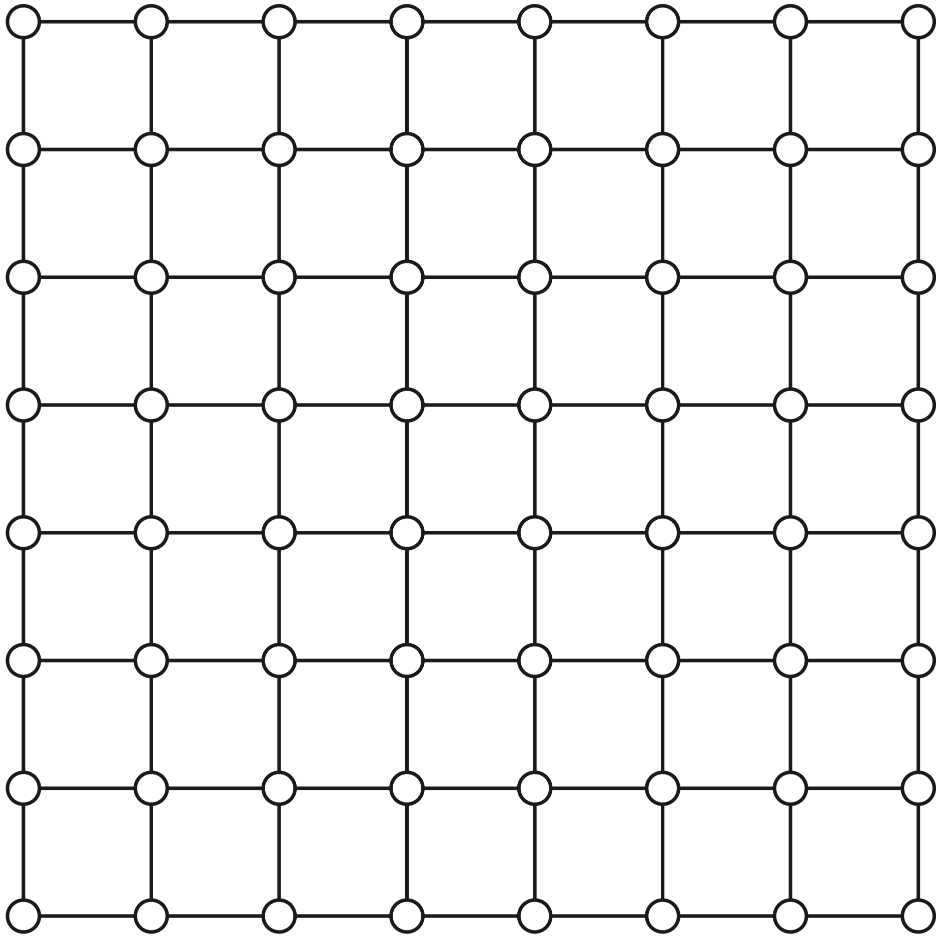
- ***Paths:*** One vertex
- ***Cycles:*** Two vertices
- ***Trees:*** One vertex *[Jordan 1869]*

Easy separators

- ***Paths:*** One vertex
- ***Cycles:*** Two vertices
- ***Trees:*** One vertex *[Jordan 1869]*
- ***Outer-planar graphs:*** Two vertices

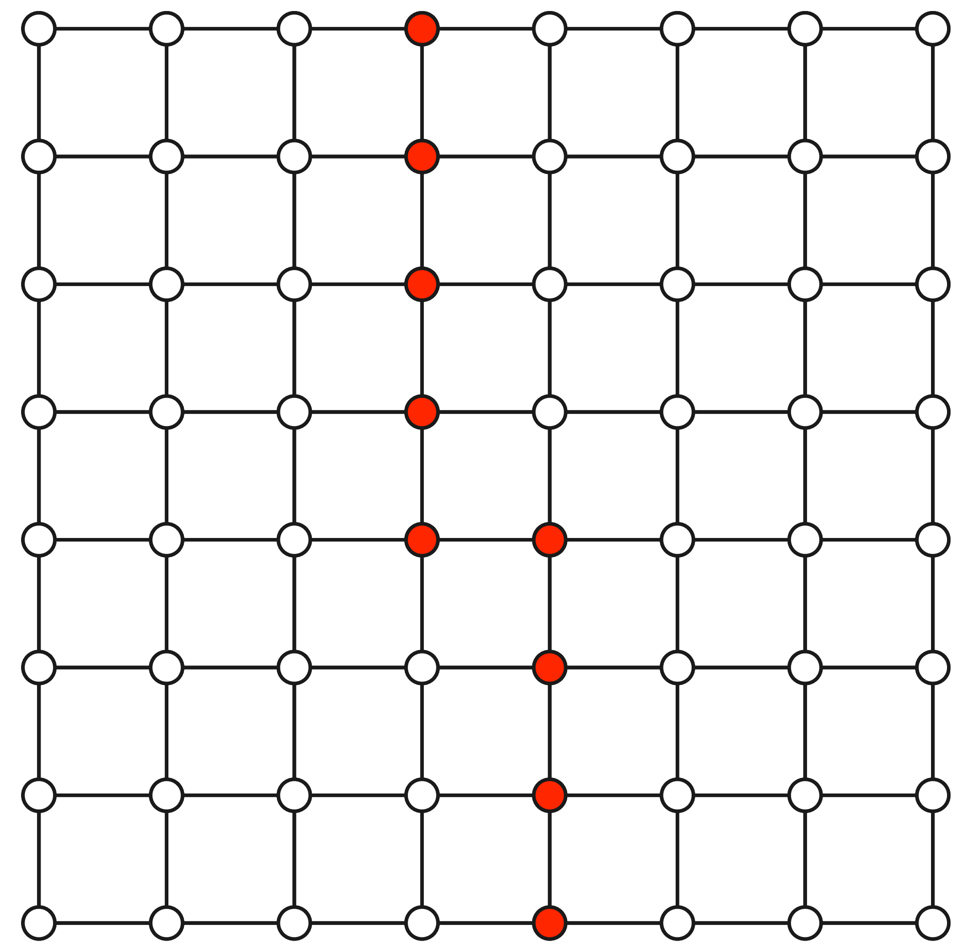
Easy separators

- ***Paths:*** One vertex
- ***Cycles:*** Two vertices
- ***Trees:*** One vertex [Jordan 1869]
- ***Outer-planar graphs:*** Two vertices
- ***$\sqrt{n} \times \sqrt{n}$ grid:*** $\sqrt{n}$ vertices



Easy separators

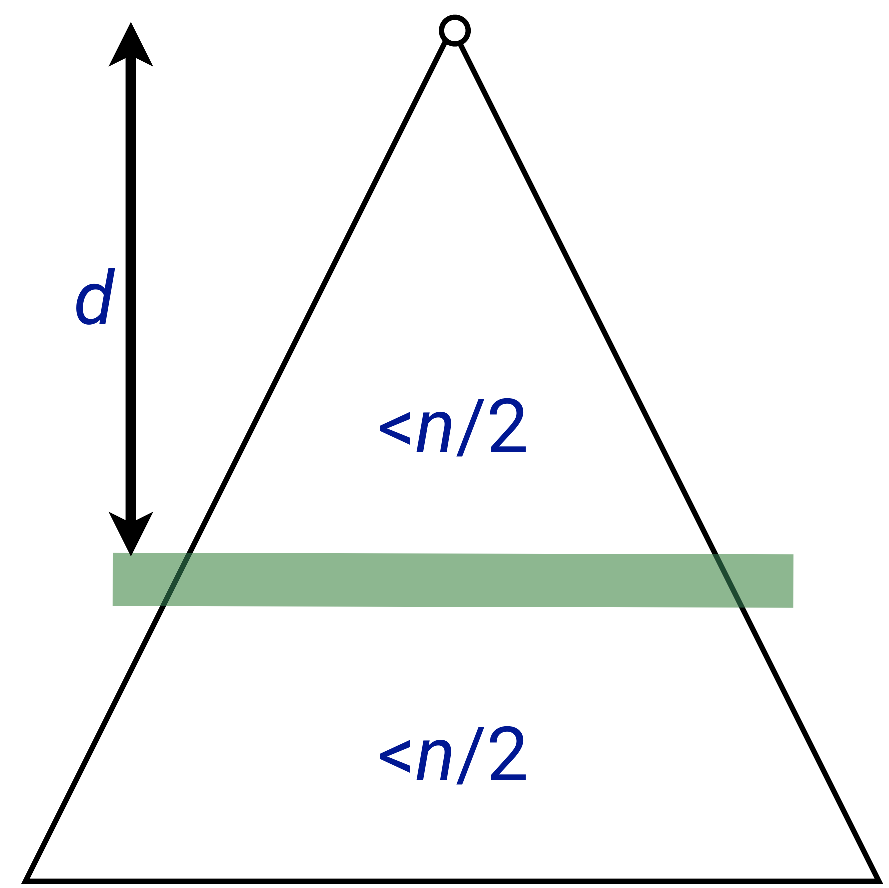
- ***Paths:*** One vertex
- ***Cycles:*** Two vertices
- ***Trees:*** One vertex *[Jordan 1869]*
- ***Outer-planar graphs:*** Two vertices
- ***√n×√n grid:*** √*n* vertices



Median BFS level

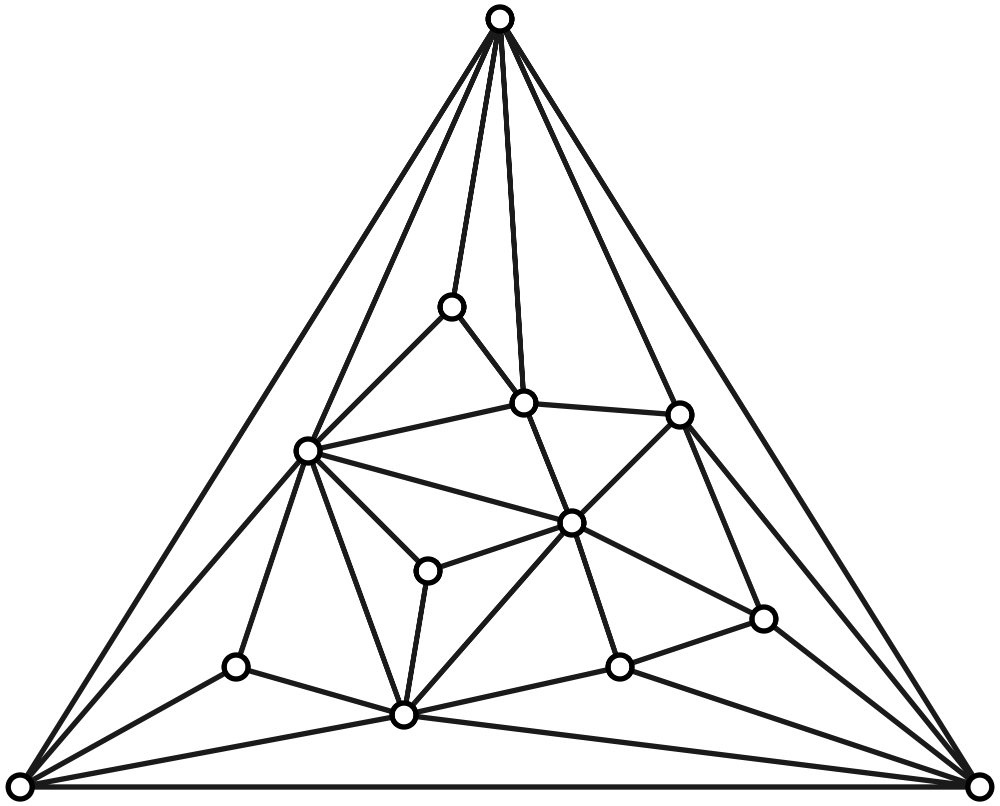
- Let T be a breadth-first search tree rooted at any vertex.
- Let V_d be the set of vertices with depth d.
- Let $V_{<d}$ be the set of vertices with depth less than d.
- Let d be the largest integer such that $|V_{<d}| < 1/2$.
- Then V_d is a separator.

- Unfortunately, V_d may be very large.



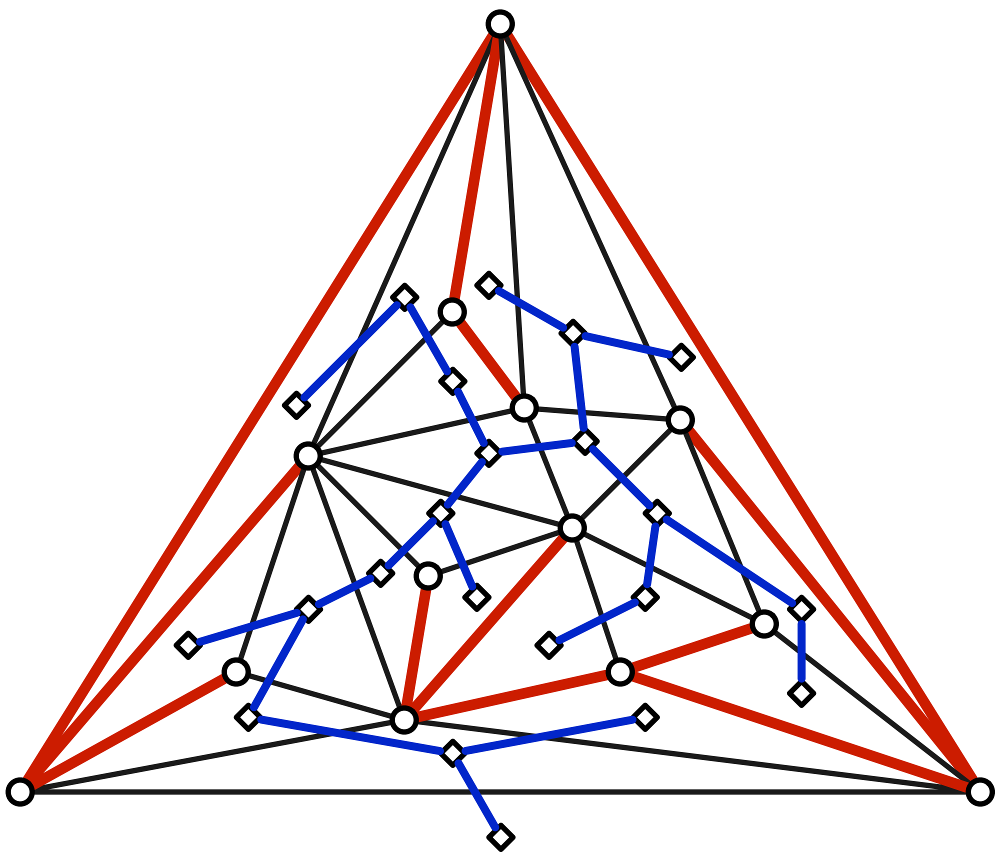
Fundamental cycle separator

- Suppose G is a planar *triangulation*.



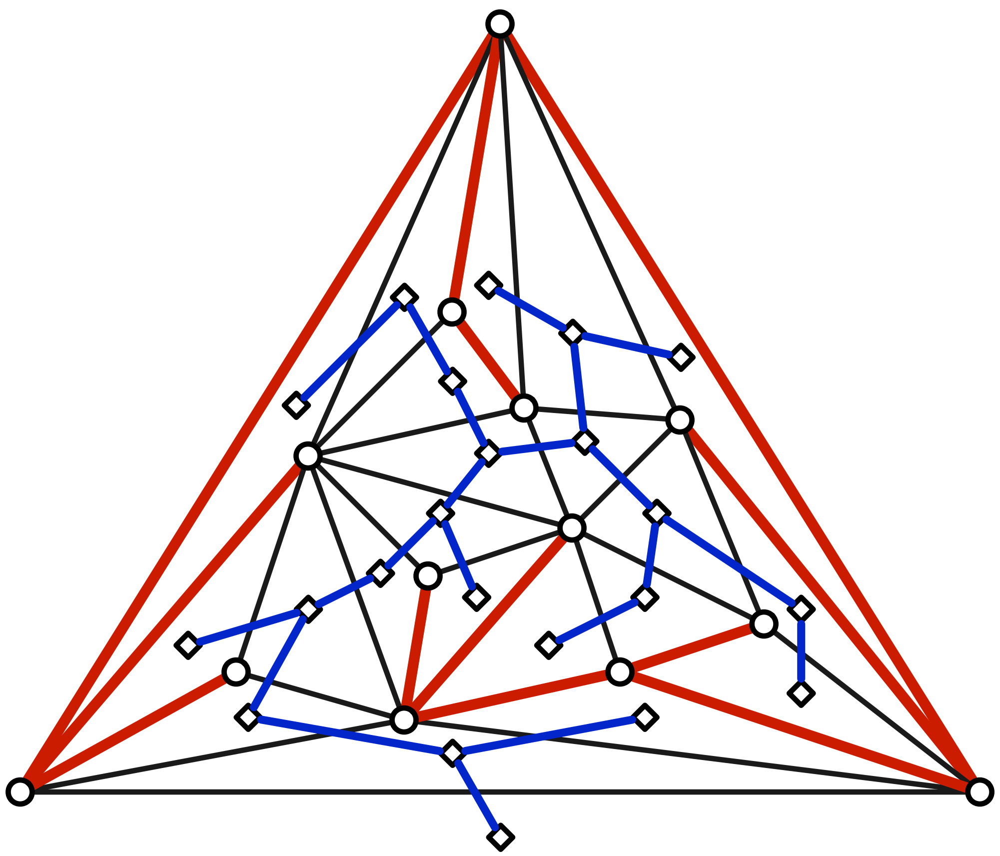
Fundamental cycle separator

- Suppose G is a planar *triangulation*.
- Let (T, C) be an arbitrary tree-cotree decomposition.



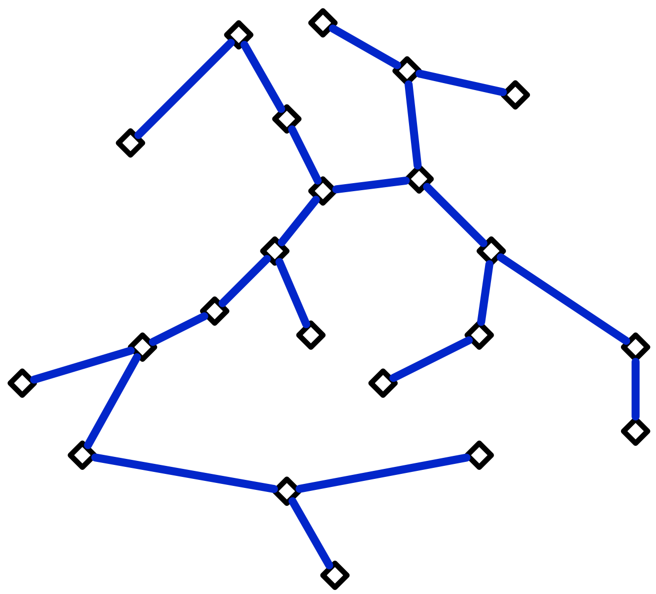
Fundamental cycle separator

- Suppose G is a planar *triangulation*.
- Let (T, C) be an arbitrary tree-cotree decomposition.
- Each vertex in C^* has degree ≤ 3.



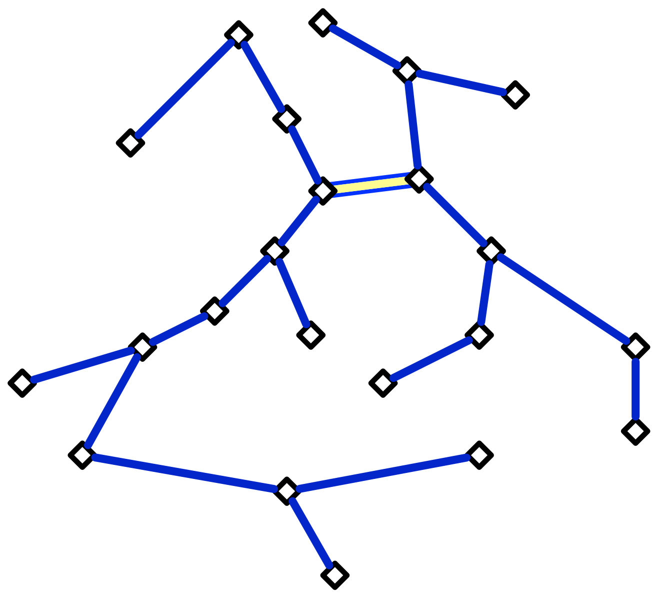
Fundamental cycle separator

- Suppose G is a planar *triangulation*.
- Let (T, C) be an arbitrary tree-cotree decomposition.
- Each vertex in C^* has degree ≤ 3.



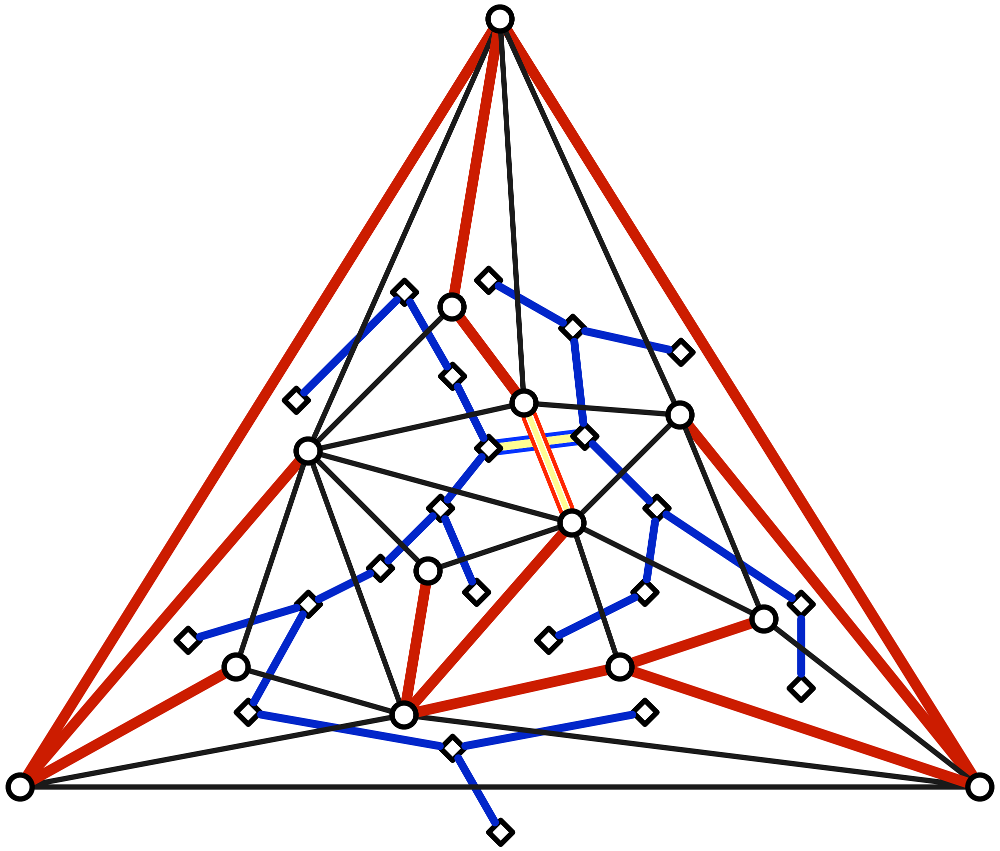
Fundamental cycle separator

- Suppose G is a planar *triangulation*.
- Let (T, C) be an arbitrary tree-cotree decomposition.
- Each vertex in C^* has degree ≤ 3.
- There is an edge e such that each component of $C^* \backslash e^*$ has at most $2F/3$ dual vertices.



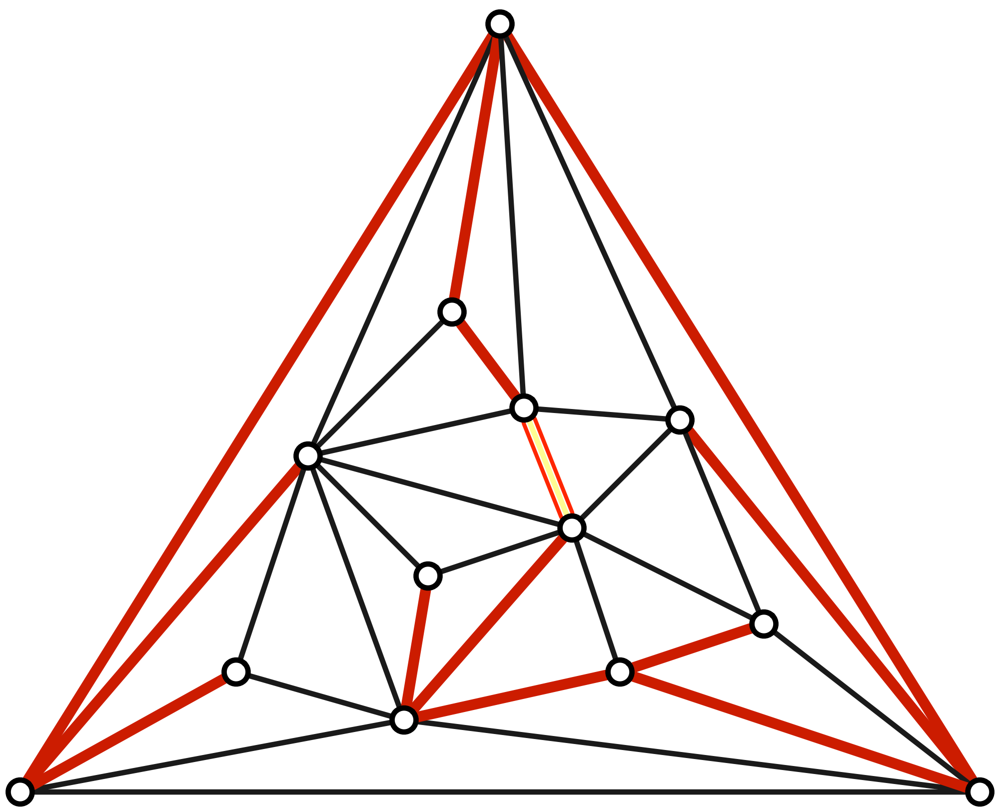
Fundamental cycle separator

- Suppose G is a planar *triangulation*.
- Let (T, C) be an arbitrary tree-cotree decomposition.
- Each vertex in C^* has degree ≤ 3.
- There is an edge e such that each component of $C^* \backslash e^*$ has at most $2F/3$ dual vertices.
- Then $cycle(T,e)$ is a (dual) separator of size $\leq 2\ depth(T) + 1$.



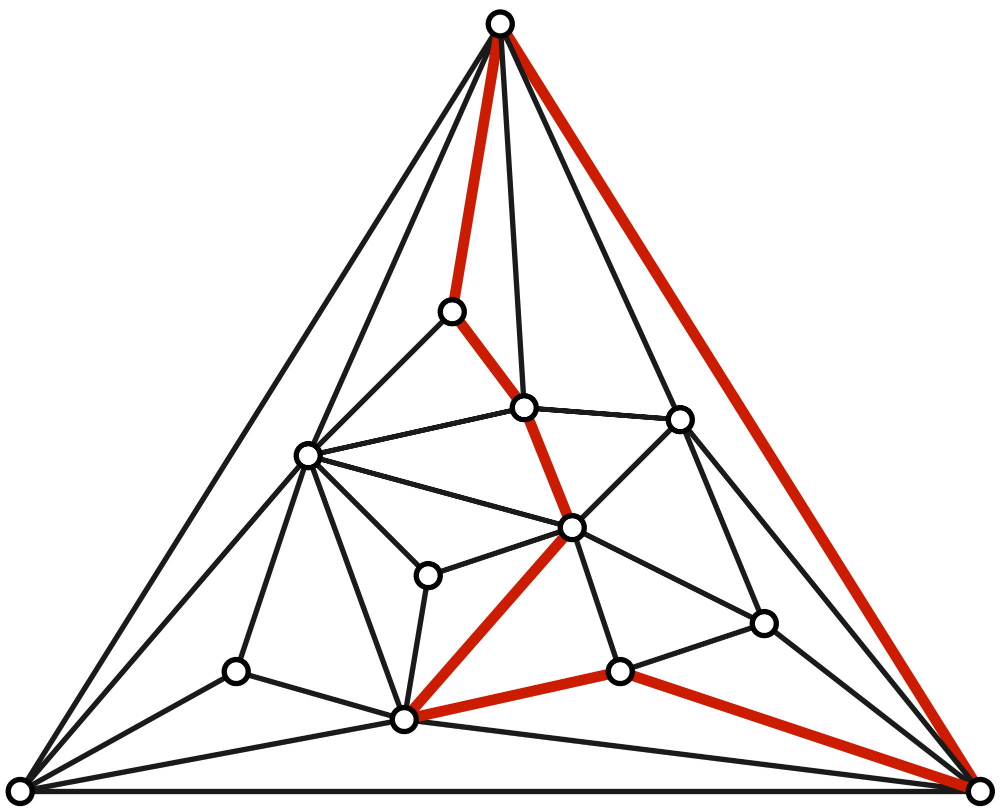
Fundamental cycle separator

- Suppose G is a planar *triangulation*.
- Let (T, C) be an arbitrary tree-cotree decomposition.
- Each vertex in C^* has degree ≤ 3.
- There is an edge e such that each component of $C^* \backslash e^*$ has at most $2F/3$ dual vertices.
- Then $cycle(T,e)$ is a (dual) separator of size $\leq 2\ depth(T) + 1$.



Fundamental cycle separator

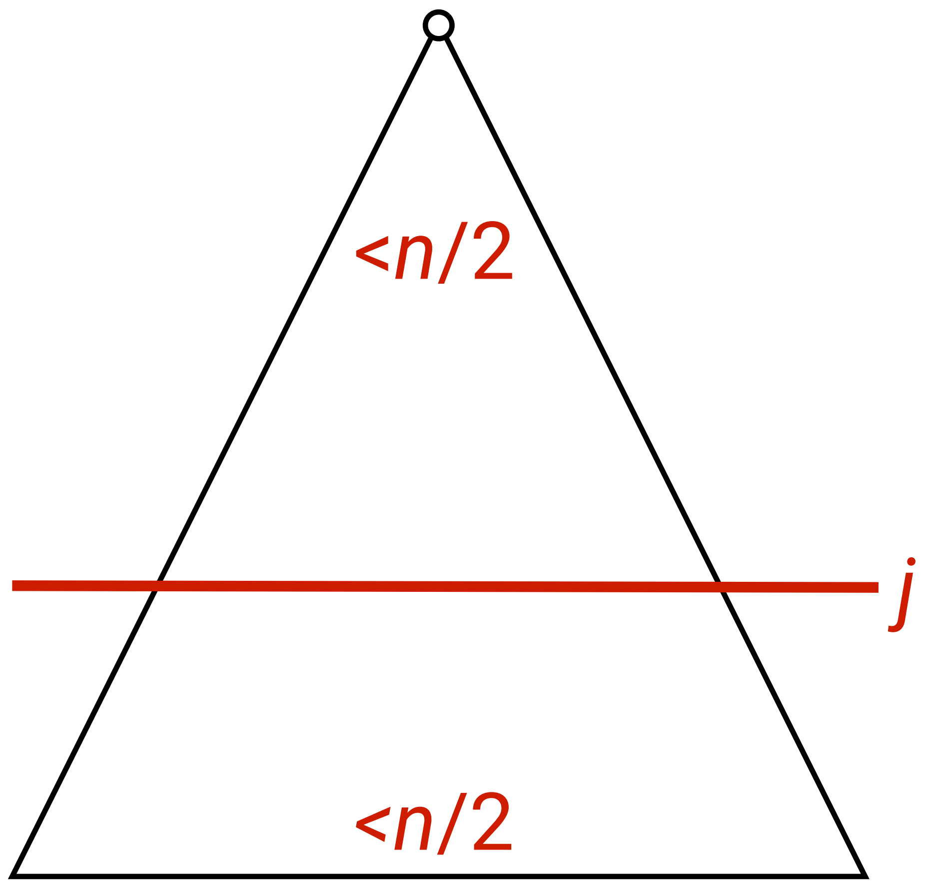
- Suppose G is a planar *triangulation*.
- Let (T, C) be an arbitrary tree-cotree decomposition.
- Each vertex in C^* has degree ≤ 3.
- There is an edge e such that each component of $C^*\backslash e^*$ has at most $2F/3$ dual vertices.
- Then *cycle*(T,e) is a (dual) separator of size ≤ 2 *depth*$(T) + 1$.



Planar separators

[Lipton Tarjan '77]

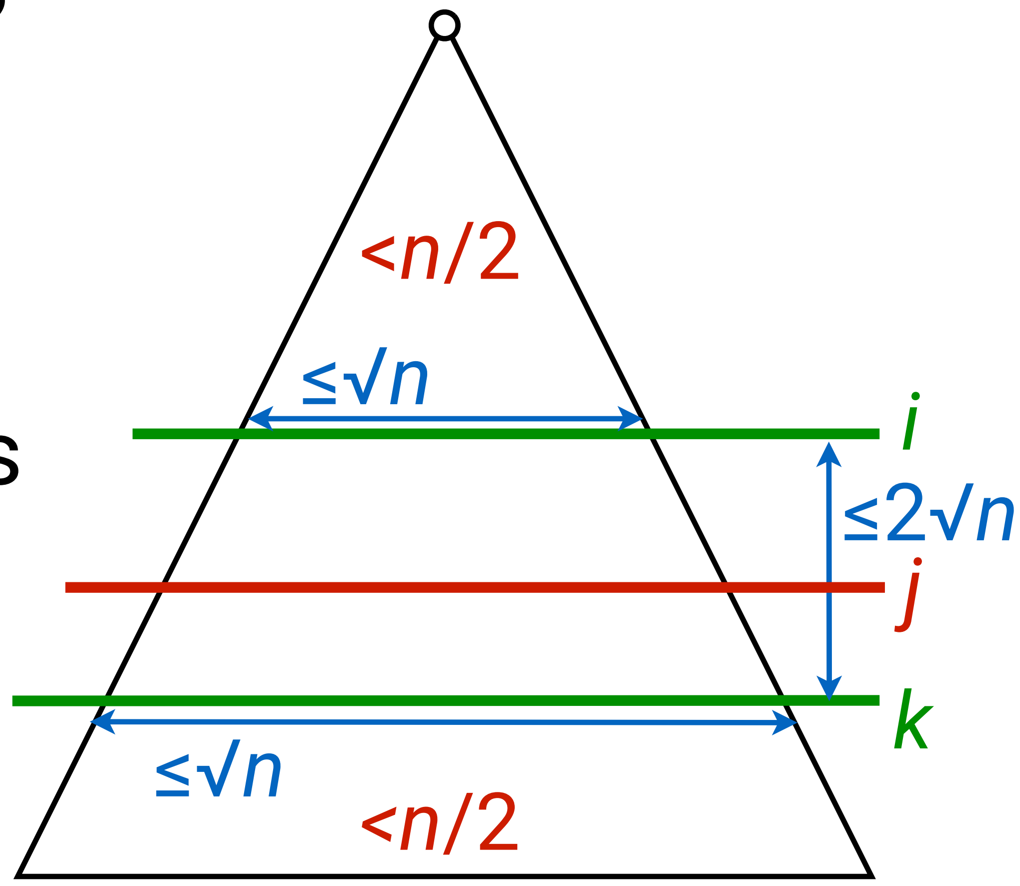
- Suppose G is a planar triangulation; let T be a BFS tree.
- Find median level j.



Planar separators

[Lipton Tarjan '77]

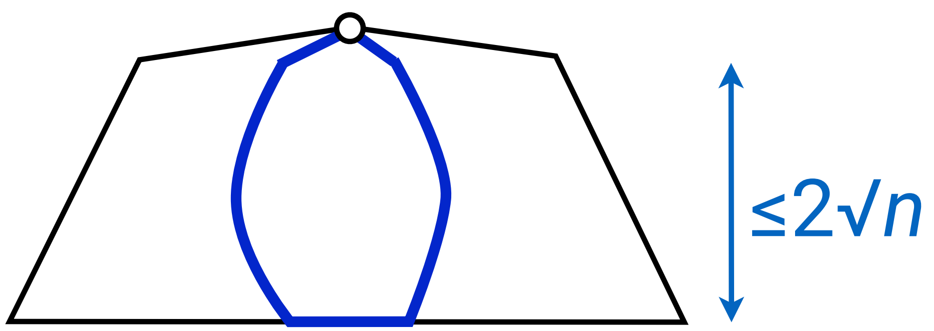
- Suppose G is a planar triangulation; let T be a BFS tree.
- Find median level j.
- Find levels V_i and V_k with $\leq\sqrt{n}$ nodes where $j-\sqrt{n} \leq i < j < k \leq j+\sqrt{n}$.



Planar separators

[Lipton Tarjan '77]

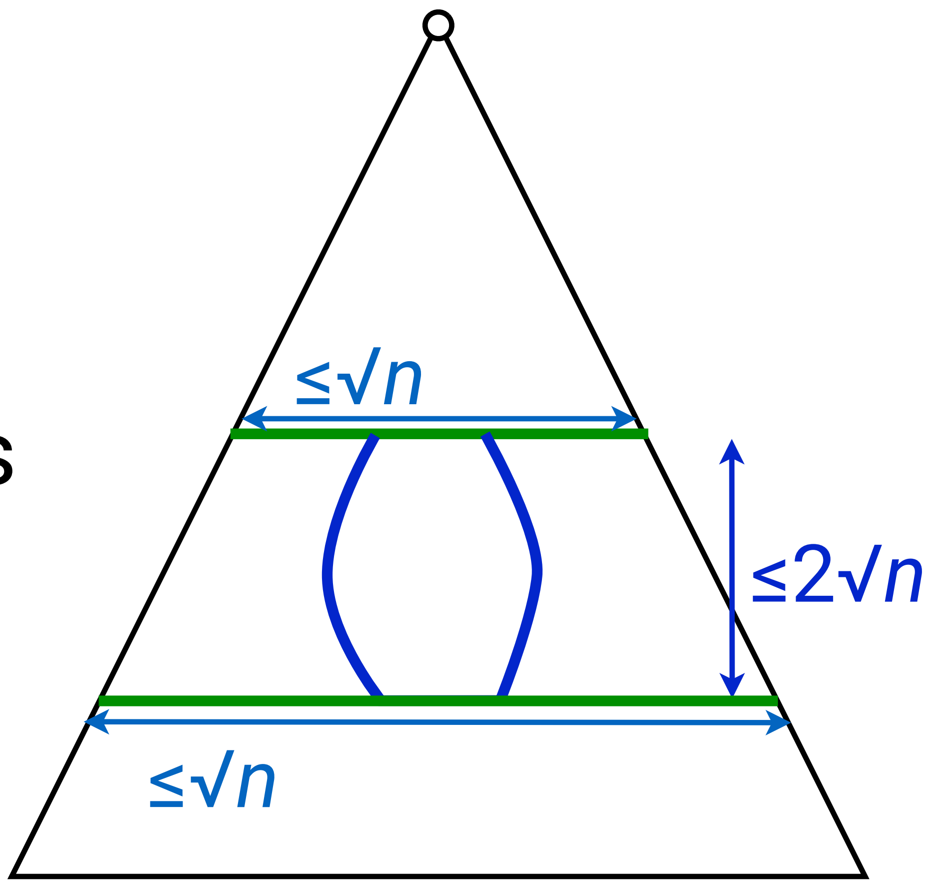
- Suppose G is a planar triangulation; let T be a BFS tree.
- Find median level j.
- Find levels V_i and V_k with $\leq\sqrt{n}$ nodes where $j-\sqrt{n} \leq i < j < k \leq j+\sqrt{n}$.
- *Contract* levels $<i$ and *delete* levels $>k$ to obtain $G(i, k)$.
- S = fund. cycle separator of $G(i, k)$.
- $S \cup V_i \cup V_k$ is a separator of size $O(\sqrt{n})$.



Planar separators

[Lipton Tarjan '77]

- Suppose G is a planar triangulation; let T be a BFS tree.
- Find median level j.
- Find levels V_i and V_k with $\leq\sqrt{n}$ nodes where $j-\sqrt{n} \leq i < j < k \leq j+\sqrt{n}$.
- *Contract* levels $<i$ and *delete* levels $>k$ to obtain $G(i, k]$.
- S = fund. cycle separator of $G(i, k]$.
- $S \cup V_i \cup V_k$ is a separator of size $O(\sqrt{n})$.



Extensions

- Planar triangulations have *cycle separators* of size $O(\sqrt{n})$.
  [Miller '84 '86]

Extensions

- Planar triangulations have *cycle separators* of size $O(\sqrt{n})$.
  [Miller '84 '86]

- *Separator hierarchy*: Recursively separate both components

Extensions

- Planar triangulations have *cycle separators* of size $O(\sqrt{n})$.
  [Miller '84 '86]

- *Separator hierarchy*: Recursively separate both components
  - Cutting across the hierarchy gives us an *r-division*: n/r pieces, each with $O(r)$ vertices and adjacent to $O(\sqrt{r})$ shallower separator vertices
    [Frederickson '87]

Extensions

- Planar triangulations have *cycle separators* of size $O(\sqrt{n})$.
  [Miller '84 '86]

- *Separator hierarchy*: Recursively separate both components
  - Cutting across the hierarchy gives us an *r-division*: n/r pieces, each with $O(r)$ vertices and adjacent to $O(\sqrt{r})$ shallower separator vertices
    [Frederickson '87]
  - For *cycle* separator hierarchies: Each piece is a disk with $O(1)$ *holes*.
    [Klein Subramanian '08]

Extensions

- Planar triangulations have *cycle separators* of size $O(\sqrt{n})$.
  [Miller '84 '86]

- *Separator hierarchy*: Recursively separate both components
  - Cutting across the hierarchy gives us an *r-division*: n/r pieces, each with $O(r)$ vertices and adjacent to $O(\sqrt{r})$ shallower separator vertices
    [Frederickson '87]
  - For *cycle* separator hierarchies: Each piece is a disk with $O(1)$ *holes*.
    [Klein Subramanian '08]
  - The *entire hierarchy* can be computed in $O(n)$ time.
    [Klein Mozes Sommer '13]

Please ask questions!

Shortest paths with arbitrary weights

- If any edges have negative weight, Dijkstra's algorithm no longer works (at least not quickly)
- But *Bellman-Ford* always works in $O(VE)$ time:

```
dist(s) ← 0
for all vertices v ≠ s
    dist(v) ← ∞

repeat V times
    for each edge u→v
        if dist(v) > dist(u) + w(u→v)
            dist(v) ← dist(u) + w(u→v)
```

[Shimbel '54, Ford '56, Moore '57, Woodbury Dantzig '57, Bellman '58, Minty '58]

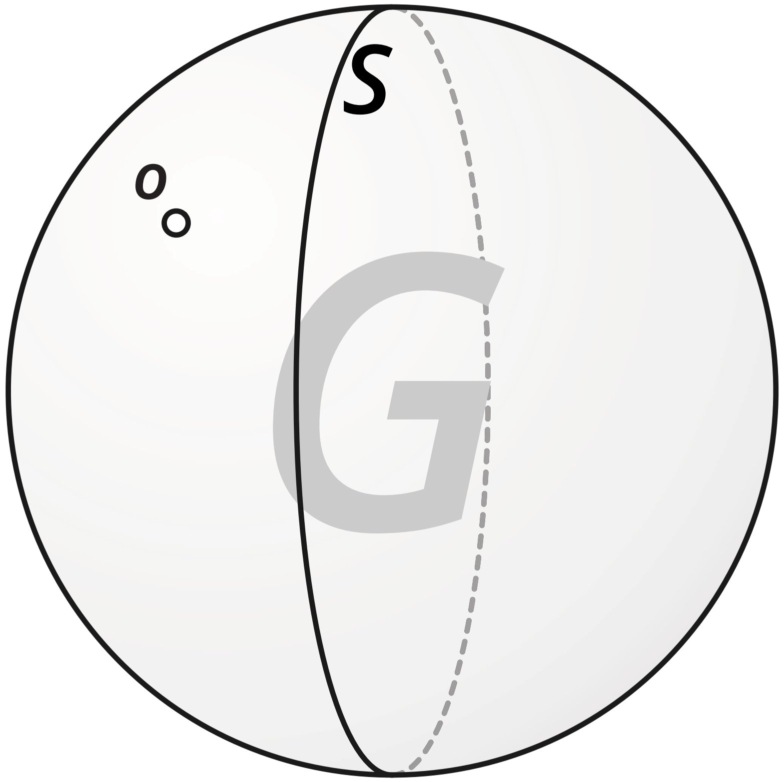
Shortest paths with negative edges

- Suppose we want to compute $dist_G(o,G)$ = distances in G from vertex o to all vertices in G.

[Mehlhorn Schmidt ’83]

Shortest paths with negative edges

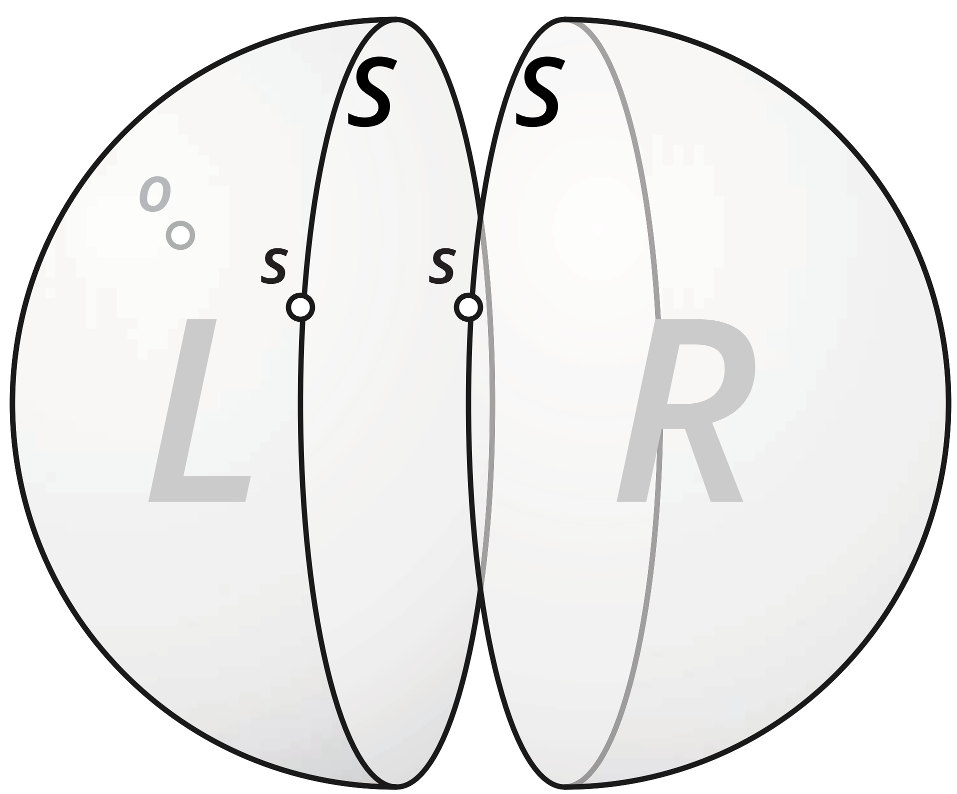
- Suppose we want to compute $dist_G(o,G)$ = distances in G from vertex o to all vertices in G.
- Find a separator S, splitting G into L and R. Pick new source vertex s in S.



[Mehlhorn Schmidt '83]

Shortest paths with negative edges

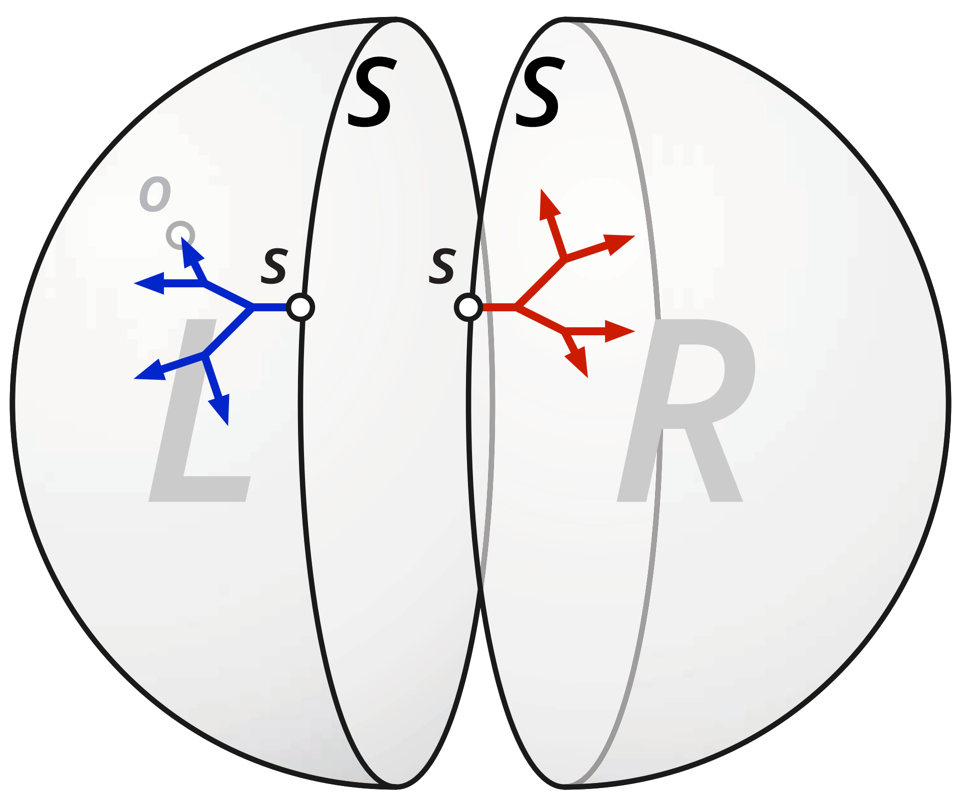
- Suppose we want to compute $dist_G(o,G)$ = distances in G from vertex o to all vertices in G.
- Find a separator S, splitting G into L and R. Pick new source vertex s in S.



[Mehlhorn Schmidt '83]

Shortest paths with negative edges

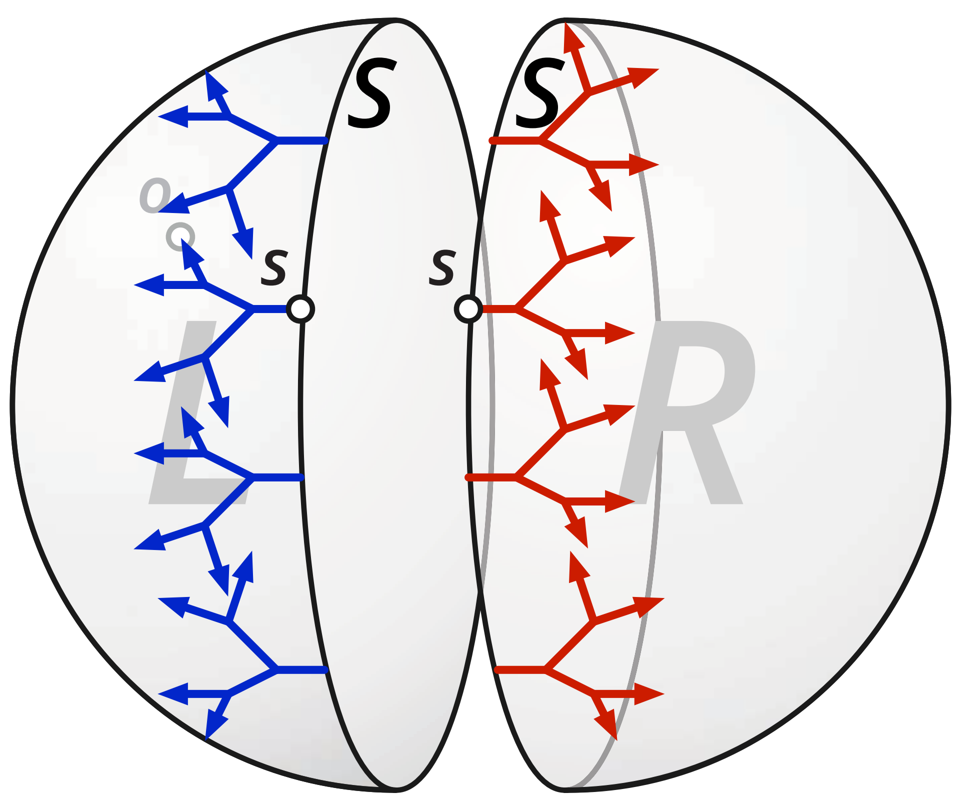
- Recursively compute $dist_L(s,L)$ and $dist_R(s,R)$.



[Mehlhorn Schmidt '83]

Shortest paths with negative edges

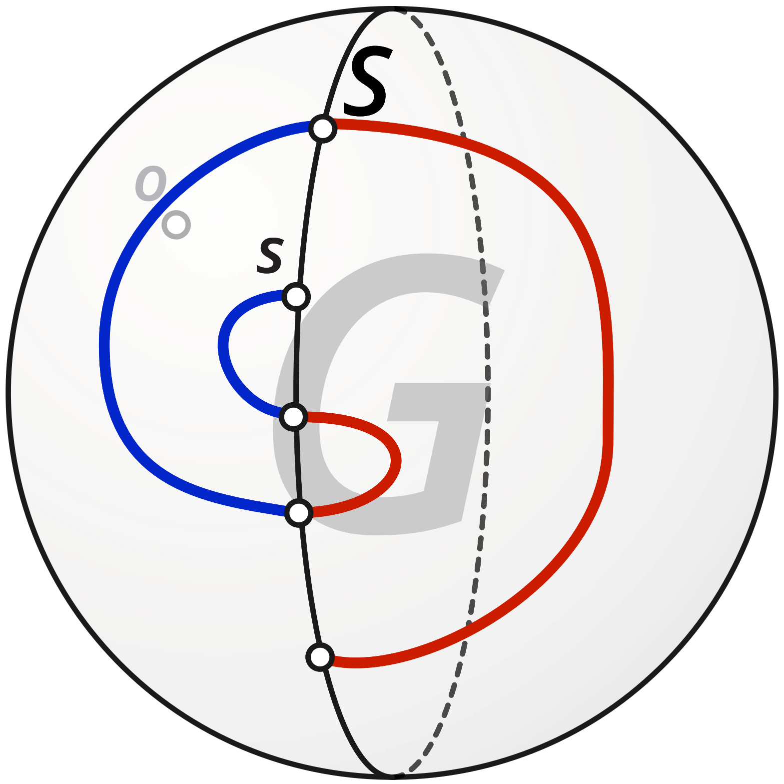
- Recursively compute $dist_L(s,L)$ and $dist_R(s,R)$.
- Reweight L and R so edges are non-negative, and compute $dist_L(S,L)$ and $dist_R(S,R)$ via Dijkstra. $\Rightarrow O(n^{3/2} \log n)$ time



[Mehlhorn Schmidt '83]

Shortest paths with negative edges

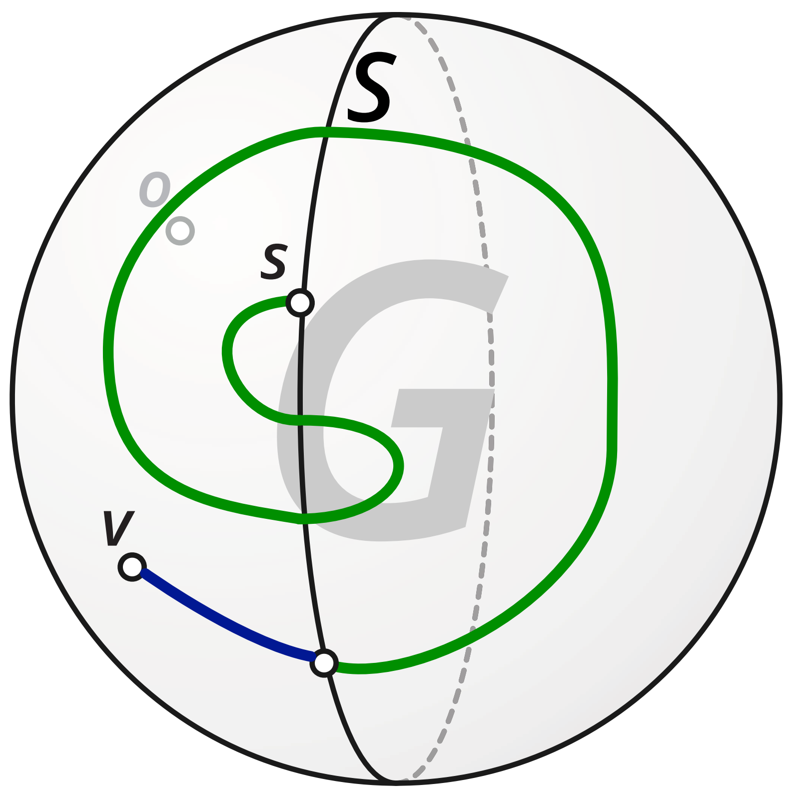
- Compute $dist_G(s,S)$ from $dist_L(S,S)$ and $dist_R(S,S)$ via Bellman-Ford.
  - Complete graph on $O(\sqrt{n})$ vertices $\Rightarrow$ $O(n^{3/2})$ time



[Mehlhorn Schmidt '83]

Shortest paths with negative edges

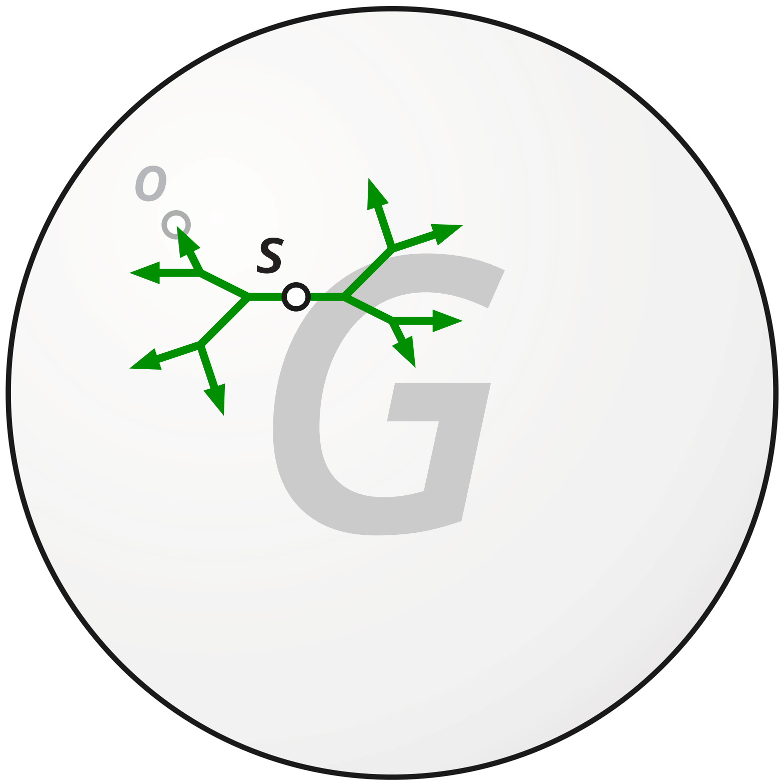
- Compute $dist_G(s,G)$ by brute force $\Rightarrow O(n^{3/2})$ time
  - If $v \in L$, then $dist_G(s,v) = \min_{t \in S} dist_G(s,t) + dist_L(t,v)$.
  - If $v \in R$, then $dist_G(s,v) = \min_{t \in S} dist_G(s,t) + dist_R(t,v)$.



[Mehlhorn Schmidt '83]

Shortest paths with negative edges

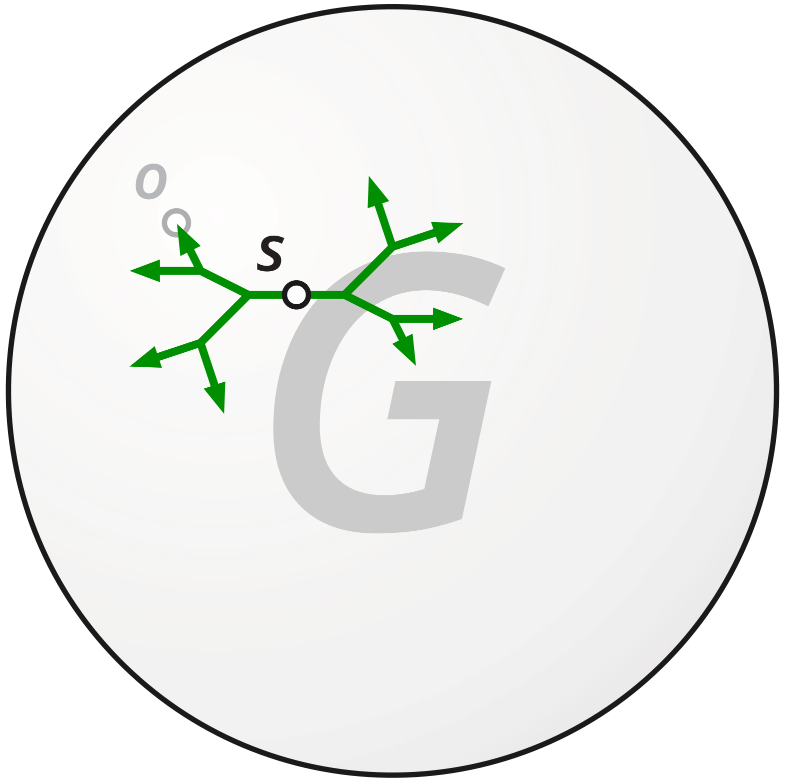
- Compute $dist_G(s,G)$ by brute force ⇒ $O(n^{3/2})$ time
  - If $v \in L$, then $dist_G(s,v) = \min_{t \in S} dist_G(s,t) + dist_L(t,v)$.
  - If $v \in R$, then $dist_G(s,v) = \min_{t \in S} dist_G(s,t) + dist_R(t,v)$.



[Mehlhorn Schmidt '83]

Shortest paths with negative edges

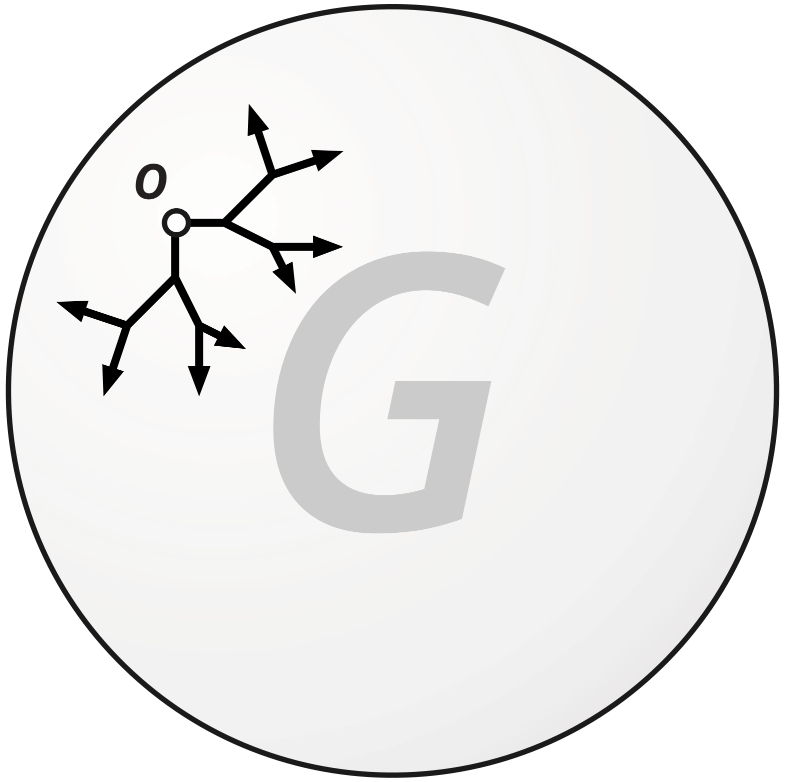
- Finally, reweight G so edges are non-negative, and compute $dist_G(o,G)$ via Dijkstra.



[Mehlhorn Schmidt '83]

Shortest paths with negative edges

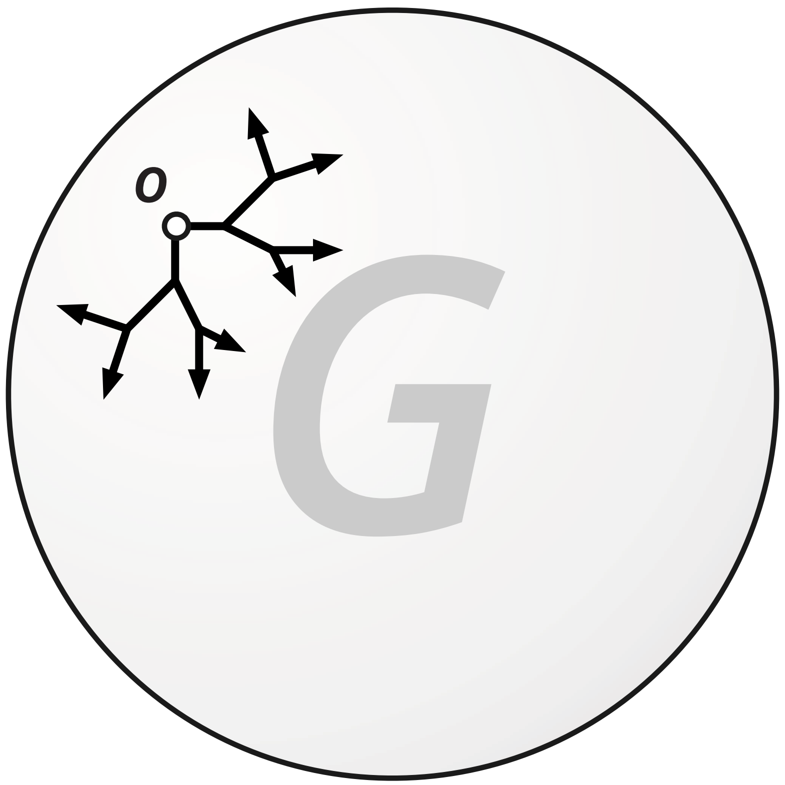
- Finally, reweight G so edges are non-negative, and compute $dist_G(o,G)$ via Dijkstra.



[Mehlhorn Schmidt '83]

Shortest paths with negative edges

- Finally, reweight G so edges are non-negative, and compute $dist_G(o,G)$ via Dijkstra.



- **Total:** $T(n) \leq T(n/3) + T(2n/3) + O(n^{3/2} \log n) = \mathbf{O(n^{3/2} \log n)}$

[Mehlhorn Schmidt '83]

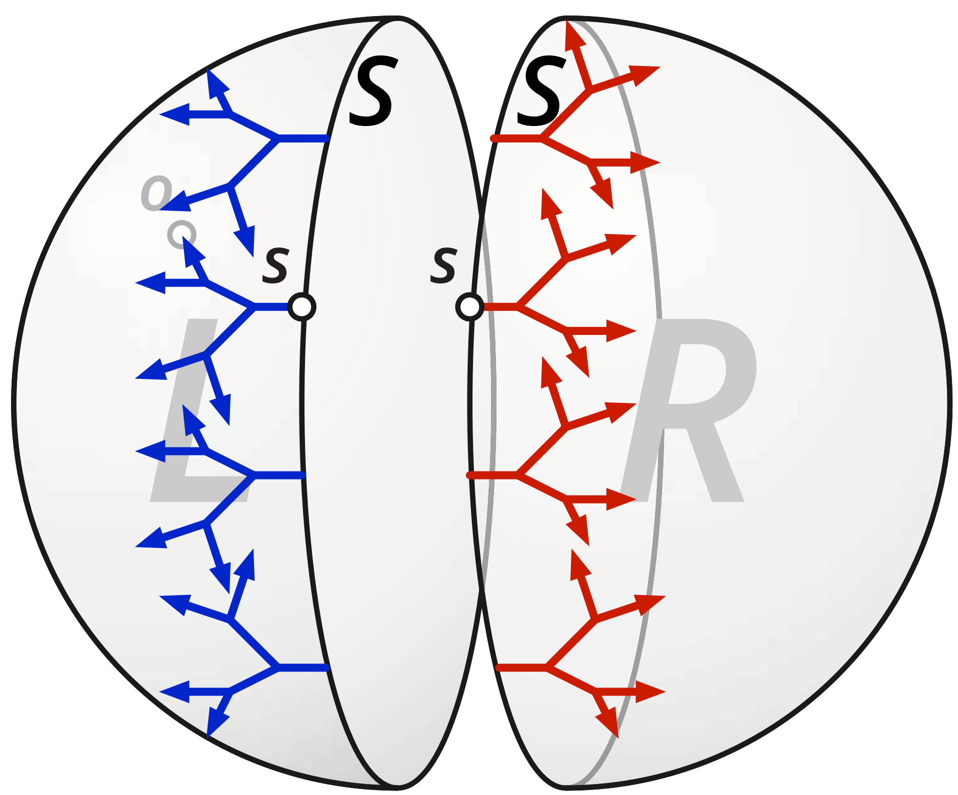
Further tricks

- Multiple-source shortest paths: We can (implicitly) compute $dist_L(S,L)$ and $dist_R(S,R)$ in only $O(n \log n)$ time.
  - Yes, even though there are $O(n^{3/2})$ distances to compute!
  - More details tomorrow!

[Klein '05]

Further tricks

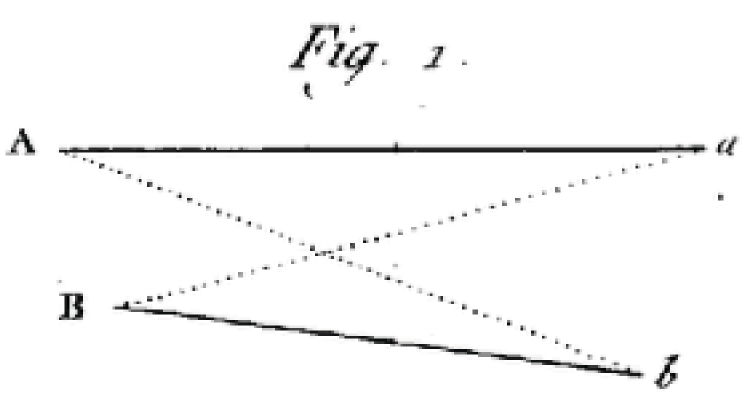
- Multiple-source shortest paths: We can (implicitly) compute $dist_L(S,L)$ and $dist_R(S,R)$ in only $O(n \log n)$ time.
  - Yes, even though there are $O(n^{3/2})$ distances to compute!
  - More details tomorrow!



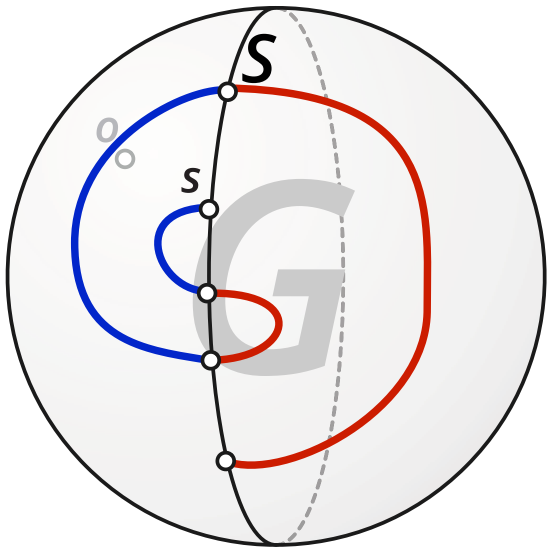
[Klein '05]

Further tricks

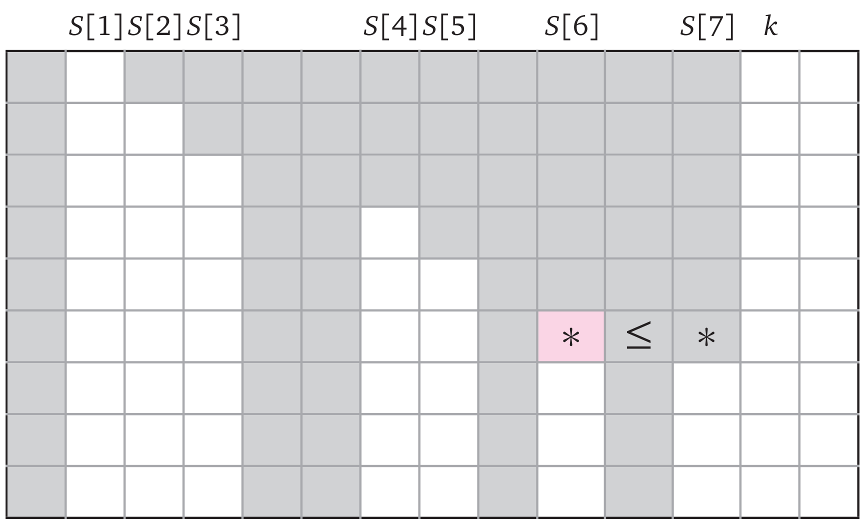
- *Monge* property: Boundary-to-boundary paths in planar graphs are shorter when they don't cross.
- We can use this property to speed up Bellman-Ford stage from $O(n^{3/2})$ time to $O(n \log n)$ time



[Monge 1781]



[Fakcharoenphol Rao '06]



[SMAWK '87]

Shortest paths with negative edges

- Using several more advanced tricks, we can compute shortest paths in planar graphs with arbitrary edge weights in $O(n \log^2 n / \log \log n)$ time.
  [Klein Mozes Weimann '10, Mozes Wulff-Nilsen '10]
  - Multiple-source shortest paths *[Klein '05]*
  - Structured r-divisions instead of single separators *[Klein Mozes Sommer '13]*
  - Exploit Monge property in Bellman-Ford *[Fakcharoenphol Rao '06]*
  - Careful parameter balancing ($r = n \cdot \alpha(n) / \log n$)

Please ask questions!

Thank you! See you tomorrow!