Flows on surface embedded graphs

Erin Wolf Chambers

Why do computational geometers care about topology?

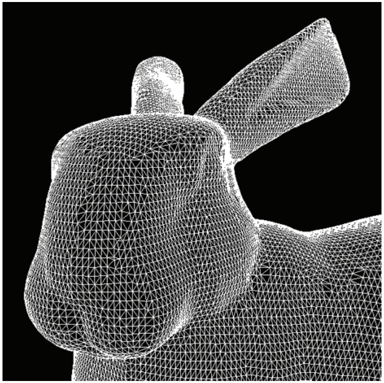
Many areas, such as graphics, biology, robotics, and networking, use algorithms which compute information about point sets.

Why do computational geometers care about topology?

Many areas, such as graphics, biology, robotics, and networking, use algorithms which compute information about point sets.

Surfaces

We've already seen algorithms that compute a mesh of these points in order to represent the original object.



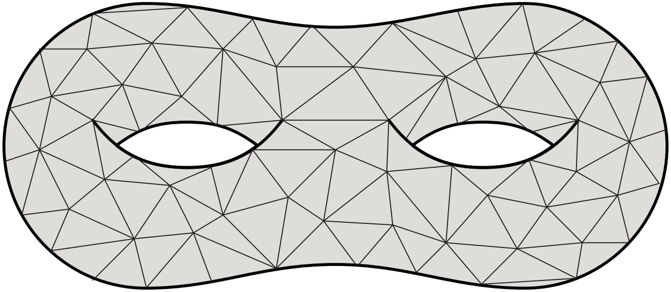
Figures courtesy of Stanford computer graphics laboratory.

Setting

This output mesh naturally leads to a setting that is at the boundaries of graph theory, topology, and geometry.

Setting

This output mesh naturally leads to a setting that is at the boundaries of graph theory, topology, and geometry.

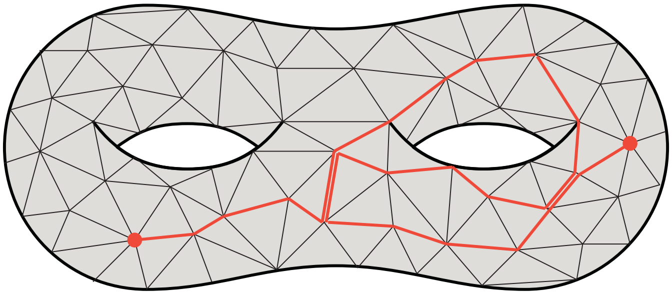


Definition

A combinatorial surface is a 2-manifold which has a weighted graph embedded on its surface so that every face of the graph is a disk.

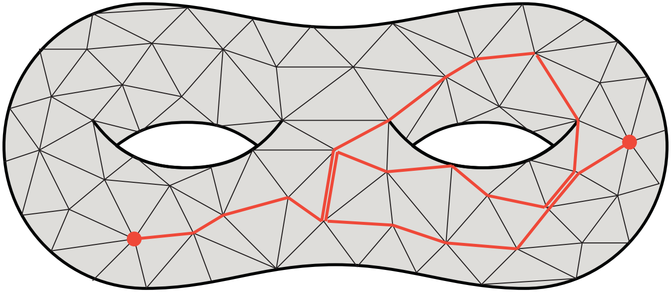
Setting

We will only consider paths and cycles which stay on the edges of the graph.



Setting

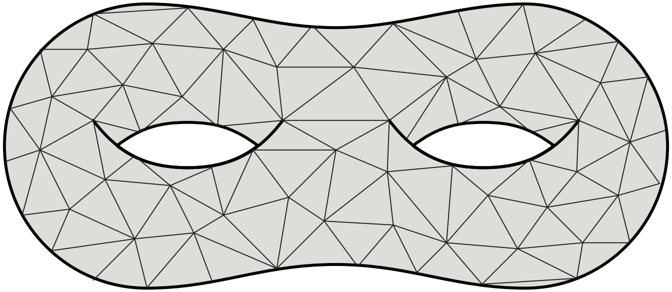
We will only consider paths and cycles which stay on the edges of the graph.



The underlying surface is actually unknown - all we have is the combinatorial structure of the graph (with weights), plus information about faces.

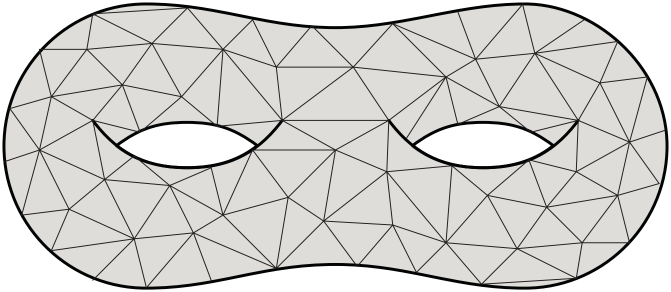
Definitions

We will let n be the total size of the graph, but this is not the only relevant parameter here.



Definitions

We will let n be the total size of the graph, but this is not the only relevant parameter here.



Any orientable surface is topologically equivalent to a sphere with some number of handles attached to it; this is the *genus* of the surface, g.

Euler characteristic

These parameters are connected:

- For any polyhedral manifold M, we know that $v - e + f = \chi(M)$, the Euler characteristic of M.

Euler characteristic

These parameters are connected:

- For any polyhedral manifold M, we know that $v - e + f = \chi(M)$, the Euler characteristic of M.
- This is independent of the triangulation on M: $\chi = 2 - 2g - k$ if the manifold is orientable, where g is the genus and k is the number of boundaries.

Euler characteristic

These parameters are connected:

- For any polyhedral manifold M, we know that $v - e + f = \chi(M)$, the Euler characteristic of M.
- This is independent of the triangulation on M: $\chi = 2 - 2g - k$ if the manifold is orientable, where g is the genus and k is the number of boundaries.
- This means that if the manifold has v vertices, then it has at most $3v - 6 + 6g$ edges and at most $2v - 4 + 4g - k$ faces. (Equality holds when every face and boundary is a triangle.)

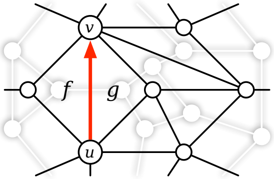
Euler characteristic

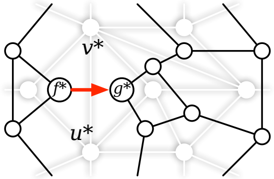
These parameters are connected:

- For any polyhedral manifold M, we know that $v - e + f = \chi(M)$, the Euler characteristic of M.
- This is independent of the triangulation on M: $\chi = 2 - 2g - k$ if the manifold is orientable, where g is the genus and k is the number of boundaries.
- This means that if the manifold has v vertices, then it has at most $3v - 6 + 6g$ edges and at most $2v - 4 + 4g - k$ faces. (Equality holds when every face and boundary is a triangle.)
- Hence, we'll let $n \leq 6v - 10 + 10g - k$ be the total number of edges, faces, and vertices.

Dual graphs

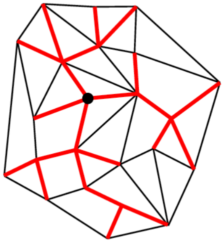
Given an embedded graphs, we can form the *dual graph*:

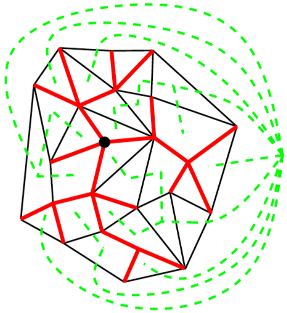




The Planar Case

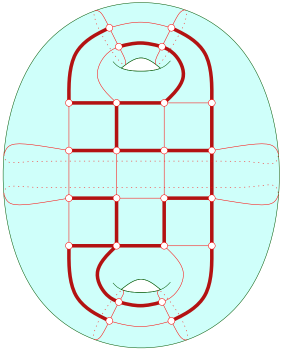
For a planar graph G with a spanning tree T, $G^* \setminus E(T)^*$ is a spanning tree of the dual graph G^*.

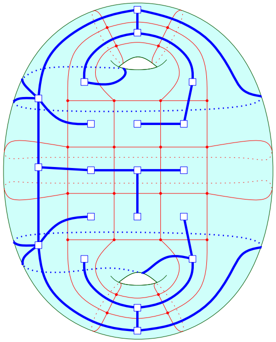




The genus g case

On a surface, we can still consider the dual of a tree, but $G^* \setminus E(T)^*$ is NOT a spanning tree of the dual graph G^*.





Instead, we can decompose into a tree, a co-tree, and $O(g)$ "extra" edges.

What can we compute?

There are many possible questions we can ask in this model, many of which are generalizations of questions on planar graphs.

What can we compute?

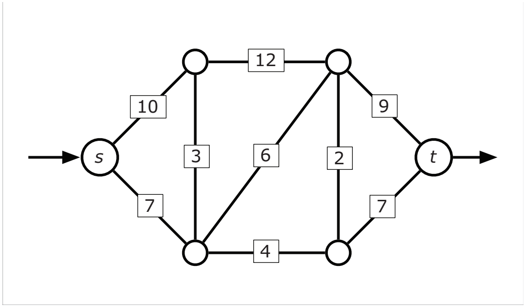
There are many possible questions we can ask in this model, many of which are generalizations of questions on planar graphs.

- How (fast) can we compute topologically interesting cycles?
- How can we tell if two curves are similar to each other?
- Can we tell if two such graphs are isomorphic?
- Can we given efficient ways to morph between two isomorphic graphs?
- Can we compute flows and cuts in these graphs?

Flow Networks

We are given:

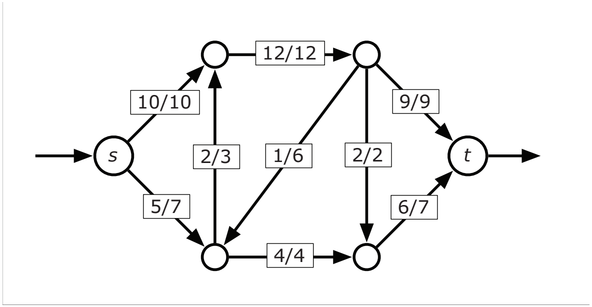
- An undirected graph $G = (V, E)$
- A capacity function $c : E \to R^+$
- Two vertices s (the source) and t (the sink)



Cuts and Flows

Minimum Cut: Compute the minimum set of edges separating s from t

Maximum Flow: Assign a direction and nonnegative weight to every edge so that flow through each edge does not exceed its capacity, flow is conserved at every vertex (other than s and t), and the flow out of s (and into t) is as large as possible.



Flows versus cuts

Theorem

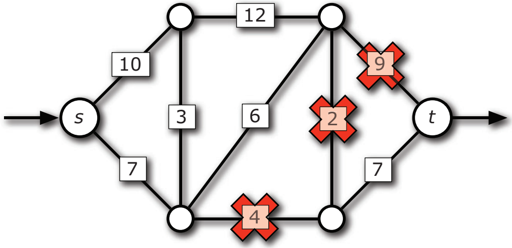
The maximum value flow is equal to the minimum capacity cut.

Flows versus cuts

Theorem

The maximum value flow is equal to the minimum capacity cut.

Given the maximum flow, it is easy to compute the minimum cut.



Applications

Maximum flows and minimum cuts are two fundamental problems in combinatorial optimization.

- Routing
- Maximum matchings
- Assignment
- Scheduling
- Load balancing
- Image segmentation
- Many more...

Previous work for planar graphs

Near linear time algorithms for planar graphs:

- Undirected $O(n \log n)$:
    - Min-Cut [Reif 83], [Frederickson 87]
    - Max-Flow [Itai and Shiloach 83], [Hassin and Johnson 85]

Previous work for planar graphs

Near linear time algorithms for planar graphs:

- Undirected $O(n \log n)$:
  - Min-Cut [Reif 83], [Frederickson 87]
  - Max-Flow [Itai and Shiloach 83], [Hassin and Johnson 85]
- Directed $O(n \log n)$:
  - Min-Cut [Janiga and Koubek 92], [Henzinger, Klein, Rao and Subramanian 97]
  - Max-Flow [Weihe 97], [Borradaile and Klein 06]

Previous work for sparse graphs

- For graphs with n vertices and $O(n)$ edges, we can compute max flows (and therefore min cuts) in:
  - Integer capacities with maximum value U: $O(n^{3/2} \log n \log U)$ time [Goldberg Rao 1998]
  - Real capacities: $O(n^2 \log n)$ time [Sleater Tarjan 1983]

Previous work for sparse graphs

- For graphs with n vertices and $O(n)$ edges, we can compute max flows (and therefore min cuts) in:
  - Integer capacities with maximum value U: $O(n^{3/2} \log n \log U)$ time [Goldberg Rao 1998]
  - Real capacities: $O(n^2 \log n)$ time [Sleater Tarjan 1983]
- If the genus is fixed, the maximum flow can be computed in $O(n^{1.595} \log C)$ time, where C is the sum of the capacities [Imai Iwano 1990].

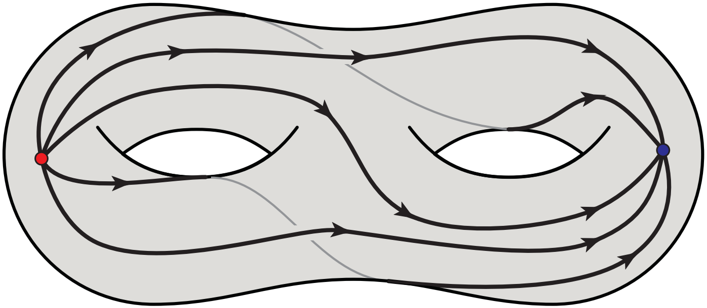
Previous work for sparse graphs

- For graphs with n vertices and $O(n)$ edges, we can compute max flows (and therefore min cuts) in:
  - Integer capacities with maximum value U: $O(n^{3/2} \log n \log U)$ time [Goldberg Rao 1998]
  - Real capacities: $O(n^2 \log n)$ time [Sleater Tarjan 1983]
- If the genus is fixed, the maximum flow can be computed in $O(n^{1.595} \log C)$ time, where C is the sum of the capacities [Imai Iwano 1990].
- If the graph is planar with k extra edges, max flow can be computed in $O(k^3 n \log n)$ time [Hochstein Weihe 07].

Flows in surface graphs

Theorem

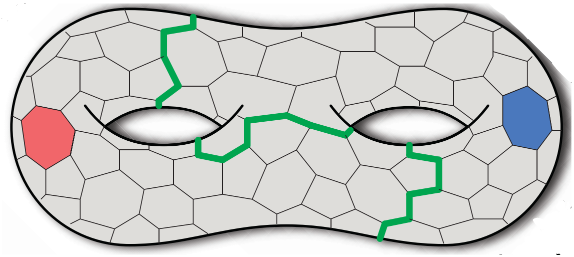
Given a directed or undirected flow network embedded on a surface of genus g, we can compute the minimum s-t flow in $g^{O(g)}n^{3/2}$ time, or in time $g^7 n \log^2 n \log^2 C$ if the edges have integer capacities that sum to C. (in STOC 2009)



Cuts in surface graphs

Theorem

Given an undirected flow network embedded on a surface of genus g, we can compute the minimum s-t cut in $g^{O(g)} n \log n$ time. (in SOCG 2009)



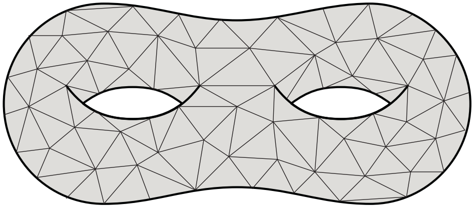
(This was later improved to $2^{O(g)} n \log \log n$, but not by me!)

Dual Graph

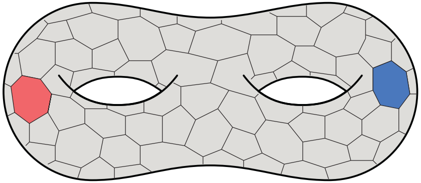
Both of these results rely heavily on dual graphs: flows will be in the main graph, but we can think of cuts as separating two faces from the dual graph. Either way, we get a set of edges.

Dual Graph

Both of these results rely heavily on dual graphs: flows will be in the main graph, but we can think of cuts as separating two faces from the dual graph. Either way, we get a set of edges.

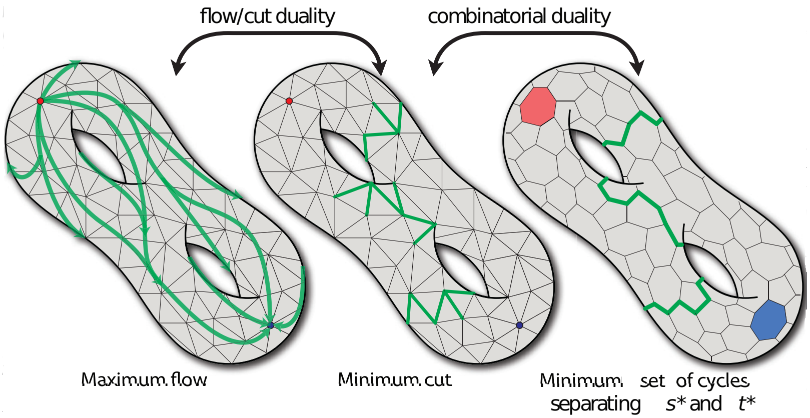


G



G^*

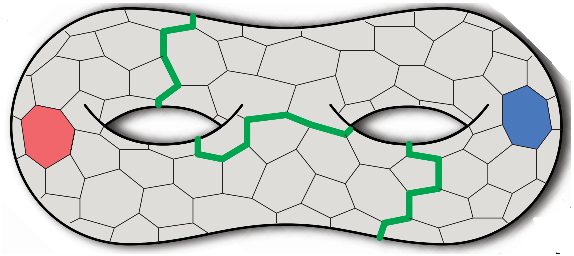
Cuts in the dual graph



Even Subgraphs

Definition

An even subgraph is a subgraph where every vertex has even degree.

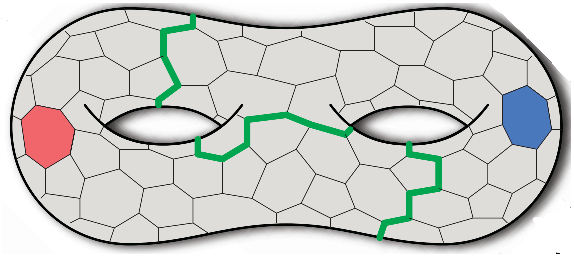


Equivalently, an even subgraph is the union of edge-disjoint cycles.

Even Subgraphs

Definition

An even subgraph is a subgraph where every vertex has even degree.



Equivalently, an even subgraph is the union of edge-disjoint cycles.

Note that the dual of the min cut is an even subgraph in G^*.

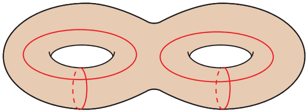
Homology

A few weeks ago, we saw the concept of *homology*. Here, we're only dealing 1-dimensional homology of surfaces, not arbitrary simplicial complexes, so things are simpler:

Homology

A few weeks ago, we saw the concept of *homology*. Here, we're only dealing 1-dimensional homology of surfaces, not arbitrary simplicial complexes, so things are simpler:

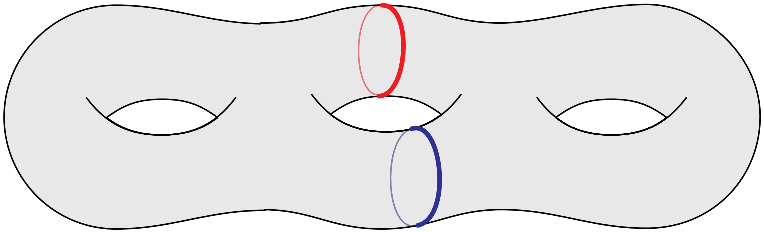
- Cycles are sets of edges which have no boundary
- Boundaries are (unions of) sets of edges that border some face.
- Homology considers all cycles, but mods out by the boundaries.



Homologous Subgraphs

Definition

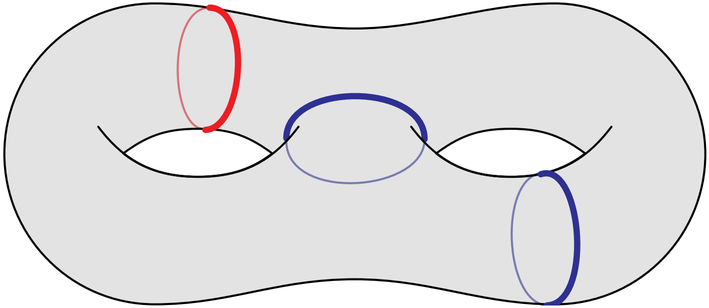
Two even subgraphs are $\mathbb{Z}_2$-homologous if their union forms a cut on the graph.



Homologous Subgraphs

Definition

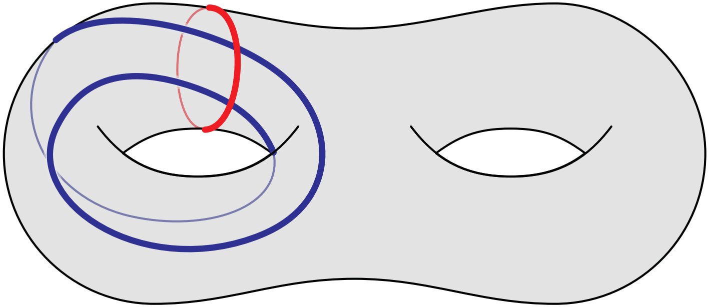
Two even subgraphs are $\mathbb{Z}_2$-homologous if their union forms a cut on the graph.



Homologous Subgraphs

Definition

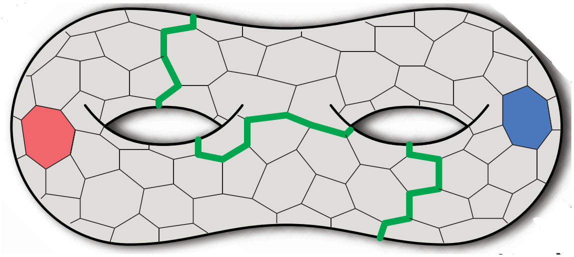
Two even subgraphs are $\mathbb{Z}_2$-homologous if their union forms a cut on the graph.



Another Characterization of Min Cuts

Lemma

The dual of the minimum cut is the minimum weight even subgraph homologous to the boundary of s^* in $G^* \setminus \{s^*, t^*\}$.

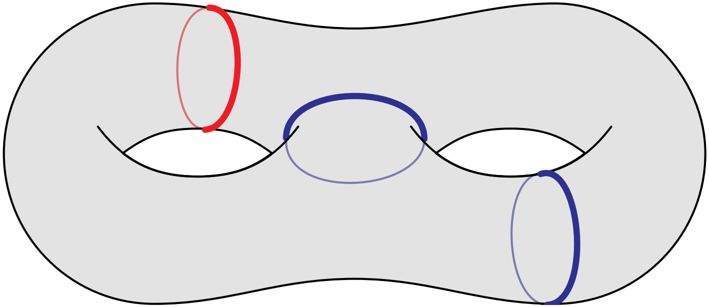


Theorem

Given any even subgraph H of an embedded graph, we can compute the minimum even subgraph homologous to H in $g^{O(g)} n \log n$ time.

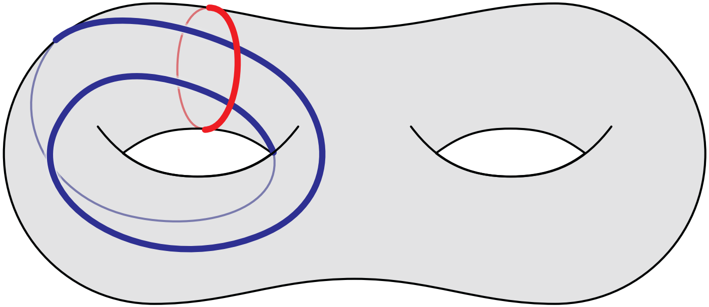
Theorem

Given any even subgraph H of an embedded graph, we can compute the minimum even subgraph homologous to H in $g^{O(g)} n \log n$ time.



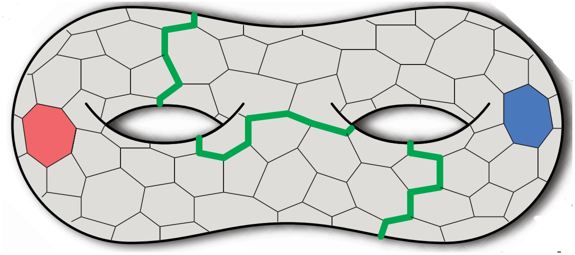
Theorem

Given any even subgraph H of an embedded graph, we can compute the minimum even subgraph homologous to H in $g^{O(g)} n \log n$ time.



Theorem

Given any even subgraph H of an embedded graph, we can compute the minimum even subgraph homologous to H in $g^{O(g)} n \log n$ time.



Since the min cut is the minimum weight even subgraph homologous to the boundary of s^*, we can compute the min cut in $g^{O(g)} n \log n$ time.

Crossing Lemma

Again, a key tool will be shortest paths:

Lemma

Let H be an even subgraph of minimum weight in its homology class. Then any shortest path crosses H at most $O(g)$ times.

Crossing Lemma

Again, a key tool will be shortest paths:

Lemma

Let H be an even subgraph of minimum weight in its homology class. Then any shortest path crosses H at most $O(g)$ times.

So we can cut the surface using shortest paths, and we know the minimum homologous subgraph can't cross our shortest paths many times.

Crossing Lemma

Again, a key tool will be shortest paths:

Lemma

Let H be an even subgraph of minimum weight in its homology class. Then any shortest path crosses H at most $O(g)$ times.

So we can cut the surface using shortest paths, and we know the minimum homologous subgraph can't cross our shortest paths many times.

This lets us brute force a solution – but it will only be exponential in g.

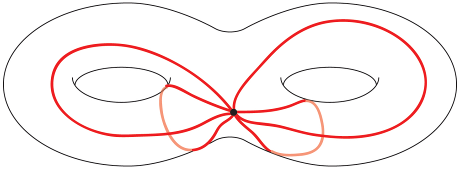
Sketch of Algorithm

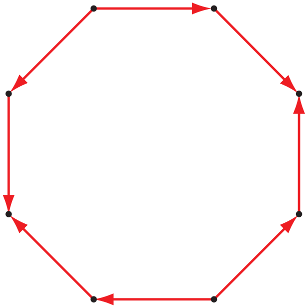
We cut the surface using a greedy system of loops [Erickson Whittlesey 2005]; each loop consists of two shortest paths.

Sketch of Algorithm

We cut the surface using a greedy system of loops [Erickson Whittlesey 2005]; each loop consists of two shortest paths.

When we cut along the greedy system, we get a topological disk.



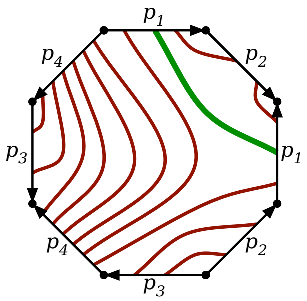


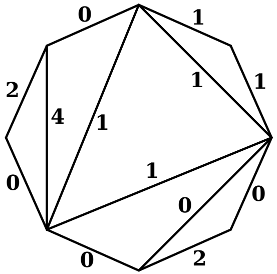
Sketch of Algorithm (cont.)

The minimum weight even homologous subgraph will cross each shortest path $O(g)$ times.

Sketch of Algorithm (cont.)

The minimum weight even homologous subgraph will cross each shortest path $O(g)$ times.





This corresponds to a labeled triangulation of a polygon with $2g$ edges, each label being a number between 0 and $O(g)$.

Parity Vectors

Definition

The crossing parity vector of an even subgraph H with respect to a system of loops is a bit vector where the i^{th} bit is equal to 1 if H crosses the i^{th} loop an odd number of times.

Parity Vectors

Definition

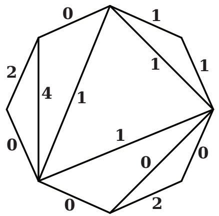
The crossing parity vector of an even subgraph H with respect to a system of loops is a bit vector where the i^{th} bit is equal to 1 if H crosses the i^{th} loop an odd number of times.

Lemma

Two even subgraphs are homologous if and only if they have the same crossing parity vectors.

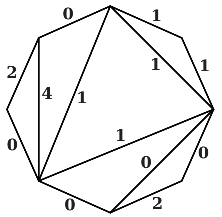
Algorithm: Final Stage

Given a weighted triangulation homologous to our original even subgraph, we can compute the shortest set of corresponding cycles in $O(gn \log n)$ time.



Algorithm: Final Stage

Given a weighted triangulation homologous to our original even subgraph, we can compute the shortest set of corresponding cycles in $O(gn \log n)$ time.



Total running time is $g^{O(g)} n \log n$ time, since there are $g^{O(g)}$ possible weighted triangulations.

NP-Hardness: The Bad News

Unfortunately, this approach will not lead to an algorithm to compute the min cut in polynomial time.

NP-Hardness: The Bad News

Unfortunately, this approach will not lead to an algorithm to compute the min cut in polynomial time.

Theorem

Given an even subgraph H of a surface embedded graph, computing the minimum weight even subgraph homologous to H is NP-Hard.

NP-Hardness: The Bad News

Unfortunately, this approach will not lead to an algorithm to compute the min cut in polynomial time.

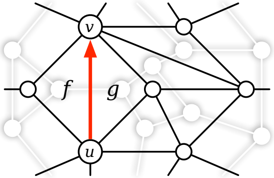
Theorem

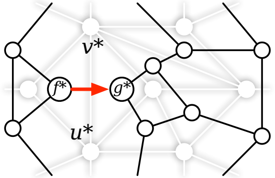
Given an even subgraph H of a surface embedded graph, computing the minimum weight even subgraph homologous to H is NP-Hard.

Reduction is from min cut in graphs with negative edges.

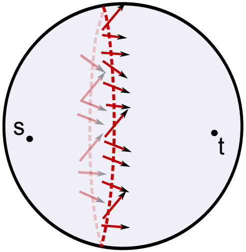
Key Idea for Flows: Cocycles

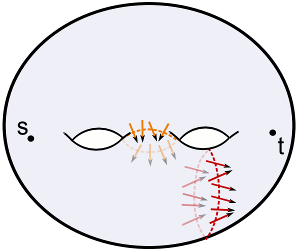
Consider the dual G^* of G:





A cocycle is the dual of a cycle.





Flows in the planar case

Lemma (Itai-Shiloach 83, Hassin-Johnson 85)

There is a feasible (s,t)-flow in G with the same value as a given (s,t)-flow ϕ if and only if the dual residual network G^*_ϕ contains no negative cycles.

Flows in the planar case

Lemma (Itai-Shiloach 83, Hassin-Johnson 85)

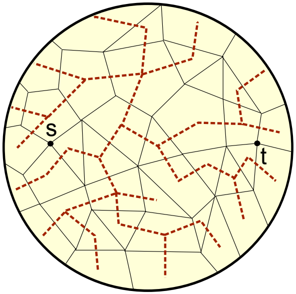
There is a feasible (s,t)-flow in G with the same value as a given (s,t)-flow ϕ if and only if the dual residual network G^*_ϕ contains no negative cycles.

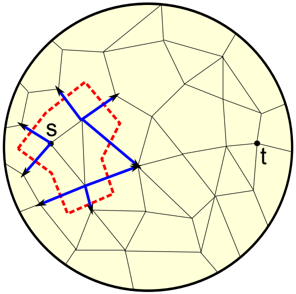
In planar graphs, this gives a way to compute flows in the primal by looking at what types of cycles are present in the weighted dual graph. If there are no negative ones, we have a valid flow.

Flows in the planar case

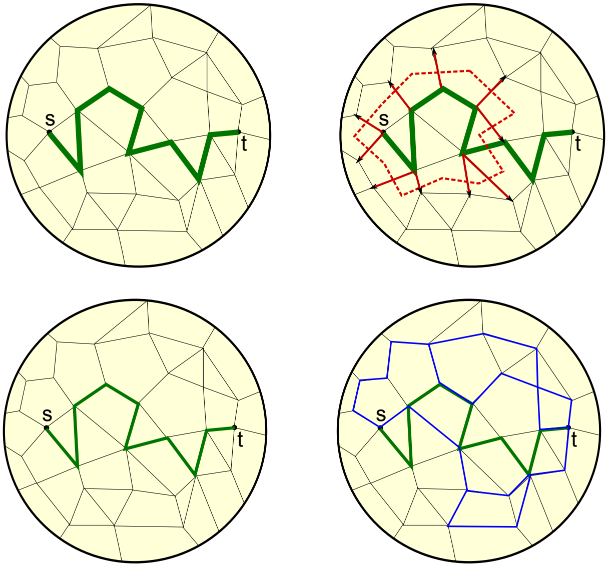
This becomes an algorithm if we consider dual shortest paths:

We can compute shortest paths in planar graphs quickly (as we discussed last time), so this gives a fast way to compute flows.





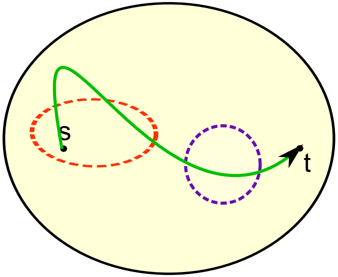
The planar algorithm (in pictures)



How to generalize?

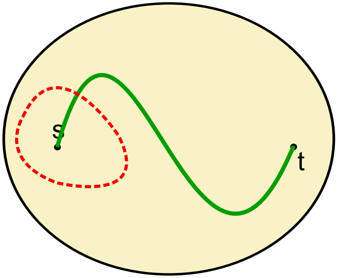
Two planar flows ϕ and ψ have the same total value if and only if they send equal flow through each cocycle of the graph.

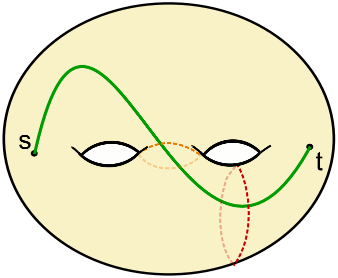
This is essentially computing a max flow by looking at which co-cycles (or potential cuts) gets saturated.



Genus g graphs

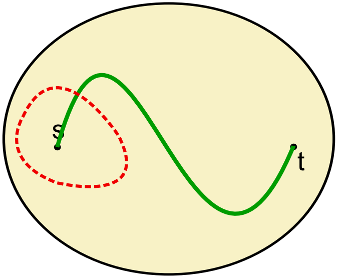
However, the flow value does not give enough information when $g > 0$: we also need to know what homology class the flow lives in.

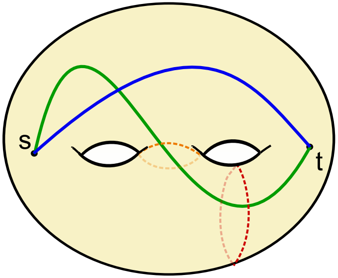




Genus g graphs

However, the flow value does not give enough information when $g > 0$: we also need to know what homology class the flow lives in.





Using Homology

Definition

Two flows ϕ and ψ are called homologous if and only if they send equal flow through each cocycle of the graph.

Using Homology

Definition

Two flows ϕ and ψ are called homologous if and only if they send equal flow through each cocycle of the graph.

Lemma (Planar Case)

There is a feasible (s,t)-flow in G with the same value as a given (s,t)-flow ϕ if and only if the dual residual network G^*_ϕ contains no negative cycles.

Using Homology

Definition

Two flows ϕ and ψ are called homologous if and only if they send equal flow through each cocycle of the graph.

Lemma (Planar Case)

There is a feasible (s, t)-flow in G with the same value as a given (s, t)-flow ϕ if and only if the dual residual network G^*_ϕ contains no negative cycles.

Lemma

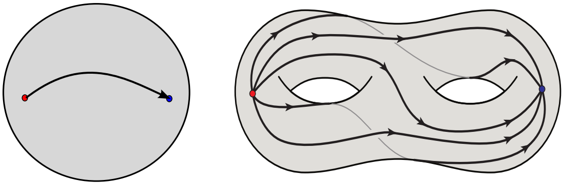
There is a feasible (s, t)-flow in G homologous to a given (s, t)-flow ϕ if and only if the dual residual network G^*_ϕ contains no negative cycles.

Computing homologous flows

- We use a generalization of [Klein, Mozes and Weimann 09] to compute the shortest paths in the dual.
- Our algorithm either returns a feasible homologous flow or a negative cocycle in $O(gn\log^2 n)$ time.

Computing homologous flows

- We use a generalization of [Klein, Mozes and Weimann 09] to compute the shortest paths in the dual.
- Our algorithm either returns a feasible homologous flow or a negative cocycle in $O(gn \log^2 n)$ time.
- The problem then reduces to finding the homology class of a max-flow, which is in R^{2g+1} for a surface of genus g (or R for the plane).



Final optimization algorithm

Each homology class can be presented by a vector, $\langle \phi_0, \ldots, \phi_{2g} \rangle$.

Final optimization algorithm

Each homology class can be presented by a vector, $\langle \phi_0, \ldots, \phi_{2g} \rangle$.
Goal:

- Maximize $\sum_{i=0}^{2g} \phi_i$,
- Such that no cocycle of the graph is oversaturated.

Final optimization algorithm

Each homology class can be presented by a vector, $\langle \phi_0, \ldots, \phi_{2g} \rangle$.
Goal:

- Maximize $\sum_{i=0}^{2g} \phi_i$,
- Such that no cocycle of the graph is oversaturated.

This gives an LP with exponential number of constraints.

Final optimization algorithm

Each homology class can be presented by a vector, $\langle \phi_0, \ldots, \phi_{2g} \rangle$.
Goal:

- Maximize $\sum_{i=0}^{2g} \phi_i$,
- Such that no cocycle of the graph is oversaturated.

This gives an LP with exponential number of constraints.

- Ellipsoid method: $O(g^7 n \log^2 n \log^2 C)$
- Multidimensional parametric search [Cohen and Megido 93]: $g^{O(g)} n^{3/2}$

The End

This is an interesting trade-off: We give algorithms that are exponential in g, but flows in general graphs are polynomial time (although higher in terms of n).

We conjectured in both of these papers (and in followups) that the right answer is $O(g^c n \log n)$ for some constant c, but that is still open.