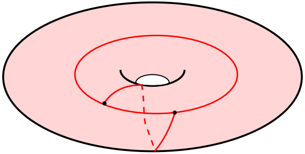
A Fixed Parameter Tractable Approximation Scheme for the Optimal Cut Graph of a Surface

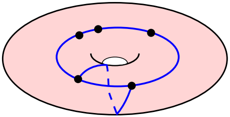
Vincent Cohen-Addad [1]   Arnaud de Mesmay [2]

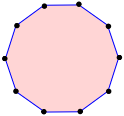
[1]École normale supérieure, Paris, France

[2]IST Austria, Klosterneuburg, Austria



From Donuts to Crêpes



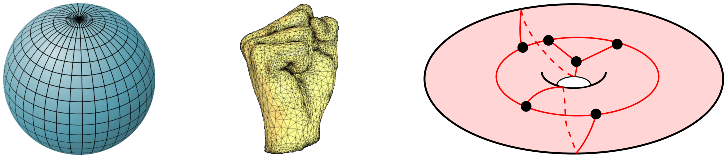


From Bretzels to Crêpes?

Surfaces and embedded graphs

- A ***surface*** is a topological space which looks locally like the plane.

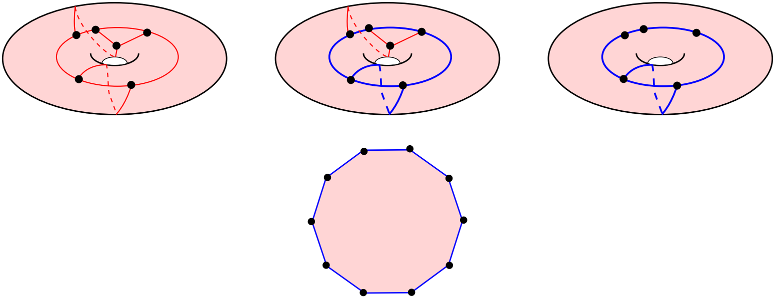
- Connected, compact surfaces without boundary are classified by their ***genus*** g.
- An ***embedding*** of a graph G on a surface S is a drawing of G on S with no crossings and every face is a topological disk.



We denote such an embedding by (S, G), we use g for the genus of S and n for the number of vertices of G.

Cut graph

A ***cut graph*** of (S, G) is a subgraph C of G such that cutting S along C gives a topological disk.



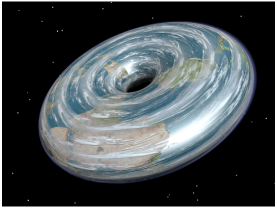
This talk is about the following problem.

Optimal cut graph

Input: Graph G embedded on S.
Output: Shortest possible cut graph C_{OPT} of (S, G).

Why should we care about (optimal) cut graphs?

- Cookie-cutter algorithm for (almost) any problem for surface-embedded graphs:
  1. Cut the surface into the plane.
  2. Solve the planar case.
  3. Recover the solution.
- More practical problems, for example texture mapping.



In all cases:

- We need ***efficient*** algorithms to do the cutting.
- The ***quality*** of the solution depends on the ***length*** of the cut graph.

Previous work on cut graphs

Optimal cut graph

Input: Graph G embedded on S.
Output: Shortest possible cut graph C_{OPT} of (S, G).

Introduced by [Erickson, Har-Peled '04].

- ***NP-hard*** by reduction from Steiner tree.
- ***Exact*** algorithm in $n^{O(g)}$.
- Polynomial algorithm to compute a $O(\log^2 g)$ ***approximation***.

Main question: Fixed parameter (in)tractability, e.g. exact algorithm in time $f(g)poly(n)$?
Our result is a FPT approximation scheme:

Theorem

Let G be a (edge-weighted) graph embedded on a surface S of genus g. For any $\varepsilon > 0$, we can compute a $(1+\varepsilon)$-approximation of the shortest cut graph of G in time $f(\varepsilon, g)n^3$ for some function f.

Section 1

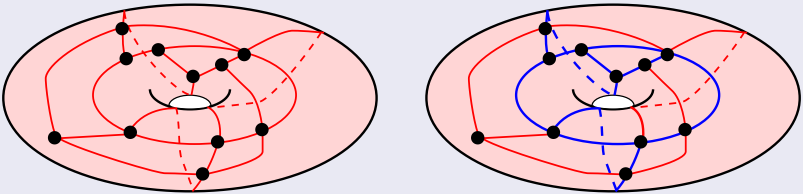
Overview of the techniques

Our techniques

- Similarity with ***connectivity problems*** like TSP, Steiner Tree, Steiner Forest, ...
- For all of these, there was a flurry of new results using the ***spanner framework*** of [Klein '05].

A ***spanner*** is a subgraph

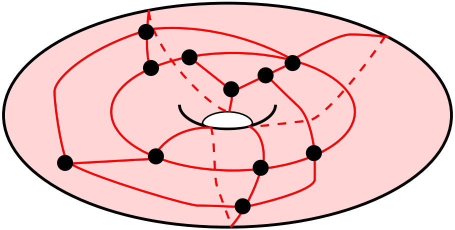
- of total length $O(f(g, \varepsilon)OPT)$.
- containing a $(1+\varepsilon)$-approximation of the optimal cut graph.



For many problems, such a spanner can be efficiently computed.

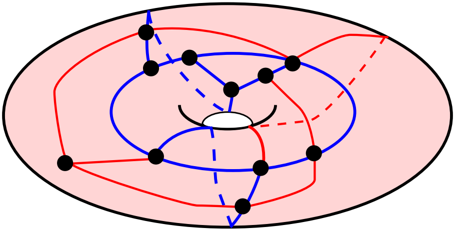
Our techniques 2

1. We compute a ***spanner*** G_{span} for the problem.
2. We ***contract*** a small set of edges of G_{span} to obtain a graph G_{tw} of reasonable treewidth.
3. We use ***dynamic programming*** on G_{tw} to compute its optimal cut graph C_{tw}.
4. We ***uncontract*** the previous edges to recover a subgraph C' of G.
5. We remove excessive edges to obtain a cut graph C



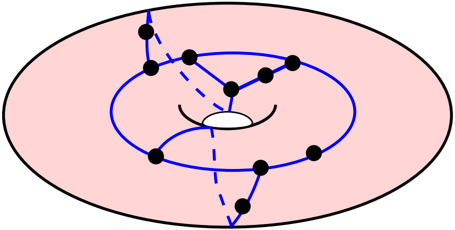
Our techniques 2

1. $\rightarrow$ We compute a ***spanner*** G_{span} for the problem.
2. We ***contract*** a small set of edges of G_{span} to obtain a graph G_{tw} of reasonable treewidth.
3. We use ***dynamic programming*** on G_{tw} to compute its optimal cut graph C_{tw}.
4. We ***uncontract*** the previous edges to recover a subgraph C' of G.
5. We remove excessive edges to obtain a cut graph C



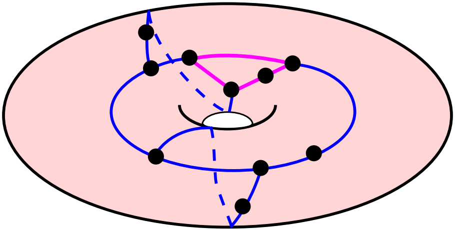
Our techniques 2

1. → We compute a ***spanner*** G_{span} for the problem.
2. We ***contract*** a small set of edges of G_{span} to obtain a graph G_{tw} of reasonable treewidth.
3. We use ***dynamic programming*** on G_{tw} to compute its optimal cut graph C_{tw}.
4. We ***uncontract*** the previous edges to recover a subgraph C' of G.
5. We remove excessive edges to obtain a cut graph C



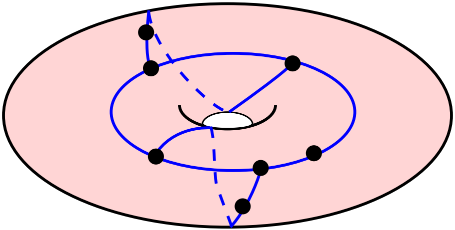
Our techniques 2

1. We compute a ***spanner*** G_{span} for the problem.
2. $\rightarrow$ We ***contract*** a small set of edges of G_{span} to obtain a graph G_{tw} of reasonable treewidth.
3. We use ***dynamic programming*** on G_{tw} to compute its optimal cut graph C_{tw}.
4. We ***uncontract*** the previous edges to recover a subgraph C' of G.
5. We remove excessive edges to obtain a cut graph C



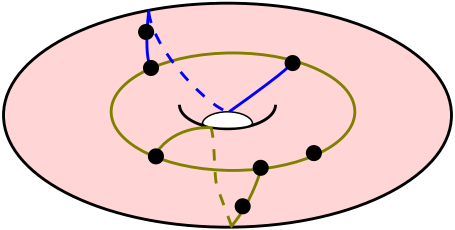
Our techniques 2

1. We compute a ***spanner*** G_{span} for the problem.
2. $\rightarrow$ We ***contract*** a small set of edges of G_{span} to obtain a graph G_{tw} of reasonable treewidth.
3. We use ***dynamic programming*** on G_{tw} to compute its optimal cut graph C_{tw}.
4. We ***uncontract*** the previous edges to recover a subgraph C' of G.
5. We remove excessive edges to obtain a cut graph C



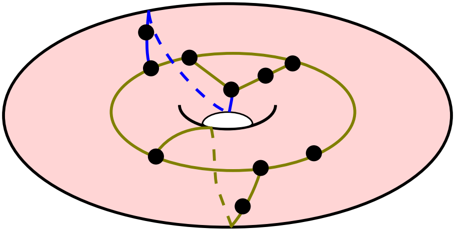
Our techniques 2

1. We compute a ***spanner*** G_{span} for the problem.
2. We ***contract*** a small set of edges of G_{span} to obtain a graph G_{tw} of reasonable treewidth.
3. $\rightarrow$ We use ***dynamic programming*** on G_{tw} to compute its optimal cut graph C_{tw}.
4. We ***uncontract*** the previous edges to recover a subgraph C' of G.
5. We remove excessive edges to obtain a cut graph C



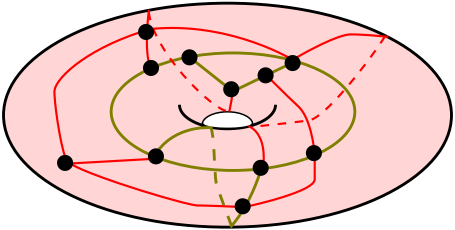
Our techniques 2

1. We compute a ***spanner*** G_{span} for the problem.
2. We ***contract*** a small set of edges of G_{span} to obtain a graph G_{tw} of reasonable treewidth.
3. We use ***dynamic programming*** on G_{tw} to compute its optimal cut graph C_{tw}.
4. $\rightarrow$ We ***uncontract*** the previous edges to recover a subgraph C' of G.
5. We remove excessive edges to obtain a cut graph C



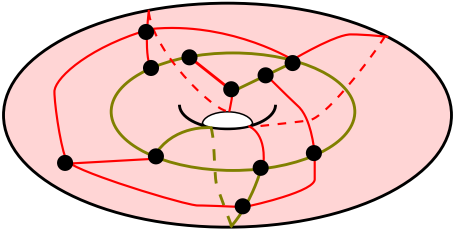
Our techniques 2

1. We compute a ***spanner*** G_{span} for the problem.
2. We ***contract*** a small set of edges of G_{span} to obtain a graph G_{tw} of reasonable treewidth.
3. We use ***dynamic programming*** on G_{tw} to compute its optimal cut graph C_{tw}.
4. $\rightarrow$ We ***uncontract*** the previous edges to recover a subgraph C' of G.
5. We remove excessive edges to obtain a cut graph C



Our techniques 2

1. We compute a ***spanner*** G_{span} for the problem.
2. We ***contract*** a small set of edges of G_{span} to obtain a graph G_{tw} of reasonable treewidth.
3. We use ***dynamic programming*** on G_{tw} to compute its optimal cut graph C_{tw}.
4. We ***uncontract*** the previous edges to recover a subgraph C' of G.
5. $\rightarrow$ We remove excessive edges to obtain a cut graph C



Our techniques 3

1. We compute a ***spanner*** G_{span} for the problem.

2. We ***contract*** a small set of edges of G_{span} to obtain a graph G_{tw} of reasonable treewidth .

3. We use ***dynamic programming*** on G_{tw} to compute its optimal cut graph C_{tw}.

4. We ***uncontract*** the previous edges to recover a subgraph C' of G.
5. We ***remove*** excessive edges.

Our techniques 3

1. We compute a ***spanner*** G_{span} for the problem.

Brick decompositions of [Borradaile, Klein, Mathieu '09], [Borradaile, Demaine, Tazari '14]

2. We ***contract*** a small set of edges of G_{span} to obtain a graph G_{tw} of reasonable treewidth .
3. We use ***dynamic programming*** on G_{tw} to compute its optimal cut graph C_{tw}.
4. We ***uncontract*** the previous edges to recover a subgraph C' of G.
5. We ***remove*** excessive edges.

Our techniques 3

1. We compute a ***spanner*** G_{span} for the problem.

Brick decompositions of [Borradaile, Klein, Mathieu '09], [Borradaile, Demaine, Tazari '14]

2. We ***contract*** a set of edges of length εOPT of G_{span} to obtain a graph G_{tw} of treewidth $O(f(g,\varepsilon))$.

3. We use ***dynamic programming*** on G_{tw} to compute its optimal cut graph C_{tw}.

4. We ***uncontract*** the previous edges to recover a subgraph C' of G.
5. We ***remove*** excessive edges.

Our techniques 3

1. We compute a ***spanner*** G_{span} for the problem.

Brick decompositions of [Borradaile, Klein, Mathieu '09], [Borradaile, Demaine, Tazari '14]

2. We ***contract*** a set of edges of length εOPT of G_{span} to obtain a graph G_{tw} of treewidth $O(f(g, \varepsilon))$.

Results of ***contraction-decomposition*** of [Demaine, Hajiaghayi, Mohar '10], [Demaine, Hajiaghayi, Kawarabayashi '11].

3. We use ***dynamic programming*** on G_{tw} to compute its optimal cut graph C_{tw}.

4. We ***uncontract*** the previous edges to recover a subgraph C' of G.
5. We ***remove*** excessive edges.

Our techniques 3

1. We compute a ***spanner*** G_{span} for the problem.

Brick decompositions of [Borradaile, Klein, Mathieu '09], [Borradaile, Demaine, Tazari '14]

2. We ***contract*** a set of edges of length εOPT of G_{span} to obtain a graph G_{tw} of treewidth $O(f(g,\varepsilon))$.

Results of ***contraction-decomposition*** of [Demaine, Hajiaghayi, Mohar '10], [Demaine, Hajiaghayi, Kawarabayashi '11].

3. We use ***dynamic programming*** on G_{tw} to compute its optimal cut graph C_{tw}.

Surface-cut decompositions of [Rué, Sau and Thilikos '14]

4. We ***uncontract*** the previous edges to recover a subgraph C' of G.
5. We ***remove*** excessive edges.

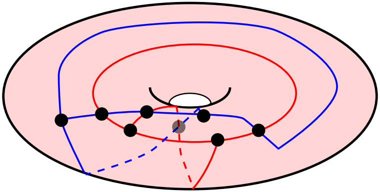
Section 2

A spanner for the optimal cut-graph

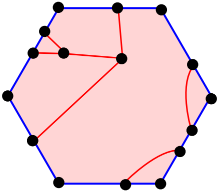
A ***spanner*** is a subgraph

- of total length $O(f(g, \varepsilon)OPT)$.
- containing a $(1 + \varepsilon)$-approximation of the optimal cut graph.

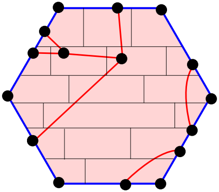
- We start with a $O(\log^2 g)$ approximation of C_{OPT}, and cut along it.



- C_{OPT} is now a ***forest*** in a disk D of boundary length $|\partial D| = O(\log^2 g)OPT$.

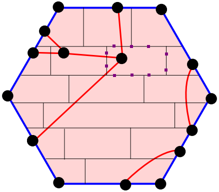


- We decompose the disk into ***bricks*** of length $f(\varepsilon)|\partial D| = f(\varepsilon, g)OPT$.

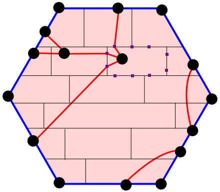


Step 1. Computing a spanner

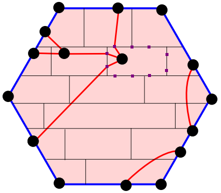
- We put regularly spaced ***portals*** on the boundaries of the bricks.



- We prove a ***structure theorem*** showing that 'pushing' C_{OPT} so that it goes through the portals does not make it much longer.



- We prove a ***structure theorem*** showing that 'pushing' C_{OPT} so that it goes through the portals does not make it much longer.

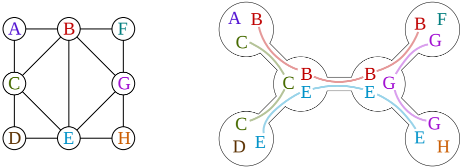


- The spanner G_{span} is obtained by computing Steiner trees for every possible subset of portals in every brick.
- Proofs similar to [Borradaile, Klein, Mathieu '09],[Borradaile, Demaine, Tazari '14]

Section 3

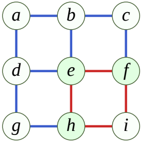
Computing an optimal cut graph in bounded tree-width

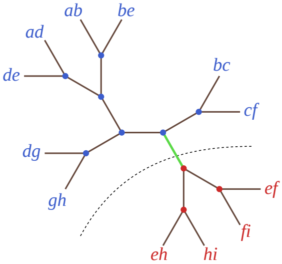
- Tree decompositions are commonly used as a basis for dynamic programming .



- For this problem they lack a ***topological*** structure.
- Instead we will rely on a variant of branch decompositions called ***Surface-cut decompositions*** [Rué, Sau, Thilikos '14].

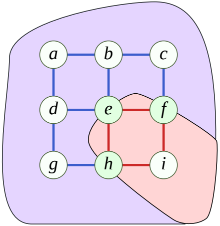
Branch decomposition:
Tree T such that every edge of the tree partitions the edges of G into subgraphs G_1 and G_2 such that $|G_1 \cap G_2|$ is not too big .

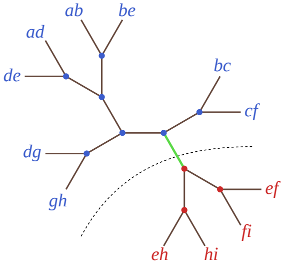




Step 3. Surface-cut decompositions

Surface-cut decomposition:
Tree T such that every edge of the tree partitions the edges of G ***and the surface*** S into subgraphs G_1 and G_2 ***and connected subsurfaces*** S_1 ***and*** S_2 such that $G_i \subseteq S_i$ and $S_1 \cap S_2$ is not too complicated.





- With a surface-cut decomposition, it is easy to design a dynamic programming algorithm to compute an optimal cut graph.

Step 3. Computing a surface-cut decomposition

Theorem (Rué, Sau, Thilikos '14)

*Given a graph G **polyhedrally** embedded [a] on a surface of genus g and branch-width k, one can compute a surface-cut decomposition of G of width $O(g+k)$ in time $2^{O(k)}n^3$.*

[a] G is polyhedrally embedded if G is 3-connected and the smallest length of a non-contractible noose is at least 3 or if G is a clique and it has at most 3 vertices

We provide two ways to circumvent the polyhedrality hypothesis:

- We provide another algorithm which does not need it, relying on a lemma of Inkmann.
- We provide a construction to make a graph embedding polyhedral while controlling its branch-width.

- Fixed parameter (in)tractability of this problem?
- What is the best approximation ratio for polynomial (in n and g).

Perspectives

- Fixed parameter (in)tractability of this problem?
- What is the best approximation ratio for polynomial (in n and g).

Thank you !