Approximation Limits and Algorithms in Practice for the Maximum Planar Subgraph Problem

Markus Chimani, Ivo Hedtke, and Tilo Wiedera
Osnabrück University, Germany

Karsten Klein
Konstanz University, Germany
Problem definition

Given a connected, simple Graph $G$, we want to find a planar subgraph $H$ of $G$ such that $|E(H)|$ is maximal.

Given non-planar graph
Problem definition

Given a connected, simple graph $G$, we want to find a planar subgraph $H$ of $G$ such that $|E(H)|$ is maximal.

Given non-planar graph

planar subgraph
Problem definition

Given a connected, simple graph $G$, we want to find a planar subgraph $H$ of $G$ such that $|E(H)|$ is maximal.

Given non-planar graph

planar subgraph

Maximal planar subgraph
Problem definition

Given a connected, simple graph $G$, we want to find a planar subgraph $H$ of $G$ such that $|E(H)|$ is maximal.

Given non-planar graph

Maximal planar subgraph

Planar subgraph

Maximum planar subgraph
Problem definition

Given a connected, simple graph $G$, we want to find a planar subgraph $H$ of $G$ such that $|E(H)|$ is maximal.

Given non-planar graph

planar subgraph

Maximal planar subgraph

Maximum planar subgraph

MaxSNP-hard
Known algorithms for approximating MPS

Euler: for any simple, planar graph $G$ it follows that $|E(G)| \leq 3|V(G)| - 6$.

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Runtime</th>
<th>Apx.-ratio</th>
<th>Tight?</th>
</tr>
</thead>
<tbody>
<tr>
<td>Spanning Tree</td>
<td>$O(m)$</td>
<td>$1/3 \approx 33.3%$</td>
<td>Yes</td>
</tr>
<tr>
<td>Cactus</td>
<td>$O(mn)$</td>
<td>$7/18 \approx 38.9%$</td>
<td>Yes</td>
</tr>
<tr>
<td>Diamond</td>
<td>$O(mn^2)$</td>
<td>$13/33 \approx 39.4%$</td>
<td>Unknown</td>
</tr>
<tr>
<td>Maximum Cactus</td>
<td>$O(m^{2/3}n \log^6 n)$</td>
<td>$4/9 \approx 44.4%$</td>
<td>Yes</td>
</tr>
</tbody>
</table>
Known algorithms for approximating MPS

Euler: for any simple, planar graph $G$ it follows that $|E(G)| \leq 3|V(G)|-6$.

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Runtime</th>
<th>Apx.-ratio</th>
<th>Tight?</th>
</tr>
</thead>
<tbody>
<tr>
<td>Spanning Tree</td>
<td>$O(m)$</td>
<td>$1/3 \approx 33.3%$</td>
<td>Yes</td>
</tr>
<tr>
<td>Cactus</td>
<td>$O(mn)$</td>
<td>$7/18 \approx 38.9%$</td>
<td>Yes</td>
</tr>
<tr>
<td>Diamond</td>
<td>$O(mn^2)$</td>
<td>$13/33 \approx 39.4%$</td>
<td>Unknown</td>
</tr>
<tr>
<td>Maximum Cactus</td>
<td>$O(m^{2/3}n \log^6 n)$</td>
<td>$4/9 \approx 44.4%$</td>
<td>Yes</td>
</tr>
</tbody>
</table>

Given: non-planar graph $G=(V,E)$

Cactus algorithm [Calinescu et al. '98]

$S := (V, \emptyset)$

for each triangle $T \in G$ :
  if all nodes of $T$ are in different components of $S$ : $S := S+T$

for each edge $e \in E$ :
  if both nodes of $e$ are in different components of $S$ : $S := S+e$

return $S$
Given: non-planar graph $G=(V,E)$

$S := (V, \emptyset)$

for each diamond $D \in G$:
    if all nodes of $D$ are in different components of $S$:
        $S := S + D$

for each triangle...
for each edge...
return $S$

Known algorithms for approximating MPS

Euler: for any simple, planar graph $G$ it follows that $|E(G)| \leq 3|V(G)| - 6$.

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Runtime</th>
<th>Apx.-ratio</th>
<th>Tight?</th>
</tr>
</thead>
<tbody>
<tr>
<td>Spanning Tree</td>
<td>$O(m)$</td>
<td>$1/3 \approx 33.3%$</td>
<td>Yes</td>
</tr>
<tr>
<td>Cactus</td>
<td>$O(mn)$</td>
<td>$7/18 \approx 38.9%$</td>
<td>Yes</td>
</tr>
<tr>
<td>Diamond</td>
<td>$O(mn^2)$</td>
<td>$13/33 \approx 39.4%$</td>
<td>Unknown</td>
</tr>
<tr>
<td>Maximum Cactus</td>
<td>$O(m^{2/3}n \log^6 n)$</td>
<td>$4/9 \approx 44.4%$</td>
<td>Yes</td>
</tr>
</tbody>
</table>
Non-planar graph $G = (V, E)$

$S := (V, \emptyset)$

for each diamond $D \in G$

  if all nodes of $D$ are in different components of $S$

  $S := S + D$

for each triangle...

for each edge...

return $S$

Euler: for any simple, planar graph $G$ it follows that $|E(G)| \leq 3|V(G)| - 6$.

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Runtime</th>
<th>Apx.-ratio</th>
<th>Tight?</th>
</tr>
</thead>
<tbody>
<tr>
<td>Spanning Tree</td>
<td>$O(m)$</td>
<td>$1/3 \approx 33.3%$</td>
<td>Yes</td>
</tr>
<tr>
<td>Cactus</td>
<td>$O(mn)$</td>
<td>$7/18 \approx 38.9%$</td>
<td>Yes</td>
</tr>
<tr>
<td>Diamond</td>
<td>$O(mn^2)$</td>
<td>$13/33 \approx 39.4%$</td>
<td>Unknown</td>
</tr>
</tbody>
</table>

Finding better Approximations for MPS is tough!

[Chalermsook & Schmid, to appear]
Part I: Computing planar subgraphs in practice

Comparison of planar subgraph heuristics with respect to

- Runtime
- Solution quality
- Implementation complexity
Part I: Computing planar subgraphs in practice

Comparison of planar subgraph heuristics with respect to

- Runtime
- Solution quality
- Implementation complexity

Implemented algorithms

- Naive ($N_i$)
- Augmented planarity test (BM, BM+)
- Cactus ($C, C+$)
- ILP-based, optimal algorithm (ILP)
Naïve algorithm

Given: non-planar graph $G=(V,E)$

$H := (V,\emptyset)$

for each $e \in E$:
    if $H+e$ is planar: // requires planarity test $\rightarrow O(n)$
        $H := H+e$

return $H$
Given: non-planar graph $G=(V,E)$

$H := (V,\emptyset)$

for each $e \in E$ :

  if $H+e$ is planar :    \textit{// requires planarity test } \rightarrow O(n)$
      $H := H+e$

return $H$

finds a maximal planar subgraph in $O(nm)$ time

+ very simple \textit{(when using a free library for planarity test)}
+ solution is maximal
  – running time
Given: non-planar graph $G=(V,E)$

$H := (V, \emptyset)$

for each $e \in E$:

    if $H+e$ is planar:  // requires planarity test $\rightarrow O(n)$
        $H := H+e$

return $H$

finds a maximal planar subgraph in $O(nm)$ time

+ very simple (when using a free library for planarity test)
+ solution is maximal
- running time
Augmented planarity test

**Idea:** Run a linear time planarity test. When detecting an edge that violates planarity, remove it and continue.
Augmented planarity test

**Idea:** Run a linear time planarity test. When detecting an edge that violates planarity, remove it and continue.

- E.g., within [Boyer, Myrvold 99]:
  - *theoretically* very simple add-on…
  …but you have to understand the planarity test and its implementation, and do some dirty work in book keeping
  - solution not maximal
  - gain maximality by running **Naïve** afterwards (but loose linear time)
  + free implementations of the planarity test available
Augmented planarity test

**Idea:** Run a linear time planarity test. When detecting an edge that violates planarity, remove it and continue.

- **E.g., within [Boyer, Myrvold 99]:**
  - **theoretically** very simple add-on…
    - …but you have to understand the planarity test and its implementation, and do some dirty work in book keeping
  - solution **not** maximal
    - gain maximality by running **Naïve** afterwards (but loose linear time)
    - free implementations of the planarity test available

- **Algorithm [Hsu 05] based on [Hsu, Shih 99]:** very similar to above, but…
  - guarantees maximal planar subgraph in linear time
  - no (free) implementation of it nor the planarity test?
ILP-based approach

$$\max \{ \sum_{e \in E} x_e \mid \sum_{e \in K} x_e \leq |K|-1 \text{ for all Kuratowski subdivisions } K \}$$

Solve an ILP via Branch-and-Cut [Jünger, Mutzel 96] → exact solution

+ solution is maximum
+ formulation is relatively simple to implement (given an ILP-framework and planarity test).
  Simple cut separation (rounding solution and testing planarity)
  - high running time (formally exponential time)
**Machine & Implementation**

Intel Xeon E5-2430 v2, 2.5 GHz, Debian 8; each process: single core, 4GB

**All**: C++ (g++ 5.3.1 –O3, 64bit), as part of OGDF (GPL, www.ogdf.net)

**ILP**: CPLEX 12.6
Experimental setup

Machine & Implementation
Intel Xeon E5-2430 v2, 2.5 GHz, Debian 8; each process: single core, 4GB
**All:** C++ (g++ 5.3.1 –O3, 64bit), as part of OGDF (GPL, www.ogdf.net)
**ILP:** CPLEX 12.6

Instances

Established benchmark sets
- North/ATT (ca. 400)
- Rome (ca. 8,000)
- SteinLib (ca. 600)

Generated Instances
- Barabási-Albert → Scale-free graphs
- Random regular graphs → Expander graphs with high probability
  - \( n \in \{100,1,000,10,000\} \), \( m/n \in \{2,3,5,10,20\} \),
  - 20 instances per type & parameters
• All upcoming plots contain average values against the number of nodes on Rome graphs (clustered to the nearest multiple of 10)
ILP takes too long

- All upcoming plots contain average values against the number of nodes on Rome graphs (clustered to the nearest multiple of 10)
- ILP has a critically **low success rate** on Rome graphs, arguably simplest instances
- Duplicating runtime limits and applying strong preprocessing* does not help
- Expander graphs on 30 nodes and 90 edges do **not terminate within 48 hours**

*Chimani, Gutwenger 09*
ILP takes too long

- All upcoming plots contain average values against the number of nodes on Rome graphs (clustered to the nearest multiple of 10)
- ILP has a critically **low success rate** on Rome graphs, arguably simplest instances
- Duplicating runtime limits and applying strong preprocessing* **does not help**
- Expander graphs on 30 nodes and 90 edges do **not terminate within 48 hours**

* [Chimani, Gutwenger 09]

→ **Heuristic approaches are required!**
The cost of maximality

Maximum Planar Subgraph

Tilo Wiedera, tilowiedera@uos.de
The cost of maximality

- Cactus+Naive yields best solutions.
- Augm. PT is a weak starting point for naive maximization.
- Augm. PT and Cactus have similarly good runtime but Augm. PT yields clearly better solutions.
The cost of maximality

- Cactus+Naive yields best solutions.
- Augm. PT is a weak starting point for naive maximization.
- Augm. PT and Cactus have similarly good runtime but Augm. PT yields clearly better solutions.

→ If runtime is crucial, omit maximality and use augmented planarity test.
→ If maximality is required, use cactus algorithm followed by naive.
Crossing minimization using planar subgraphs

Use large planar subgraphs as a starting point for the planarization heuristic to minimize crossings.

Stronger subgraph algorithms lead to smaller crossing numbers

Simple
[Batini et al. 1984]
Initial subgraphs are important
Crossing minimization using planar subgraphs

Use large planar subgraphs as a starting point for the **planarization heuristic** to minimize crossings.

Stronger subgraph algorithms lead to smaller crossing numbers.

**Simple**
[Batini et al. 1984]
Initial subgraphs are **important**

**State-of-the-art**
[Chimani, Gutwenger 12]
initial subgraphs become more and more irrelevant
Good approximation is important (even in practice!)

Formerly best known, well implementable approximation algorithm (cactus, 7/18) is the foundation of the practically strongest algorithm.

→ Investigating better approximation algorithms is worthwhile also from a practical perspective.

• Can we achieve better approximations?

• What classes of algorithms are the most promising?

• What are the problems that we face when employing certain algorithmic approaches?
Part II: Limits of greedily approximating MPS

Seemingly promising ideas for novel greedy MPS approximations

- based on planarity testing (DFS)
- what about BFS-based algorithms?
- Greedy edge selection
  Iteratively pick an edge that minimizes the number of arising forbidden edges
- Greedy triangle selection (like above)
- Simple subgraph selection
  Iteratively pick a (dense & planar) subgraph, e.g., $K_4$.

<table>
<thead>
<tr>
<th>Algorithm class</th>
<th>bound</th>
<th>subproblem</th>
</tr>
</thead>
<tbody>
<tr>
<td>DFS-based</td>
<td>2 / 3</td>
<td>NP-hard</td>
</tr>
<tr>
<td>BFS-based</td>
<td>2 / 3</td>
<td>NP-hard</td>
</tr>
<tr>
<td>Gr. edge selection</td>
<td>1 / 3</td>
<td>-</td>
</tr>
<tr>
<td>Gr. triangle selection</td>
<td>7 / 18</td>
<td>NP-hard</td>
</tr>
<tr>
<td>Gr. subgraph selection</td>
<td>1 / 2</td>
<td>NP-hard</td>
</tr>
</tbody>
</table>

\[
\begin{align*}
E_r & \quad (V \text{ and } E) \\
B_v & \quad \forall v \in V
\end{align*}
\]

\[
\begin{align*}
B_{r,v}^{m+1} & \quad \forall v \in V \\
V & \quad \text{and } E \\
B_{v,s}^{m+1} & \quad \forall v \in V
\end{align*}
\]
Solving MPS-DFS (or MPS-BFS) optimally approximates MPS with at most 2/3

**DFS Idea:** path as degenerate DFS-tree cuts through 1/3 of the edges

**BFS Idea:** triangulated but 3-colorable graph, BFS-tree with only 3 levels (second level has 3 nodes corresponding to the colors)
Simple subgraph selection approximates MPS with at most $1/2$.
Summary

- Approximating MPS is **harder than it seems**
  19 year old 4/9-approximation is still the best known result

- Approximation is **relevant for practical** algorithms
Approximating MPS is **harder than it seems**
19 year old 4/9-approximation is still the best known result

Approximation is **relevant for practical** algorithms

**Open Questions**

- Is the algorithm by Chalermsook & Schmid tight?
  How does it perform in practice?

- Is the approximation ratio of **1/2** achievable
  by **selecting denser subgraphs**?

- Can we bound the **approximation ratio in general**?