Approximation Algorithms for NP-Complete Problems on Planar Graphs

Marius Knabben

December 10, 2013
1 Introduction

2 The Approximation Algorithm (for maximization problems)

3 Maximum Independent Set, an example

4 Running Time

5 Modification for minimization problems

6 Conclusion
1 Introduction
- What is our goal?
- Outerplanar Graphs
- Basic idea of the algorithm

2 The Approximation Algorithm (for maximization problems)

3 Maximum Independent Set, an example

4 Running Time

5 Modification for minimization problems

6 Conclusion
Goal for this Presentation

Develop approximation techniques for several NP-Complete problems on planar graphs.
Goal for this Presentation

Develop approximation techniques for several NP-Complete problems on planar graphs.

- Maximization and minimization problems.
- At most \((k + 1)/k\) optimal for minimization problems.
- At least \(k/(k + 1)\) optimal for maximization problems.
- *Polynomial* time effort.
Goal for this Presentation

Develop approximation techniques for several NP-Complete problems on planar graphs.

- Maximization and minimization problems.
- At most \((k + 1)/k\) optimal for minimization problems.
- At least \(k/(k + 1)\) optimal for maximization problems.
- \emph{Polynomial} time effort.

In this presentation we show for Maximum Independent Set how we find an approximate solution on a planar graph. \((k = 1)\)
1- and k-Outerplanar Graphs

Definition (1-Outerplanar Graphs)

A graph $G$ is outerplanar if $G$ can be drawn in the plane without crossings of edges and all vertices belong to the outer face.
**1- and k-Outerplanar Graphs**

**Definition (1-Outerplanar Graphs)**

A graph $G$ is outerplanar if $G$ can be drawn in the plane without crossings of edges and all vertices belong to the outer face.
Definition (1-Outerplanar Graphs)

A graph $G$ is outerplanar if $G$ can be drawn in the plane without crossings of edges and all vertices belong to the outer face.
1- and k-Outerplanar Graphs

Definition (k-Outerplanar Graphs)

A graph $G$ is $k$-outerplanar if $G$ can be drawn in the plane without crossings of edges and by removing all vertices that belong to the outer face the resulting graph is $(k-1)$-outerplanar.
1- and k-Outerplanar Graphs

**Definition (k-Outerplanar Graphs)**

A graph $G$ is $k$-outerplanar if $G$ can be drawn in the plane without crossings of edges and by removing all vertices that belong to the outer face the resulting graph is $(k-1)$-outerplanar.
1- and k-Outerplanar Graphs

Definition (k-Outerplanar Graphs)

A graph $G$ is $k$-outerplanar if $G$ can be drawn in the plane without crossings of edges and by removing all vertices that belong to the outer face the resulting graph is $(k-1)$-outerplanar.
1- and k-Outerplanar Graphs

Definition (k-Outerplanar Graphs)

A graph $G$ is k-outerplanar if $G$ can be drawn in the plane without crossings of edges and by removing all vertices that belong to the outer face the resulting graph is (k-1)-outerplanar.
Definition (k-Outerplanar Graphs)

A graph $G$ is $k$-outerplanar if $G$ can be drawn in the plane without crossings of edges and by removing all vertices that belong to the outer face the resulting graph is $(k-1)$-outerplanar.
1- and k-Outerplanar Graphs

**Definition (k-Outerplanar Graphs)**

A graph $G$ is $k$-outerplanar if $G$ can be drawn in the plane without crossings of edges and by removing all vertices that belong to the outer face the resulting graph is $(k-1)$-outerplanar.

For every planar graph $G$, there is a $k$ so that $G$ is $k$-outerplanar.
Level i-nodes
Level i-nodes
Level i-nodes
Level i-nodes
Approximation

The algorithm *divides* the given graph and then *conquers* on its subgraphs.
Approximation

The algorithm **divides** the given graph and then **conquers** on its subgraphs.

**Divide**

- **k+1** Divisions
- Multiple components per division
Approximation

The algorithm divides the given graph and then conquers on its subgraphs.

**Divide**
- $k+1$ Divisions
- Multiple components per division

**Conquer**
- For each division: Take the union of each components solution.
- The maximum of all the unions is the solution.
1 Introduction

2 The Approximation Algorithm (for maximization problems)
   - The Algorithm
   - Visualization

3 Maximum Independent Set, an example

4 Running Time

5 Modification for minimization problems

6 Conclusion
The Algorithm

Input: The graph $G$ and $k$.

1. Calculate the level for each node
The Algorithm

Input: The graph $G$ and $k$.

1. Calculate the level for each node
2. for $i = 1, 2, ..., k + 1$
   1. $G^i$ is constructed by deleting all level-$(i, (k + 1) + i, 2(k + 1) + i, ...)$ nodes of $G$
   2. $G^i$ is composed of multiple components $G^i_1$ to $G^i_m$ which are at most $k$ outerplanar

Output: $S$
The Algorithm

Input: The graph \( G \) and \( k \).

1. Calculate the level for each node
2. for \( i = 1, 2, \ldots, k + 1 \)
   1. \( G^i \) is constructed by deleting all level-\((i, (k + 1) + i, 2(k + 1) + i, \ldots)\) nodes of \( G \)
   2. \( G^i \) is composed of multiple components \( G_1^i \) to \( G_m^i \) which are at most \( k \) outerplanar
3. for \( j = 1, \ldots, m \)
   1. \( S_j^i = \text{Maximum Independent Set}(G_j^i) \)
The Algorithm

Input: The graph $G$ and $k$.

1. Calculate the level for each node
2. for $i = 1, 2, ..., k + 1$
   1. $G^i$ is constructed by deleting all level-$(i, (k + 1) + i, 2(k + 1) + i, ...)$ nodes of $G$
   2. $G^i$ is composed of multiple components $G^i_1$ to $G^i_m$ which are at most $k$ outerplanar
3. for $j = 1, ..., m$
   1. $S^i_j = \text{Maximum Independent Set}(G^i_j)$
4. $S^i = \bigcup_j S^i_j$
3. $S = \max_i S^i$
**The Algorithm**

Input: The graph $G$ and $k$.

1. Calculate the level for each node
2. for $i = 1, 2, \ldots, k + 1$
   1. $G^i$ is constructed by deleting all level-$(i, (k + 1) + i, 2(k + 1) + i, \ldots)$ nodes of $G$
   2. $G^i$ is composed of multiple components $G^i_1$ to $G^i_m$ which are at most $k$ outerplanar
3. for $j = 1, \ldots, m$
   1. $S^i_j = \text{Maximum Independent Set}(G^i_j)$
4. $S^i = \bigcup_j S^i_j$
3. $S = \max_i S^i$

Output: $S$
Visualization - Divide

$G$ is 8-Outerplanar, $k = 3$
Visualization - Divide

$G$ is 8-Outerplanar, $k = 3$

$i = 1$

Level 2
Level 3
Level 4
Level 6
Level 7
Level 8
Visualization - Divide

$G$ is 8-Outerplanar, $k = 3$

$i = 2$

- Level 1
- Level 3
- Level 4
- Level 5
- Level 7
- Level 8
Visualization - Divide

$G$ is 8-Outerplanar, $k = 3$

$i = 3$

- Level 1
- Level 2
- Level 4
- Level 5
- Level 6
- Level 8
Visualization - Divide

$G$ is 8-Outerplanar, $k = 3$

\[ i = 4 \]
The Approximation Algorithm (for maximization problems)

- Visualization

Visualization - Conquer

Now we have four approximate solutions ($S^1$ to $S^4$)
Now we have four approximate solutions ($S^1$ to $S^4$)

- In at least one of those four subgraphs at most $1/4$ of all nodes of $G$ are deleted
Now we have four approximate solutions ($S^1$ to $S^4$)

- In at least one of those four subgraphs at most 1/4 of all nodes of $G$ are deleted
- This subgraphs solution is at least 3/4 optimal
The Approximation Algorithm (for maximization problems)

Visualization

Visualization - Conquer

Now we have four approximate solutions ($S^1$ to $S^4$)

In at least one of those four subgraphs at most $1/4$ of all nodes of $G$ are deleted

This subgraphs solution is at least $3/4$ optimal

Reintroducing $k$: The solution is $k/(k + 1)$ optimal
Introduction

The Approximation Algorithm (for maximization problems)

Maximum Independent Set, an example
- Step 1: Create a Tree
- Step 2: Calculate the Maximum Independent Set

Running Time

Modification for minimization problems

Conclusion
Our Graph
Our Graph
Maximum Independent Set, an example

Step 1: Create a Tree

Tree
Maximum Independent Set, an example

Step 1: Create a Tree

Tree

(a,a)
Maximum Independent Set, an example

Step 1: Create a Tree

Tree

(a,a)

(a,b)
Maximum Independent Set, an example

Step 1: Create a Tree

Tree

(a,a)  (a,b)  (b,d)
Maximum Independent Set, an example

Step 1: Create a Tree

Tree

(a,a)
(a,b)  (b,d)
(b,c)
Maximum Independent Set, an example

Step 1: Create a Tree

Tree

(a,a)  (a,b)  (b,d)  (b,c)  (c,d)
Maximum Independent Set, an example

Step 1: Create a Tree

Tree
Maximum Independent Set, an example

Step 1: Create a Tree

Tree

\[
\begin{align*}
(a,a) & \quad (a,b) \quad (b,d) \quad (d,e) \quad (e,e) \\
(a,b) & \quad (b,c) \quad (c,d)
\end{align*}
\]
Maximum Independent Set, an example

Step 1: Create a Tree

Tree
Maximum Independent Set, an example

Step 1: Create a Tree

Tree
Maximum Independent Set, an example

Step 1: Create a Tree

Tree

```
(a,a)
/   \
(a,b) (b,d) (d,e) (e,e)
/   /   /   /
(b,c) (c,d) (e,f) (f,g) (g,e)
```

```
g e d b c f
```

```
(a,a) (a,b) (b,d) (b,c) (c,d) (d,e) (e,e) (e,f) (f,g) (g,e)
```
Maximum Independent Set, an example

Step 1: Create a Tree
Maximum Independent Set, an example

Step 1: Create a Tree
Algorithm (for Maximum Independent Set)

- Recursively defined
- Only once executed per tree-node
Maximum Independent Set, an example

Step 2: Calculate the Maximum Independent Set

Algorithm (for Maximum Independent Set)

- Recursively defined
- Only once executed per tree-node
- Generates small tables for each leaf node and merges them to their parent nodes
- The root’s table contains the solution
Algorithm (for Maximum Independent Set)

- Recursively defined
- Only once executed per tree-node
- Generates small tables for each leaf node and merges them to their parent nodes
- The root’s table contains the solution

There is also a linear time algorithm for k-outerplanar graphs. But much more complicated. (Too much for this presentation)
### Table Structure

A table $T_{ab}$ representing an edge from $a$ to $b$ in our graph consists of four entries: $(t_{\emptyset}, t_a, t_b, t_{ab})$
Table Structure

A table $T_{ab}$ representing an edge from $a$ to $b$ in our graph consists of four entries: $(t_\emptyset, t_a, t_b, t_{ab})$

- $t_\emptyset$: Neither $a$ nor $b$ are in our solution
- $t_a$: $a$ but not $b$ is in our solution
- $t_b$: $b$ but not $a$ is in our solution
- $t_{ab}$: $a$ and $b$ are in our solution
A table $T_{ab}$ representing an edge from $a$ to $b$ in our graph consists of four entries: $(t_{\emptyset}, t_a, t_b, t_{ab})$

- $t_{\emptyset}$: Neither $a$ nor $b$ are in our solution
- $t_a$: $a$ but not $b$ is in our solution
- $t_b$: $b$ but not $a$ is in our solution
- $t_{ab}$: $a$ and $b$ are in our solution

The table for a leafnode $(a, b)$ is always like this: $T_{ab} = (0, 1, 1, -\infty)$
Maximum Independent Set, an example

Step 2: Calculate the Maximum Independent Set

On our Tree

\[
\begin{align*}
T_{ab} &= (0, 1, 1, -\infty) \\
T_{bc} &= (0, 1, 1, -\infty) \\
T_{cd} &= (0, 1, 1, -\infty) \\
T_{da} &= (0, 1, 1, -\infty)
\end{align*}
\]
Merge two neighboring tables $T_{ab}$ and $T'_{bc}$:

<table>
<thead>
<tr>
<th></th>
<th>$t_0$</th>
<th>$t_a$</th>
<th>$t_b$</th>
<th>$t_{ab}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$t_0'$</td>
<td>$w$</td>
<td>$x$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$t_b'$</td>
<td></td>
<td></td>
<td>$w' + 1$</td>
<td>$x' + 1$</td>
</tr>
<tr>
<td>$t_c'$</td>
<td>$y$</td>
<td>$z$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$t_{bc}'$</td>
<td></td>
<td></td>
<td>$y' + 1$</td>
<td>$z' + 1$</td>
</tr>
</tbody>
</table>
Merge two neighboring tables $T_{ab}$ and $T'_{bc}$:

<table>
<thead>
<tr>
<th></th>
<th>$t_0'$</th>
<th>$t_a$</th>
<th>$t_b$</th>
<th>$t_{ab}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$t_0'$</td>
<td>$w$</td>
<td>$x$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$t_b'$</td>
<td></td>
<td></td>
<td>$w' + 1$</td>
<td>$x' + 1$</td>
</tr>
<tr>
<td>$t_c'$</td>
<td>$y$</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$t_{bc}'$</td>
<td></td>
<td></td>
<td>$y' + 1$</td>
<td>$z' + 1$</td>
</tr>
</tbody>
</table>

$T_{ac} = (\max(w, w'), \max(x, x'), \max(y, y'), \max(z, z'))$
Merge

\[
T_{ab} = (0, 1, 1, -\infty) \\
T_{bc} = (0, 1, 1, -\infty) \\
T_{cd} = (0, 1, 1, -\infty) \\
T_{da} = (0, 1, 1, -\infty)
\]

Now we got \((a, a)\) as our parent node. We have to adjust:

1. It is impossible to have node \(a\) in our solution and at the same
time not \(a\) in our solution.

\[\Rightarrow\]
2. \(a\) can't be twice in our solution.

\[\Rightarrow\]
3. Last entry \(-1\)

So our Solution is \(\max(2, -\infty, -\infty, 2) = 2\).
Maximum Independent Set, an example

Step 2: Calculate the Maximum Independent Set

Merge

(a,a)

\[ T_{ac} = (1, 1, 1, 2) \]

\[ T_{cd} = (0, 1, 1, -\infty) \]

\[ T_{da} = (0, 1, 1, -\infty) \]

Now we got (a,a) as our parent node. We have to adjust:

1. It is impossible to have node a in our solution and at the same time not a in our solution.

2. a can’t be twice in our solution.

So our Solution is max(2, -\infty, -\infty, 2) = 2.
Maximum Independent Set, an example

Step 2: Calculate the Maximum Independent Set

Now we got ($a, a$) as our parent node. We have to adjust:

1. It is impossible to have node $a$ in our solution and at the same time not $a$ in our solution.
2. $a$ can't be twice in our solution.
3. Last entry $−1$

So our Solution is max(2, $−\infty$, $−\infty$, 2) = 2.
Maximum Independent Set, an example

Step 2: Calculate the Maximum Independent Set

Merge

Now we got \((a, a)\) as our parent node. We have to adjust:

1. It is impossible to have node \(a\) in our solution and at the same time not \(a\) in our solution.

\[
T_{aa} = (2, 2, 2, 3)
\]
Maximum Independent Set, an example

Step 2: Calculate the Maximum Independent Set

Now we got \((a, a)\) as our parent node. We have to adjust:

1. It is impossible to have node \(a\) in our solution and at the same time not \(a\) in our solution. \(\Rightarrow\) Second and third entry \(= -\infty\)
Merge

Now we got \((a, a)\) as our parent node. We have to adjust:

1. It is impossible to have node \(a\) in our solution and at the same time not \(a\) in our solution. \(\Rightarrow\) Second and third entry = \(-\infty\)
2. \(a\) can’t be twice in our solution.
Maximum Independent Set, an example

Step 2: Calculate the Maximum Independent Set

Merge

Now we got \((a, a)\) as our parent node. We have to adjust:

1. It is impossible to have node \(a\) in our solution and at the same time not \(a\) in our solution. \(\Rightarrow\) Second and third entry = \(-\infty\)
2. \(a\) can’t be twice in our solution. \(\Rightarrow\) Last entry \(-1\)
Maximum Independent Set, an example

Step 2: Calculate the Maximum Independent Set

Merge

\[ T_{aa} = (2, -\infty, -\infty, 2) \]

Now we got \((a, a)\) as our parent node. We have to adjust:

1. It is impossible to have node \(a\) in our solution and at the same time not \(a\) in our solution. \(\Rightarrow\) Second and third entry \(= -\infty\)

2. \(a\) can’t be twice in our solution. \(\Rightarrow\) Last entry \(-1\)

So our Solution is \(\max(2, -\infty, -\infty, 2) = 2\).
1 Introduction

2 The Approximation Algorithm (for maximization problems)

3 Maximum Independent Set, an example

4 Running Time

5 Modification for minimization problems

6 Conclusion
Running Time

Assume:

$n$ is the number of nodes in $G$.

$\delta$ is the running time of the underlying algorithm (per node).
Running Time

Assume:

- \( n \) is the number of nodes in \( G \).
- \( \delta \) is the running time of the underlying algorithm (per node).

- Calculate level for each node: Linear time, *proven by Hopcroft and Tarjan [1974]*
Running Time

Assume:

- $n$ is the number of nodes in $G$.
- $\delta$ is the running time of the underlying algorithm (per node).

- Calculate level for each node: Linear time, proven by Hopcroft and Tarjan [1974]
- Generating $k$ subgraphs of $G$: $O(kn)$

Running Time

Assume:

- $n$ is the number of nodes in $G$.
- $\delta$ is the running time of the underlying algorithm (per node).

- Calculate level for each node: Linear time, proven by Hopcroft and Tarjan [1974]
- Generating $k$ subgraphs of $G$: $O(kn)$
Assume:

- $n$ is the number of nodes in $G$.
- $\delta$ is the running time of the underlying algorithm (per node).

- Calculate level for each node: Linear time, proven by Hopcroft and Tarjan [1974]
- Generating $k$ subgraphs of $G$: $O(kn)$
- Running the problem’s algorithm: $O(\delta kn)$
Running Time

Assume:

- $n$ is the number of nodes in $G$.
- $\delta$ is the running time of the underlying algorithm (per node).

- Calculate level for each node: Linear time, *proven by Hopcroft and Tarjan [1974]*
- Generating $k$ subgraphs of $G$: $O(kn)$
- Running the problem’s algorithm: $O(\delta kn)$
- Union for the subgraphs: $O(n)$
- Max of approx. Solutions: $O(k)$

Overall running time: $O(\delta kn)$

For Maximum Independent Set:

$\delta = 8k$
Running Time

Assume:

- $n$ is the number of nodes in $G$.
- $\delta$ is the running time of the underlying algorithm (per node).

- Calculate level for each node: Linear time, proven by Hopcroft and Tarjan [1974]
- Generating k subgraphs of $G$: $O(kn)$
- Running the problem’s algorithm: $O(\delta kn)$
- Union for the subgraphs: $O(n)$
- Max of approx. Solutions: $O(k)$

Overall running time: $O(\delta kn)$
Running Time

Assume:

\( n \) is the number of nodes in \( G \).
\( \delta \) is the running time of the underlying algorithm (per node).

- Calculate level for each node: Linear time, proven by Hopcroft and Tarjan [1974]
- Generating \( k \) subgraphs of \( G \): \( \mathcal{O}(kn) \)
- Running the problem’s algorithm: \( \mathcal{O}(\delta kn) \)
- Union for the subgraphs: \( \mathcal{O}(n) \)
- Max of approx. Solutions: \( \mathcal{O}(k) \)

Overall running time: \( \mathcal{O}(\delta kn) \)
For Maximum Independent Set: \( \delta = 8^k \)
1. Introduction

2. The Approximation Algorithm (for maximization problems)

3. Maximum Independent Set, an example

4. Running Time

5. Modification for minimization problems

6. Conclusion
Minimization

The shown algorithm works for many maximization problems. But what about minimization?
Minimization

The shown algorithm works for many maximization problems. But what about minimization?

We have to modify the splitting into the $k$-outerplanar subgraphs:

- **Max.**: Nodes are deleted (Resulting supgraphs: Level 2-4 and 6-8)
- **Min.**: Nodes are duplicated (Resulting supgraphs: Level 1-3, 3-5 and 5-8)
Minimization

The shown algorithm works for many maximization problems. But what about minimization?

We have to modify the splitting into the k-outerplanar subgraphs:

- **Max.**: Nodes are deleted (Resulting supgraphs: Level 2-4 and 6-8)
- **Min.**: Nodes are duplicated (Resulting supgraphs: Level 1-3, 3-5 and 5-8)

And there is at least one solution with at most $1/(k + 1)$ of all nodes duplicated.
Minimization

The shown algorithm works for many maximization problems. But what about minimization?

We have to modify the splitting into the k-outerplanar subgraphs:

- **Max.:** Nodes are deleted (Resulting supgraphs: Level 2-4 and 6-8)
- **Min.:** Nodes are duplicated (Resulting supgraphs: Level 1-3, 3-5 and 5-8)

And there is at least one solution with at most \(1/(k+1)\) of all nodes duplicated.
So our solution is at most \((k+1)/k\) optimal.
1 Introduction

2 The Approximation Algorithm (for maximization problems)

3 Maximum Independent Set, an example

4 Running Time

5 Modification for minimization problems

6 Conclusion
A General Algorithm for Many Problems

Since we have a strategy on how we can approximate minimization and maximization problems, we can change the inner algorithm to solve many other problems:
A General Algorithm for Many Problems

Since we have a strategy on how we can approximate minimization and maximization problems, we can change the inner algorithm to solve many other problems:

- Minimum Vertex Cover
- Partition Into Triangles
- Minimum Dominating Set
- Minimum Edge Dominating Set
- Maximum H-Matching
- ...
Minimum Vertex Cover

**Instance:** A graph \( G = (V, E) \), and a positive integer \( K \leq |V| \)

**Question:** Is there a vertex cover of size \( K \) or less for \( G \), that is, a subset \( V' \subseteq V \) with \( |V'| \leq K \) such that for each edge \( (u, v) \in E \) at least one of \( u \) or \( v \) belongs to \( V' \)?
Partition Into Triangles

**Instance:** A graph $G = (V, E)$, with $|V| = 3q$ for some integer $q$.

**Question:** Can the vertices of $G$ be partitioned into $q$ disjoint sets $V_1, V_2, \ldots, V_q$, each containing exactly 3 vertices, such that each of these $V_i$ is the node set of a triangle in $G$?
Minimum Dominating Set

**Instance:** A graph $G = (V, E)$, and a positive integer $K \leq |V|$

**Question:** Is there a dominating set of size $K$ or less for $G$, that is, a subset $V' \subseteq V$ with $|V'| \leq K$ such that for all $u \in V - V'$ there is a $v \in V'$ for which $(u, v) \in E$?
Minimum Edge Dominating Set

**Instance:** A graph $G = (V, E)$, and a positive integer $K \leq |V|$

**Question:** Is there a set $E' \subseteq E$ of $K$ or fewer edges such that every edge in $E$ shares at least one endpoint with some edge in $E'$?
**Maximum H-Matching**

**Instance:** Let $H$ be a connected graph with 3 or more nodes. A graph $G = (V, E)$, and a positive integer $k$

**Question:** Does $G$ contain $k$ or more node-disjoint subgraphs isomorphic to $H$?
THANK YOU!!!