Approximation Algorithms for NP-Complete Problems on Planar Graphs

Marius Knabben

December 10, 2013

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.
- *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

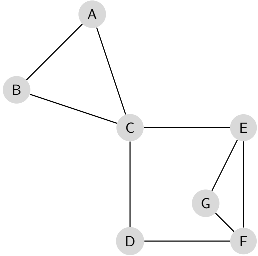
Definition (1-Outerplanar Graphs)

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)

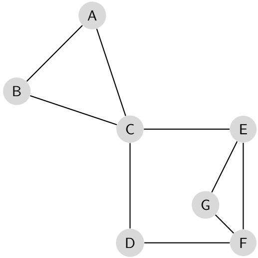
A graph G is outerplanar if G can be drawn in the plane without crossings of edges and all vertices belong to the outer face.

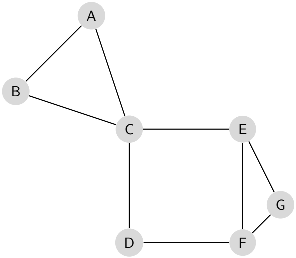


1- and k-Outerplanar Graphs

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

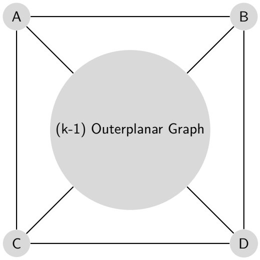
Definition (k-Outerplanar Graphs)

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)

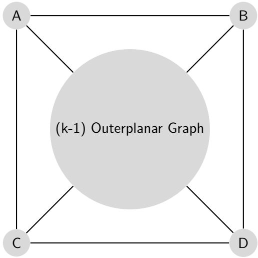
A graph G is k-outerplanar if G can be drawn in the plane without crossings of edges and by removing all vertices that belong to the outer face the resulting graph is (k-1)-outerplanar.

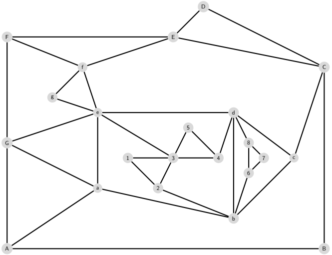


1- and k-Outerplanar Graphs

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.

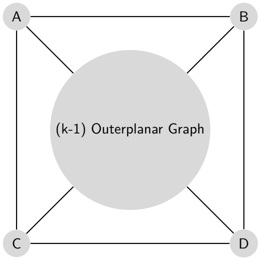


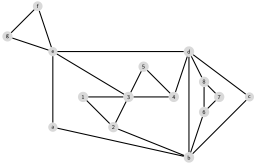


1- and k-Outerplanar Graphs

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.

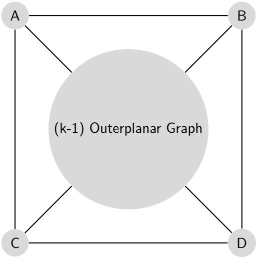


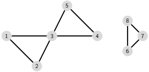


1- and k-Outerplanar Graphs

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.

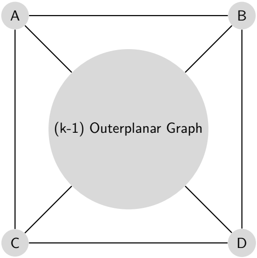


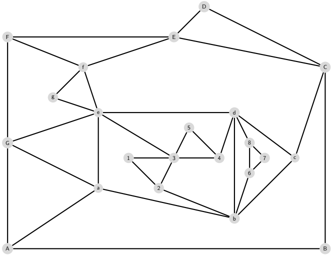


1- and k-Outerplanar Graphs

Definition (k-Outerplanar Graphs)

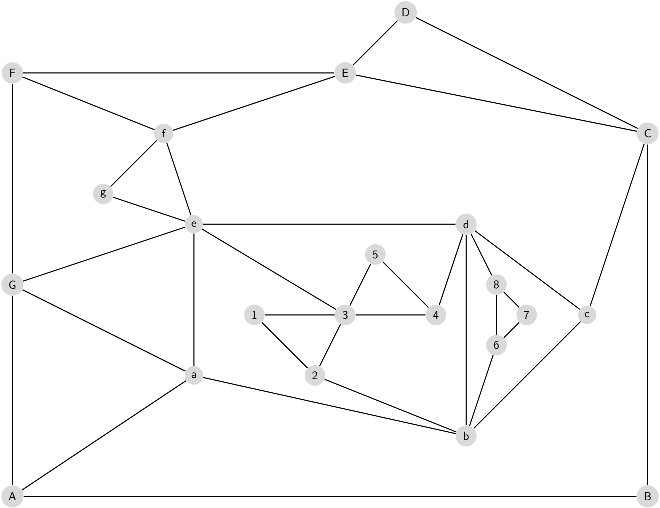
A graph G is k-outerplanar if G can be drawn in the plane without crossings of edges and by removing all vertices that belong to the outer face the resulting graph is (k-1)-outerplanar.





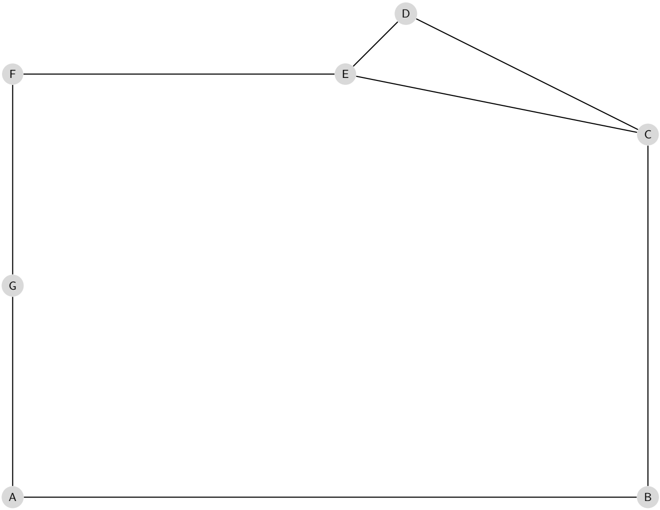
For every planar graph G, there is a k so that G is k-outerplanar.

Level i-nodes



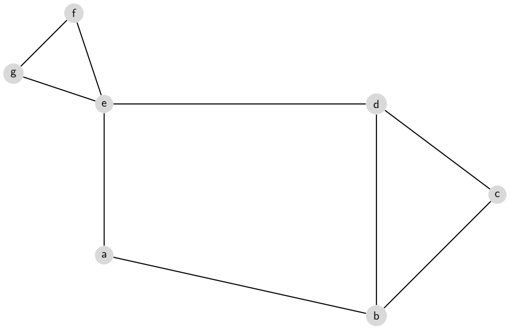
- Level 1
- Level 2
- Level 3

Level i-nodes



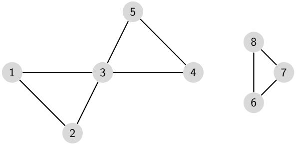
- Level 1
- Level 2
- Level 3

Level i-nodes



- Level 1
- Level 2
- Level 3

Level i-nodes



- Level 1
- Level 2
- Level 3

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.

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

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_1^i to G_m^i which are at most k outerplanar
    3. for j = 1, ..., m
        1. S_j^i = Maximum Independent Set(G_j^i)

The Algorithm

Input: The graph G and k.

1. Calculate the level for each node
2. for i $= 1, 2, \ldots, k+1$
   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 $= 1, \ldots, m$
      1. $S^i_j =$ Maximum Independent Set(G^i_j)
   4. $S^i = \bigcup_j S^i_j$
3. $S = \max_i S^i$

The Algorithm

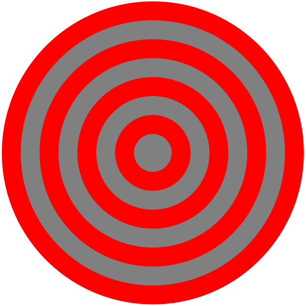
Input: The graph G and k.

1. Calculate the level for each node
2. for i $= 1, 2, ..., k + 1$
    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 = Maximum Independent Set(G^i_j)
    4. $S^i = \bigcup_j S^i_j$
3. $S = \max_i S^i$

Output: S

Visualization - Divide

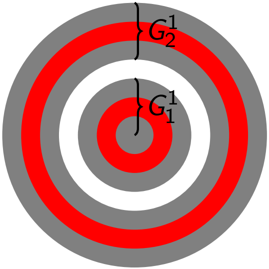
G is 8-Outerplanar, $k = 3$



- Level 1
- Level 2
- Level 3
- Level 4
- Level 5
- Level 6
- Level 7
- Level 8

Visualization - Divide

G is 8-Outerplanar, $k = 3$



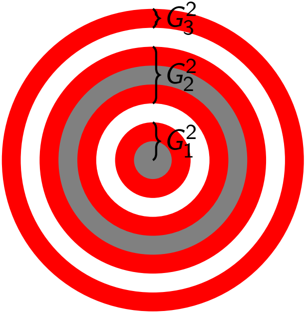
- Level 2
- Level 3
- Level 4

- Level 6
- Level 7
- Level 8

$i = 1$

Visualization - Divide

G is 8-Outerplanar, $k = 3$



- Level 1

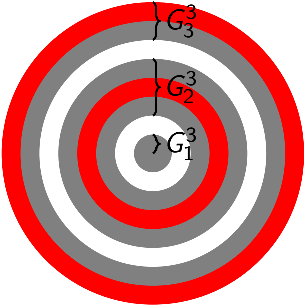
- Level 3
- Level 4
- Level 5

- Level 7
- Level 8

$i = 2$

Visualization - Divide

G is 8-Outerplanar, $k = 3$

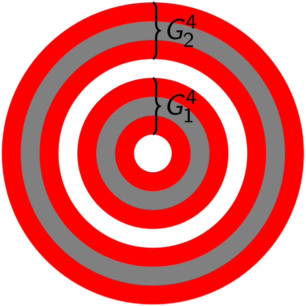


$i = 3$

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

Visualization - Divide

G is 8-Outerplanar, $k = 3$

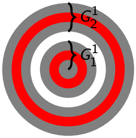


- Level 1
- Level 2
- Level 3

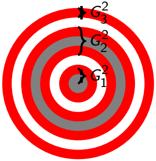
- Level 5
- Level 6
- Level 7

$i = 4$

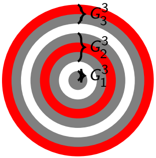
Visualization - Conquer



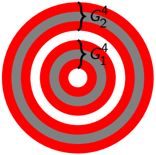
$S^1 = S(G_1^1) \cup S(G_2^1)$



$S^2 = S(G_1^2) \cup S(G_2^2) \cup S(G_3^2)$



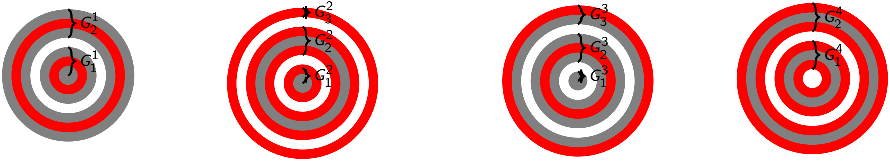
$S^3 = S(G_1^3) \cup S(G_2^3) \cup S(G_3^3)$



$S^4 = S(G_1^4) \cup S(G_2^4)$

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

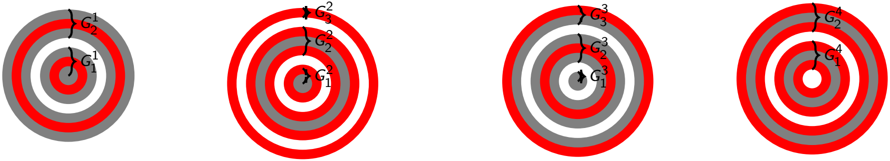
Visualization - Conquer



$S^1 = S(G_1^1) \cup S(G_2^1)$ $S^2 = S(G_1^2) \cup S(G_2^2) \cup S(G_3^2)$ $S^3 = S(G_1^3) \cup S(G_2^3) \cup S(G_3^3)$ $S^4 = S(G_1^4) \cup S(G_2^4)$

- 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

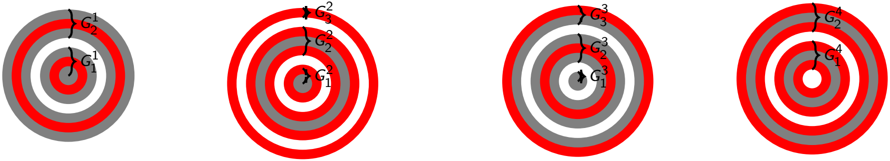
Visualization - Conquer



$S^1 = S(G_1^1) \cup S(G_2^1)$ $S^2 = S(G_1^2) \cup S(G_2^2) \cup S(G_3^2)$ $S^3 = S(G_1^3) \cup S(G_2^3) \cup S(G_3^3)$ $S^4 = S(G_1^4) \cup S(G_2^4)$

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

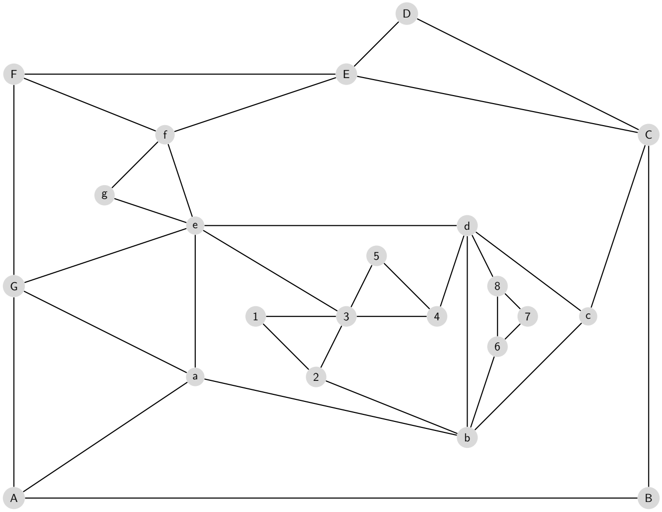
Visualization - Conquer



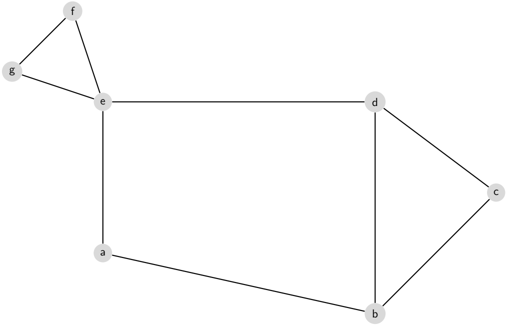
$S^1 = S(G_1^1) \cup S(G_2^1)$ $S^2 = S(G_1^2) \cup S(G_2^2) \cup S(G_3^2)$ $S^3 = S(G_1^3) \cup S(G_2^3) \cup S(G_3^3)$ $S^4 = S(G_1^4) \cup S(G_2^4)$

- 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
- This subgraphs solution is at least $3/4$ optimal
- Reintroducing k: The solution is $k/(k+1)$ optimal

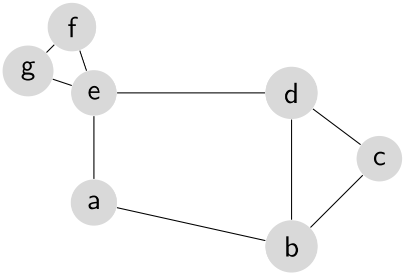
Our Graph



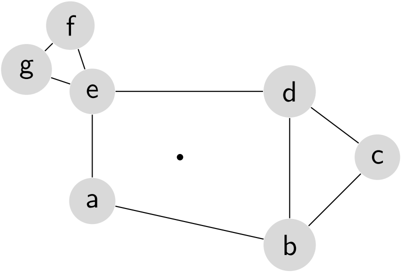
Our Graph



Tree

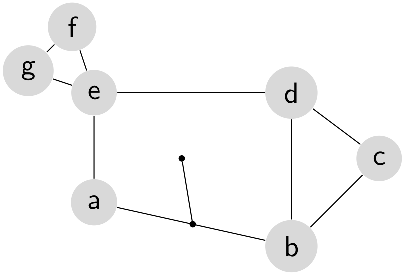


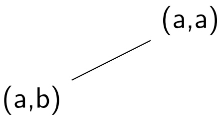
Tree



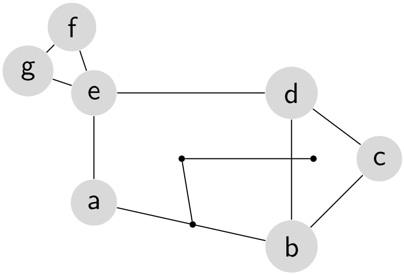
(a,a)

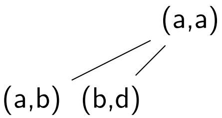
Tree



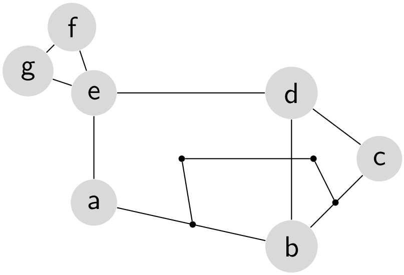


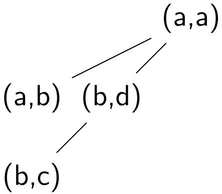
Tree



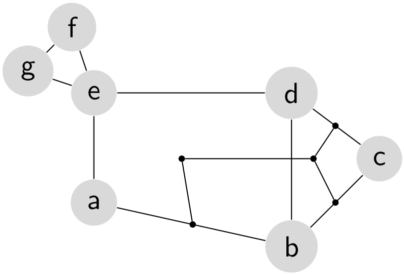


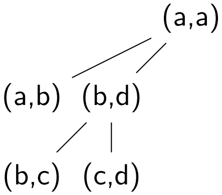
Tree



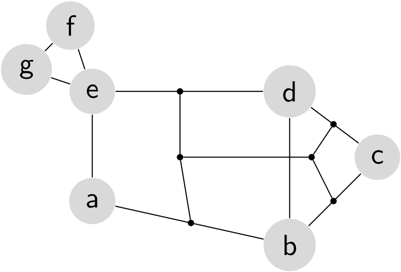


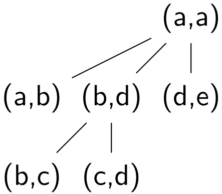
Tree



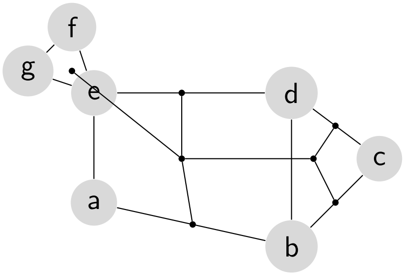


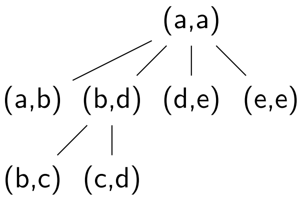
Tree



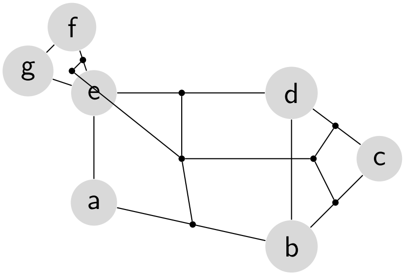


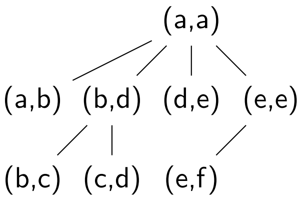
Tree



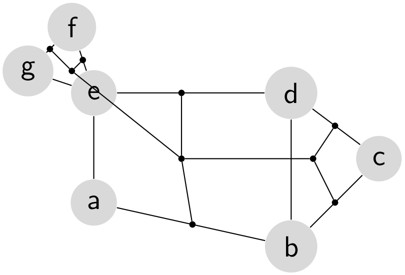


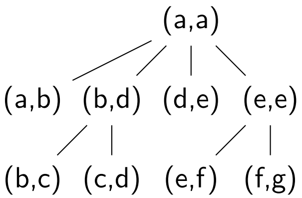
Tree



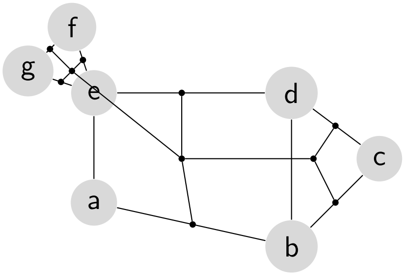


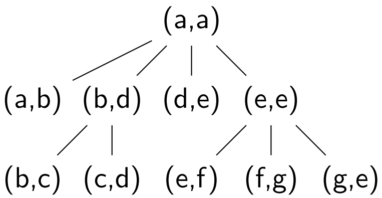
Tree



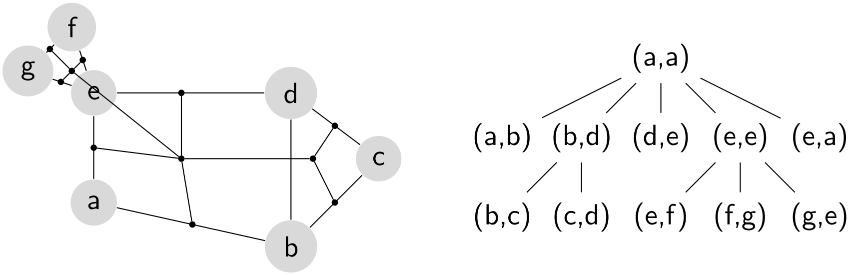


Tree

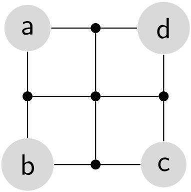


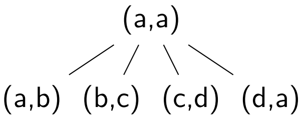


Tree



Tree





Algorithm (for Maximum Independent Set)

- Recursively defined
- Only once executed per tree-node

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

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

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

On our Tree

(a,a)

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

Merge

Merge two neighboring tables T_{ab} and T'_{bc}:

$+$	$t_\emptyset$	t_a	t_b	t_{ab}
$t'_\emptyset$	w	x		
t'_b			$w' + 1$	$x' + 1$
t'_c	y	z		
t'_{bc}			$y' + 1$	$z' + 1$

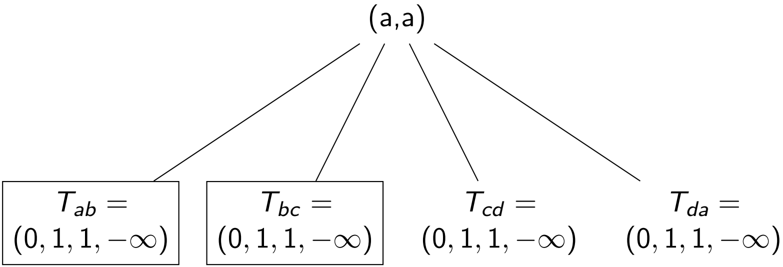
Merge

Merge two neighboring tables T_{ab} and T'_{bc}:

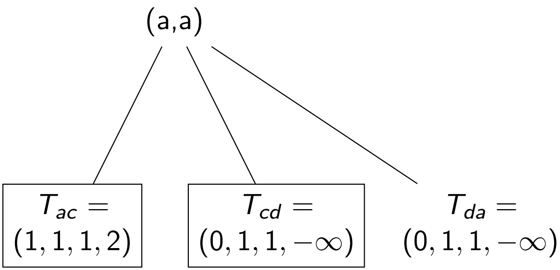
$+$	$t_\emptyset$	t_a	t_b	t_{ab}
$t'_\emptyset$	w	x		
t'_b			$w'+1$	$x'+1$
t'_c	y	z		
t'_{bc}			$y'+1$	$z'+1$

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

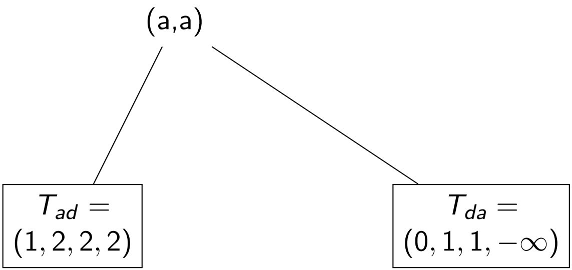
Merge



Merge



Merge



Merge

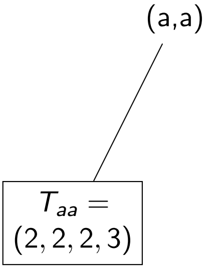
(a,a)

$T_{aa} =$
$(2, 2, 2, 3)$

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.

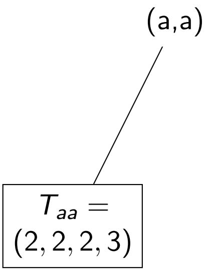
Merge



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 $= -\infty$

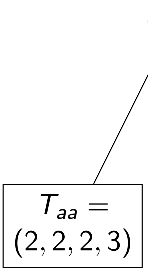
Merge



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.

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

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 $\max(2, -\infty, -\infty, 2) = 2$.

Running Time

Assume:

n is the number of nodes in G.

δ is the running time of the underlying algorithm (per node).

Running Time

Assume:

n is the number of nodes in G.

δ 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.

δ 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 Time

Assume:

n is the number of nodes in G.

δ 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: $\mathcal{O}(\delta kn)$

Running Time

Assume:

n is the number of nodes in G.

δ 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: $\mathcal{O}(k)$

Running Time

Assume:

n is the number of nodes in G.

δ 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: $\mathcal{O}(\delta kn)$

Running Time

Assume:

n is the number of nodes in G.

δ 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: $\delta = 8^k$

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.

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, thas 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, ... , 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 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!!!