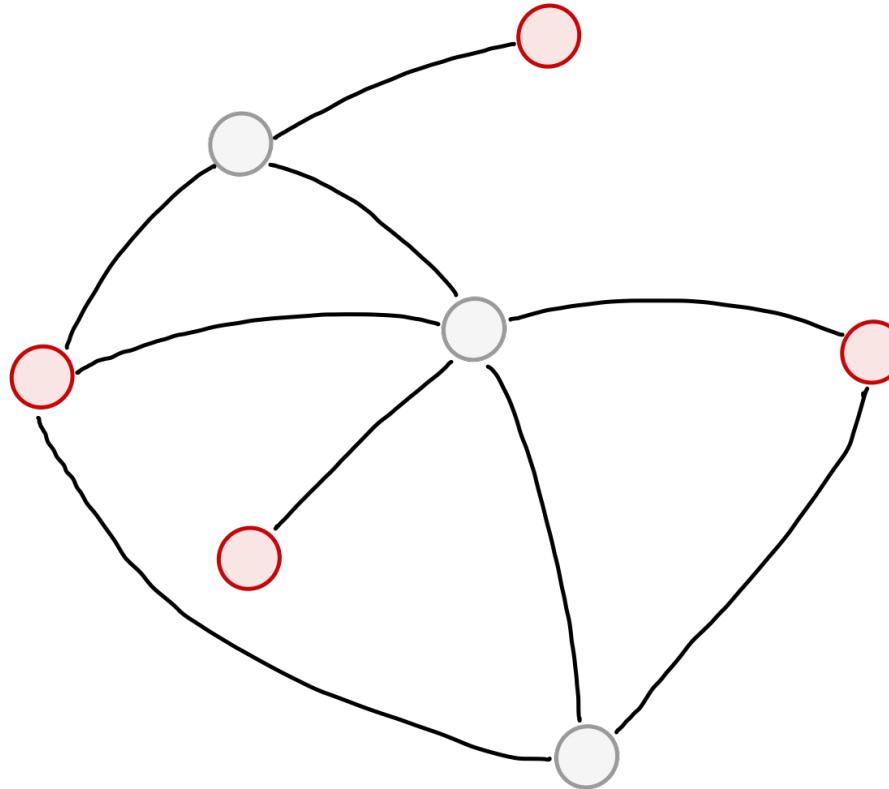


PTAS auf planaren Graphen

Approximation des Maximum Independent Sets

Manuel Wächtershäuser, Gian Saß, Tom Leimbrock, Jan Hindges

Maximum Independent Set (NP-Hard)



Gliederung

- Einleitung
- Reduktionen
- Generierung und Darstellung planarer Graphen
- Polynomial-Time Approximation Scheme (PTAS)
- Lösung mit Baumzerlegung
- Ausblick

Einleitung

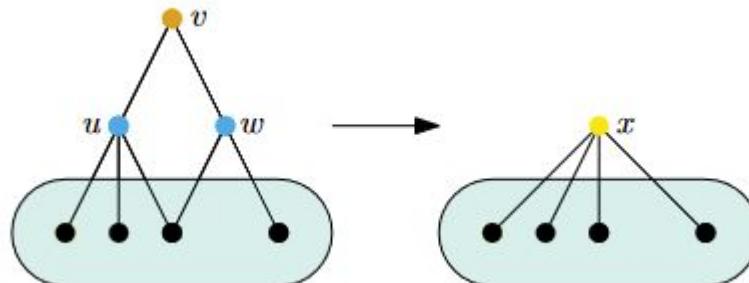
- Arbeit mit planaren Graphen
- Reduktion
- Zerlegung in Ringe der Dicke k durch Entfernung von Knoten
- Baumzerlegung auf den einzelnen Schichten
- Lösung des Problems für jede einzelne Schicht
- Kombination der Lösungen

Reduktionen

- Idee: Graph kleiner machen und wissen, wie sich die Lösung verändert
- Entfernung oder Zusammenfügen von Knoten
- Durchführung auf schneller Datenstruktur für Graphen
- anschließend Durchführung auf DCEL
- Wiederherstellung des ursprünglichen Graphen und der korrekten Lösung

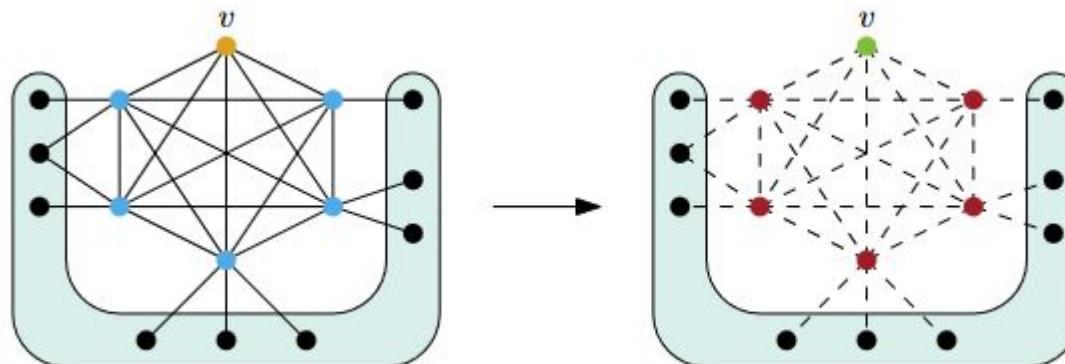
Nodal Fold Reduction

- kann auf Knoten mit zwei Nachbarn angewandt werden, die **nicht** miteinander adjazent sind
- Merge nacheinander beide Nachbarn in einen solchen Knoten
- Wenn der neue Knoten Teil der Lösung → Nachbarn gehören zur Lösung
- wenn der neue Knoten nicht Teil der Lösung → ursprünglicher Knoten gehört zur Lösung



Isolated Clique Reduction

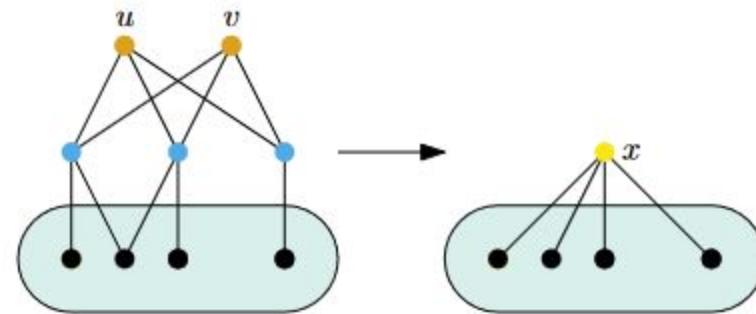
- Isolierte Clique: Knoten, der mit allen seinen Nachbarn eine Clique bildet
- gesamte Clique entfernen
- isolierten Knoten zur Lösung hinzufügen



Twin Reduction

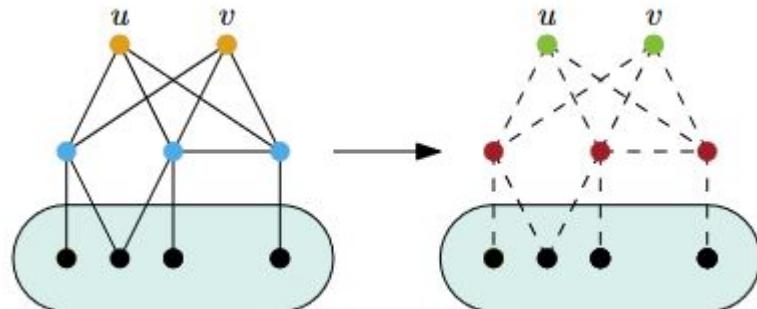
- zwei nicht-adjazente Knoten mit denselben drei Nachbarn
- Entfernung der beiden Knoten und der drei gemeinsamen Nachbarn
- Ersatz durch Knoten mit der Vereinigungsmenge der Nachbarn der drei gemeinsamen Nachbarn als Nachbarn
- Wenn neuer Knoten Teil der Lösung → drei gemeinsame Nachbarn in der Lösung
- sonst → die ursprünglichen beiden Knoten in der Lösung

Twin Reduction

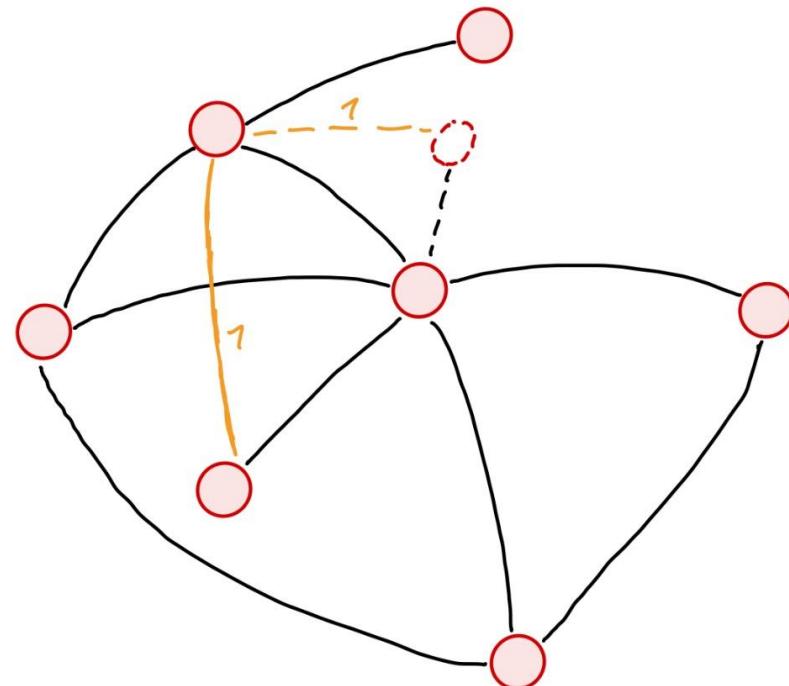
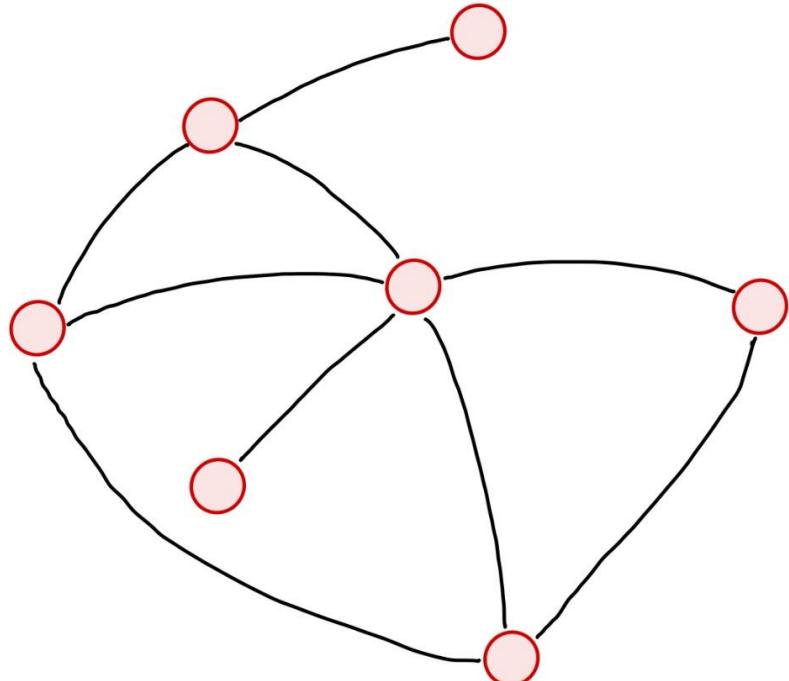


Twin-Reduktion

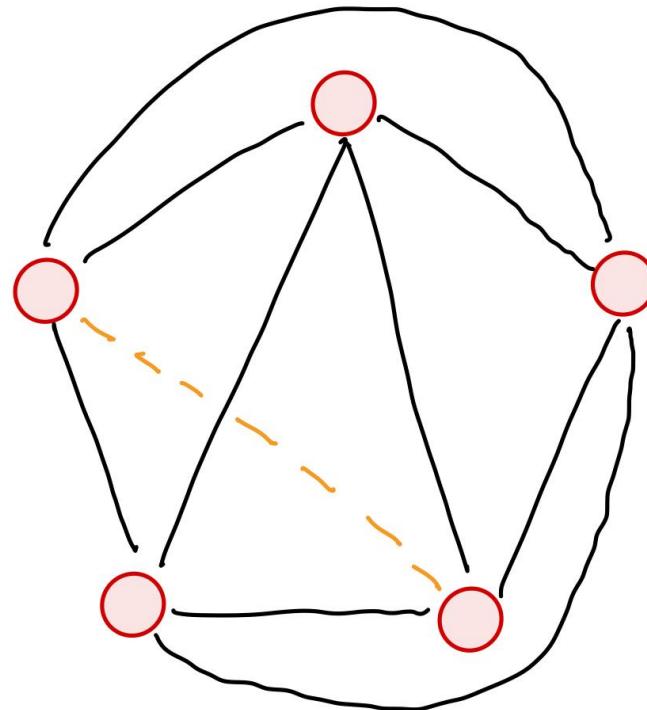
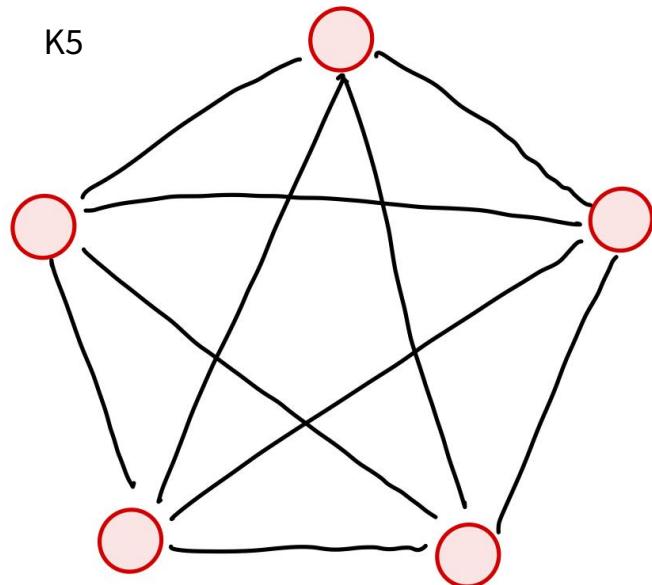
- Wenn zwei der drei gemeinsamen Nachbarn adjazent \rightarrow entferne die beiden ursprünglichen Knoten und die drei gemeinsamen Nachbarn ersatzlos
- Füge die beiden ursprünglichen Knoten zur Lösung hinzu



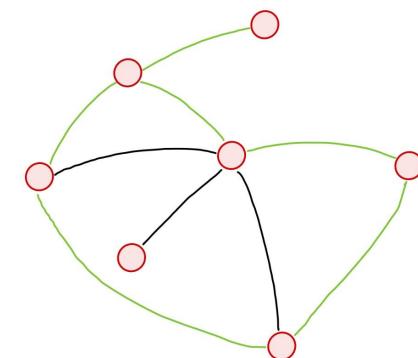
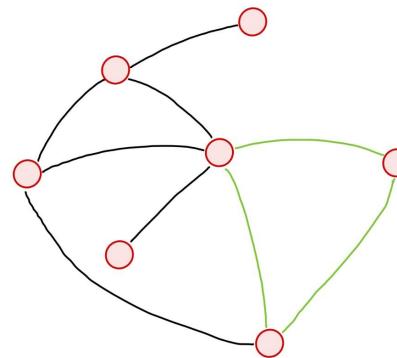
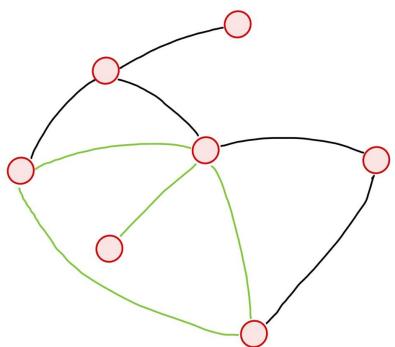
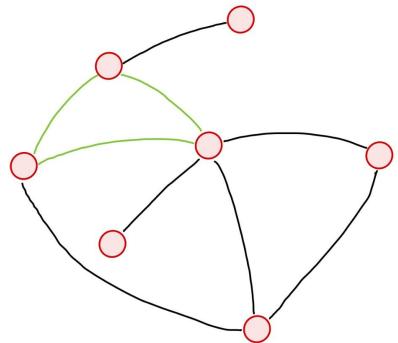
Planare Graphen



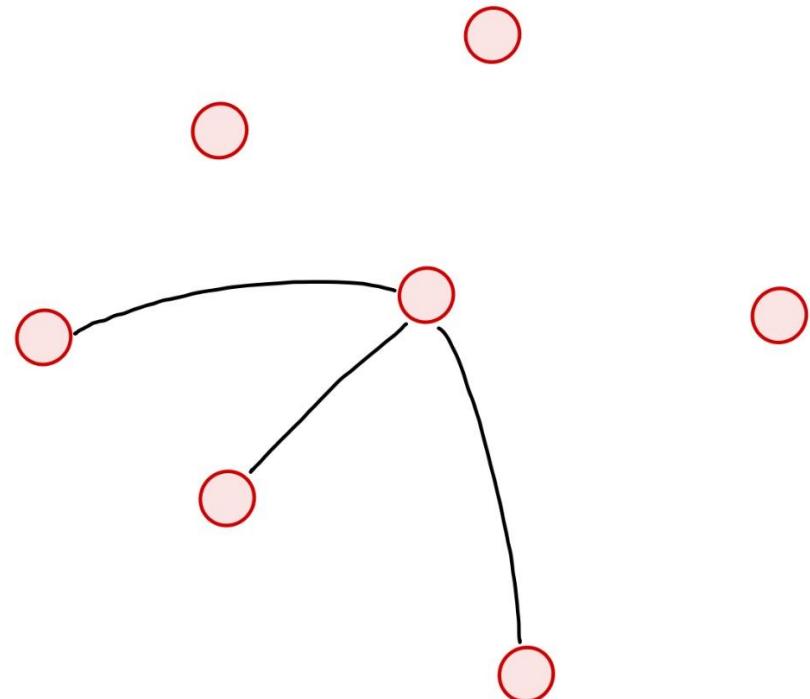
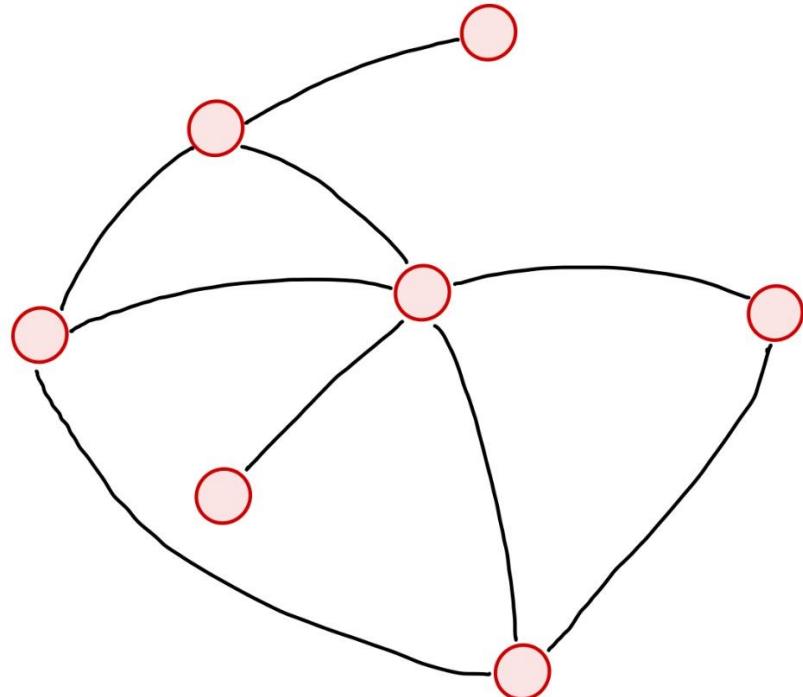
(Maximal) Planare Graphen



Faces



K-Aussenplanar



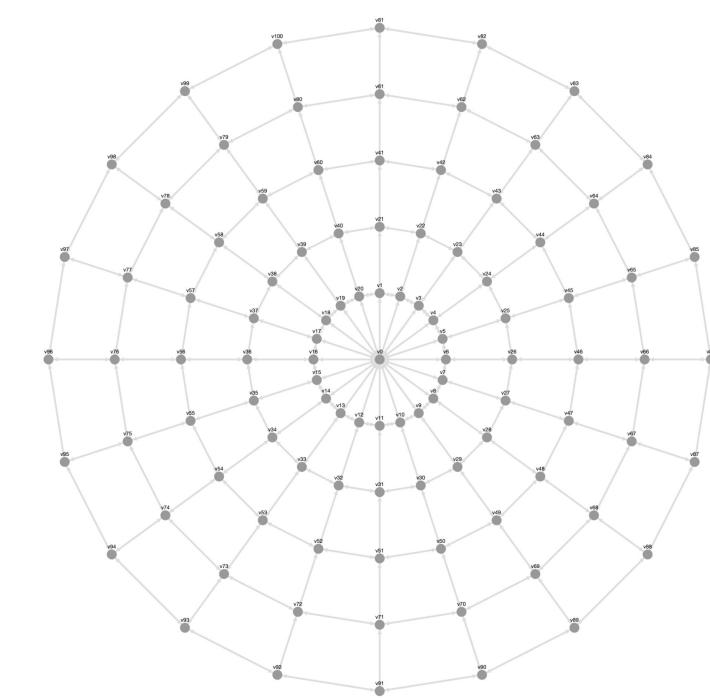
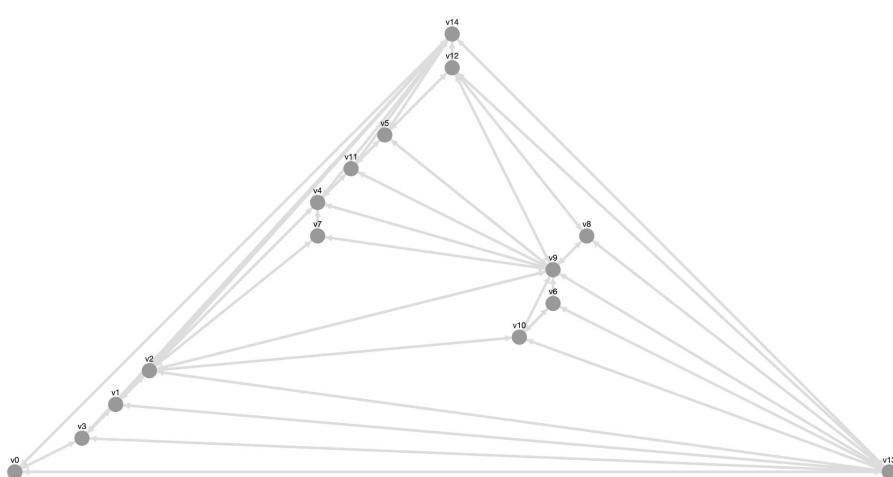
Generierung von Graphen

- Python-Programm mit *networkx*
- Ein *nx.Graph* kann in ein *nx.PlanarEmbedding* konvertiert werden
 - *nx.is_planar* Methode
- *nx.PlanarEmbedding* liefert ein *Combinatorial Embedding*, das für die Erstellung der DCEL benötigt wird
- *Combinatorial Embedding*: Kanten eines Knoten sind gegen den Uhrzeigersinn sortiert
- Speicherung des CEs und des Layout des Graphen für die Visualisierung

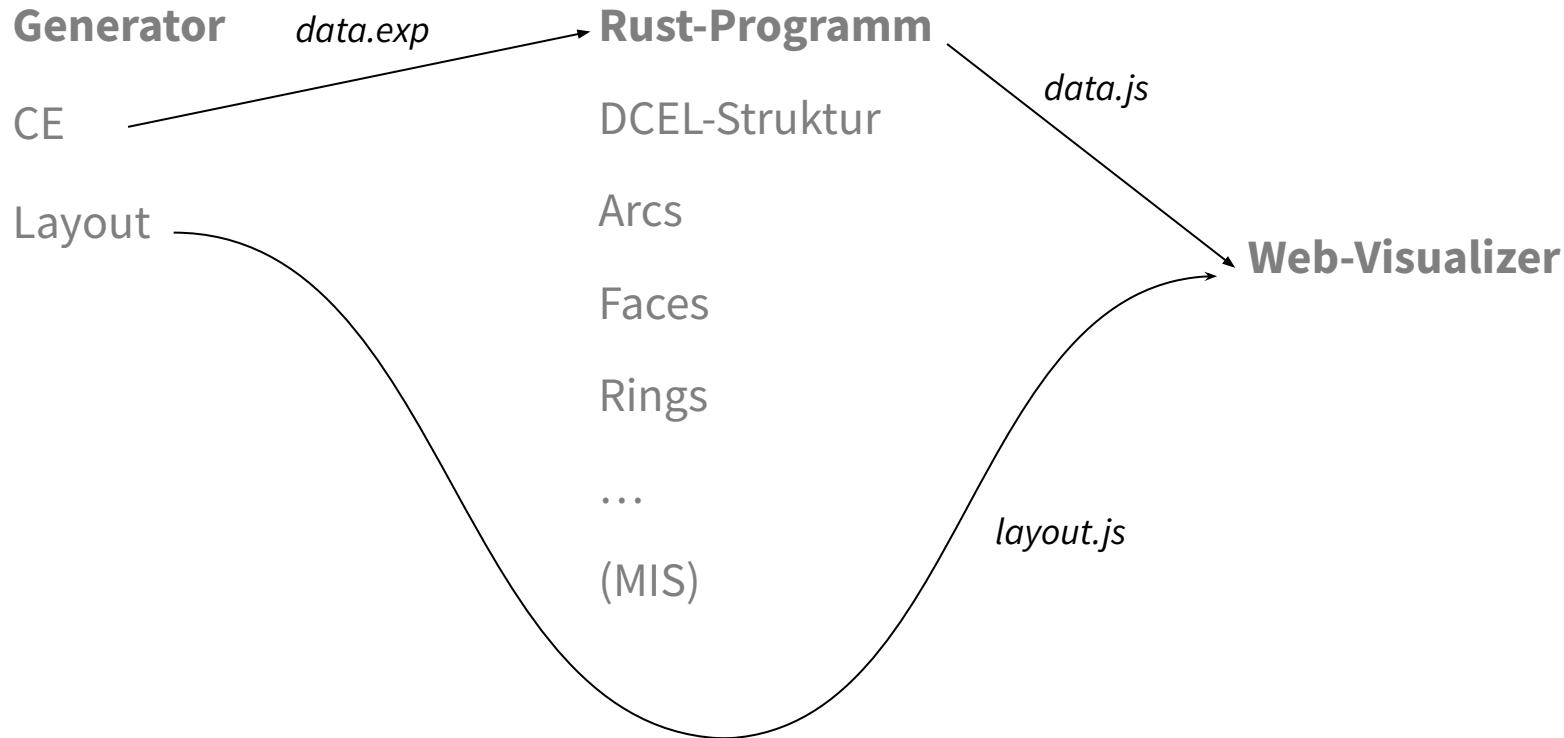
Generierung von Graphen

- Es werden zwei Arten von Graphen unterstützt
- Zufälliger Graph
 - Inkrementales Hinzufügen von Kanten, solange Planarity nicht verletzt wird
 - Schlecht zu visualisieren
- Zufälliger “Kreisgraph”
 - Graph besteht aus einer Vielzahl mehreren Ringen
 - Gut zu visualisieren
 - Hilfreich um den Algorithmus zu testen und zu debuggen

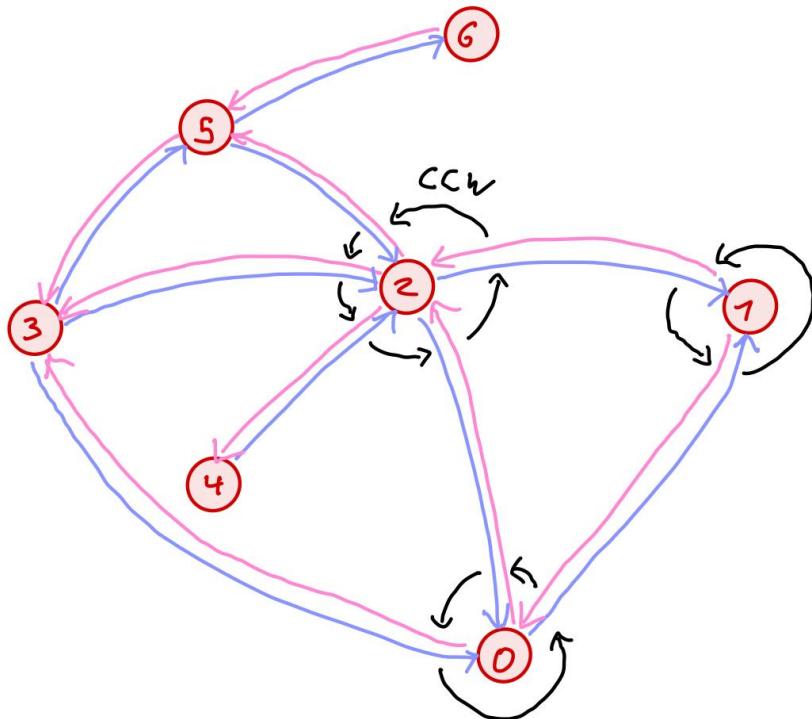
Visualisierung von Graphen



Visualisierung von Graphen



Doubly-Connected Edge List

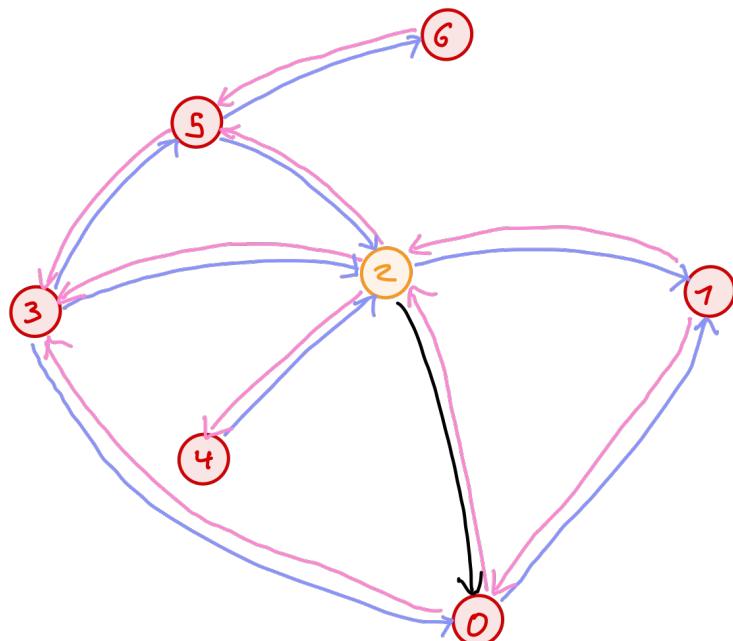


Input data

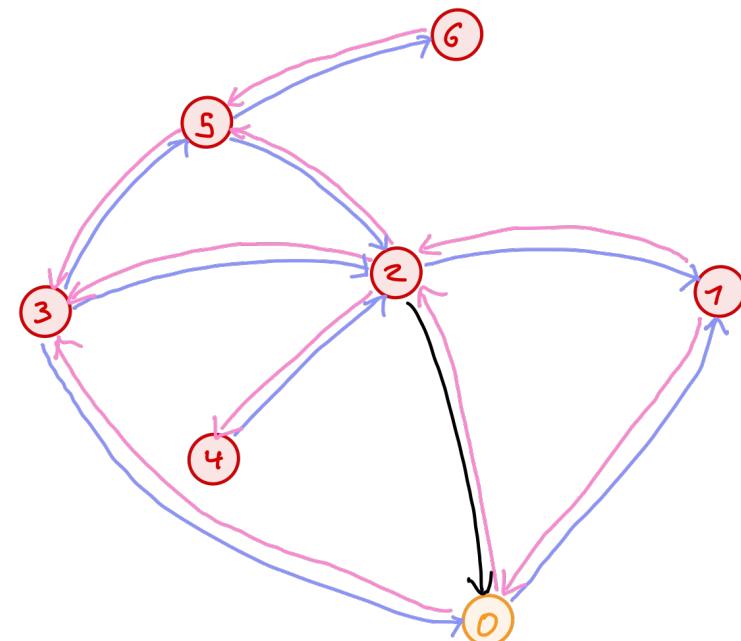
7	
9	
0 1	3 0
0 2	3 2
0 3	3 5
1 0	4 2
1 2	
2 0	5 2
2 1	5 6
2 5	5 3
2 3	6 5
2 4	

Doubly-Connected Edge List

Twin

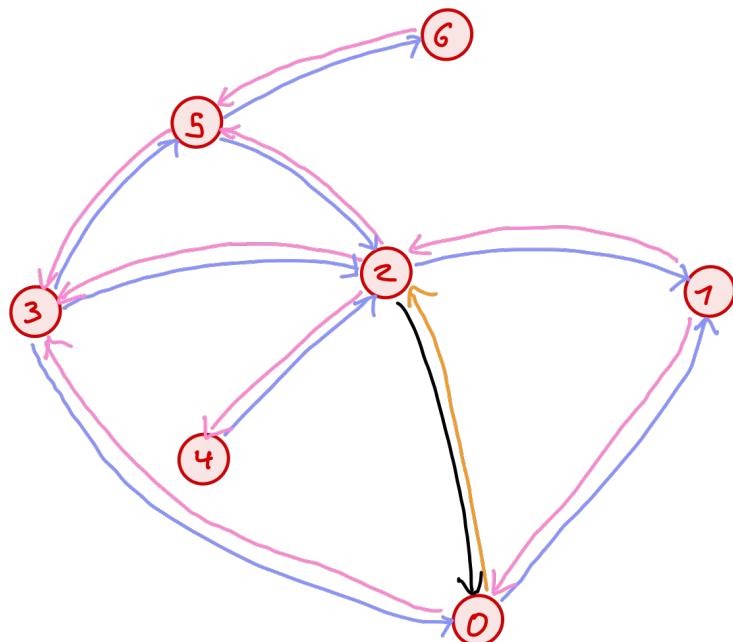


Destination

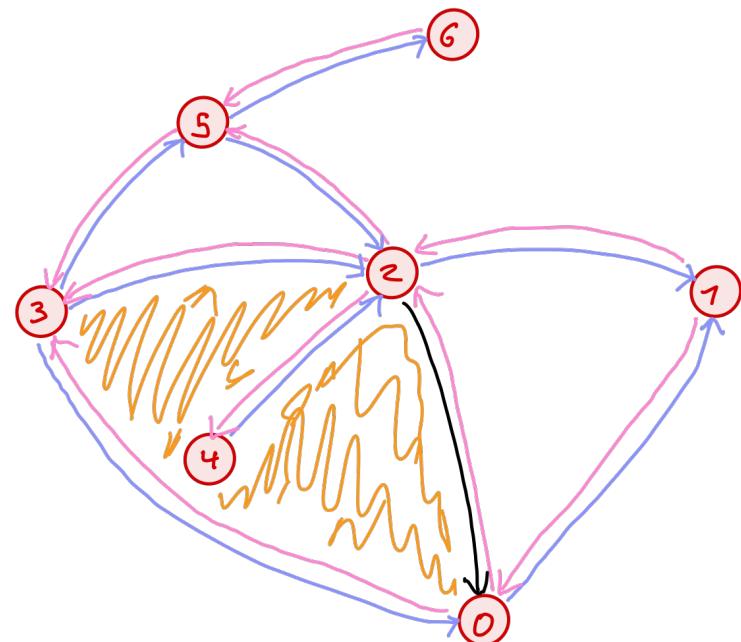


Doubly-Connected Edge List

Twin

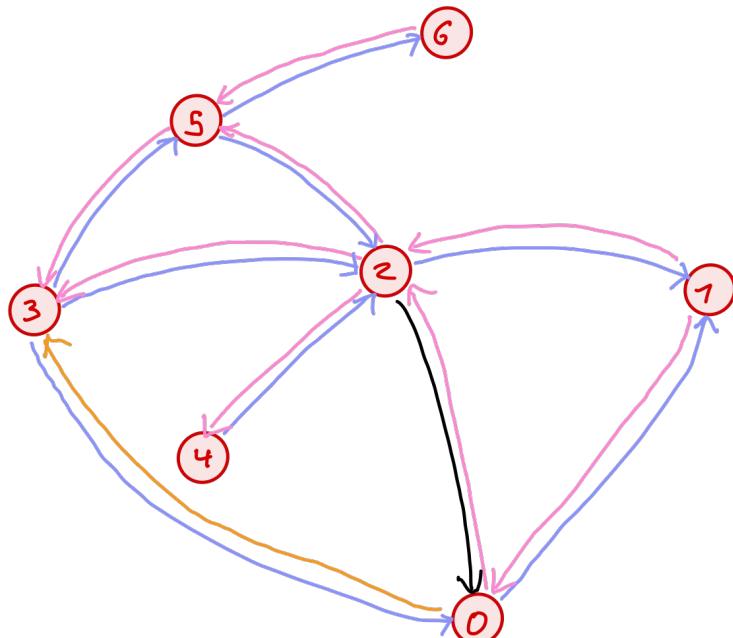


Face

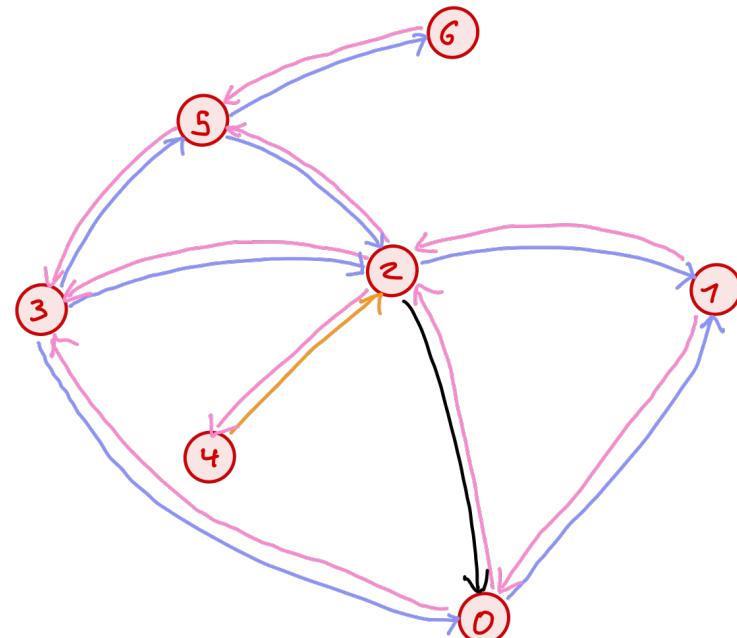


Doubly-Connected Edge List

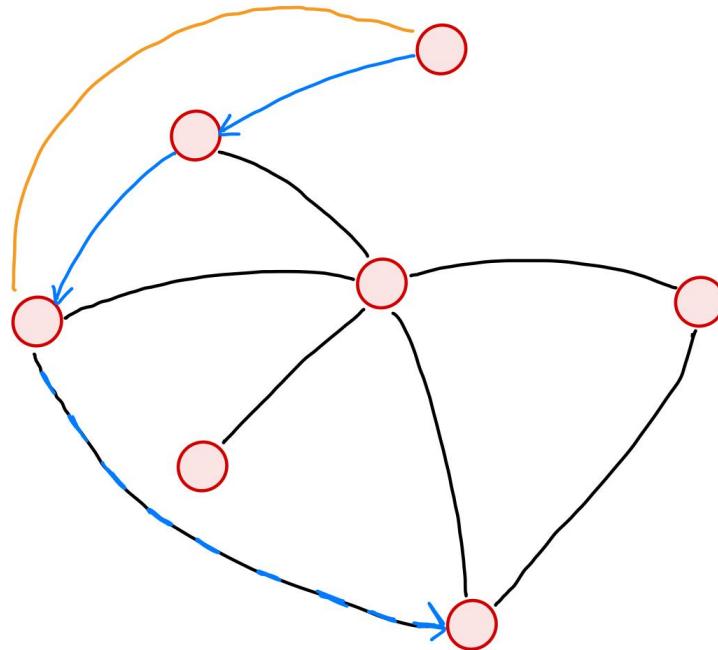
Next



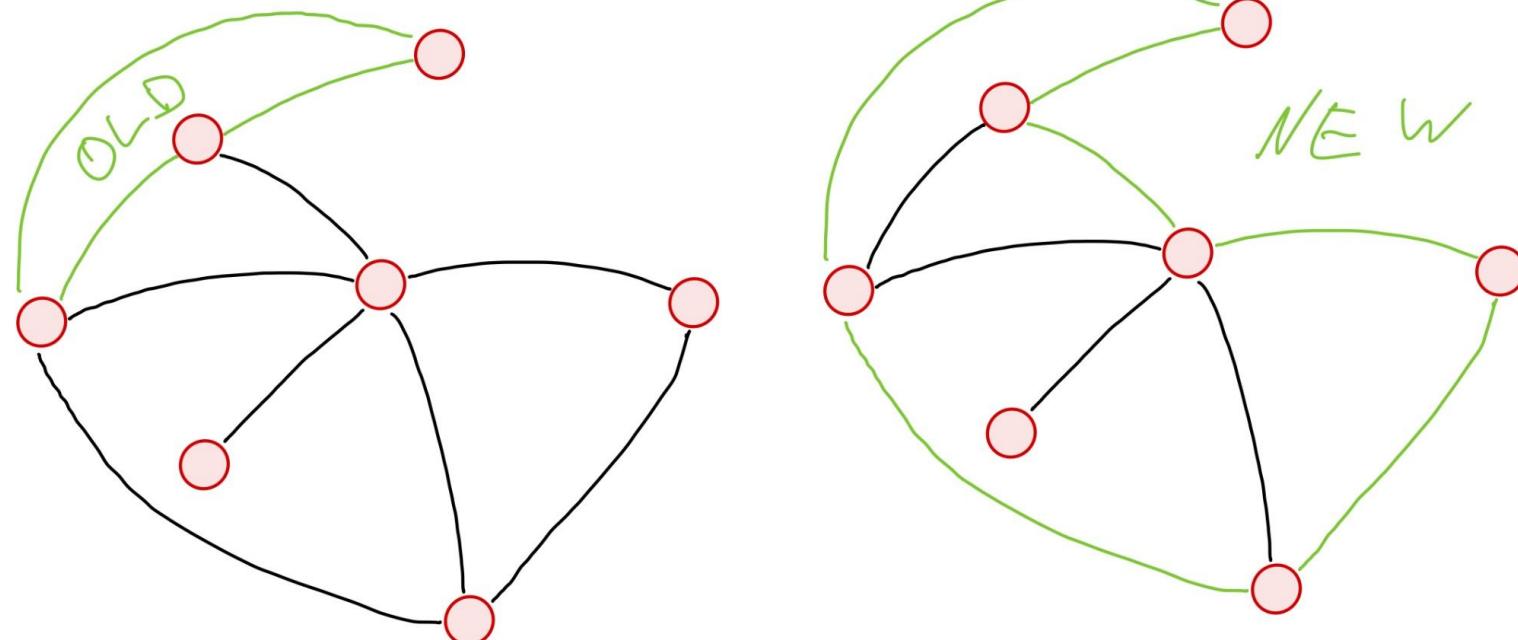
Prev



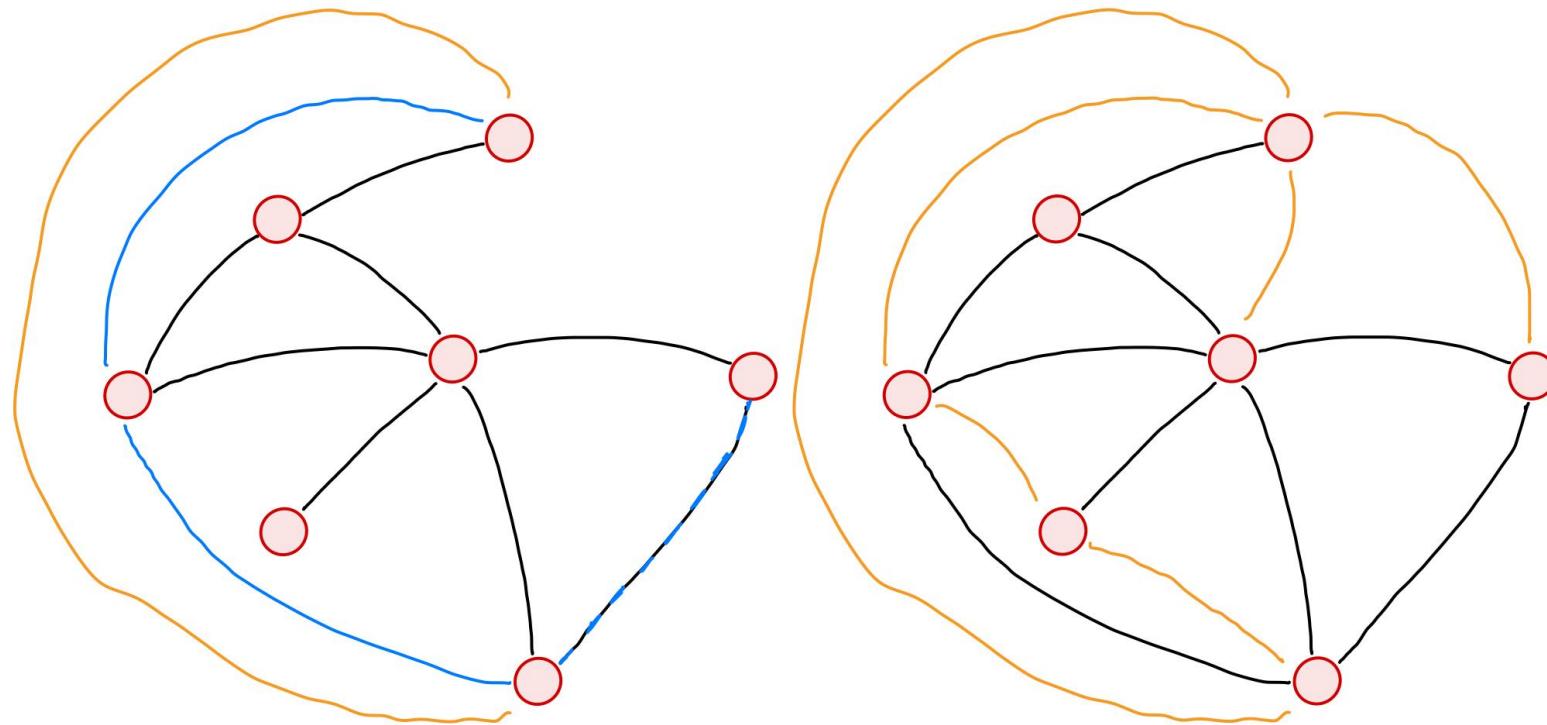
Triangulation



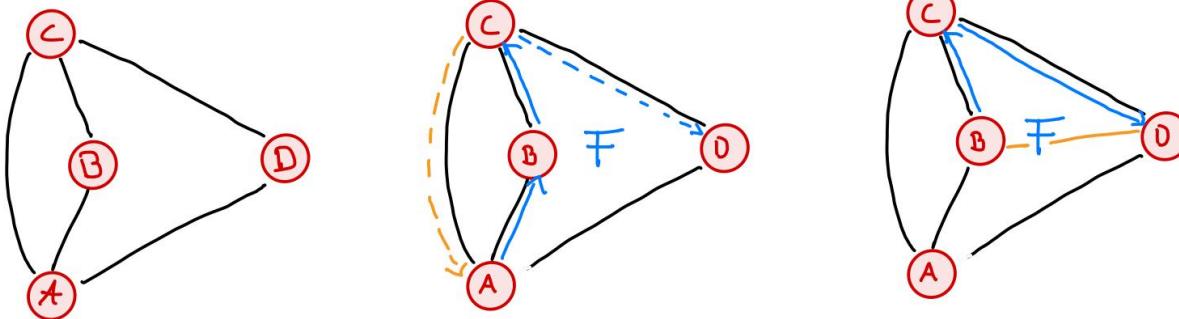
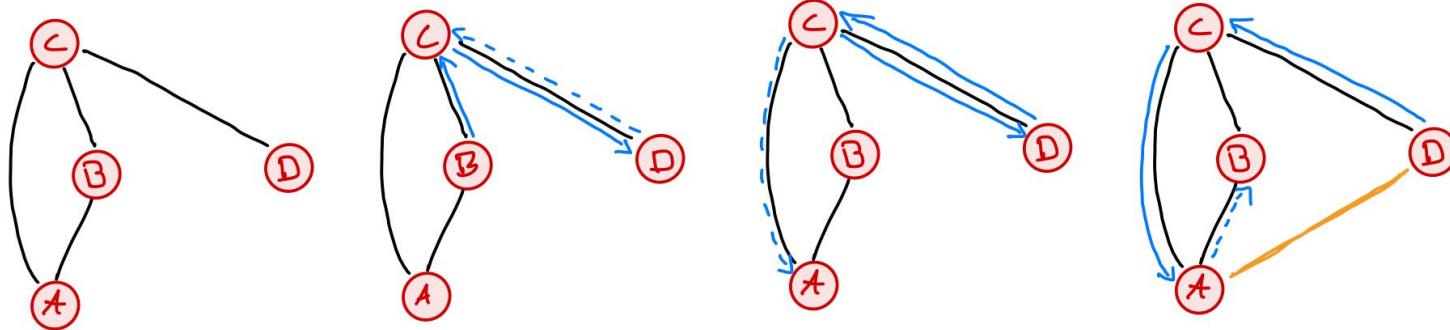
Triangulation



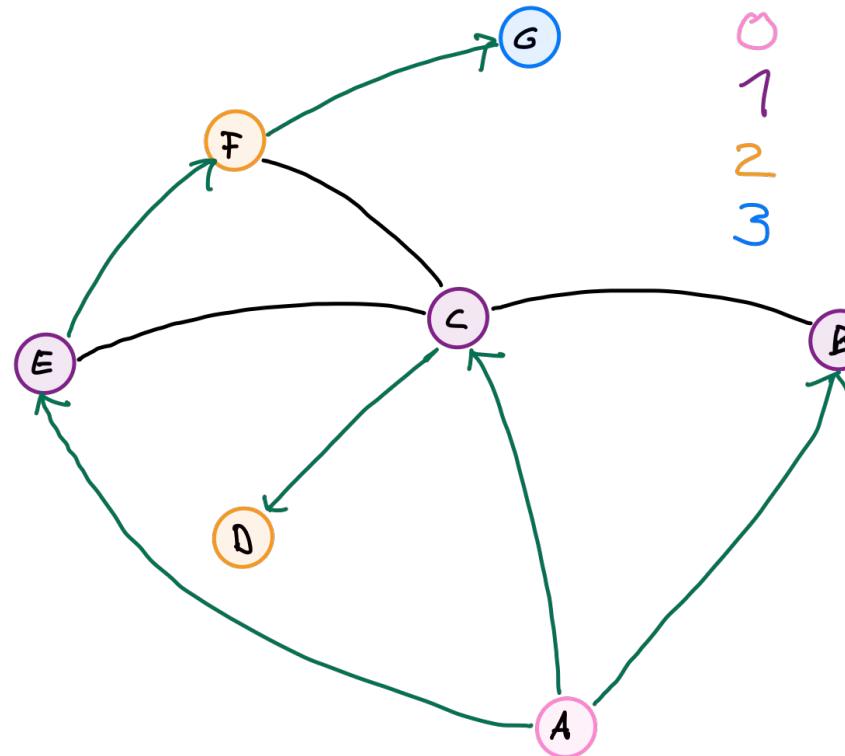
Triangulation



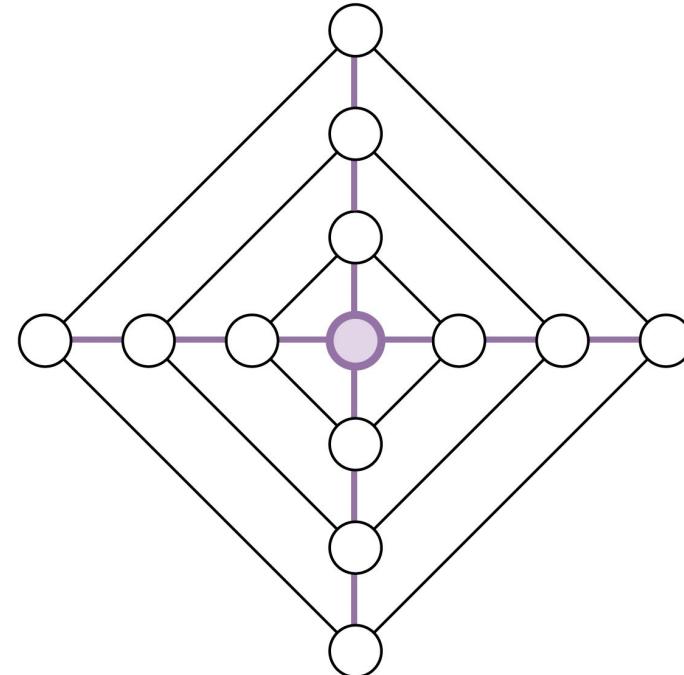
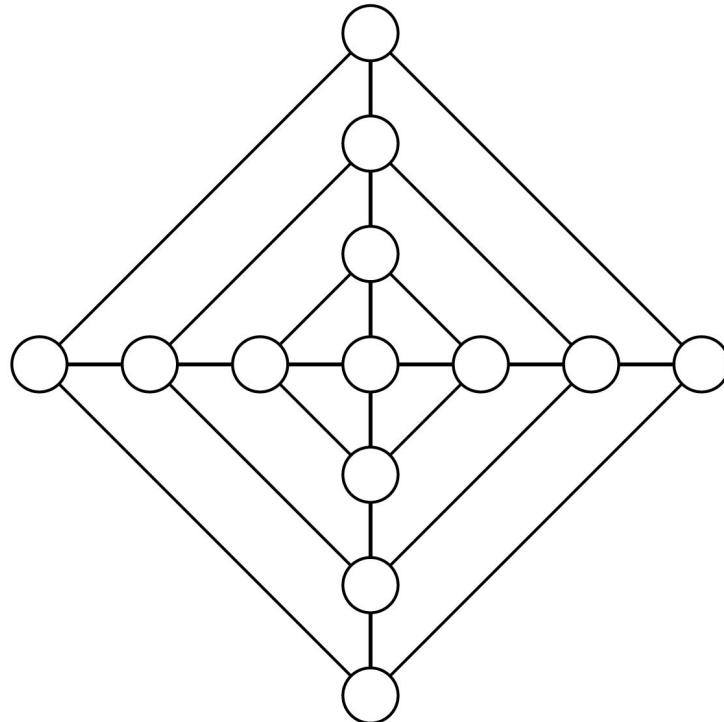
Triangulation Ausnahmen



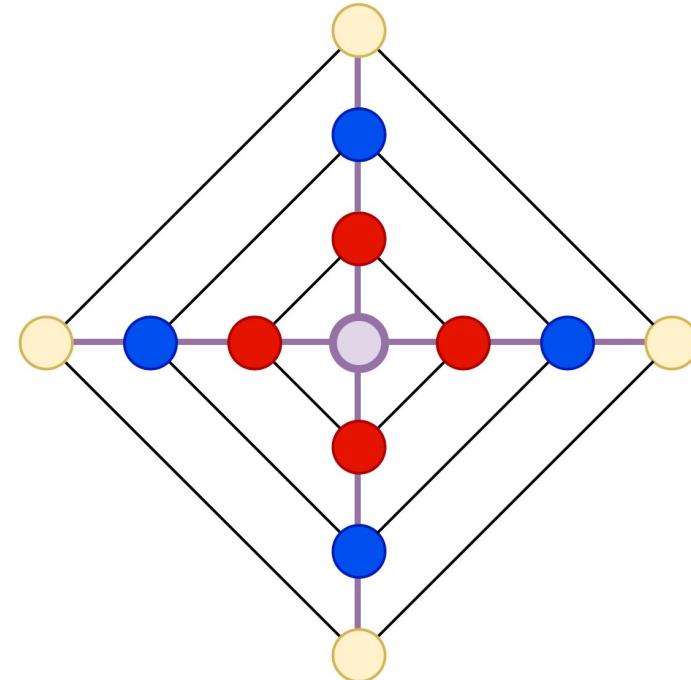
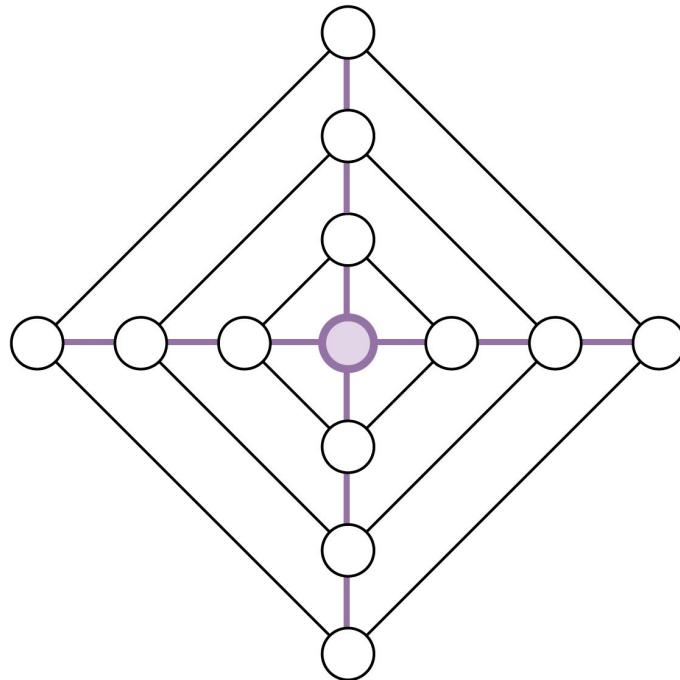
Spannbaum



ringe



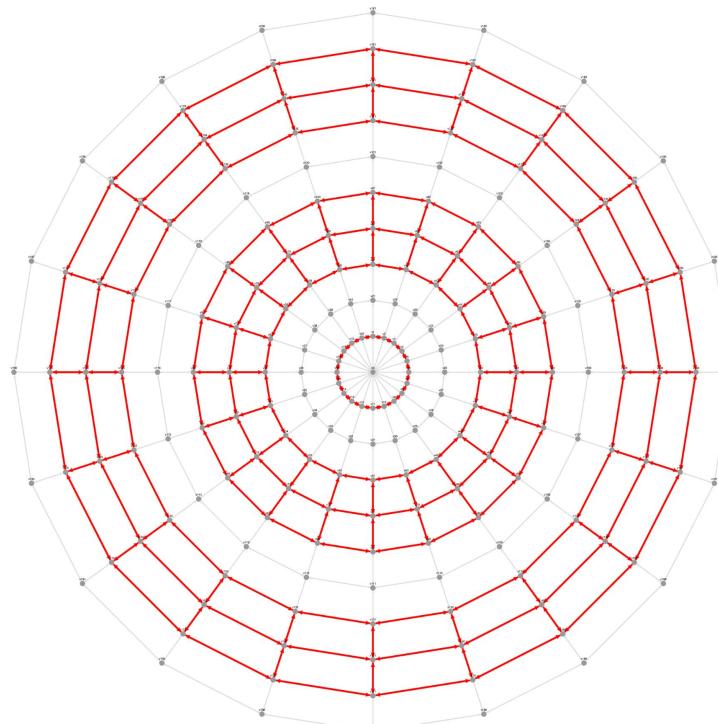
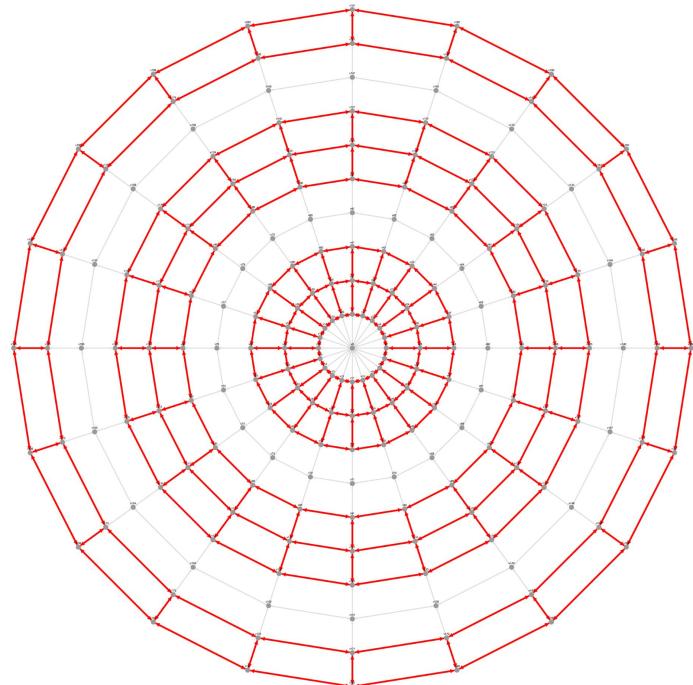
Ringe



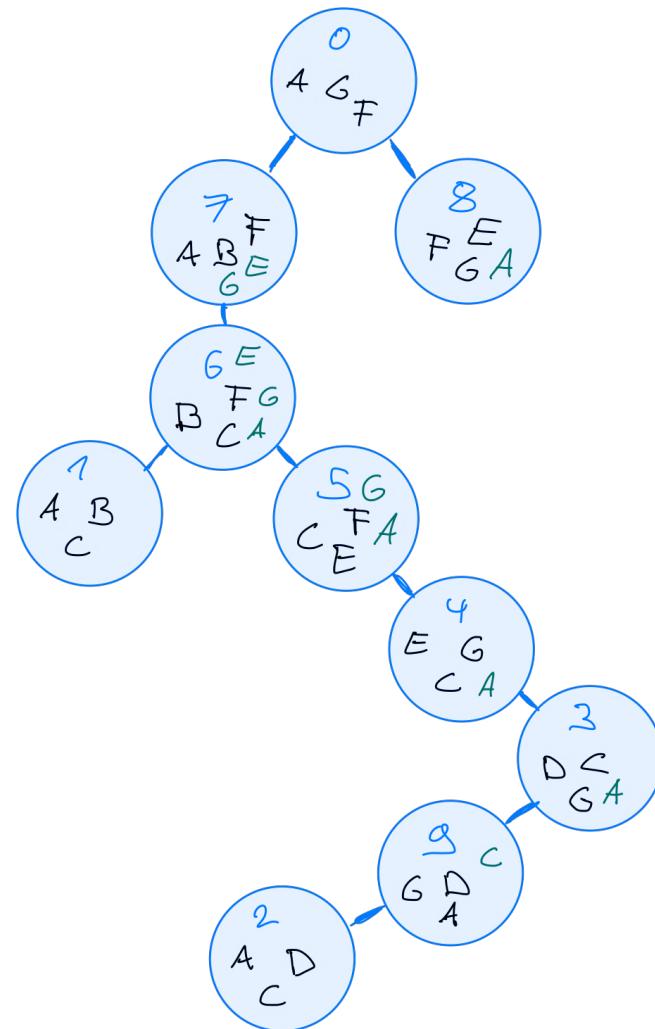
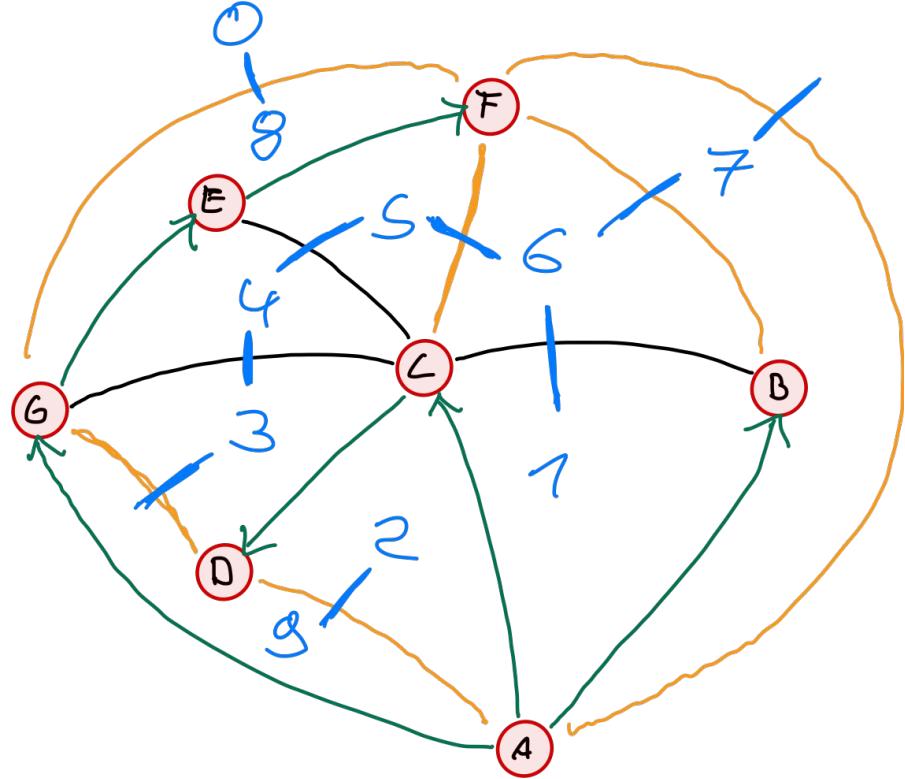
Approximation mit Donuts

- “Baker’s Approximation Scheme”
- Gegeben: Ganzzahl k
- Für jedes i , sodass $0 \leq i \leq k$
 - Schneide jeden Ring wo $\text{Tiefe mod } (k+1) = i$ raus
 - Es verbleiben disjunkte Teilgraphen, Ringe mit max. Dicke k (Donuts/Slices)
 - Approximiere MIS jedes Teilgraphen und füge Ergebnisse zusammen
- Suche nach der besten i -ten Lösung

Donuts mit $k=3$, $i=0$ und $i=2$



Tree Decomposition



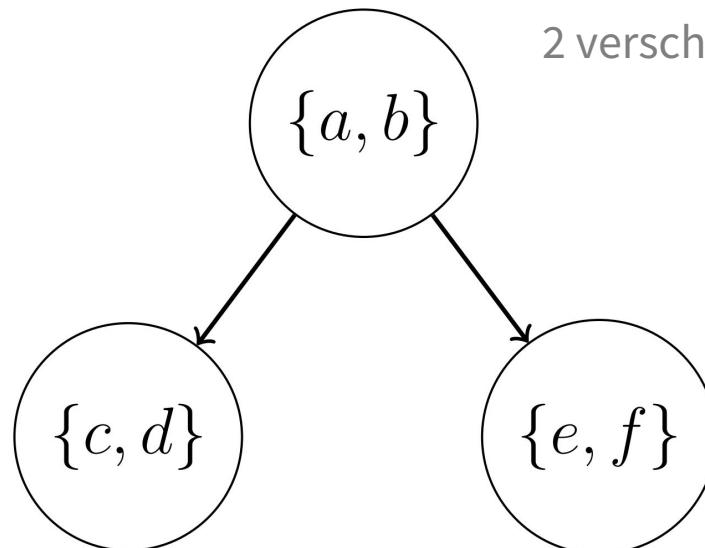
Tree Decomposition aus dem Planaren Graph

- Knoten der Faces sind die Knoten in einem Bag
- Kanten im Dualgraphen(außer Spannbaumkanten) sind die Kanten des Baums
- Für alle Knoten im Face den Spannbaum “zurück” laufen und Knoten auf dem Weg dem Bag hinzufügen
- Graph ist trianguliert → jedes Face beinhaltet 3 Knoten
- Spannbaum Rückweg enthält max K-Knoten → Bag maximal 3K Knoten

Konstruktion: Nice Tree Decomposition

Für alle Bags gilt:

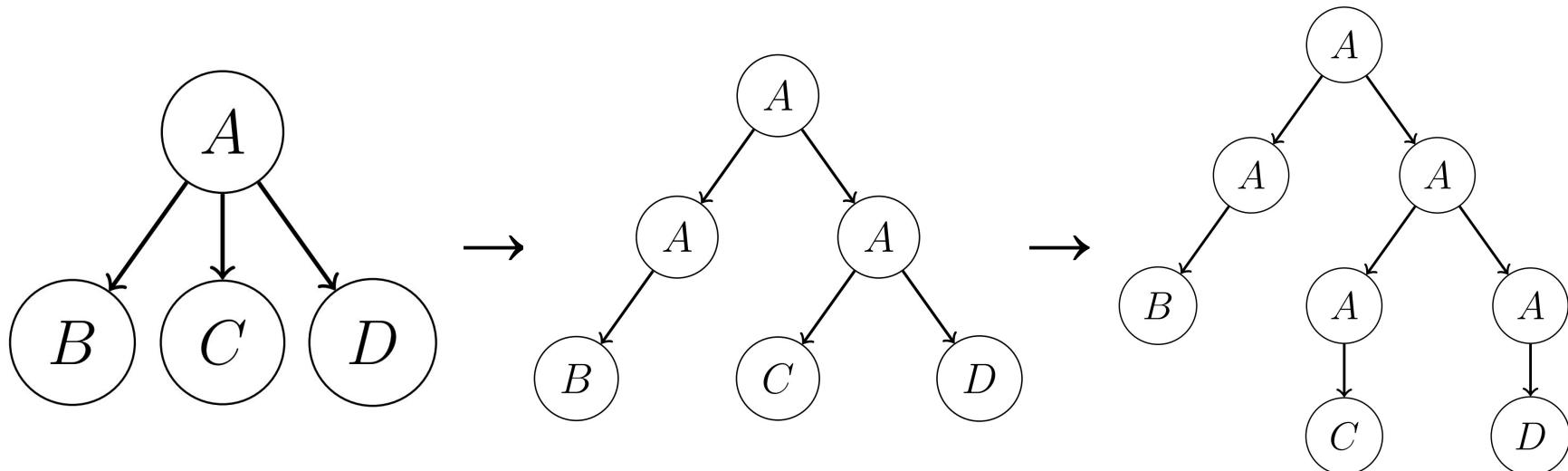
- Es sind keine Kind-Bags mit unterschiedlichen Mengen erlaubt
- Ein Kind-Bag darf nur einen Knoten mehr oder weniger als der Elternknoten haben



2 verschiedene Kinder → Keine Nice TD.

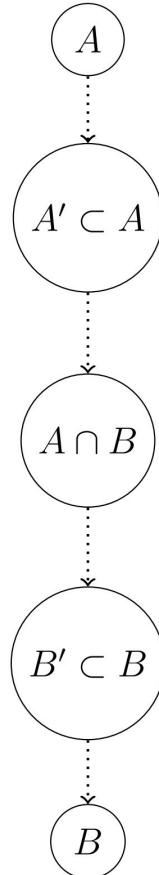
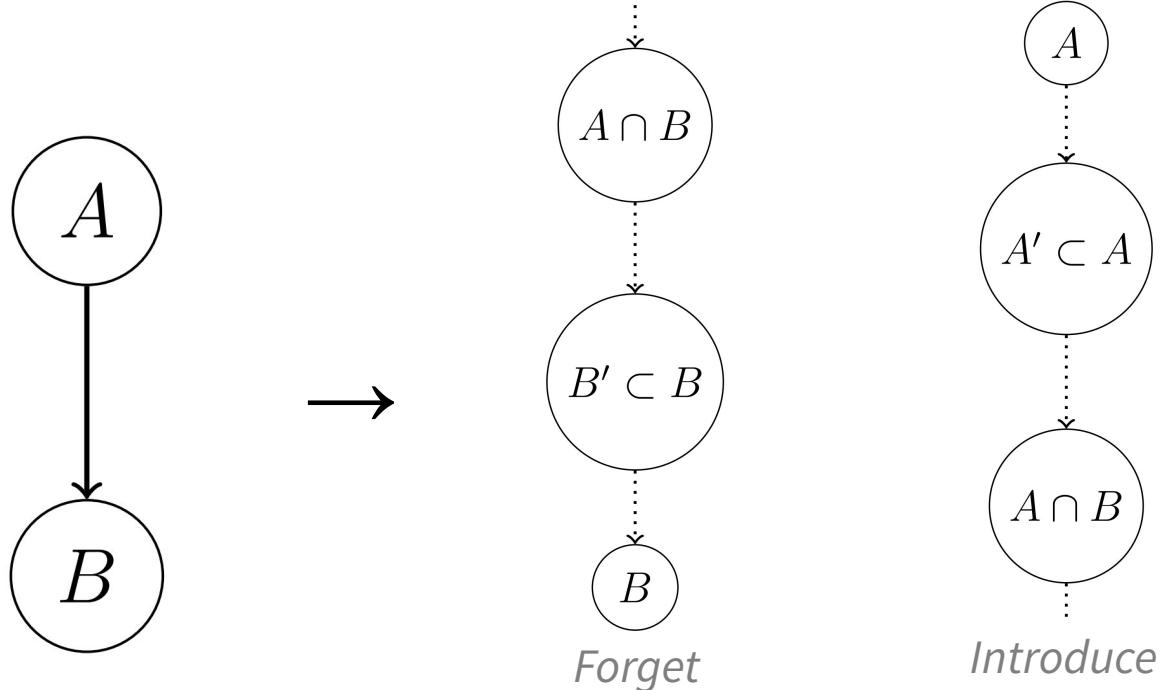
Konstruktion: Nice Tree Decomposition

1. Joins: Parent **duplizieren**, bis kein Knoten mehr unterschiedliche Kindknoten hat.



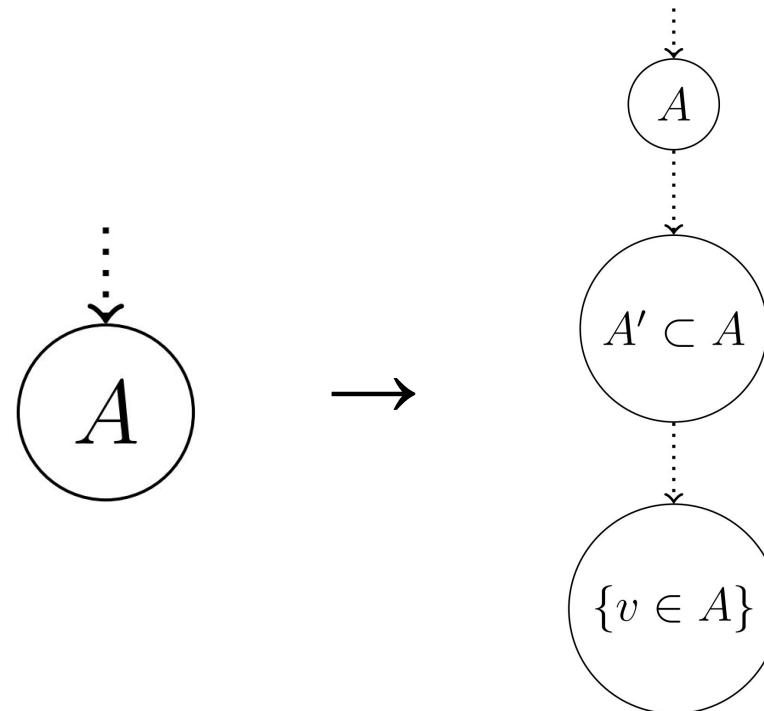
Konstruktion: Nice Tree Decomposition

2. Innere Knoten: *Introduce* und *Forget*, bis zur **Schnittmenge**.

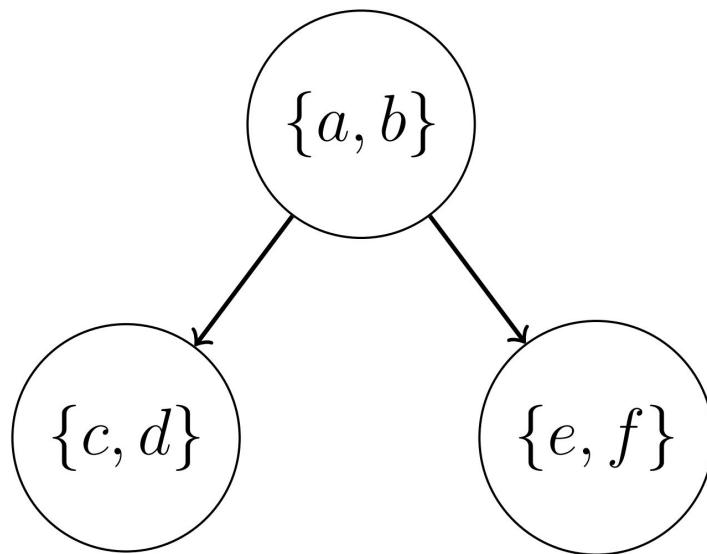


Konstruktion: Nice Tree Decomposition

3. Blattknoten: *Introduce* bis noch **ein Element** im Bag ist.

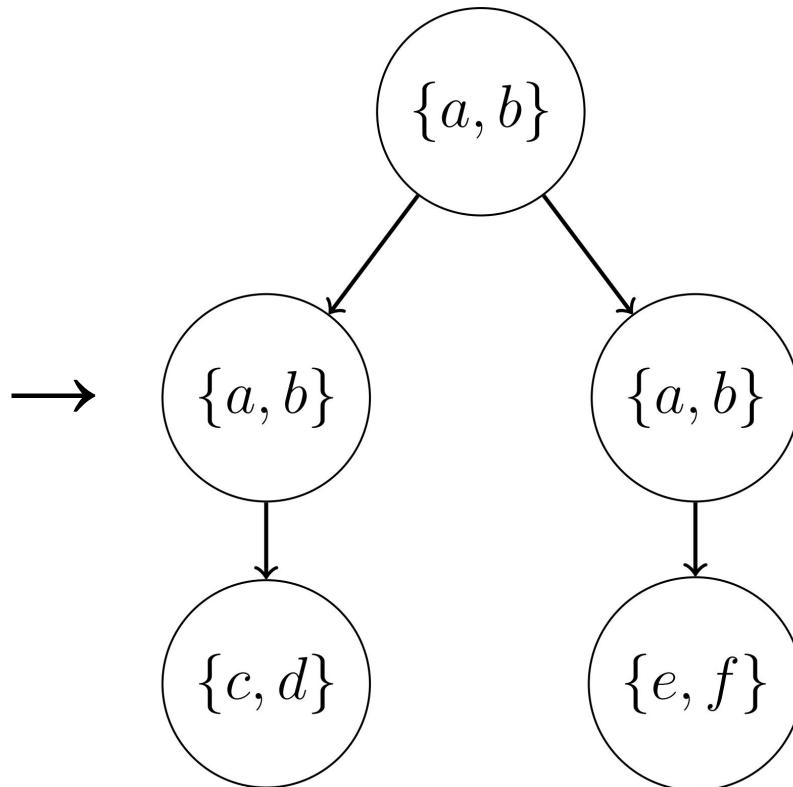


Konstruktion: Nice Tree Decomposition



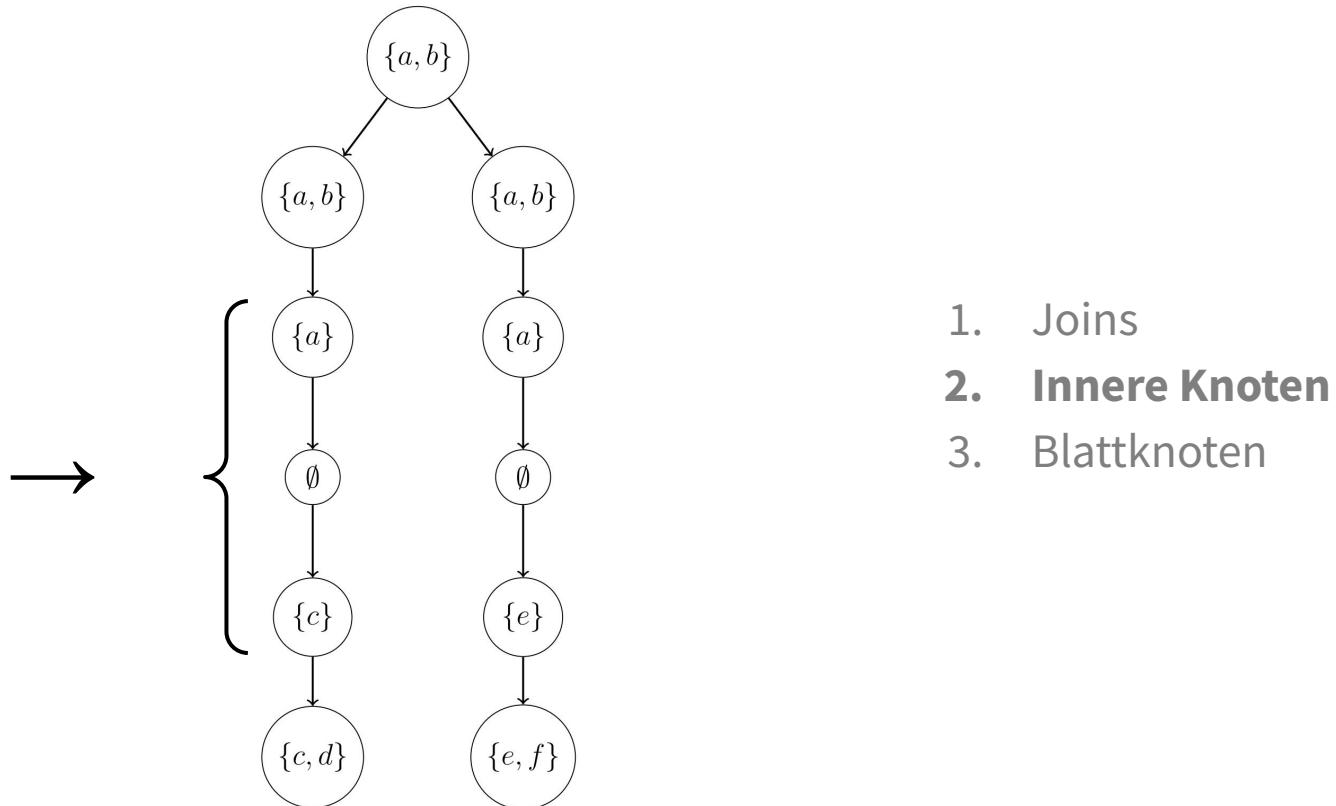
1. Joins
2. Innere Knoten
3. Blattknoten

Konstruktion: Nice Tree Decomposition

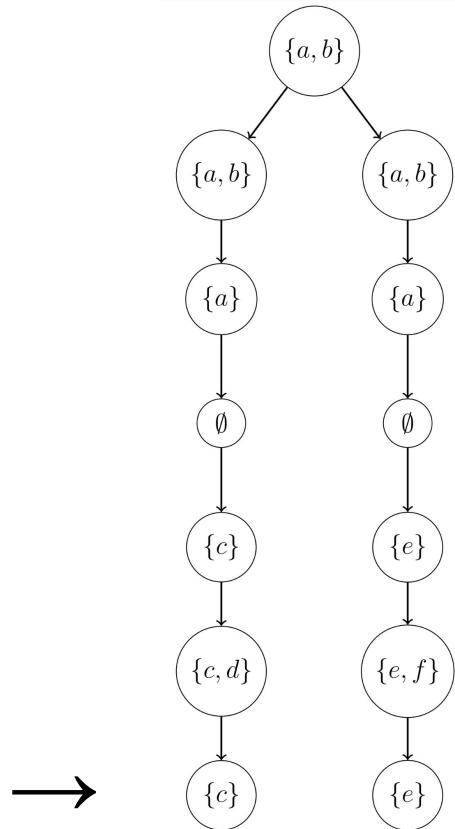


1. **Joins**
2. Innere Knoten
3. Blattknoten

Konstruktion: Nice Tree Decomposition



Konstruktion: Nice Tree Decomposition



1. Joins
2. Innere Knoten
3. Blattknoten

Code zur NTD und MIS

```
impl From<&DualGraph<'_>> for TreeDecomposition {
    fn from(dual_graph: &DualGraph) -> Self {
        ...
    }
}

pub trait NiceTreeDecomposition {
    fn make_nice(&self) -> TreeDecomposition;
    fn find_max_independent_set(&self) -> HashSet<usize>;
}

let td      = TreeDecomposition::from(&dual_graph);
let nice_td = td.make_nice();
let mis     = nice_td.find_max_independent_set();
```

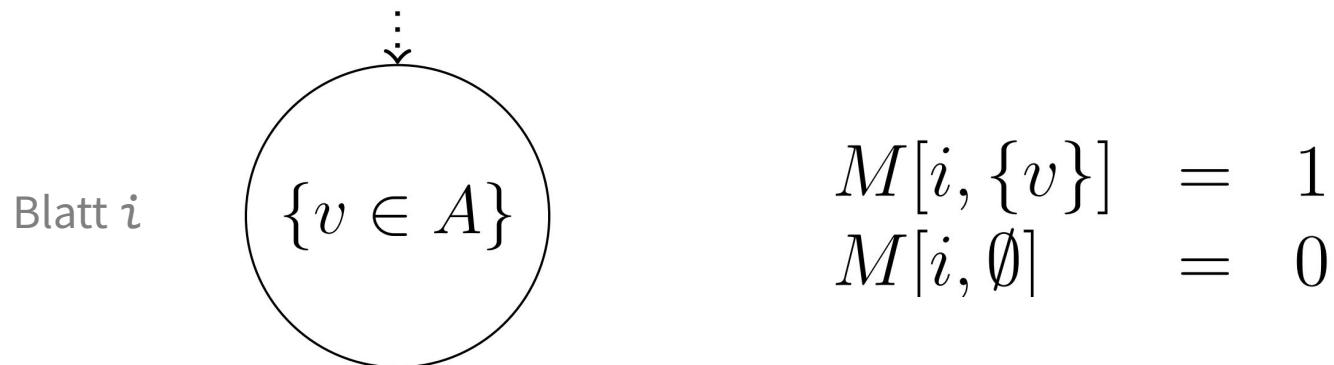
MIS mit dynamic programming auf Nice TD

- Post-Order Baumtraversierung (links, rechts, parent) → bottom-up Analyse
- Unterscheidung nach Knotenart
- Tabelle speichert die Größe des (Teil-) MIS, mit einem Rekonstruktionsalgorithmus kann die Menge berechnet werden

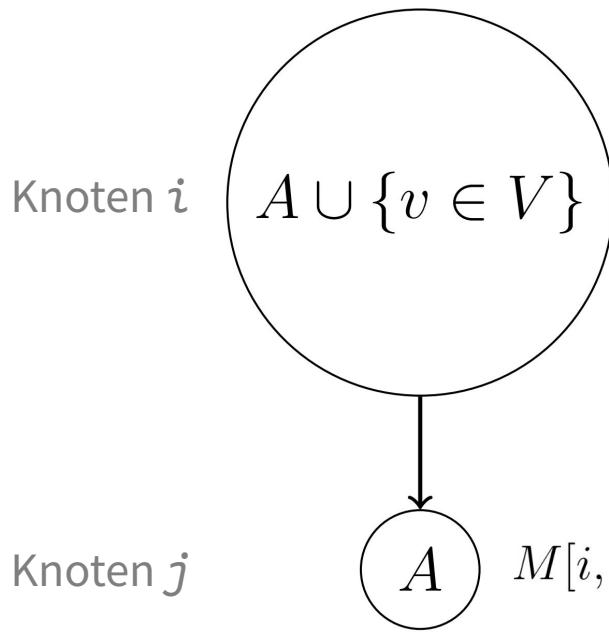
$$M : \underbrace{\mathbb{N}}_{\text{Bag-ID}} \times \underbrace{\mathcal{P}(V)}_{\text{MIS}} \longrightarrow \mathbb{Z}$$
$$(i, S) \longmapsto M[i, S]$$

MIS mit dynamic programming auf Nice TD

- Post-Order Baumtraversierung (links, rechts, parent) → bottom-up Analyse
- Unterscheidung nach Knotenart



MIS mit dynamic programming auf Nice TD



Introduce

$$M[i, \{v, w\}] = -\infty$$

$$M[i, \{w\}] = 1$$

$$M[i, \{v\}] = 1$$

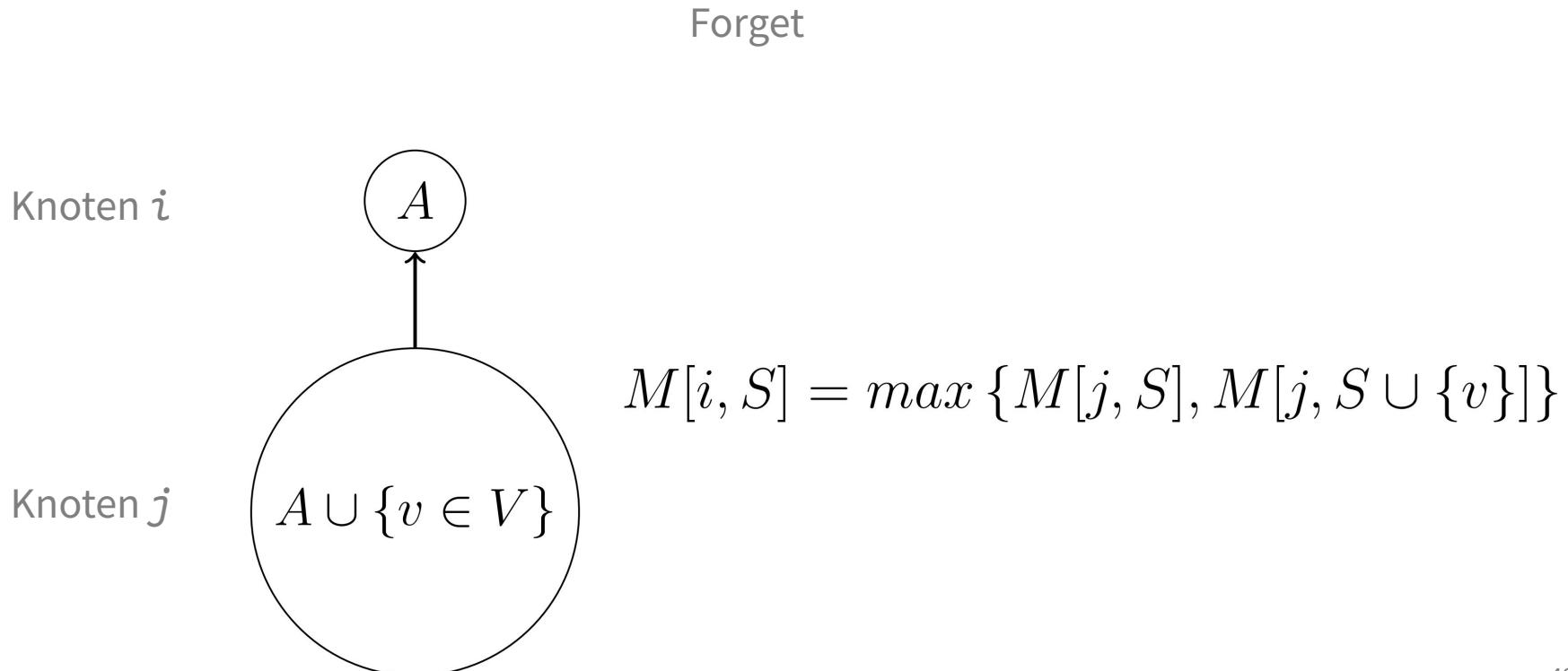
$$M[i, \emptyset] = 0$$

$$M[j, \{w\}] = 1$$

$$M[j, \emptyset] = 0$$

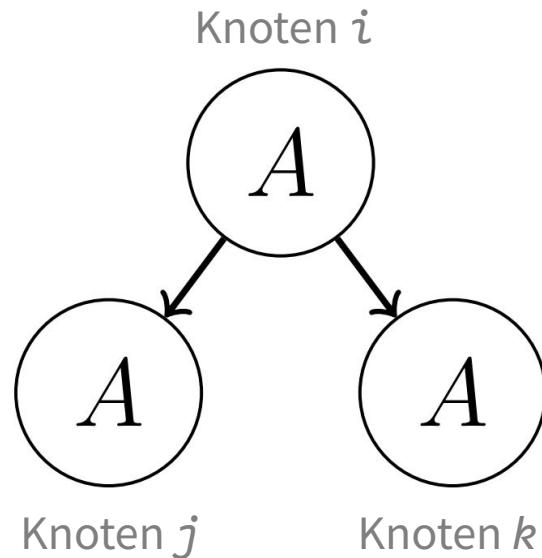
$$M[i, S] = \begin{cases} M[j, S] & \text{falls } v \notin S \\ M[j, S \setminus \{v\}] + 1 & \text{falls } v \in S \text{ und } S \text{ ist unabhängig} \\ -\infty & \text{sonst} \end{cases}$$

MIS mit dynamic programming auf Nice TD



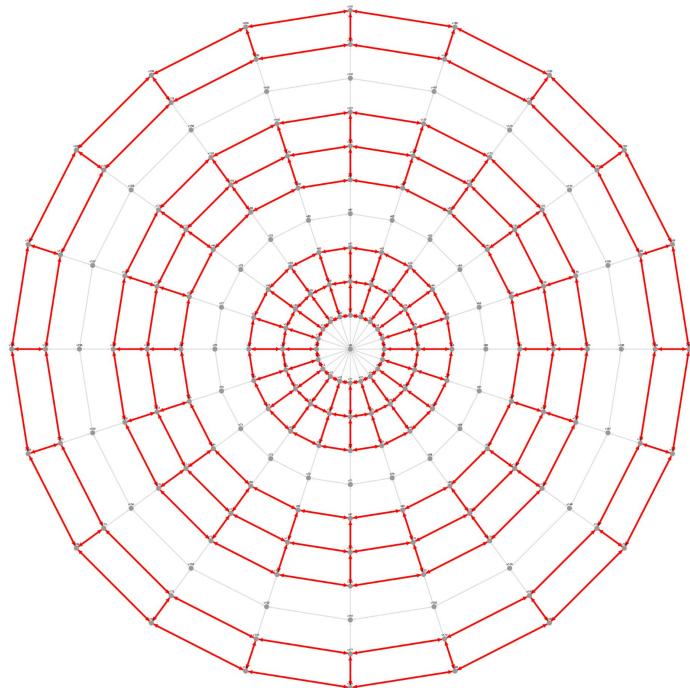
MIS mit dynamic programming auf Nice TD

Join



$$M[i, S] = M[j, S] + M[k, S] - |S|$$

MIS vereinen



Laufzeit : $\mathcal{O}(8^k \cdot k \cdot n)$

Fazit und Ausblick

- Sehr aufwändige Vorbereitung
 - Nicht lohnenswert für kleine Graphen
- Teilsysteme sind in einem fortgeschrittenen Stadium, aber noch nicht zusammengefügt
- Messungen von Performance und Speicherbedarf für verschiedene Eingaben
- Vergleich mit anderen Zwischenberechnungen oder Lösungen
 - Z.B. Tree decomposition von arboretum-td berechnen lassen
 - exakte Lösungen