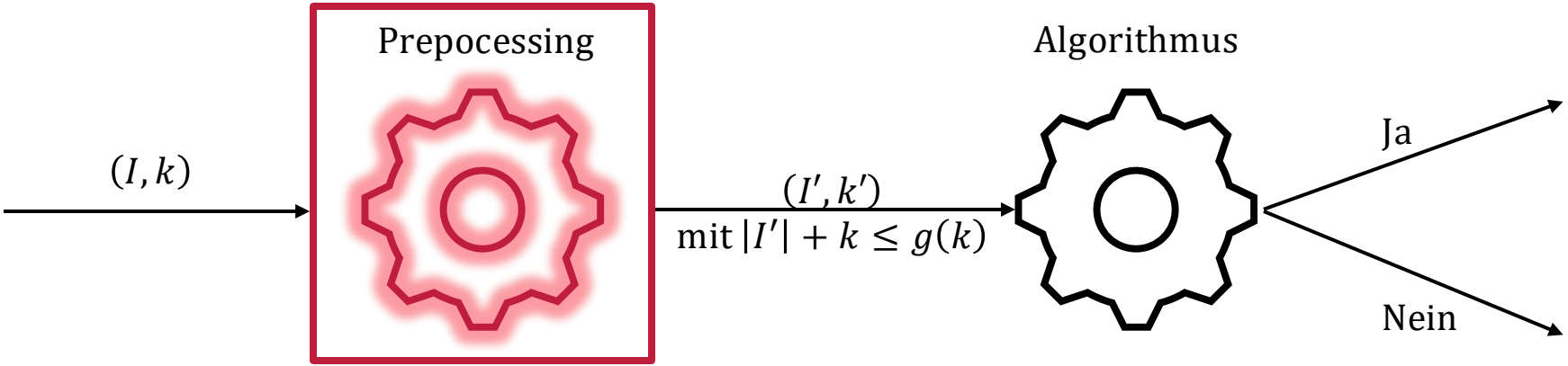
Technische Universität Braunschweig

Parametrisierte Algorithmen

Arne Schmidt

Wiederholung

Parametrisierter Algorithmus mit Kernelisation



Polynomielle Zeit

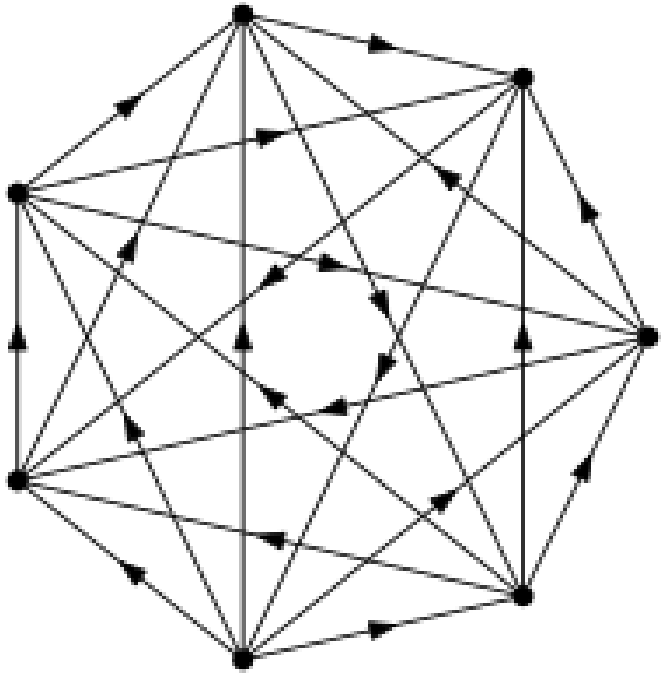
i.d.R Anwenden von Reduktionsregeln

Kapitel 2.2.2 – Feedback Arc Set in Tournament Graphen

Tournament Graphen

Definition 2.5
Ein gerichteter Graph $D = (V, A)$ heißt **Tournament Graph**, wenn für je zwei Knoten $u, v \in V$ entweder $(u, v) \in A$ oder $(v, u) \in A$ (aber nicht beide).

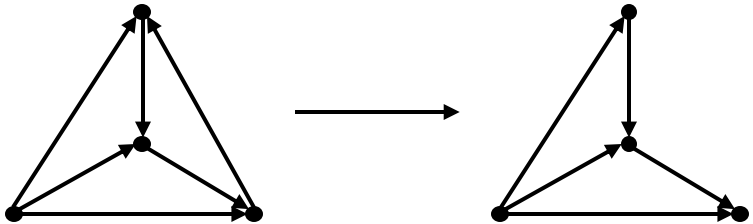
Lemma 2.6
Existiert in einem Tournament Graphen ein gerichteter Kreis, dann existiert ein gerichteter Kreis mit drei Knoten ("Dreieck").



Feedback Arc Set in Tournament Graphen

Problem 2.7 (FAST)
Gegeben: Tournament Graph $D = (V, A)$, Zahl $k \in \mathbb{N}$.
Frage: Exsitiert eine Kantenmenge $F \subseteq A$, sodass $D' = (V, A \setminus F)$ azyklisch ist?

Zwischenfrage:
Gilt FAST $\in$ XP?



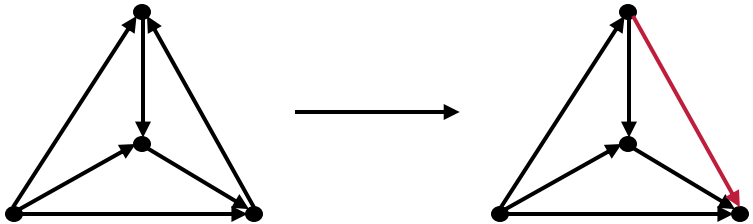
$k = 1$

Lemma 2.8
Sei $D = (V, A)$ ein Tournament Graph und $F \subseteq A$. Dann ist F genau dann ein inklusions-minimales Feedback Arc Set, wenn F eine inklusions-minimale Menge ist, sodass D durch Umdrehen der Kanten in F azyklisch wird.

Feedback Arc Set in Tournament Graphen

Problem 2.7 (FAST)
Gegeben: Tournament Graph $D = (V, A)$, Zahl $k \in \mathbb{N}$.
Frage: Exsitiert eine Kantenmenge $F \subseteq A$, sodass $D' = (V, A \setminus F)$ azyklisch ist?

Lemma 2.8
Sei $D = (V, A)$ ein Tournament Graph und $F \subseteq A$. Dann ist F genau dann ein inklusions-minimales Feedback Arc Set, wenn F eine inklusions-minimale Menge ist, sodass D durch Umdrehen der Kanten in F azyklisch wird.



$k = 1$

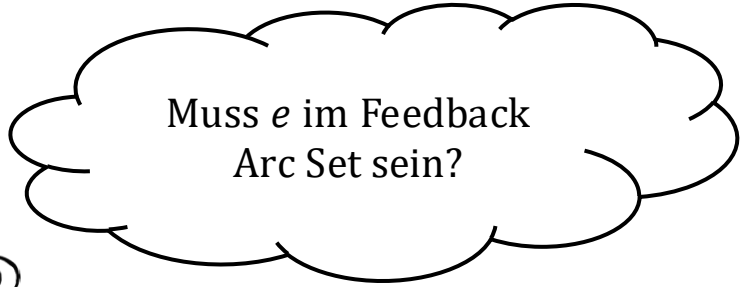
Konzentriere dich also auf folgende 2 Punkte:

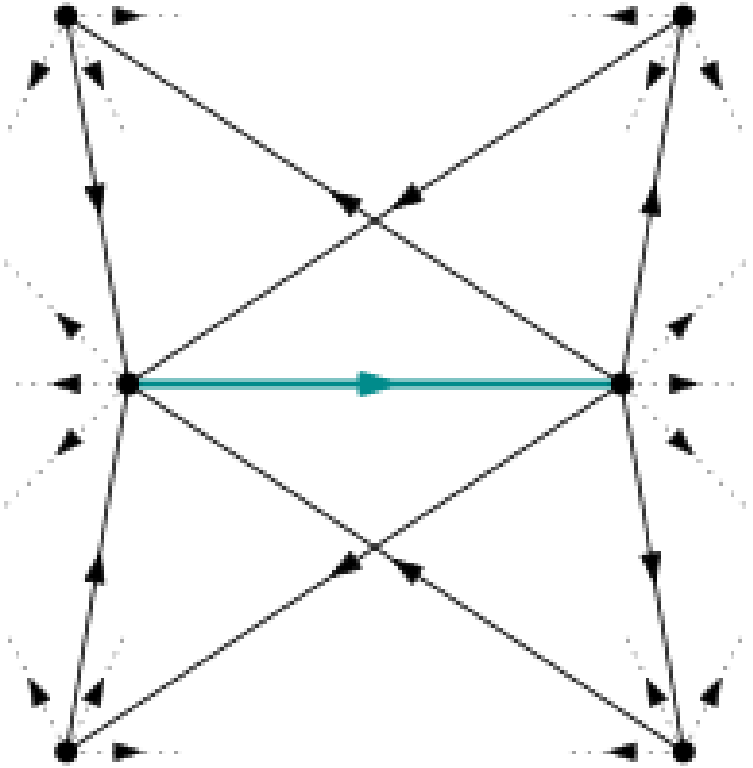
Drehe maximal k Kanten um.

Betrachte als Kreise nur Dreiecke.

Idee für Reduktionsregel

Angenommen, eine Kante e liegt auf $> k$ Dreiecken.





Reduktionsregeln FAST

Regel FAST1:
Existiert eine Kante e, welche auf $> k$ Dreiecken liegt, dann drehe e um und reduziere k um 1 .

Etwas schwieriger zu sehen:

Regel FAST2:
Existiert ein Knoten v, welcher auf keinem Dreieck liegt, dann lösche v.

Lemma 2.9
Regeln FAST1 und FAST2 sind sicher.

Beweis: Tafel!

Reduktionsregeln FAST

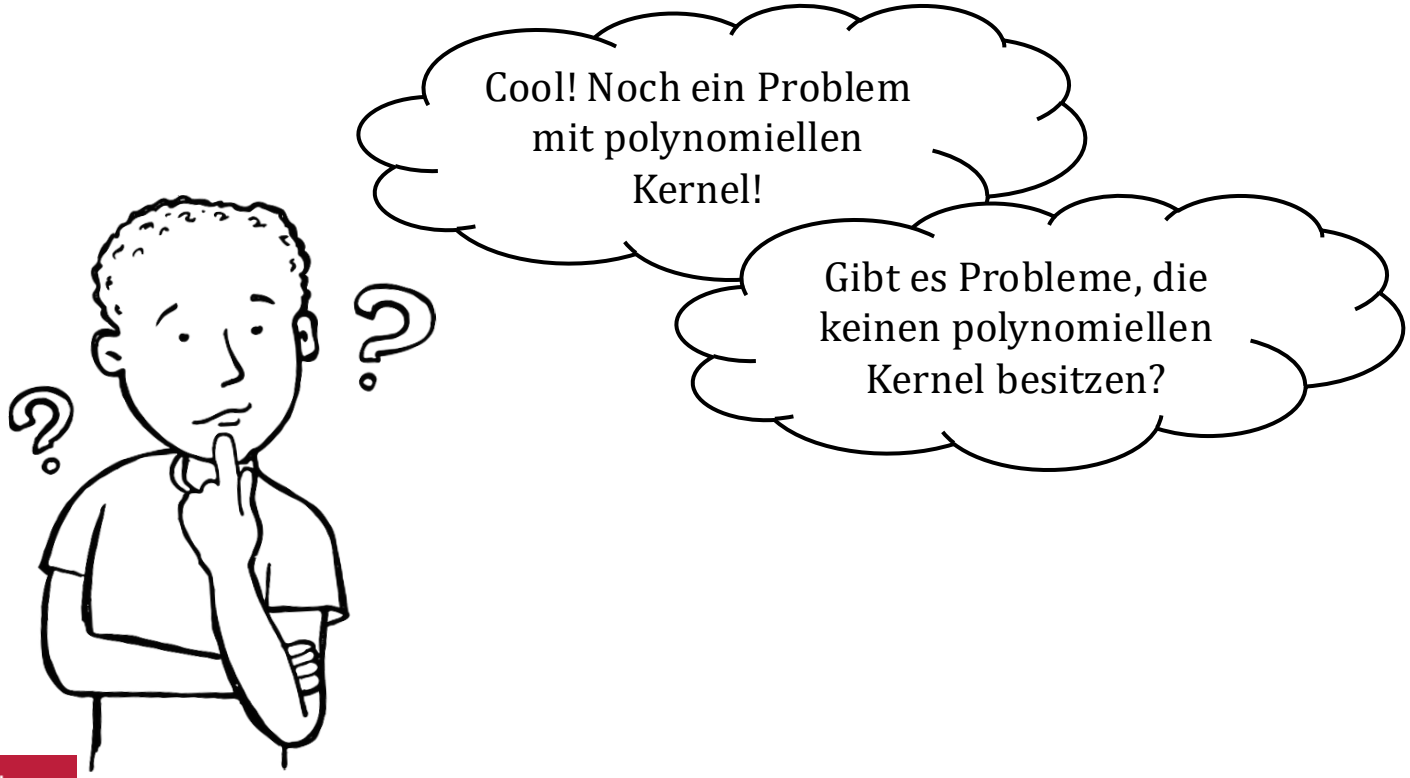
Regel FAST1:
Existiert eine Kante e, welche auf $> k$ Dreiecken liegt, dann drehe e um und reduziere k um 1 .

Etwas schwieriger zu sehen:

Regel FAST2:
Existiert ein Knoten v, welcher auf keinem Dreieck liegt, dann lösche v.

Theorem 2.10
FAST erlaubt einen Kernel mit maximal $k^2 + 2k$ Knoten.

Beweis: Tafel!


Cool! Noch ein Problem mit polynomiellen Kernel!
Gibt es Probleme, die keinen polynomiellen Kernel besitzen?

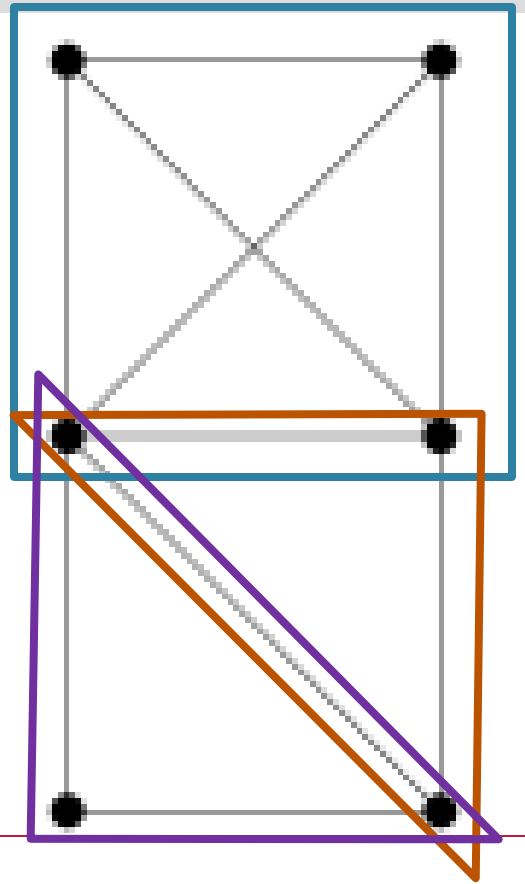
Kapitel 2.2.3 – Edge Clique Cover

Edge Clique Cover

Problem 2.11
Gegeben: Ein Graph $G = (V, E)$, eine Zahl $k \in \mathbb{N}$.
Frage: Existieren Cliquen $C_1, \dots, C_k$ in G, sodass

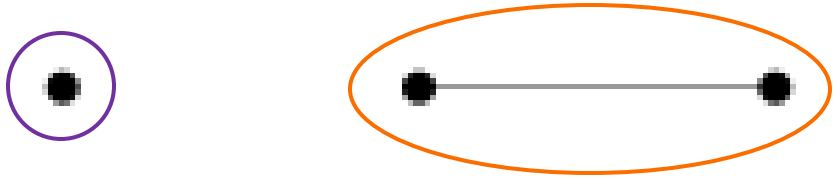
$$\bigcup_{i=1}^{k} E(C_i) = E$$

Zur Erinnerung:
In einer Clique ist jeder mit jedem verbunden.

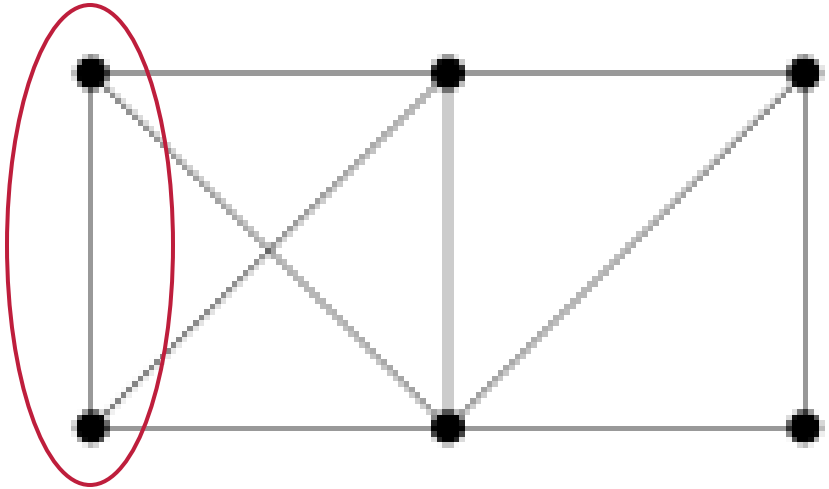


Reduktionsregeln

Regel ECC1
Existiert ein isolierter Knoten, entferne diesen.



Regel ECC2
Existiert eine isolierte Kante, entferne diese und reduziere k um 1.



Regel ECC3
Gilt für eine Kante $\{u, v\} \in E$: $N[u] = N[v]$, dann entferne v.
Für $v \in V$ ist $N[v] := N(v) \cup \{v\}$.

ECC Kernel

Regel ECC1
Existiert ein isolierter Knoten, entferne diesen.

Regel ECC2
Existiert eine isolierte Kante, entferne diese und reduziere k um 1.

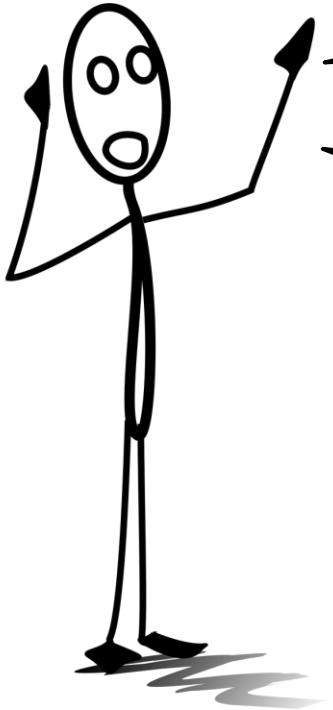
Regel ECC3
Gilt für eine Kante $\{u, v\} \in E$: $N[u] = N[v]$, dann entferne v.
Für $v \in V$ ist $N[v] := N(v) \cup \{v\}$.

Lemma 2.12
Die Regeln ECC1, ECC2 und ECC3 sind sicher.

Beweis: Tafel

Theorem 2.13
ECC besitzt einen Kernel der Größe 2^k.

Beweis: Tafel



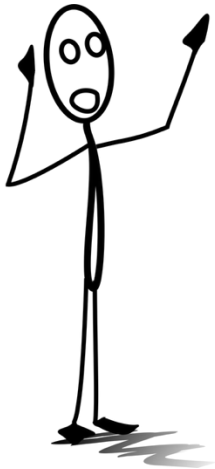
Der Kernel sagt uns: ECC liegt in FPT!

Es wird vermutet, dass kein polynomieller Kernel existiert.

Recap

Idee: Wende einfache Regeln an, um die Instanz zu verkleinern.

Ist nach Anweden der Regeln (insgesamt poly-Zeit!) die Größe der Instanz nur durch den Parameter nach oben beschränkt, ist das Problem in FPT.



Es gibt beim Design des Kernels verschiedene Herausforderungen

1. Minimiere die Größe des Kernels

2. Minimiere die Laufzeit zum Berechnen des Kernels