# Fopra - Erweiterung der JGraphT-Bibliothek

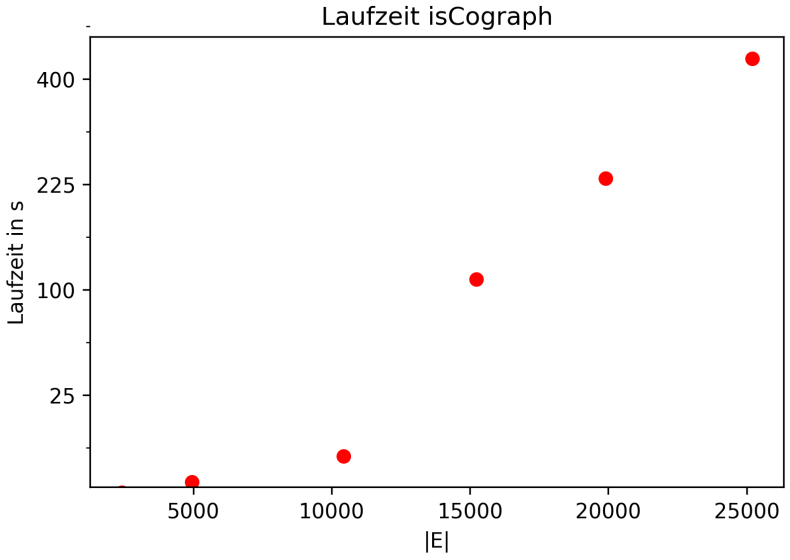
Lars Rafeldt Marcian Seeger Lennart Wolf
Lars Reuter

August 3, 2020

JGraphT ist eine Java-Bibliothek, die zahlreiche Datenstrukturen zum Verwalten von Graphen, sowie Algorithmen, welche auf selbigen operieren, zur Verfügung stellt. Ziel der Arbeit in diesem Fortgeschrittenenpraktikum war es, die JGraphT-Bibliothek zu erweitern. Ein besonderes Augenmerk wurde dabei auf Algorithmen zur Erkennung von bestimmten Graphklassen gelegt, welche im Folgenden vorgestellt werden.

## Cographen

Für die Klasse der Cographen existieren einige äquivalente Definitionen. So ist zum Beispiel die Klasse der Cographen äquivalent zur Klasse der P4-freien Graphen. Ein Cograph enthält somit keine knotendisjunkten Kanten, deren Knoten durch genau eine weitere Kante verbunden sind. Um diese Überprüfung durchzuführen wird für jede Kante des Graphen über die Menge aller anderen Kanten iteriert, weshalb sich eine Laufzeit von $\mathcal{O}(|E|^2)$ ergibt. Die Laufzeit dieses Algorithmus wurde mithilfe von Cliquen verschiedener Größen überprüft. Alle Cliquen sind Cographen, weshalb der Algorithmus keine P4s finden kann, sondern alle Kantenquadrupel überprüfen muss und somit in Worst-Case-Laufzeit terminiert.




## Thresholdgraphen

Ein Graph mit einem einzigen Knoten ist ein Thresholdgraph. Alle anderen Thresholdgraphen erhält man durch beliebig häufiges Hinzufügen von einzelnen Knoten zur Knotenmenge oder Hinzufügen eines dominierenden Knoten, also einem Knoten, der mit jedem anderen Knoten des bisherigen Graphen eine Kante teilt. Sortiert man alle Knoten nach ihrem Grad, muss also entweder ein Knoten den Grad 0 haben oder aber maximalen Grad. Diesen Knoten kann man dann entfernen und den Restgraphen betrachten. Wenn einmal kein Knoten maximalen Grad oder Grad 0 hat, kann es sich nicht um einen Thresholdgraphen handeln. Kann die Überprüfung bis zu einem leeren Restgraphen wiederholt werden, ist der Graph ein Thresholdgraph. Es ergibt sich eine lineare Laufzeit von $\mathcal{O}(|V| + |E|)$.

# Quasi-Thresholdgraphen

Ein Graph mit einem einzigen Knoten ist ein Quasi-Thresholdgraph. Das Hinzufügen eines dominierenden Knotens, resultiert wieder in einem Quasi-Threshold Graphen. Die disjunkte Vereinigung zweier Quasi-Thresholdgraphen ist wieder ein Quasi-Thresholdgraph. Durch rekursives Ausführen dieser Schritte können größere Quasi-Thresholdgraphen aufgebaut werden. Der folgende Algorithmus[2] hat eine Laufzeit $\mathcal{O}(|V|^2 + |E|)$.

**Algorithm QT.** *Test whether a graph is a quasi-threshold graph.*
*Input: A graph $G = (V,E)$ with $V = \{v_1, v_2, \ldots, v_n\}$.*
*Output: A rooted forest representation $F$ of $G$ if $G$ is a quasi-threshold graph. Otherwise output "no."*
*Method.*
calculate $deg(v_i)$ for each vertex $v_i$ in $G$;
**for** $i = 1$ **to** $n$ **do** $indegree(v_i) \longleftarrow 0$;
**for** each edge $\{v_i, v_j\}$ in $G$ **do**
  **if** $deg(v_i) > deg(v_j)$ **or** $(deg(v_i) = deg(v_j)$ **and** $i < j)$
  **then** $indegree(v_j) \longleftarrow indegree(v_j) + 1$
  **else** $indegree(v_i) \longleftarrow indegree(v_i) + 1$;
$F \longleftarrow \emptyset$; /* a digraph with vertex set $\{v_1, v_2, \ldots, v_n\}$ and no edges */
**for** $j = 1$ **to** $n$ **do**
  **if** $indegree(j) \geqslant 1$
  **then** choose a vertex $v_i \in N_G[v_j]$ such that $deg(v_i) > deg(v_j)$ or
    $(deg(v_i) = deg(v_j)$ and $i < j)$ and indegree($i$) is largest;
  add the arc $(i, v_j)$ into $F$;
  **end do**;
use a depth-first search to compute the number $anc(v_j)$ of ancestors of $v_j$ in $F$ for each $j$;
**if** $indegree(v_j) = anc(v_j)$ for all $j$ **then** output $F$ **else** answer "no".

# Permutationsgraphen

Ein Graph ist ein Permutationsgraph, wenn die Knoten Elemente einer Permutation und die Kanten des Graphen die Elementpaare, die bei der Permutation vertauscht wurden, darstellen. In der Permutation müssen die Elemente mit denen ein Element verbunden und die einen kleineren Wert als jenes Element haben, rechts von diesem Element stehen. Zum Erkennen von Permutationsgraphen erstellt man also eine Permutation anhand des gegebenen Graphen und prüft ob diese mit der Permutation dieses Graphen übereinstimmt. Der implementierte Algorithmus[4] hat eine Laufzeit von $\mathcal{O}(|V|^2 + |E|)$.

**Algorithm**

1. Let $U_{11} = \{i \in V : (1,i) \in E\}$ and $U_{12} = V - E_{11}$, and then record
$$U_{11} \quad 1 \quad U_{12}$$
If $U_{11}$ or $U_{12}$ is empty, then drop the empty set from the record.
2. Let $a$ and $b$ be the smallest number in the set $U_{11}$ and $U_{12}$, respectively. Let $U_{a1} = \{i \in U_{11} : (a,i) \in E\}$, $U_{a2} = U_{11} - U_{a1}$ and $U_{b1} = \{j \in U_{12} : (b,j) \in E\}$, $U_{b2} = U_{12} - U_{b1}$, and then record
$$U_{a1} \quad a \quad U_{a2} \quad 1 \quad U_{b1} \quad b \quad U_{b2}$$
If any set of the above is empty then drop it.
3. Apply the same process for each nonempty set as in (2) until no nonempty set is left.
4. The resulting numbers
$$i_1, i_2, \ldots, i_n$$
is the desired permutation $\pi$, where $\pi(j) = i_j$.

# P5-Free-Graphen

Ein Graph ist dann ein P5-frei, wenn er eine dominierende Clique oder einen dominierenden P3 enthält. Der implementierte Algorithmus bestimmt erst eine schwache Zusammenhangskomponente $V = (A, N, R)$ des Graphen. Dabei ist $A$ eine beliebige Knotenmenge, sodass die Bedingungen aus Algorithmus 1 gelten, $N$ die Nachbarschaft von $A$ und $R$ die Menge aller Knoten des Graphen ohne $A$ und $N$. Im Laufe des Algorithmus 1 werden $A$, $N$ und $R$ so verändert, dass eine maximale schwache Dekomposition des Graphen erreicht wird. Mithilfe der schwachen Dekomposition kann dann in Algorithmus 2 der Graph auf P5s überprüft werden. Der implementierte Algorithmus[5] hat eine Laufzeit von $\mathcal{O}(|V| \cdot (|V| + |E|))$.

**Algorithm 1:** Weak decomposition of a graph [23]

*Input:* $G = (V, E)$ connected graph that have two or more nonadjacent vertices.
*Output:* $V = (A, N, R)$ partition in that $G(A)$ is connected, $N = N(A)$, $A \not\sim R = \overline{N}(A)$.

*Begin*
  $A :=$ any set of vertices such that,
  $V \neq A \cup N(A)$;
  $N := N(A)$;
  $R := V - A \cup N(A)$;
  *While* $(\exists r \in R, \exists n \in N$ such that $nr \notin E)$ *Do*
    $N := (N - \{n\}) \cup (N(n) \cap R)$;
    $A := A \cup \{n\}$;
    $R := R - (N(n) \cap R)$;
  *EndWhile*
*End*

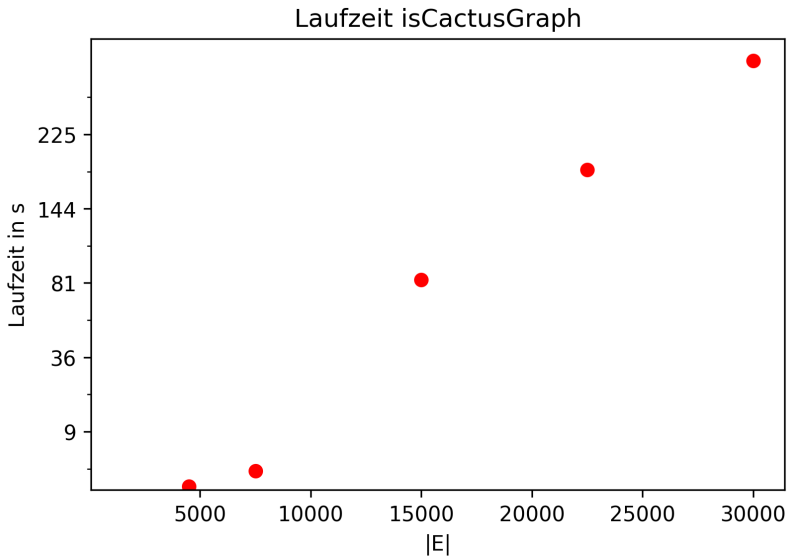
**Algorithm 2:** Recognition algorithm for $P_5$-free graphs

*Input:* $G = (V, E)$ a connected bipartite graph with two or more nonadjacent vertices.
*Output:* The answer to the issue: Is $G$ a $P_5$-free graph?

*Begin*
  $L = \{G\}$; / $L$ represents a list of graphs.
  Let $H$ be in $L$.
  *While* $(|V(H)| > 5)$ *Do*
    1. Determine the degree of each vertex
    2. Determine a weak decomposition $(A, N, R)$ with $N \sim R$ for $H$;
    3. Determine $B = N_H(N) - R$ and $C = A - B$;
    4. Let: $nr := |N|; r := |R|; b := |B|$;
    5. *If* $(\exists v \in R$ such that $d_H(v) \neq nr)$ *Then* The graph $G$ is not $P_5$-free
      *ElseIf* $(\exists v \in N$ so that $d_H(v) \neq b + r)$ *Then*
      Graph $G$ is not $P_5$-free
        *Else*
        Insert, in $L$, the induced subgraph of $A$ (at each iteration the graph is called $H$, so $H = [A]$) of order strictly higher than 5.
    *EndIf*
  *EndWhile*
  6. Graph $G$ is $P_5$-free
*End*

## Kaktusgraphen

Kaktusgraphen sind Graphen, in denen keine Kante Teil von mehr als zwei Zyklen ist. JGraphT stellt eine Funktion zur Ermittlung einer CycleBasis nach Paton bereit. Diese besteht aus einer minimalen Menge von Zyklen im Graphen, aus denen sich alle größeren Zyklen zusammensetzen lassen. Die Laufzeit dieses Algorithmus beträgt im Worst Case $\mathcal{O}(|V|^3)$. Für jede Kante im Graphen muss dann über die Cycle Base iteriert werden, welche im Worst Case aus disjunkten Kantenmengen besteht, deren Vereinigung der Menge aller Kanten entspricht. Es ergibt sich hierfür eine Laufzeit von $\mathcal{O}(|E|^2)$. Zur Überprüfung wurden Graphen verschiedener Größen erstellt, die ausschließlich aus kantendisjunkten Zyklen bestehen. So wird die CycleBasis maximal groß und der Algorithmus terminiert nicht bevor alle Kanten überprüft worden sind, da ein Kaktusgraph vorliegt.

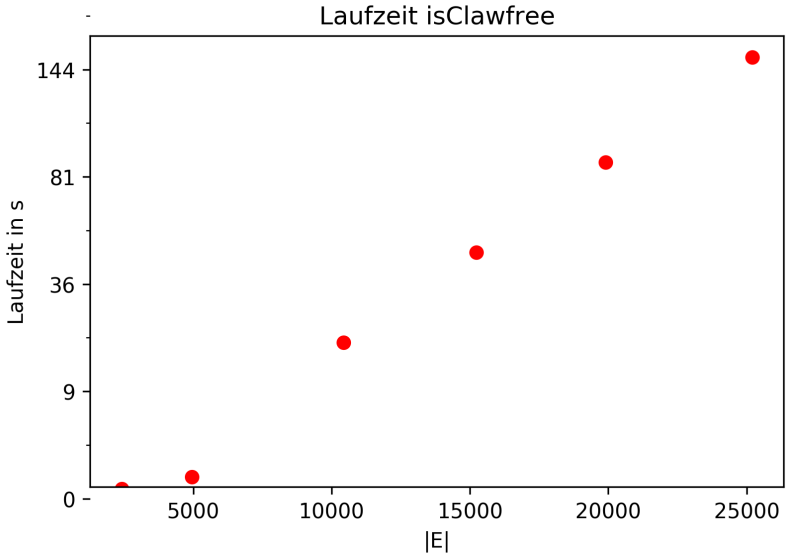



## k-Zusammenhangskomponente

Unter dem k-Zusammenhang eines Graphen wird anschaulich die kleinste Zahl k von Kanten verstanden, nach deren Löschung ein Graph nicht mehr zusammenhängend ist. Die Ford-Fulkerson-Methode löst das Max-Flow-Problem, welches nach dem Max-Flow-Min-Cut-Theorem dem Min-Cut-Problem entspricht. Da es sich bei den betrachteten Graphen um ungewichtete Kanten handelt, entspricht die Größe des kleinsten Min-Cut im Graphen (für jedes Quelle-Senke-Paar existiert ein Min-Cut) gerade k. JGraphT stellt den Edmonds-Karp-Algorithmus zur Lösung des Max-Flow-Problems bereit, welcher eine Laufzeit von $\mathcal{O}(|V| \cdot |E|^2)$ aufweist. Es müssen genau $|V| \cdot (|V| - 1)$ Max-Flow-Probleme gelöst werden, weshalb sich insgesamt eine Laufzeit von $\mathcal{O}(|V|^2 \cdot |E|^2)$ ergibt.

## Clawfreegraphen

Die sogenannte Claw ist eine Graphstruktur aus vier Knoten, wobei ein Knoten der zentrale Knoten ist, von dem jeweils eine Kante zu den anderen drei Knoten führt. Die anderen drei Knoten haben untereinander jedoch keine gemeinsamen Kanten. In clawfreien Graphen sind alle induzierten Subgraphen keine Claws. Ein naiver Lösungsansatz besteht darin, alle Knotenquadrupel auf eine Claw zu überprüfen. Hierfür ergibt sich eine Laufzeit von $\mathcal{O}(|V|^4)$. Der hier umgesetzte Ansatz hat die gleiche Worst-Case-Laufzeit, ist für weniger dichte Graphen aber schneller. Es wird über die Nachbarschaften aller Knoten iteriert. Wenn in einer dieser Nachbarschaften drei Knoten existieren, die keine gemeinsame Kante haben, ist eine Claw gefunden. Äquivalent lässt sich sagen, dass ein Graph clawfrei ist, wenn kein Komplement einer Knotennachbarschaft über einen Dreierzyklus verfügt. Zur Überprüfung der Laufzeit wurden wieder Cliquen verwendet, da diese eine maximale Anzahl von Kanten haben und stets clawfree sind.




## HolefreeGraphen

Ein ungerichteter Graph G besitzt dann ein Hole, wenn G einen Kreis mit $u_0u_1 \dots u_k$ mit k  4 besitzt, sodass $u_iu_{i+1}u_{i+2}u_{i+3}$ für jedes i = 0, 1, . . . , k-3, und $u_{k-2}u_{k-1}u_ku_0$ P4s von G bilden. Also ist ein Kreis mit vier oder mehr Knoten ein Loch, wobei die Knoten im Kreis nur zu jeweils zwei Nachbarn aus dem Kreis eine Kante haben dürfen und zu keinem anderen Knoten im Kreis eine Kante besteht. Der implementierte Algorithmus[6] hat eine Laufzeit von $\mathcal{O}(|V| + |E|^2)$.

*Hole-Detection Algorithm*
*Input:* a connected undirected graph $G$.
*Output:* yes, if $G$ contains a hole; otherwise, no.

1. Initialize the entries of the arrays `not_in_hole[]` and `in_path[]` to 0;
   compute the adjacency matrix `A[]` of $G$;
2. For each vertex $u$ of $G$ do
   2.1 `in_path[`$u$`]` $\leftarrow 1$;
   2.2 for each edge $vw$ of $G$ do
   if $u$ is adjacent to $v$ and non-adjacent to $w$
   and `not_in_hole[`$(u,v),w$`]` $= 0$
   then `in_path[`$v$`]` $\leftarrow 1$;
   `process(`$u,v,w$`)`;
   `in_path[`$v$`]` $\leftarrow 0$;
   2.3 `in_path[`$u$`]` $\leftarrow 0$;
3. Print that $G$ does not contain a hole.

where the procedure `process()` is as follows:

`process(`$a,b,c$`)`
1. `in_path[`$c$`]` $\leftarrow 1$;
2. for each vertex $d$ adjacent to $c$ in $G$ do
   2.1 if $d$ is adjacent to neither $a$ nor $b$ in $G$
   then {*abcd is a* $P_4$ *of* $G$}
   2.2 if `in_path[`$d$`]` $= 1$
   then print that $G$ has a hole; Stop.
   else if `not_in_hole[`$(b,c),d$`]` $= 0$
   then `process(`$b,c,d$`)`;
3. `in_path[`$c$`]` $\leftarrow 0$;
4. `not_in_hole[`$(a,b),c$`]` $\leftarrow 1$;
   `not_in_hole[`$(c,b),a$`]` $\leftarrow 1$;

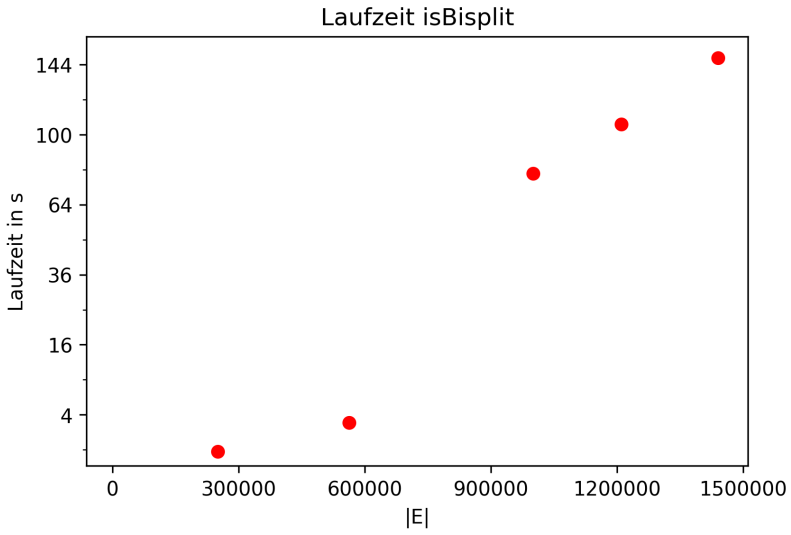
# Bisplitgraph

Ein Graph G ist ein Bisplitgraph oder auch Bipartite-Chain-Graph, wenn folgende zwei Bedingungen erfüllt sind:
1. Die Knoten von G lassen sich in zwei Knotenmengen/Partitionen $V_1$ und $V_2$ aufteilen, sodass

$$(u \in V_1 \wedge v \in V_1) \vee (u \in V_2 \wedge v \in V_2) \Rightarrow (u, v) \notin E$$

gilt, es existieren also keine Kanten zwischen Knoten aus den gleichen Partitionen.
2. Es existiert eine lineare Ordnung der Nachbarschaften aller Knoten aus der gleichen Partition:

$$(u \in V_1 \wedge v \in V_1) \Rightarrow (\{(u, u') \mid (u, u') \in E\} \subseteq \{(v, v') \mid (v, v') \in E\})$$




# Quellen

[1] https://www.graphclasses.org/classes/gc_151.html
[2] https://dblp.org/rec/journals/dam/YanCC96.html
[3] https://dl.acm.org/doi/pdf/10.1145/363219.363232
[4] https://www.researchgate.net/publication/266939196_Determination_of_permutation_graphs
[5] https://dblp.org/rec/journals/symmetry/TalmaciuDSLI20
[6] https://dblp.org/rec/journals/algorithmica/NikolopoulosP07