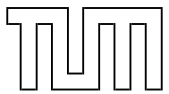
Fortgeschrittene Netzwerk- und Graph-Algorithmen

Prof. Dr. Hanjo Täubig

Lehrstuhl für Effiziente Algorithmen
(Prof. Dr. Ernst W. Mayr)
Institut für Informatik
Technische Universität München

Wintersemester 2010/11



Aufzählen von Cliquen

Wie kann man Cliquen aufzählen?

- Ausgabe ist oft exponentiell in der Eingabelänge

$\Rightarrow$ etwas anderer Effizienzbegriff

polynomielle Gesamtzeit:
bei Ausgabe von C Konfigurationen Begrenzung der Zeit durch ein Polynom in C und in der Eingabegröße n
(output-sensitiv)

- vollständige Suche ist *nicht* in polynomieller Gesamtzeit
- Aufzählung aller maximalen Cliquen geht in polynomieller Gesamtzeit (ein klassischer Algorithmus läuft z.B. erst $\mathcal{O}(n^2 C)$ Schritte ohne Ausgabe und gibt dann alle maximalen Cliquen auf einmal aus)
- Aufzählung aller Maximum-Cliquen geht nur in polynomieller Gesamtzeit, falls $\mathcal{P} = \mathcal{NP}$

'Effiziente' Aufzählungsalgorithmen

Andere Möglichkeit:

- polynomielle Verzögerung (polynomial delay):
  Zeit bis zur Ausgabe der ersten Konfiguration, zwischen zwei aufeinanderfolgnden Ausgaben von Konfigurationen und von der Ausgabe der letzten Konfiguration bis zum Stop ist polynomiell in der Eingabegröße

Aufzählung maximaler Cliquen

Satz

Es gibt einen Algorithmus, der alle maximalen Cliquen mit (polynomieller) Verzögerung $\mathcal{O}\left(n^3\right)$ und (linearem) Platzverbrauch $\mathcal{O}(m+n)$ aufzählt.

Algorithmus zur Aufzählung maximaler Cliquen

- Konstruiere Binärbaum mit n Leveln, dessen Blätter sich nur auf Level n befinden
- Jedes Level ist einem Knoten von G zugeordnet, auf Level i betrachtet man Knoten v_i.
- Knoten von Level i des Baums entsprechen den maximalen Cliquen des induzierten Teilgraphen $G[\{v_1, \ldots, v_i\}]$.

$\Rightarrow$ Die Blätter sind genau die maximalen Cliquen von G.

- Für ein gegebenes Level i und eine maximale Clique U in $G[\{v_1, \ldots, v_i\}]$ wollen wir die Kinder auf dem nächsten Level $i+1$ bestimmen:
  1. Alle Knoten von U sind adjazent zu v_{i+1} in G.
  2. Es existiert ein Knoten in U, der nicht zu v_{i+1} adjazent ist in G

Algorithmus zur Aufzählung maximaler Cliquen

Fallunterscheidung:

1. Alle Knoten von U sind adjazent zu v_{i+1} in G.
   $\Rightarrow$ $U \cup \{v_{i+1}\}$ ist maximale Clique in $G[\{v_1, \ldots, v_{i+1}\}]$
   - Das ist die einzige Möglichkeit, eine maximale Clique in $G[\{v_1, \ldots, v_{i+1}\}]$ zu bekommen, die U enthält
     In diesem Fall hat U nur ein einziges Kind im Baum.
2. Es existiert ein Knoten in U, der nicht zu v_{i+1} adjazent ist in G
   Man kann 2 maximale Cliquen in $G[\{v_1, \ldots, v_{i+1}\}]$ erhalten:
   1. U ist selbst eine maximale Clique
   2. $\left(U \setminus \bar{N}(v_{i+1})\right) \cup \{v_{i+1}\}$ ist eine Clique,
      wobei $\bar{N}(v_{i+1})$ alle nicht mit v_{i+1} adjazenten Knoten sind
      - ⋆ Wenn die Menge eine *maximale* Clique ist, hätte U zwei Kinder im Baum
      - ⋆ $\left(U \setminus \bar{N}(v_{i+1})\right) \cup \{v_{i+1}\}$ könnte aber Kind von mehreren sein

      $\Rightarrow$ Kind der lexikographisch kleinsten Menge U (falls maximal)

$\Rightarrow$ Binärbaum
(interne Knoten haben 1 oder 2 Kinder, Blätter nur in Level n)

Algorithmus zur Aufzählung maximaler Cliquen

- Traversiere den Binärbaum per Tiefensuche (DFS)
- Ausgabe aller Blätter
- Gegeben Knoten U auf Level i:
  - `Parent`(U,i):
    Vaterknoten von U ist die lexikographisch kleinste maximale Clique in $G[\{v_1,\ldots,v_{i-1}\}]$, die $U \setminus \{v_i\}$ enthält
    $\Rightarrow$ Grundfunktion, in $\mathcal{O}(m+n)$
  - `LeftChild`(U,i):
    - ★ falls $U \subseteq N(v_{i+1})$ (1. Fall), dann $U \cup \{v_{i+1}\}$
    - ★ falls $U \not\subseteq N(v_{i+1})$ (Teil des 2. Falls), dann U
    - ★ Unterscheidung der Fälle kostet $\mathcal{O}(m+n)$ Zeit
  - `RightChild`(U,i):
    - ★ falls $U \subseteq N(v_{i+1})$, dann existiert kein rechtes Kind
    - ★ falls $U \not\subseteq N(v_{i+1})$, dann
      $\left(U \setminus \bar{N}(v_{i+1})\right) \cup \{v_{i+1}\}$ falls es maximale Clique ist und
      $U = \texttt{Parent}\left(\left(U \setminus \bar{N}(v_{i+1})\right) \cup \{v_{i+1}\}, i+1\right)$
      sonst keins
    - ★ Kosten: $\mathcal{O}(m+n)$ Zeit

Algorithmus zur Aufzählung maximaler Cliquen

- Längster Pfad zwischen zwei Blättern im Baum ist $2n-2$ und geht durch $2n-1$ Knoten
- pro Knoten Zeitaufwand $\mathcal{O}(m+n)$
- Jeder Unterbaum hat ein Blatt auf Level n

$\Rightarrow$ Ausgabeverzögerung $\mathcal{O}(n^3)$

- Wenn ein Knoten bearbeitet wird, muss nur die Menge U, das Level i, sowie ein Label zur Unterscheidung von linkem/rechten Kind gespeichert werden

$\Rightarrow$ Speicheraufwand $\mathcal{O}(m+n)$

Aufzählung in lexikographischer Reihenfolge

- Es ist $\mathcal{NP}$-vollständig für einen Graphen G und eine maximale Clique U von G zu entscheiden, ob es eine maximale Clique U' gibt, die lexikographisch größer ist als U.
- Falls $\mathcal{P} \neq \mathcal{NP}$, dann gibt es keinen Algorithmus, der in Polynomialzeit zu einem Graphen G und einer maximalen Clique U von G die lexikographisch nächstgrößere maximale Clique generiert.
- Überraschenderweise kann man trotzdem alle maximalen Cliquen in lexikographischer Ordnung mit polynomieller Verzögerung aufzählen.

Aufzählung in lexikographischer Reihenfolge

- Idee: während der Generierung der aktuellen Ausgabe wird zusätzliche Arbeit in die Erzeugung lexikographisch größerer Cliquen investiert

$\Rightarrow$ Diese werden in einer Priority Queue gespeichert, die dann u.U. exponentiell viele Cliquen enthält und damit auch exponentiell viel Speicherplatz braucht.

Aufzählung in lexikographischer Reihenfolge

Algorithmus 7 : Alg. von Johnson, Papadimitriou und Yannakakis

$U_0 :=$ erste (lexikographisch kleinste) maximale Clique;
Füge U_0 in Priority Queue Q ein;
while Q *ist nicht leer* **do**
    $U :=$ExtractMin(Q);
    Ausgabe U;
    foreach *Knoten* v_j *von* G, *der zu einem Knoten* $v_i \in U$ *mit* $i < j$ *nicht adjazent ist* **do**
        $U_j := U \cap \{v_1, \ldots, v_j\}$;
        if $(U_j - \overline{N}(v_j)) \cup \{v_j\}$ *ist eine maximale Clique in* $G[\{v_1, \ldots, v_j\}]$ **then**
            Sei T die lexikographisch kleinste maximale Clique, die $(U_j - \overline{N}(v_j)) \cup \{v_j\}$ enthält;
            Füge T in Q ein

Aufzählung in lexikographischer Reihenfolge

Satz

Der Algorithmus von Johnson, Papadimitriou und Yannakakis zählt alle maximalen Cliquen eines Graphen mit n Knoten in lexikographischer Reihenfolge und mit Delay $\mathcal{O}(n^3)$ auf.

Aufzählung in lexikographischer Reihenfolge

Beweis.

Korrektheit:

- Menge T ist beim Einfügen in Q (bei Betrachtung von U) lexikographisch größer als U
  (denn es wird ja aus U zumindest Knoten v_i entnommen während lediglich v_j hinzukommt und es gilt $i < j$)

$\Rightarrow$ Es werden nur Mengen in der Queue gespeichert, die erst nach U ausgegeben werden dürfen.

$\Rightarrow$ Die ausgegebenen maximalen Cliquen sind lexikographisch aufsteigend.

Aufzählung in lexikographischer Reihenfolge

Beweis.

Vollständigkeit:

- Falls U die lexikographisch kleinste noch auszugebende Clique ist, dann ist U in Q.
- Induktionsanfang: Für $U = U_0$ ist das korrekt.
- Induktionsschritt: Sei U lexikographisch größer als U_0.
  - Sei j der größte Index, dass $U_j = U \cap \{v_1, \ldots, v_j\}$ *keine* maximale Clique in $G[\{v_1, \ldots, v_j\}]$.
    (Muss existieren, weil sonst $U = U_0$. Außerdem muss $j < n$ sein, weil U eine maximale Clique des Gesamtgraphen G ist.)
  - Aufgrund der Maximalität von j muss gelten $v_{j+1} \in U$.
  - $\exists$ Menge S: $U_j \cup S$ ist maximale Clique in $G[\{v_1, \ldots, v_j\}]$

Aufzählung in lexikographischer Reihenfolge

Beweis.

- Induktionsschritt (Fortsetzung):
  - Aufgrund der Maximalität von j ist v_{j+1} nicht adjazent zu allen Knoten in S.

  $\Rightarrow$ $\exists$ maximale Clique U', die zwar $U_j \cup S$, aber nicht v_{j+1} enthält
  - $U' \leq U$, weil sie sich in S unterscheiden
  - U' wurde schon ausgegeben (Ind.voraussetzung)
  - Als U' ausgegeben wurde, wurde festgestellt, dass v_{j+1} nicht adjazent ist zu einem Knoten $v_i \in U'$ mit $i < j+1$
  - $(U'_{j+1} \setminus \bar{N}(v_{j+1})) \cup \{v_{j+1}\} = U_{j+1}$
    und U_{j+1} ist maximale Clique in $G[\{v_1, \ldots, v_{j+1}\}]$

  $\Rightarrow$ Die lexikographisch kleinste maximale Clique, die U_{j+1} enthält, wurde in Q eingefügt.

Aufzählung in lexikographischer Reihenfolge

Beweis.

- Induktionsschritt (Fortsetzung):
  - Aufgrund der Maximalität von j stimmen U und T in den ersten $j+1$ Knoten überein.
  - Annahme: $U \neq T$
    Sei k der kleinste Index eines Knoten v_k, der in genau einer der beiden Mengen ist.
  - $k > j+1$
  - Da $T \leq U$, gilt $v_k \in T$ und $v_k \notin U$.

  $\Rightarrow$ U_k ist nicht maximale Clique in $G[\{v_1, \ldots, v_k\}]$
  (Widerspruch zur Maximalität von j)

  $\Rightarrow$ $U = T$

  $\Rightarrow$ U ist in der Priority Queue Q

Aufzählung in lexikographischer Reihenfolge

Komplexität:

- Extraktion der lexikographisch kleinsten maximalen Clique aus Q: $\mathcal{O}(n \log C)$
- n Berechnungen von maximalen Cliquen, die eine gegebene Menge enthalten: $\mathcal{O}(m+n)$ pro Menge
- Einfügen einer maximalen Clique in Q: $\mathcal{O}(n \log C)$ pro Clique
- Da $C \leq 3^{\lceil \frac{n}{3} \rceil}$, folgt dass die Verzögerung $\mathcal{O}(n^3)$ ist.

Plex

Relaxiere Cliquenbegriff, so dass konstant viele Verbindungen zu Gruppenmitgliedern bei jedem Knoten fehlen dürfen.

Definition

Sei $G = (V, E)$ ein ungerichteter Graph und sei $k \in \{1, \ldots, n-1\}$ eine natürliche Zahl.
Dann wird ein Subset $U \subseteq V$ als k-Plex bezeichnet, falls $\delta(G[U]) \geq |U| - k$.

k-Plex-Eigenschaften

- Jede Clique ist ein 1-Plex.
- Jedes k-Plex ist auch ein $(k+1)$-Plex.
- Ein *maximales k-Plex* ist in keinem größeren k-Plex echt enthalten.
- Ein *Maximum k-Plex* hat maximale Kardinalität unter allen k-Plexen in G.
- Jeder induzierte Teilgraph eines k-Plex ist auch ein k-Plex, d.h. die k-Plex-Eigenschaft ist abgeschlossen unter Exklusion.