Parameterisierte Algorithmen

SoSe 2013 in Trier

Henning Fernau
fernau@uni-trier.de

Parameterisierte Algorithmen | Gesamtübersicht

- Einführung
- Grundbegriffe
- Problemkerne
- Suchbäume
- Graphparameter
- Weitere Methoden
- Komplexitätstheorie—parameterisiert

Themen für letzte Woche und heute (und noch etwas nächste Woche):

- Win-Win
- iteratives Verbessern
- Farbkodierungen
- Wohl-Quasi-Ordnungen

Iteratives Verbessern: Der Maximierungsfall

Am ehesten entspricht dem wohl die Idee der so genannten Greedy Localization aus der Literatur:
Ausgehend von einer gefundenen (inklusions-)maximalen Lösung wird iterativ nach einer Verbesserung gesucht.
Wir betrachten im Folgenden ein Packungsproblem:
DREIERMENGENPACKEN

Eingabe: Ein System $\mathcal{M}$ von Dreiermengen über Grundmenge X
Parameter: eine natürliche Zahl k
Frage: Gibt es wenigstens k paarweise disjunkte Mengen in $\mathcal{M}$, also eine Dreiermengenpackung der Größe k ?

Dreiermengenpacken—kombinatorisch

Lemma 1 *Gibt es eine Dreiermengenpackung mit* $j+1$ *Mengen und kennen wir bereits eine Packung* $P = \{p_1, \ldots, p_j\}$ *mit* j *Mengen, so gibt es eine Dreiermengenpackung* $Q = \{q_1, \ldots, q_{j+1}\}$ *(genannt* $j+1$*-Erweiterung von* P*), sodass sich aus jeder Menge* p_i *mindestens zwei Elemente in* Q *wiederfinden.*

Beweis: Wir können davon ausgehen, dass wir eine solche Erweiterung Q von P betrachten, die möglichst viele Mengen aus P komplett übernimmt.
Wäre die Behauptung falsch, so gäbe es ein $p \in P$, aus dem sich höchstens ein Element in Q wiederfindet.
1. Findet sich gar kein Element in Q wieder, so können wir ein $q \in Q \setminus P$ nach dem Schubfachprinzip finden und durch p ersetzen. So erhalten wir ein Q' mit ebenfalls $j+1$ Mengen, das mehr Mengen aus P vollständig übernimmt. $\lightning$
2. Findet sich nur ein Element $x \in p$ in Q wieder, so gibt es genau eine Dreiermenge $q \in Q$ mit $x \in q$. $Q' = (Q \setminus \{q\}) \cup \{p\}$ führt wieder zu einer Erweiterung von P, die mehr Mengen als Q aus P komplett übernimmt. $\lightning$

Dreiermengenpacken—Eine erste Idee

Wir starten mit einer maximalen Dreiermengenpackung P (Greedy).
Enthält diese Packung k Mengen, so sind wir (positiv) fertig.
Im Allgemeinen wird sie j Mengen enthalten, $j < k$.
Aus jeder der j Mengen raten wir nun (Suchbaum !) zwei der drei Elemente, die in einer größeren Packung wiederverwendet werden sollen (Lemma !).
Das liefert uns die Elemente $a_1, \ldots, a_{2j}$.
Woher kommen aber die noch fehlenden $j + 3$ Elemente ?! Wir machen uns das Leben "einfach" und führen $j + 3$ Elementvariablen $x_1, \ldots, x_{j+3}$ ein.
Da P inklusionsmaximal, brauchen wir den Fall, dass eine der "neuen Mengen" nur durch Elementvariablen repräsentiert ist, nicht zu betrachten.
Wir betrachten also sämtliche Aufteilungen von $\{a_1, \ldots, a_{2j}, x_1, \ldots, x_{j+3}\}$ in Dreiermengen, sodass jede Dreiermenge mindestens ein Element von $a_1, \ldots, a_{2j}$ enthält.
Problem: Wie belegen wir die Variablen ?

Dreiermengenpacken

Problem: Wie belegen wir die Variablen ?

Eine erste Lösung ist wieder die Greedy-Methode:

Solange möglich, ersetzen wir die x_i durch "richtige" Elemente, um natürlich möglichst Mengen zu konstruieren, welche auch in unserem ursprünglichen Mengensystem liegen.

Sollten wir dabei feststellen, dass es solche konsistenten Belegungen gar nicht (mehr) gibt, so muss das daran liegen (jedenfalls im vorliegenden Suchast), dass eine der auf diese Weise und in dieser Phase gemachten gefräßigen Entscheidungen falsch war.

Es gibt aber nur höchstens $j + 2 \leq k + 1$ viele solcher potentiell falschen Entscheidungen. Diese werden nun durch einen Suchbaumschritt mit höchsten $k+1$ Gabeln aufgelöst.

Dadurch haben wir eine (weitere) der Elementvariablen entgültig festgelegt.

Dann wird diese Phase erneut gestartet.

Insgesamt führt dies auf eine Laufzeit der Form $(ck)^k$ für eine Konstante c.

Farbkodierungen

. . . bieten gegenüber “Greedy Localization” den (theoretischen) Vorteil, Algorithmen zu produzieren mit Laufzeit $\mathcal{O}^*(\mathrm{d}^{\mathrm{k}})$.

Allerdings ist die Konstante d meist recht groß.

Im Folgenden: Was sind Farbkodierungen ?

Abschließende Kommentare zu Greedy Localization:

Außerdem ist nicht klar, ob die Verzweigungsbreite des Suchbaums (so wie sie soeben mit k abgeschätzt wurde; meist ist sie linear in k bei dieser Technik) auch wirklich erreicht wird.

Man beachte, dass “frühe Kollisionen” zu evtl. deutlich kleineren Verzweigungsbreiten führen. Für die “wahre Laufzeitabschätzung” wären “Erwartungswerte” für den Kollisionsauftritt erforderlich.

Ein geeigenetes (“richtiges”) stochastisches Modell fehlt jedoch.

Homomorphieprobleme

Homomorphismen sind strukturerhaltende Abbildungen.
Diese heißen Monomorphismen, wenn sie injektiv sind.
Das Homomorphieproblem fragt bei Eingabe zweier Strukturen $\mathbb{A}$ und $\mathbb{B}$, ob es einen Homomorphismus von $\mathbb{A}$ in $\mathbb{B}$ gibt; eine natürliche Parameteriserung ist die Größe von $\mathbb{A}$.

Beispiel: $\mathbb{A}$ ist ein Weg der Länge k und $\mathbb{B}$ ein beliebiger Graph.
Das Homomorphieproblem fragt also, ob es einen Weg der Länge k in $\mathbb{B}$ gibt, wobei sich Knoten und Kanten wiederholen dürfen. Dieses Problem ist (und das ist typisch) polynomiell lösbar. (Wie ?)
Hingegen ist das entsprechende Monomorphieproblem *NP*-vollständig.
Dieses fragt also nach einem schlichten Weg der Länge k in $\mathbb{B}$ (also, ohne wiederholt auftretende Knoten oder Kanten).
Monomorphieprobleme heißen auch Einbettungsprobleme.

Ein Trick: farbenfrohe Strukturen

Bei diesen können das Homomorphie- und das Monomorphieproblem übereinstimmen.

Die Hoffnung ist: damit ist vielleicht auch das Monomorphieproblem “irgendwie leicht”.

Beispiel: $\mathbb{A}$ ist ein Weg der Länge k; jeder Knoten ist (injektiv) mit einer Farbe j gefärbt, $1 \leq j \leq k$.

$\mathbb{B}$ ist ein Graph; auch hier ist jeder Knoten mit einer Farbe j gefärbt, $1 \leq j \leq k$.

Eine strukturerhaltende Abbildung (im strengen Sinne) muss jetzt auch “farberhaltend” sein.

Daher ist jeder Homomorphismus auch ein Monomorphismus.

Achtung: Färbung meint hier lediglich eine Injektion der Knotenmenge in die Farbmenge; es ist nichts über “Nachbarfarben” ausgesagt.

Algorithm 1 Dynamisches Programmieren für farbenfrohe Wege

Input(s): a graph $G = (V, E)$, a coloring $c : V \to \{1, \ldots, k\}$, a color sequence $c_1 \cdots c_k$ with $\{1, \ldots, k\} = \{c_1, \ldots, c_k\}$.

Output(s): Is there a path $\pi_1, \ldots, \pi_k$ in G with $c(\pi_i) = c_i$?

for all $s \in V$ with $c(s) = c_1$ **do**
  for all $v \in V$ **do**
    $M_0[v] = 0$ iff $v \neq s$; $M_0[v] = 1$ iff $v = s$.
  end for
  for $i \leftarrow 1, \ldots, k-1$ **do**
    for all $v \in V$ **do**
      $M_i[v] = 0$
    end for
    for all $uv \in E$ **do**
      if $c(v) = c_{i+1}$ AND $M_{i-1}[u] = 1$ **then**
        $M_i[v] \leftarrow 1$
      end if
    end for
  end for
  if $M_{k-1}[v] = 1$ for some v **then**
    return YES
  end if
end for
return NO

Ein Trick: farbenfrohe Strukturen

Folgerung 2 *Das Wegeinbettungsproblem auf farbenfrohen Graphen ist in Polynomzeit lösbar.*

Woher nehmen wir aber die Färbungen, so wir diese nur als Hilfe für die Gewährleistung der Injektivität benötigen ?
Und würde uns das überhaupt etwas helfen ? Wir betrachten also zunächst:
FARBENFROHE WEGE

Eingabe: Graph $G = (V, E)$, (injektive) Färbung $c : V \to \{1, \ldots, k\}$
Parameter: natürliche Zahl k
Frage: Gibt es einen farbenfrohen Weg in (G, c), also einen Weg $p = v_1 \cdots v_k$ mit $\{c(v_i) \mid 1 \leq i \leq k\} = \{1, \ldots, k\}$?

Ein Trick: farbenfrohe Strukturen

Satz 3 *Das Problem* FARBENFROHE WEGE *liegt in FPT.*

Das kann man leicht mit Alg. 1 einsehen:
Man muss ja in einer Eingangsschleife nur alle Farbfolgen (Permutationen von $1, \ldots, k$) durchgehen.
Das führt also zu einem Algorithmus der Laufzeit $\mathcal{O}^*(k!) \approx \mathcal{O}^*(k^k)$.
Wir haben eingangs Laufzeiten der Bauart $\mathcal{O}^*(d^k)$ versprochen.
Außerdem ist ja noch unklar, woher die Färbungen von G eigentlich kommen sollen (Problem 1).
Dazu entscheidende Beobachtung: Entscheidend ist die "Farbenfrohheit", nicht so sehr die konkrete Einfärbung; daher genügt es, mit dynamischem Programmieren "farbenfrohe Wege" ausfindig zu machen (Problem 2).

Randomisierte parameterisierte Algorithmen

sind ein erster Schritt zur Lösung von Problem 1.
Wir werden anschließend zeigen, wie wir durch geeignete Derandomisierung den Zufall wieder herausbekommen.

Es sei (P, k) ein parametrisiertes Problem. Ein Monte-Carlo-FPT-Algorithmus für (P, k) ist ein randomisierter parameterisierter Algorithmus A (FPT bzgl. k), der für alle Eingaben (x, k) erfüllt:
Ist (x, k) eine JA-Instanz, so akzeptiert A mit Wahrscheinlichkeit von (mind.) $1/2$.
Ist (x, k) eine NEIN-Instanz, so akzeptiert A mit Wahrscheinlichkeit Null.

Zufallsfärbungen (im Folgenden spezialisiert auf Graphen)
Eine k-Zufallsfärbung der Knotenmenge V eines Graphen entsteht durch Zuweisung einer Farbe aus $\{1, \ldots, k\}$ zu jedem Knoten mit gleicher Wahrscheinlichkeit und unabhängig.
Eine Knotenmenge A heißt farbenfroh bzgl. einer Knotenfärbung c, wenn die Einschränkung von c auf A injektiv ist.

Lemma 4 *Ist c eine k-Zufallsfärbung der Knotenmenge V von G und ist A eine Knotenmenge mit k Elementen, so ist A mit Wahrscheinlichkeit von wenigstens e^{-k} farbenfroh bzgl. c.*

Beweis: Die Wahrscheinlichkeit, dass c eine feste Farbfolge $c_1 \cdots c_k$ realisiert, liegt bei k^{-k}. Wie schon beobachtet, ist dies nicht nötig; die Permutationen von $\{1, \ldots, k\}$ muss man wieder "rausrechnen". Die gefragte Wahrscheinlichkeit liegt also bei $\frac{k!}{k^k}$, was mit der Stirlingschen Abschätzung die Behauptung liefert:

$$\left(\frac{k}{e}\right)^k \cdot \sqrt{2\pi k} \leq k! \leq \left(\frac{k}{e}\right)^k \cdot \sqrt{2\pi k} \cdot e^{(1/12k)}$$

Algorithm 2 Dynamisches Programmieren für farbenfrohe Wege

Input(s): a graph $G = (V, E)$, a coloring $c : V \to \{1, \ldots, k\}$
Output(s): Is there a colorful path $\pi_1, \ldots, \pi_k$ in G ?

for all $s \in V$ {initial vertex} **do**
  for all $v \in V$ **do**
    $M_0[v] = \emptyset$ iff $v \neq s$; $M_0[v] = \{\{c(s)\}\}$ iff $v = s$.
  end for
  for $i \leftarrow 1, \ldots, k-1$ {extending paths} **do**
    for all $v \in V$ **do**
      $M_i[v] \leftarrow \emptyset$
    end for
    for all $uv \in E$ **do**
      for all $C_{u,i-1} \in M_{i-1}[u]$ **do**
        if $c(v) \notin C_{u,i-1}$ **then**
          $M_i[v] \leftarrow M_i[v] \cup \{(C_{u,i-1} \cup \{c(v)\})\}$
        end if
      end for
    end for
  end for
  if $M_{k-1}[v] \neq \emptyset$ for some v **then**
    return YES
  end if
end for
return NO

Ein randomisierter *FPT*-Algorithmus

Satz 5 *Für das Problem, einen schlichten Weg der Länge* k *in einem gegebenen Graphen* $G = (V, E)$ *zu finden, gibt es einen Monte-Carlo-FPT-Algorithmus mit Laufzeit* $\mathcal{O}^*((2e)^k))$.

Beweis: Nach obigem Lemma müssen wir e^k k-Zufallsfärbungen des Graphen erzeugen, um mit Wahrscheinlichkeit von wenigstens $1/2$ jede Menge A von k Knoten irgendwann einmal farbenfroh zu färben.
Die Wahrscheinlichkeit, dass A bei einer konkreten k-Zufallsfärbung nicht farbenfroh ist, liegt bei weniger als $(1 - e^{-k})$. Die Wahrscheinlichkeit, dass jene Menge A bei keiner von e^k k-Zufallsfärbungen nicht farbenfroh ist, liegt daher bei $(1 - e^{-k})^{e^k} \leq e^{-e^{-k} \cdot e^k} = e^{-1} < 1/2$.
Der angegebene Algorithmus zum dynamischen Programmieren wird schlimmstenfalls alle Farbteilmengen durchgehen, was (für jede k-Zufallsfärbung) eine Laufzeit von $\mathcal{O}^*(2^k)$ zeitigt.

Noch ein randomisierter *FPT*-Algorithmus

Satz 6 *Für das Problem, eine Dreiermengenpackung der Größe* k *in einem gegebenen Dreiermengensystem zu finden, gibt es einen Monte-Carlo-FPT-Algorithmus mit Laufzeit* $\mathcal{O}^*((24 \cdot e)^k)$.

Beweis: Wir müssen den Verbesserungsschritt ersetzen.
Die Grundidee ist die folgende, so eine maximale Packung P, $|P| = j$, vorliegt:
1. Wir eliminieren einen der drei Knoten jeder Menge $p \in P$.
2. Wir erhalten so $2j$ o.E. "gefärbte" Knoten in G (Lemma eingangs!).
3. Wir "würfeln" weitere $j + 3$ "neugefärbte" Knoten (und dies Würfeln geschieht genügend oft).
4. Wir gewinnen eine verbesserte Lösung (so sie existiert) durch dynamisches Programmieren auf den $3j + 3$ Farben.
Im schlimmsten Fall kostet Schritt 1-2 $\mathcal{O}^*(3^k)$ Zeit, Schritt 3 kostet $\mathcal{O}^*(e^k)$ Zeit (Monte-Carlo !) und Schritt 4 (s.u.) kostet $\mathcal{O}^*(2^{3k})$ Zeit.

Algorithm 3 Dynamisches Programmieren für farbenfrohe Dreiermengen

Input(s): a 3-set system $\mathcal{M}$ over X, a coloring $c : X \to \{1, \ldots, 3k\}$
Output(s): Is there a colorful 3-set packing $\pi_1, \ldots, \pi_k$ in $\mathcal{M}$?

for all $C \subset \{1, \ldots, 3k\}$ {initialization} **do**
  $M[C] \leftarrow 0$
end for
$M[\emptyset] = 1$
for all $p \in \mathcal{M}$ **do**
  if p is colorful, i.e., $|c(p)| = 3$ **then**
    for all $C \subset \{1, \ldots, 3k\}$ {extending packings} **do**
      if $M[C] = 1$ AND $c(p) \cap C = \emptyset$ **then**
        $M[C \cup c(p)] \leftarrow 1$
      end if
    end for
  end if
end for
return $M[\{1, \ldots, 3k\}] = 1$

Derandomisierung

Es seien M, N endliche Mengen und $k \in \mathbb{N}$. Eine Familie Γ von Abbildungen $g : M \to N$ heißt k-perfekte Familie von Hash-Funktionen gdw. für jede Teilmenge $K \subseteq M$ mit $|K| = k$ gibt es ein $c \in \Gamma$, welches K farbenfroh färbt.

Satz 7 *Es gibt eine k-perfekte Familie von Hash-Funktionen Γ mit $\Gamma \in \mathcal{O}^*(6.1^k)$.*

Dieses Ergebnis wurde auf der SODA 2006 vorgestellt. Eine “schlechtere” Familie findet sich im Buch von Flum/Grohe beschrieben (S. 346–354).
k-perfekte Familien von Hash-Funktionen können offenbar den Zufallsschritt in den angegebenen Algorithmen ersetzen.

Derandomisierung

Benutzen wir die vorgestellte Existenz k-perfekter Familien von Hash-Funktionen als Black Box, so können wir schlussfolgern:

Folgerung 8 *Für das Problem, einen schlichten Weg der Länge* k *in einem gegebenen Graphen* $G = (V, E)$ *zu finden, gibt es einen FPT-Algorithmus mit Laufzeit* $\mathcal{O}^*((2 \cdot 6.1)^k))$.

Folgerung 9 *Für das Problem, eine Dreiermengenpackung der Größe* k *in einem gegebenen Dreiermengensystem zu finden, gibt es einen FPT-Algorithmus mit Laufzeit* $\mathcal{O}^*((24 \cdot 6.1)^k))$.

Grapheigenschaften

Eine Grapheigenschaft Π heißt vererbbar gdw.
Ist $G \in \Pi$ und H ein induzierter Teilgraph von G, so gilt $H \in \Pi$.

Beispiele vererbbarer Eigenschaften:
Zweifärbbarkeit, Vollständigkeit, Planarität, Maximal-Durchmesser d, Unabhängigkeit, Kreisfreiheit

Ausschlussmengen

Eine Grapheigenschaft Π besitzt eine Kennzeichnung durch Ausschlussmengen (engl.: obstruction sets), wenn es eine Grapheigenschaft Ψ gibt, sodass $G \in \Pi$ gdw. für alle induzierten Teilgraphen H von G gilt: $H \notin \Psi$.

Lemma 10 *Eine Eigenschaft Π ist vererbbar gdw. Π besitzt Kennzeichnung durch Aussschlussmengen.*

Beweis: $\Leftarrow$: klar; $\Rightarrow$: Da Π vererbbar, Ausschlussmenge gegeben durch minimale Elemente in der Quasiordnung (s.u.) "ist induzierter Teilgraph von", welche nicht Π erfüllen.

Von besonderem Interesse: endliche Ausschlussmengen,
d.h., Ψ "enthält" nur endlich viele nicht-isomorphe Graphen.

Graphmodifikationsprobleme

Viele Graphprobleme lassen sich als Graphmodifikationsprobleme begreifen.
Das bedeutet für eine Grapheigenschaft Π z.B.:

Kantenlöschproblem Können wir k Kanten aus G löschen, um Graphen aus Π zu erhalten ?

Knotenlöschproblem Können wir k Knoten aus G löschen, um . . .

Kantenadditionsproblem Können wir k Kanten in G hinzufügen, um . . .

Selbstverständlich sind auch Kombinationen denkbar.
Beispiel für Knotenlöschproblem: Knotenüberdeckungsproblem mit Π: die kantenlosen Graphen.

Graphmodifikationsprobleme allgemein:

$\Pi_{i,j,k}$-Graphmodifikationsproblem:

Eingabe: Graph $G = (V, E)$

Parameter: $i, j, k \in \mathbb{N}$

Frage: Können wir bis zu i Knoten und j Kanten aus G entfernen und bis zu k Kanten hinzufügen, um einen Graphen aus Π zu erhalten ?

Satz 11 *(Leizhen Cai) Für (vererbbare) Grapheigenschaften Π mit endlicher Kennzeichnung durch Ausschlussmengen ist das $\Pi_{i,j,k}$-Graphmodifikationsproblem in FPT; genauere Laufzeitangabe: $\mathcal{O}(N^{i+2j+2k}|G|^{N+1})$, wobei N die Knotenzahl des größten Graphen in der Ausschlussmenge ist (also konstant).*

Lemma 12 *Es sei* Π *eine vererbbare Grapheigenschaft, die in* $T(|G|)$ *Schritten überprüft werden kann.*
Dann kann für jeden Graphen $G = (V, E) \notin \Pi$ *in* $\mathcal{O}(|V|T(|G|))$ *vielen Schritten ein minimaler verbotener induzierter Teilgraph* H *gefunden werden.*

"Minimal" ist so zu verstehen, dass H selbst keinen verbotenen induzierten echten Teilgraphen besitzt.
Der Beweis ergibt sich durch eine einfache Greedy-Strategie: ausgehend von G werden sukzessive Knoten gelöscht und die Grapheigenschaft getestet (s.u.).
Es bliebe zu beweisen, dass diese Strategie wirklich einen minimalen verbotenen induzierten Teilgraphen findet.

Der Beweis des Lemmas:

Algorithm 4 How to find forbidden subgraphs.

Input(s): a graph $G = (V, E)$, an algorithm A that answers $H \in \Pi$ in time $T(|H|)$.
Output(s): A minimal forbidden induced subgraph $F \subseteq V$, if it exists; $\emptyset$ otherwise.

$F \leftarrow \emptyset$; $W \leftarrow V$;
while $W \neq \emptyset$ **do**
  Choose $v \in W$; $W \leftarrow W \setminus \{v\}$.
  if $A(G - v)$ **then**
    $F \leftarrow F \cup \{v\}$
  else
    $G \leftarrow G - v$
  end if
end while
return F

Der Beweis des Satzes:

Algorithm 5 A schematics for parameterized graph modification algorithms.

Input(s): a graph G, together with parameters i, j, k
Output(s): $G \in \Pi_{i,j,k}$?

for all minimal forbidden induced subgraphs H of G **do**
  while $i + j + k > 0$ **do**
    Modify G by deleting a vertex or an edge from H OR by adding an edge to H; change the parameter accordingly.
    if $G \in \Pi$ **then**
      return YES
    end if
  end while
end for
return NO

Der Beweis des Satzes:

Zur Korrektheit:

Grundidee: Die verbotenen Teilgraphen müssen “zerstört” werden.

Zur Komplexität:

Offenbar gilt $T(|G|) = \mathcal{O}(|V^N|)$ durch Testen aller $\leq N$-Knoten-Mengen von G.

Nach Lemma kostet das Finden von H daher Zeit $\mathcal{O}(|V|^{N+1})$.

Es gibt nur höchstens N Möglichkeiten, einen Knoten aus H zu löschen.

Es gibt nur je $\binom{N}{2}$ Möglichkeiten, eine Kante aus H zu löschen oder zu H hinzuzufügen.

$\rightsquigarrow$ behauptete Gesamtlaufzeit

Bipartisierung

ist offenbar ein Graphmodifikationsproblem.

Die Eigenschaft Π “bipartit” (oder “paar” oder “zweifärbbar”) ist offenbar vererbbar.

Eine endliche Ausschlussmenge ist jedoch unbekannt.

Die “natürliche Ausschlussmenge” sind alle Kreise ungerader Länge; das ist aber eine unendliche Ausschlussmenge.

Noch ein Beispiel: Warum *FPT*-Mitgliedschaft ?

Zur Erinnerung: (siehe VL 7)
Eine Clique ist ein vollständiger Teilgraph.
Ein Cluster ist eine Clique, die eine Zusammenhangskomponente ist.
Bei einem Clustergraph bildet jede Zusammenhangskomponente eine Clique.

CLUSTERKNOTENPROBLEM (CLUSTER VERTEX DELETION CVD)
Eingabe: Ein Graph $G = (V, E)$
Parameter: $k \in \mathbb{N}$
Frage: Gibt es eine Knotenmenge $D \subseteq V$, sodass $|D| \leq k$ und $G - D$ ist ein Clustergraph ?

Lemma 13 *Enthält* G *einen induzierten* P_2*, so ist* G *kein Clustergraph (und umgekehrt).*

Satz 14 CVD *liegt in FPT.*

Fragen: Wie sieht ein “Ausschlussmengenargument” aus ?
Welches allgemeinere Problem liegt also auch in *FPT* ?
Können Sie für CVD auch einen Algorithmus finden, der auf Iterativem Komprimieren beruht?

Nachweise

“Greedy Localization” wird im Buch von Niedermeier ausführlich behandelt.

Eine gute (und umfassende) Darstellung zu Farbkodierungen findet man im Buch von Flum und Grohe.

Bei der Darstellung des Dreiermengenpackungsproblems folgen wir eher den Originalarbeiten von J. Chen und seinen Koautoren, teilweise mit eigenen Beweisen.

Die beste Darstellung von Graphmodifikationsproblemen findet sich m.E. in der Originalarbeit von Leizhen Cai: Fixed-parameter tractability of graph modification problems for hereditary properties. *Information Processing Letters*, 38:171–176, 1996.

Zu CVD mehr in: Fixed-Parameter Algorithms for Cluster Vertex Deletion. LATIN 2008.