Institut für Theoretische Informatik
Lehrstuhl Prof. Dr. D. Wagner

Ferien-Übungsblatt 8 – Lösungsvorschläge

Vorlesung Algorithmentechnik im WS 09/10

Problem 1: Probabilistische Komplexitätsklassen [vgl. Kapitel 8 im Skript] **

Studieren Sie die Komplexitätsklassen aus Kapitel 8 im Skript.

(a) Zeigen Sie: Für jedes Polynom $q(n) \geq 1$ kann man in der Definition von $\mathcal{RP}$ anstelle von $1/2$ auch $1 - 2^{-q(n)}$ einsetzen, also Fehlerwahrscheinlichkeit $2^{-q(n)}$ statt $1/2$ fordern, ohne an der Klasse was zu ändern.

Lösung. Sei zu einem Problem Π aus $\mathcal{RP}$ mit $\mathcal{A}$ ein $\mathcal{RP}$-Algorithmus bezeichnet, der Π löst. Um die gewünschte Fehlerwahrscheinlichkeit zu erreichen, konstruieren wir einen "Meta"-Algorithmus $\mathcal{A}'$, der $\mathcal{A}$ wie folgt aufruft.

Sei $\mathcal{I}$ mit $|\mathcal{I}| = n$ eine beliebige Eingabe von $\mathcal{A}$. Dann verwaltet $\mathcal{A}'$ einen Zähler $\mathfrak{z}$ der mit $q(n)$ initialisiert wird. In jeder Iteration von Algorithmus $\mathcal{A}'$ wird $\mathfrak{z}$ um 1 dekrementiert und der Algorithmus $\mathcal{A}$ mit $\mathcal{I}$ aufgerufen. Ist die Ausgabe von $\mathcal{A}(\mathcal{I})$ "Ja", brechen wir die Schleife ab und geben "Ja" aus. Andernfalls terminieren wir nach $q(n)$ Iterationen und geben "Nein" aus. Siehe dazu auch Algorithmus 1.

Algorithmus 1 : Meta-Algorithmus zur Verringerung der Fehlerwahrscheinlichkeit auf $2^{-q(n)}$

Eingabe : $\mathcal{RP}$-Algorithms $\mathcal{A}$, Instanz $\mathcal{I}$

```
1 Für 𝔷 ← q(n) ... 1
2     result ← A(I)
3     Wenn result = "Ja"
4         return "Ja"
5 return "Nein"
```

Ist nun $\mathcal{I} \in \Pi$, so gilt, dass $\mathcal{A}'$ genau dann eine falsche Antwort (nämlich "Nein") liefert, wenn in allen $q(n)$ Iterationen von $\mathcal{A}'$ der Algorithmus $\mathcal{A}$ für $\mathcal{I}$ ebenfalls die falsche Antwort liefern würde. In jeder Iteration liefert $\mathcal{A}(\mathcal{I})$ mit Wahrscheinlichkeit $< 1/2$ die falsche Antwort, für die $q(n)$ unabhängigen Aufrufe von $\mathcal{A}(\mathcal{I})$ ist die Wahrscheinlichkeit dass jedes Mal die falsche Antwort geliefert wird also $< (1/2)^{q(n)} = 2^{-q(n)}$. Das heißt

$$\mathcal{I} \in \Pi \longrightarrow \Pr[\mathcal{A}'(\mathcal{I}) \text{ ist "Ja"}] \geq 1 - 2^{-q(n)}.$$

Ist hingegen $\mathcal{I} \notin \Pi$, so liefert $\mathcal{A}(\mathcal{I})$ in jeder Iteration die korrekte Antwort – nämlich "Nein", und somit liefert auch $\mathcal{A}'(\mathcal{I})$ die korrete Antwort, das heißt

$$\mathcal{I} \notin \Pi \longrightarrow \Pr[\mathcal{A}'(\mathcal{I}) \text{ ist "Nein"}] = 0.$$

Da $\mathcal{A}(\mathcal{I})$ Laufzeit $\mathcal{O}(p(n))$ hat, und dieser $\mathcal{O}(q(n))$ mal aufgerufen wird, ist die Gesamtlaufzeit von $\mathcal{A}'(\mathcal{I})$ in $\mathcal{O}(p(n)q(n))$, also ebenfalls polyonmiell. □

Die Klasse co-$\mathcal{RP}$ enthält die Entscheidungsprobleme Π für die es einen polynomiellen, randomisierten Algorithmus $\mathcal{A}$ gibt, so dass für alle Instanzen $\mathcal{I}$ von Π gilt:

$$\begin{cases} \mathcal{I} \in Y_\Pi \longrightarrow \Pr[\mathcal{A}(\mathcal{I}) \text{ ist “Nein”}] = 0 \\ \mathcal{I} \notin Y_\Pi \longrightarrow \Pr[\mathcal{A}(\mathcal{I}) \text{ ist “Nein”}] \geq 1/2 \end{cases}$$

Die Klasse $\mathcal{ZPP}$ (zero-error probabilistic polyn.-time) ist definiert durch $\mathcal{ZPP} := \mathcal{RP} \cap co-\mathcal{RP}$.

(b) Welche Art von Algorithmen sind in $\mathcal{ZPP}$ enthalten? Begründen Sie Ihre Antwort.

Lösung. Die Klasse $\mathcal{ZPP}$ enthält genau die Las-Vegas Algorithmen mit erwartet polynomieller Laufzeit. Das heißt, Algorithmen aus $\mathcal{ZPP}$ liefern stets das korrekte Ergebnis (mit Fehlerwahrscheinlichkeit 0). Dies lässt sich dadurch einsehen dass für ein Problem $\Pi \in \mathcal{RP} \cap co-\mathcal{RP}$ sowohl ein Algorithmus existiert, der für jede Nein-Instanz Π das korrekte Ergebnis liefert, alsauch ein Algorithmus der für jede Ja-Instanz von Π das korrekte Ergebnis liefert. Ein zusammengesetzter Algorithmus der *immer* das korrekte Ergebnis liefert, ist in der nächsten Aufgabe vorgestellt. □

(c) Zu einem Problem Π sei $\mathcal{A}$ ein $\mathcal{RP}$-Algorithmus und $\overline{\mathcal{A}}$ ein co-$\mathcal{RP}$-Algorithmus wobei $\mathcal{A}$ und $\overline{\mathcal{A}}$ jeweils Laufzeit $p(n)$ haben. Geben Sie für Π einen $\mathcal{ZPP}$-Algorithmus $\mathcal{A}'$ an, der $\mathcal{A}$ und $\overline{\mathcal{A}}$ verwendet. Geben Sie weiterhin die erwartete Laufzeit von $\mathcal{A}'$ an.

Hinweis: Es gilt $2 - \sum_{i=0}^{k} i2^{-i} = (k+2)2^{-k}$.

Lösung. Die Idee für einen $\mathcal{ZPP}$-Algorithmus $\mathcal{A}'$ ist wie folgt. Wir machen uns zu Nutze, dass $\mathcal{A}$ und $\overline{\mathcal{A}}$ beides Algorithmen zur Lösung von Π sind, wobei $\mathcal{A}$ ein $\mathcal{RP}$-Algorithmus, und $\overline{\mathcal{A}}$ ein co-$\mathcal{RP}$-Algorithmus ist, das heißt, die beiden Algorithmen haben jeweils gegensätzlichen *einseitigen Fehler*, liefern also zu einer Instanz $\mathcal{I}$ für einen der Fälle $\mathcal{I} \in \Pi$ oder $\mathcal{I} \notin \Pi$ immer die korrekte Antwort. Wir führen nun in einer “Endlosschleife” unsere Instanz $\mathcal{I}$ sowohl auf $\mathcal{A}$ alsauch auf $\overline{\mathcal{A}}$ aus, und zwar bis die Ergebnisse der beiden Algorithmen übereinstimmen. Algorithmus 2 zeigt unseren Ansatz.

Algorithmus 2 : $\mathcal{ZPP}$-Algorithmus $\mathcal{A}'$

Eingabe : Instanz $\mathcal{I}$ für Π, $\mathcal{RP}$-Algorithmus $\mathcal{A}$ und co-$\mathcal{RP}$-Algorithmus $\overline{\mathcal{A}}$
Ausgabe : “Ja” falls $\mathcal{I} \in \Pi$, “Nein” falls $\mathcal{I} \notin \Pi$

```
wiederhole
    r1 ← A(I)
    r2 ← Ā(I)
bis r = r'
r ← r1(= r2)
return r
```

Dass r immer die korrekte Antwort enthält, lässt sich relativ leicht einsehen. Wir beweisen folgende zwei Fälle.

1. $\mathcal{I} \in \Pi$: In diesem Fall ist $\Pr[r_1 =$ “Ja”$] \geq 1/2$ und $\Pr[r_2 =$ “Nein”$] = 0$, also $\Pr[r_2 =$ “Ja”$] = 1$. Es ist also stets $r_2 =$ “Ja“; $\overline{\mathcal{A}}$ liefert also immer die korrekte Antwort. Liefert zudem $\mathcal{A}$ die korrekte Antwort, also $r_1 =$ ”Ja“, dann ist $r_1 = r_2$ und wir geben $r =$ ”Ja“ zurück.
2. $\mathcal{I} \notin \Pi$: In diesem Fall ist $\Pr[r_1 =$ ”Ja“$] = 0$, also $\Pr[r_1 =$ ”Nein“$] = 1$. Das heißt, $\mathcal{A}$ gibt immer die korrekte Antwort, und es ist $r_1 =$ ”Nein“. Für $\overline{\mathcal{A}}$ gilt $\Pr[r_2 =$ ”Nein“$] \geq 1/2$. Die Schleife terminiert also nur wenn $r_1 = r_2 =$ ”Nein“, und wir geben $r =$ ”Nein“ zurück.

Sei nun $p(n)$ die (maximale) Laufzeit von $\mathcal{A}(\mathcal{I})$ bzw. $\overline{\mathcal{A}}(\mathcal{I})$. Dann hat jeder Schleifendurchlauf trivialerweise Laufzeit $2p(n)$. Zudem ist die Fehlerwahrscheinlichkeit für jeden Schleifendurch-

lauf unabhängig, das heißt die erwartete Laufzeit setzt sich zusammen durch

$$\begin{aligned} E[t_{\mathcal{A}'}(\mathcal{I})] &\leq \frac{1}{2}2p(n) + \frac{1}{4}4p(n) + \frac{1}{8}6p(n) + \dots \\ &= 2p(n) \underbrace{\sum_{i=1}^{\infty} 2^{-i} i}_{=:\ \Psi}. \end{aligned}$$

Wegen $2 - \sum_{i=0}^{k} 2^{-i} i = (k+2)2^{-k}$ folgt

$$\begin{aligned} 2 - \Psi &= \lim_{k\to\infty} 2 - \sum_{i=0}^{k} 2^{-i} i \\ &= \lim_{k\to\infty} \frac{k+2}{2^k} \\ &= 0 \end{aligned}$$

womit $\Psi = 2$ folgt. Insgesamt ergibt sich also eine erwartete Laufzeit von $E[t_{\mathcal{A}'}] = 4p(n)$.

□

Problem 2: Vergleich von Wörtern [vgl. Kapitel 8 im Skript] **

In dieser Aufgabe widmen wir uns folgender Problemstellung. Gegeben seien zwei Bitfolgen $a_1 \cdots a_n$ und $b_1 \cdots b_n$, von denen wir uns vorstellen, dass sie an verschiedenen Orten vorliegen. Möchte man nun überprüfen, ob die Bitfolgen identisch sind, so lässt sich das natürlich bewerkstelligen, indem man alle n Bits von einem Ort zum anderen überträgt. Der Monte-Carlo-Algorithmus 3 benötigt dafür jedoch bloß $\Theta(\log n)$ Bits. Eine Bitfolge $a_1 \cdots a_n$ kann auf natürliche Weise als eine Zahl $a := \sum_{i=1}^{n} a_i 2^{i-1}$ interpretiert werden.

Algorithmus 3 : Überprüfe ob Bitfolgen gleich sind

Eingabe : Bitfolgen $a_1 \cdots a_n$ und $b_1 \cdots b_n$
Ausgabe : Entscheidung ob Bitfolgen gleich sind

```
1 p ← ⟨Primzahl, zufällig gleichverteilt aus denen kleiner oder gleich n² log n² gewählt⟩
2 a ← Σ_{i=1}^{n} a_i 2^{i-1}
3 b ← Σ_{i=1}^{n} b_i 2^{i-1}
4 Wenn a mod p ≡ b mod p
5 |  return ja
6 sonst
7 |  return nein
```

(a) Beschreiben Sie, wie Sie Algorithmus 3 dazu benutzen können um über die Distanz zu überprüfen ob $a_1 \cdots a_n = b_1 \cdots b_n$ ist. Was müssen Sie dazu übertragen? Zeigen Sie, dass die Anzahl übertragener Bits in $\mathcal{O}(\log n)$ liegt.

Lösung. Angenommen die Bits $a_1 \cdots a_n$ liegen an einem Ort A und die Bits $b_1 \cdots b_n$ an einem Ort B. Um nun zu überprüfen ob $a_1 \cdots a_n = b_1 \cdots b_n$ gilt, wählt o. B. d. A. A die Primzahl $p \in [2, n^2 \log n^2]$. Dann berechnen wir in A den Wert $a \mod p =: z$ und übertragen das Tupel (z, p) zu B. Bei B wird nun $b \mod p =: z'$ berechnet und überprüft ob $z = z'$ gilt.

Was die Anzahl übertragener Bits betrifft, so gilt für die Primzahl p dass $p \leq n^2 \log n^2$. Dafür sind $\log(n^2 \log n^2)$ Bits notwendig. Es gilt

$$\log(n^2 \log n^2) = 2\log(n \log n^2) = 2\log n + 2\log\log n^2 \in \mathcal{O}(\log n).$$

Somit genügen für p bereits $\mathcal{O}(\log n)$ Bits. Da für z wegen $z = a \mod p$ gilt dass $z \leq p$, folgt somit dass für z ebenfalls $\mathcal{O}(\log n)$ Bits ausreichen.

□

(b) Zeigen Sie, dass die Wahrscheinlichkeit, dass Algorithmus 3 ein falsches Ergebnis liefert in $\mathcal{O}(1/n)$ liegt. Benutzen Sie dafür die folgenden beiden Resultate:

Satz 2.1 (Chebyshew)**.** *Für die Anzahl* $\mathfrak{p}(n)$ *der Primzahlen kleiner oder gleich* n *gilt*

$$\frac{7}{8}\frac{n}{\ln n} \leq \mathfrak{p}(n) \leq \frac{9}{8}\frac{n}{\ln n}$$

Es ist also $\mathfrak{p}(n) \in \Theta(n/\ln n)$.

Satz 2.2. *Die Anzahl* k *verschiedener Primteiler einer Zahl* $m \leq 2^n$ *ist höchstens* n*.*

Lösung. Für eine Primzahl p bezeichne $F_p : \mathbb{Z} \to \mathbb{Z}_p$ die Abbildung, die $x \mapsto x \mod p$ berechnet. Dann überprüft Algorithmus 3 ob $F_p(a) = F_p(b)$ gilt. Eine falsche Antwort wird von Algorithmus 3 genau dann geliefert, wenn $F_p(a) = F_p(b)$ aber $a \neq b$ gilt. Dies ist der Fall wenn p ein Teiler von $c := |b - a|$ ist, wobei $c \neq 0$.

Offensichtlich gilt $a, b \leq 2^n$, und somit ist auch $c \leq 2^n$, damit gilt dass wegen Satz 2.2 die Zahl c höchstens n Primteiler hat. Sei nun $t \in \mathbb{N}$ die obere Schranke, so dass Algorithmus 3 genau eine der Primzahlen im Bereich $[2, t]$ gleichverteilt wählt. Nach Satz 2.1 gibt es in diesem Intervall $\mathfrak{p}(t) \in \Theta(t \log t)$ Primzahlen. Für die Wahrscheinlichkeit ein p so zu wählen, dass $F_p(a) = F_p(b)$ aber $a \neq b$, also $\Pr[F_p(a) = F_p(b) \mid a \neq b]$ ist demnach mit der Methode "Anzahl (un)günstige Fälle durch Anzahl mögliche Fälle“ in

$$\mathcal{O}\Big(\frac{n}{t/\log t}\Big).$$

Wählt man also wie im Algorithmus $t := n^2 \log n^2$ so ist die Fehlerwahrscheinlichkeit

$$\begin{aligned}
\mathcal{O}\Big(\frac{n}{n^2 \log n^2 / \log(n^2 \log n^2)}\Big) &= \mathcal{O}\Big(\frac{n \log(n^2 \log n^2)}{n^2 \log n^2}\Big) \\
&= \mathcal{O}\Big(\frac{\log n + \log\log n}{n \log n}\Big) \\
&= \mathcal{O}\Big(\frac{\log n + \log n}{n \log n}\Big) \\
&= \mathcal{O}\Big(\frac{1}{n}\Big)
\end{aligned}$$

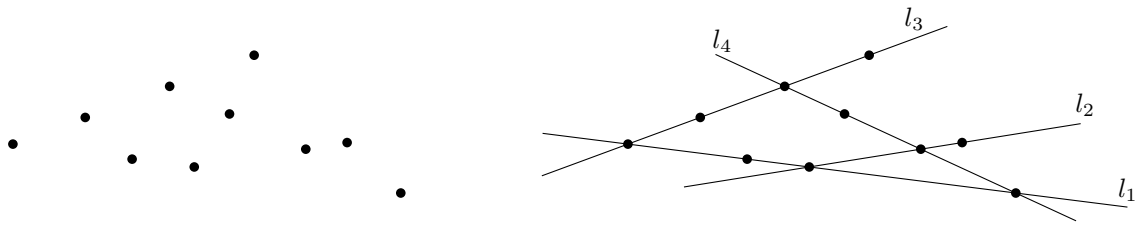
Damit ist die Behauptung bewiesen.

□

Problem 3: Line-Shooting Problem [vgl. Kapitel 10 im Skript] ***

Gegeben sind n Punkte $P \subset \mathbb{R}^2$ ($|P| < \infty$) in der Ebene mit $p_i := (x_i, y_i)$ für alle $p_i \in P$ sowie eine Zahl $k \in \mathbb{N}$. Gefragt ist nun ob es eine Menge L von k Geraden gibt so, dass die Geraden $l_j \in L$ alle Punkte $p_i \in P$ überdecken – das heißt für jeden Punkt $p_i \in P$ eine Gerade $l_j \in L$ existiert so, dass $p_i \in l_j$. Eine Instanz des Line-Shooting Problems ist dann durch $\mathcal{I} := (P, k)$ gegeben.

(a) Beweisen Sie, dass das Line-Shooting Problem FPT bezüglich k ist. Geben Sie dazu einen Algorithmus mit beschränkter Suchbaumtiefe an, der eine Instanz (P, k) exakt löst. Dabei soll



(a) Ja-Instanz $\mathcal{I} = (P, 4)$. **(b)** Lösung von $\mathcal{I}$ mit 4 Geraden.

Abbildung 1: Beispiel für eine Instanz $\mathcal{I} = (P, k)$ des Line-Shooting Problems mit gegebener Punktemenge P und $k = 4$. Eine zulässige Lösung ist in Abbildung (b) dargestellt.

die Suchbaumtiefe und der Verzweigungsgrad polynomiell in k, und der Aufwand pro Knoten polynomiell in $|P| =: n$ sein. Analysieren Sie die Laufzeit Ihres Algorithmus im $\mathcal{O}$-Kalkül.

Hinweis: Verwalten Sie in jedem Baumknoten k Bins, die maximal zwei Punkte aufnehmen können (und damit ggf. eine Gerade definieren). Ein Suchbaum der Höhe $\mathcal{O}(k)$ genügt.

Lösung. Die Idee für einen exakten Such-Algorithmus mit beschränkter Suchbaumtiefe ist wie folgt. Jeder Knoten v in dem Suchbaum besteht aus k Bins die maximal zwei Punkte $p_i \in P$ aufnehmen können. Wir nennen einen Bin b *offen*, wenn $|b| < 2$, ansonsten heißt b *voll*. Jeder volle Bin definiert eindeutig eine Gerade l die durch dessen Punkte induziert wird. Dabei kann l durchaus mehr als zwei Punkte in P überdecken.

Der Suchbaum wird nun wie folgt konstruiert: Zunächst ordnen wir die Punktemenge P in einer beliebigen Reihenfolge, das heißt es ist $P = \{p_1, \ldots, p_n\}$. Dann wird die Wurzel des Baumes mit k leeren Bins initialisiert. Betrachten wir nun einen beliebigen Knoten v auf Level i des Baumes. Die zugehörigen Bins von v seien mit $b_1(v), \ldots, b_k(v)$ bezeichnet. Für jeden offenen Bin $b_j(v)$ erzeugen wir einen Nachfolgeknoten w_j wie folgt. Die Bins von w_j werden zunächst von v übernommen, das heißt es gilt $b_\nu(w_j) := b_\nu(v)$ für alle $1 \leq \nu \leq k$. In den j-ten Bin von w_j wird allerdings noch der Punkt p_{i+1} hinzugefügt, also $b_j(w_j) := b_j(v) \cup \{p_{i+1}\}$. In Worten heißt das, wir untersuchen alle Möglichkeiten den "nächsten" Punkt p_{i+1} auf die Bins zu verteilen (dies sind maximal k Stück, da es maximal k offene Bins geben kann). Damit ist der Verzweigungsgrad für jeden Knoten höchstens k – insbesondere also polynomiell in k.

Wird beim erzeugen eines Knotens im Baum ein Bin $b_j(v)$ voll, so werden alle durch die $b_j(v)$ induzierte Gerade l überdeckten Punkte "gelöscht" – das heißt diese werden nicht länger für die Nachfolgeknoten in Betracht gezogen. Dafür ist die für jeden vollen Bin überdeckte Menge von Punkten zu ermitteln. Dies geht in Zeit $\mathcal{O}(kn)$. Für die Blätter des Baumes muss schließlich überprüft werden, ob die durch die Bins induzierten Geraden *alle* (verbliebenen) Punkte überdecken. Dies ist ebenfalls in $\mathcal{O}(kn)$ berechenbar. Die Höhe des Suchbaums ist höchstens $2k$, da wir k Bins der Größe 2 haben, und für jeden Nachfolgeknoten genau in einen der Bins einen Punkt hinzufügen. Da der Verzweigungsgrad des Baumes höchstens k ist, hat der Baum insgesamt $\mathcal{O}(k^{2k})$ Knoten. Das heißt unser Algorithmus ist ein Algorithmus mit Suchbaumtiefe polynomiell in k, und die Gesamtlaufzeit beträgt $\mathcal{O}(k^{2k}kn) = \mathcal{O}(k^{2k+1}n)$. □

(b) Argumentieren Sie, warum bezüglich der Punktemenge P aus Abbildung 1a die Instanz $\mathcal{I} = (P, 3)$ eine Nein-Instanz des Problems ist.

Lösung. Betrachten wir Abbildung 1b. Die drei Geraden l_1, l_3 und l_4 decken jeweils $4 > 3$ Punkte ab, damit müssen sie zwangsweise in einer Lösung von $\mathcal{I} = (P, 3)$ enthalten sein. Wäre beseispielsweise l_1 nicht Teil der Lösung, so bräuchte man 4 Geraden, um die durch l_1 überdeckten Punkte abzudecken, was natürlich schon zu viel wäre. Da wir nun wissen dass l_1, l_3 und l_4 Teil der Lösung sein müssen, bleibt uns keine Gerade mehr übrig, um den rechtesten durch l_2 überdeckten Punkt zu überdecken. Das heißt, die Punktemenge P lässt sich nicht mit 3 Geraden überdecken. □

(c) Gegeben sei eine Instanz $\mathcal{I} = (P, k)$ des Line-Shooting Problems mit $n = |P|$. Geben Sie ein

Algorithmus zur Kernbildung von $\mathcal{I}$ an, der die Instanz $\mathcal{I}$ auf eine Instanz $\mathcal{I}'$ reduziert, so dass $|\mathcal{I}'|$ polynomiell in k ist. Beweisen Sie die Korrektheit der Transformation und geben Sie die Gesamtlaufzeit zur Lösung von $\mathcal{I}$ mit Ihrem Verfahren an. Ist Ihr Verfahren effizienter, als das aus Aufgabe (a)?

Lösung. Sei $\mathcal{I} := (P, k)$ eine Instanz des Line-Shooting Problems mit $n = |P|$. Für die Kernbildung betrachten wir die Menge L aller Geraden die durch alle Punktepaare $(p_i, p_j) \in P^2$ induziert werden. Dies sind höchstens n^2, also $|L| \in \mathcal{O}(n^2)$. Achtung: Für jede Menge an kollinearen Punkten gibt es nur eine Gerade in L, da L als Menge definiert ist.

Jede Gerade $l \in L$ die mindestens $k+1$ Punkte überdeckt, muss zwangsweise in einer Lösung von $\mathcal{I}$ enthalten sein. Wäre nämlich l nicht Teil der Lösung von $\mathcal{I}$, so bräuchten wir $k+1$ Geraden um alle Punkte von l abzudecken. Dies sind jedoch bereits mehr als k; Die Gerade l muss also Teil der Lösung sein.

Betrachten wir nun die Menge $F \subseteq L$ derjenigen Geraden, die mindestens $k+1$ Punkte überdecken. Ist $|F| > k$, so lässt sich $\mathcal{I}$ nicht lösen. Andernfalls, löschen wir in P alle Punkte, die durch die Geraden in F überdeckt werden. Die resultierende Punktemenge sei mit P' bezeichnet. Weiterhin setzen wir $k' := k - |F|$, da wir für die Punkte in P' nurnoch $k - |F|$ Geraden zur Verfügung haben.

Wir erhalten so eine neue Instanz $\mathcal{I}' := (P', k')$. Jede Gerade in $L \setminus F$ überdeckt höchstens k Punkte in P', andernfalls wäre sie ja in F. Ist also $|P'| > kk'$, so kann die Menge P' nicht durch k' Geraden abgedeckt werden (wir haben ja schließlich nurnoch k' Geraden zur Verfügung, von denen jede allerdings nur maximal k Punkte überdeckt), und damit kann auch P nicht mit k Geraden überdeckt werden.

Für den Fall, dass $|P'| \leq kk' \leq k^2$, können wir mit dem Algorithmus aus Aufgabe (a) in Zeit $\mathcal{O}(k^{2k+2})$ überprüfen, ob P' mit k' Geraden überdeckt werden kann. Die Kernbildung lässt sich in Zeit $\mathcal{O}(n^2)$ durchführen, da wir lediglich für alle Punktepaare die Geraden erzeugen müssen und dabei diejenigen Punkte löschen, die von Geraden überdeckt werden, die mehr als $k+1$ Punkte überdecken. Die Gesamtlaufzeit ist damit $\mathcal{O}(n^2 + k^{2k+2})$. Das Verfahren ist somit effizienter als der Suchalgorithmus mit beschränkter Baumtiefe. □