seit 1558

Benutzergeführtes Anonymisieren von Daten mit Pattern Clustering: Algorithmen und Komplexität

Diplomarbeit

zur Erlangung des akademischen Grades

Diplom-Informatiker

Friedrich-Schiller-Universität Jena

Fakultät für Mathematik und Informatik

eingereicht von Thomas Köhler

geb. am 03.07.1986 in Mühlhausen

Betreuer: Dipl.-Inf. Robert Bredereck (TU Berlin)
Jun.-Prof. Dr. Markus Chimani
Dipl.-Inf. André Nichterlein (TU Berlin)
Prof. Dr. Rolf Niedermeier (TU Berlin)

Jena, 16. Dezember 2011

Zusammenfassung

Eine Matrix M ist k-anonym, wenn jede Zeile aus M zu mindestens $k-1$ anderen Zeilen in M identisch ist. Um eine beliebige Matrix in eine k-anonyme Matrix zu überführen, werden einige Zeichen in M durch $\star$-Symbole ersetzt, das heißt sie werden gelöscht. Eine Matrix durch eine minimale Anzahl an Löschungen in eine k-anonyme Matrix zu überführen, ist ein gut untersuchtes NP-schweres Problem. Die Idee einer k-anonymen Ausgabematrix wurde von Bredereck et al. [9] erweitert, sodass der Benutzer Einfluss auf den Vorgang des Löschens nehmen kann. Der Benutzer erhält hierbei die Möglichkeit Muster vorzugeben, mit welchen sich bestimmen lässt, welche Stellen in der Matrix gelöscht werden dürfen. Das Problem, ob sich aus einem gegebenen Muster und einer Eingabematrix eine k-anonyme Matrix bilden lässt, ist als NP-schwer bekannt, selbst für Spezialfälle. In dieser Arbeit wird ebenfalls die NP-Schwere gezeigt, allerdings mit einer einfacheren Reduktion und für eingeschränktere Spezialfälle. Für NP-schwere Spezialfälle wird fixed-parameter-tractability gezeigt: Zum einen wird dies bezüglich der Anzahl an Mustern und zum anderen bezüglich dem kombinierten Parameter aus der Anzahl Löschungen und der Anzahl verschiedener Muster gezeigt. Des Weiteren werden verschiedene Kostenfunktionen vorgestellt, sodass der Benutzer mit der Wahl der Kostenfunktion weiteren Einfluss auf die Gestalt der k-anonymen Ausgabematrix nehmen kann.

Inhaltsverzeichnis

1 Einführung **4**
- 1.1 Verwandte Arbeiten 5
- 1.2 Pattern Clustering - Anwendungsmöglichkeiten 5

2 Definitionen **8**
- 2.1 Parametrisierte Komplexitätstheorie 10
- 2.2 Ergebnisse 13
- 2.3 Kostenfunktionen 14
  - 2.3.1 Standardkostenfunktion 14
  - 2.3.2 Binäre Kostenfunktion 14
  - 2.3.3 Allgemeine Kostenfunktion 17
  - 2.3.4 Clustering-Kostenfunktion 18

3 Komplexität **20**
- 3.1 Zur Nichtexistenz polynomieller Problemkerne 20
- 3.2 Reduktion von Constraint Bipartite Vertex Cover 25

4 Algorithmen **29**
- 4.1 Parameter p und $k = 1$ 31
- 4.2 Parameter p und $k \in \mathbb{N}$ 33
- 4.3 Partieller Problemkern für spezielle Kostenfunktionen 40
- 4.4 Parameter s 43

5 Ausblick **53**

Literaturverzeichnis **55**

1 Einführung

Mit der ständig wachsenden Erhebung personenbezogener Daten zur statistischen Verwertung stellt sich auch die Frage wie man die Privatsphäre des Einzelnen schützen kann. Somit ist das Anonymisieren von zu veröffentlichenden Daten zu einem interessantem Forschungsgebiet geworden. Ein zentraler Begriff in diesem Gebiet ist k-Anonymity. Daten werden in Matrizen gespeichert, dabei entspricht eine Zeile den Daten einer Person. Mit einer k-anonymen Matrix bezeichnet man eine Matrix, in der jede Zeile zu mindestens $k-1$ anderen Zeilen identisch ist. Die Idee dabei ist, dass man kein Individuum eindeutig einer Zeile zuordnen kann, wenn keine Zeile einmalig vorkommt. Im Allgemeinen ist eine Matrix nicht k-anonym für $k > 1$. Um die gewünschte Eigenschaft zu erhalten, werden in ausgewählten Zeilen einige Einträge durch ein $\star$-Symbol ersetzt; man sagt sie werden gelöscht. Das klassische k-Anonymity fragt für eine gegebene Matrix welche Einträge gelöscht werden müssen, damit die Ausgabematrix k-anonym wird und möglichst wenige $\star$-Symbole enthält. Da die veröffentlichten Daten von Nutzern weiterverarbeitet werden, können, je nach Benutzer, gewisse Muster in der Gestalt der Ausgabematrix erwünscht sein.

Zum Beispiel könnte ein Statistiker k-anonyme Daten erhalten haben, aber aufgrund der Struktur dieser Daten wurden vorrangig die Stellen gelöscht in denen relevante Daten zur Verwertung standen. Liegt diese Struktur auch bei anderen in Frage kommenden Datensätzen vor, so hat dieser Statistiker kaum eine Möglichkeit einen aussagekräftigen anonymen Datensatz zu bekommen. Einfluss auf die Gestalt der Ausgabematrix kann im klassischen k-Anonymity nicht vorgenommen werden. Bredereck et al. [9] haben ein Konzept vorgeschlagen wie der Benutzer Einfluss auf die Form der Anonymisierung nehmen kann. Dieses Konzept heißt Pattern Clustering. Wird die k-anonyme Matrix mit Pattern Clustering erzeugt, so kann der vorher erwähnte Statistiker Löschmuster vorgeben, sodass die relevanten Stellen seltener oder gar nicht mehr gelöscht werden. Neben der zu anonymisierenden Eingabematrix erhält Pattern Clustering außerdem vom Benutzer gewählte Muster welche vorschreiben, wie gelöscht werden darf. Diese Muster (englisch Pattern) werden in Form von Vektoren über einem binären Alphabet dargestellt, sogenannten Patternvektoren. Jeder Patternvektor besteht aus so vielen Zeichen wie die Eingabematrix Spalten hat und gibt dabei an, welche Spalten einer Zeile gelöscht werden und welche nicht. Eine Lösung für Pattern Clustering weist jeweils mindestens k Zeilen der Eingabematrix auf einen Patternvektor zu. Diese Eingabezeilen müssen in den Spalten, in denen nicht gelöscht wird, identisch sein. Somit wandelt eine

Lösung eine Eingabematrix in eine k-anonyme Ausgabematrix um. Diese Arbeit greift komplexitätstheoretische und algorithmische Resultate von Bredereck et al. [9] auf und ergänzt sie durch weitere Ergebnisse für interessante Spezialfälle.

1.1 Verwandte Arbeiten

Im Jahr 2002 hat Sweeney [24] k-Anonymity, ein formales Modell zum Schutz der Privatsphäre beim Veröffentlichen von Daten, vorgeschlagen. Dieses Modell wurde schnell populär. Bonizzoni et al. [6] zeigten, dass k-Anonymity NP-schwer ist, auch wenn die Eingabematrix nur 8 Spalten hat und $k = 4$ gilt. Außerdem ist es NP-schwer für $k = 3$ [20], selbst bei einer Alphabetgröße von zwei [6]. Blocki und Williams haben gezeigt für $k = 2$ ist k-Anonymity in polynomieller Zeit lösbar [3].

Die parametrisierte Komplexität von k-Anonymity wurde zuerst von Evans et al. [14] untersucht. Sie zeigten unter anderen fixed-parameter tractability für den kombinierten Parameter "Anzahl an Löschungen" und "Anzahl an Zeilen" in der Eingabematrix, beziehungsweise bezüglich den kombinierten Parameter "Anzahl an Löschungen" und "Anzahl an Spalten" in der Eingabematrix. Bonizzoni et al. [7] setzten die Untersuchungen fort und zeigten fixed-parameter tractability für den kombinierten Parameter "Anzahl an Spalten" und Alphabetgröße, sowie W[1]-Schwere bezüglich des Parameters "Anzahl an Löschungen". Bredereck et al. [8] haben NP-Schwere für den Spezialfall, dass es nur zwei verschiedene Ausgabezeilentypen gibt, gezeigt. Außerdem wurde ein Algorithmus vorgestellt, der fixed-parameter tractability bezüglich dem Parameter "Anzahl der Eingabezeilentypen" nachweist. Neben dem klassischen k-Anonymity wurden bereits Erweiterungen zu dem Modell entwickelt, wie zum Beispiel l-Diversity [19] p-Sensitive [11, 25] oder Domain Generalization Hierachies [23, 21]. Aggarwal et al. [1] stellten eine Möglichkeit vor, mittels Clustering zu Anonymisieren: Sie definierten zwei Probleme, bei welchen jeweils angenommen wird die Daten wären Punkte in einem Raum. Dabei soll der Raum die Eigenschaft haben, dass Punkte die nah beieinander liegen ähnlich sind. Daten welche nah beieinander sind, werden Zusammengefasst und anstatt der Daten wird das Datum des gemeinsamen Mittelpunkts, der Radius und die Größe des Clusters veröffentlicht. Dabei wird zum Beispiel der größte Radius der Cluster minimiert. Die Anonymität wird durch eine Mindestgröße der Cluster erreicht.

1.2 Pattern Clustering - Anwendungsmöglichkeiten

Anonymisierte Daten werden oft, beispielsweise in der Medizin, zu statistischen Auswertungen genutzt. Dabei gilt es stets den Spagat zwischen dem Schutz der persönlichen

Daten der Patienten und der Aussagekraft der anonymisierten Daten zu meistern. Klassische Verfahren, wie zum Beispiel k-Anonymity, können hier durchaus versagen. In einem Beispielszenario zeigen wir die Vorteile von Pattern Clustering gegenüber k-Anonymity auf: Angenommen ein Biostatistiker möchte den Zusammenhang von möglichen Erkrankungen und der beruflichen Tätigkeit von Patienten und zugleich den Einfluss von Tabakkonsum auf die jeweilige Krankheit untersuchen. Dabei sei nicht von Interesse wie sich die Gewohnheit zu rauchen zu dem jeweiligen Beruf verhält. Der Statistiker fordert nun Daten von Krankenhäusern an, mit der Information, dass folgende drei Daten für ihn relevant sind: ob der Patient raucht, welche Krankheit er hat und welche berufliche Tätigkeit er ausübt. Ein Krankenhaus verfüge nun über einen Datensatz wie in folgender Tabelle:

Raucher	Krankheit	Beruf
Ja	Hautkrankheit	Chemiker
Nein	Hautkrankheit	Chemiker
Ja	Lungenkrankheit	Chemiker
Ja	Lungenkrankheit	Arzt
Ja	Lungenkrankheit	Arzt
Ja	Herzkrankheit	Arzt
Nein	Herzkrankheit	Chemiker

Das Krankenhaus ist laut Bundesdatenschutzgesetz § 3 Absatz 6 BDSG, beziehungsweise Krankenhausdatenschutzgesetz der Länder, verpflichtet die Daten zu anonymisieren. Es entscheidet sich für 2-Anonymity mit der geringsten Anzahl an Schwärzungen. Eine Lösung ist in Tabelle 1.1 abgebildet.

Raucher	Krankheit	Beruf
Ja	$\star$	Chemiker
Nein	$\star$	Chemiker
Ja	$\star$	Chemiker
Ja	$\star$	Arzt
Ja	$\star$	Arzt
Ja	$\star$	Arzt
Nein	$\star$	Chemiker

Tabelle 1.1: Abgebildet ist die kostenoptimale Lösung für 2-Anonymity.

In diesem Fall wurde dabei die wichtige Spalte mit den Krankheiten vollständig ge-

schwärzt. Zusammenhänge bezüglich der Krankheiten sind damit in diesem ungünstigen Fall verloren. Wenn der Statistiker die Anonymisierung steuern könnte, so würde er diesen ungünstigen Fall verhindern. Eine Form der benutzergeführten Anonymisierung ist Pattern Clustering. Dabei kann der Statistiker Muster zum Löschen vorgeben. Diese Muster werden in Form von Zeilen repräsentiert. Je nach Symbol in diesen Zeilen kann in der entsprechenden Spalte, in der das Symbol steht, gelöscht werden oder nicht. Diese Zeilen werden Patternvektoren genannt. Das $\star$-Symbol in dem Patternvektor q bedeutet: In jeder Zeile, die auf q zugewiesen wird, wird die Spalte des $\star$-Symbols gelöscht. Das $\square$-Symbol gibt an, dass an entsprechender Stelle nicht gelöscht wird. Der Statistiker wählt also einen Patternvektor aus, welcher den Zusammenhang zwischen Krankheit und Rauchen erhält, und einen Patternvektor, der den Zusammenhang zwischen der Krankheit und dem Beruf erhält. Wegen möglichen Datenzeilen, welche nicht auf die bisherigen Patternvektoren passen, wählt er noch einen Patternvektor, der alles löscht außer der Krankheit. Da die Anzahl an gelöschten Zeichen minimiert wird, wird dieser letzte Patternvektor von einer optimalen Lösung nur so oft benutzt wie notwendig, denn er löscht doppelt so viele Zeichen wie die sonstigen Patternvektoren. Die Patternvektoren sind in Tabelle 1.2 abgebildet. Außerdem ist in Tabelle 1.2 eine resultierende Lösung zu sehen. Sie enthält den Zusammenhang, dass Chemiker erhöht an Hautkrankheiten leiden und Raucher erhöht an Lungenkrankheiten. Dieses einfache Beispiel zeigt die Vorteile von benutzergeführter Anonymisierung.

	Raucher	Krankheit	Beruf
	$\star$	Hautkrankheit	Chemiker
Patternmatrix P	$\star$	Hautkrankheit	Chemiker
($\star$, $\square$, $\square$)	Ja	Lungenkrankheit	$\star$
($\square$, $\square$, $\star$)	Ja	Lungenkrankheit	$\star$
($\star$, $\square$, $\star$)	Ja	Lungenkrankheit	$\star$
	$\star$	Herzkrankheit	$\star$
	$\star$	Herzkrankheit	$\star$

Tabelle 1.2: Diese Abbildung zeigt die Patternmatrix P und eine Lösung für Pattern Clustering mit einer minimalen Anzahl an gelöschten Zeichen.

2 Definitionen

Anonyme Datensätze. *Datensätze*, wie zum Beispiel Umfrageergebnisse oder Patientendaten, können unterschiedlich dargestellt werden. In dieser Arbeit wird ein Datum $v \in \Sigma^m$ als Zeilenvektor der Länge m über einem Alphabet Σ aufgefasst. Der Vektor v wird auch Datumszeile genannt. Mit $v[i]$ wird der Wert von v an der Stelle i bezeichnet. Als *Datensatz* $M \in \Sigma^{n \times m}$ wird eine Matrix oder Tabelle bezeichnet. In dieser Tabelle besitzt jede der n Datenzeilen m Einträge. Um mengentheoretische Operationen auf die Zeilen der Matrix anwenden zu können, muss man sie als Menge darstellen. Bei einer einfachen Überführung in eine Menge gingen Datenzeilen verloren, welche doppelt sind. Daher wird hier das Konzept der Multimenge benutzt. In einer Multimenge können Elemente mehrfach enthalten sein. Die Multimenge $R(A)$ einer beliebigen $n \times m$-Matrix A enthält alle Zeilen aus A und wird Zeilenmenge genannt. Es gilt stets $|R(A)| = n$.

Da es im Zusammenhang mit Anonymität üblich ist, dass ein Datum nicht einmalig ist, wird mit $\#v$ stets die Anzahl von Elementen bezeichnet, die in $R(M)$ identisch mit v sind. Es gilt $\#v = |\{u : u \in R(M) \wedge u = v\}|$. Als *Eingabezeilentyp* t wird eine maximale Menge von identischen Eingabezeilen bezeichnet.

Definition 1. *Ein Datensatz M ist* k-anonym *wenn jede Datenzeile mindestens k mal vorkommt, das heißt für alle $v \in R(M)$ gilt $\#v \geq k$.*

Siehe Abbildung 2.1 für eine Illustration von zwei Datensätzen.

Patternvektoren. Im Allgemeinen ist ein Datensatz nicht k-anonym. Um diese Eigenschaft zu erhalten, werden in einigen Datenvektoren Einträge geschwärzt um so Gleichheit verschiedener Zeilen zu erreichen. Genauer wird die Stelle j im Datum v geschwärzt, indem $v[j] = \star$ gesetzt wird, wobei das $\star$-Symbol als eine Art geschwärztes Zeichen interpretiert werden kann, welches nicht im ursprünglichem Alphabet Σ ist. In dieser Arbeit werden geschwärzte Zeichen auch gelöschte Zeichen genannt. Bei *benutzergeführter Anonymisierung* wird ein Muster vorgegeben, um Schwärzungen an entsprechenden Stellen zu erreichen. Formal besteht das Muster aus Patternvektoren. Ein Patternvektor $q \in \{\square, \star\}^m$ ist ein Vektor der Länge m und besteht aus $\square$-Symbolen und $\star$-Symbolen, wobei das $\square$-Symbol bedeutet, dass das entsprechende Zeichen erhalten wird und das $\star$-Symbol, dass es geschwärzt wird. Mit $P \in \{\square, \star\}^{p \times m}$ werden alle zur Verfügung stehenden Patternvektoren bezeichnet. Die *Patternmatrix* P ist ein Teil der Eingabe. Als

1	2	3
3	2	1
1	2	4
1	3	4

1	2	3
1	2	3
1	2	4
1	2	4

Abbildung 2.1: Diese Abbildung zeigt zwei Datensätzen mit jeweils vier Eingabezeilen. Der links abgebildete Datensatz ist 1-anonym, der andere Datensatz ist 2-anonym.

($\star$, $\star$, $\square$) ($\square$, $\square$, $\star$)

Abbildung 2.2: Diese Abbildung zeigt zwei Patternvektoren der Länge drei. Der linke Patternvektor würde zwei Zeichen schwärzen, nämlich das erste und das zweite. Der rechts abgebildete Patternvektor würde nur das letzte Zeichen schwärzen.

Patternvektorentyp t wird eine maximale Menge von identischen Patternvektoren bezeichnet. Mit p_t wird die Anzahl an verschiedenen Patternvektorentypen in P bezeichnet. Siehe Abbildung 2.2 für eine Illustration.

Zuweisungen. Wie bereits erwähnt, werden zum Anonymisieren die Datenvektoren auf Patternvektoren zugewiesen. Eine solche Zuweisung wird mit $\varphi : M \to P$ bezeichnet. Im Folgenden wird gesagt, dass φ dem Patternvektor q die Eingabezeile v zuweist, wenn $\varphi(v) = q$ gilt.

Eine solche Zuweisung φ heißt total genau dann, wenn für alle $v \in M$ gilt, dass $\varphi(v) \in P$. Für jede Eingabezeile v_i ist $\varphi(v_i)$ ein Patternvektor und $\varphi(v_i)[j]$ ist das j-te Symbol dieses Patternvektors und $M[i][j]$ ist das Matrixelemente in der i-ten Zeile und j-ten Spalte.

Definition 2. *Eine Zuweisung heißt* ***konsistent*** *genau dann, wenn sie total ist und für alle Urbilder v, x mit dem gleichen Bild $q = \varphi(v) = \varphi(x)$ gilt: $q[j] = \square \Rightarrow v[j] = x[j]$. Als* ***Ergebnis*** *$M'$ einer Zuweisung wird ebenfalls eine $n \times m$ Matrix bezeichnet, welche durch folgende Funktion erzeugt wird.*

$$M'[i][j] = \begin{cases} M[i][j] & \text{falls } \varphi(v_i)[j] = \square \\ \star & \text{falls } \varphi(v_i)[j] = \star \end{cases}$$

Definition 3. *Eine Eingabezeile v* passt konsistent *auf einen Patternvektor q auf den schon eine andere Eingabezeile u zugewiesen wurde, wenn für $q = \varphi(u)$ gilt: $q[j] = \square \Rightarrow v[j] = u[j]$. Der resultierende* Ausgabezeilentyp *t ist ein Vektor der Länge m, der sich von v nur dadurch unterscheidet, dass er an den Stellen der $\star$-Symbole in q ebenfalls $\star$-Symbole hat. Man sagt t wurde von q erzeugt.*

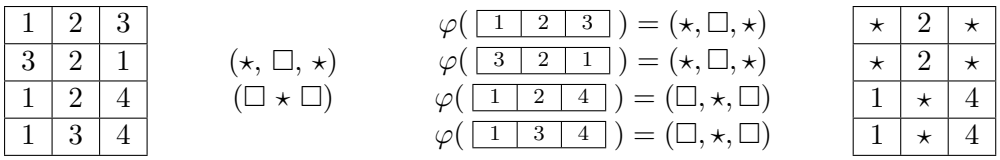


Abbildung 2.3: Abgebildet ist hier ein kleines Beispiel. Von links nach rechts ist die Eingabematrix, zwei Patternvektoren, eine Abbildung φ und die 2-anonyme Ergebnismatrix zu sehen.

Definition 4. *Eine Zuweisung φ heißt für M genau dann* k-anonym*, wenn das Ergebnis M' von φ k-anonym ist.*

Siehe Abbildung 2.3 für die Illustration einer Abbildung.

In Tabelle 2.1 sind alle wichtigen Bezeichner und Symbole dieser Arbeit dargestellt. Mit diesen Definitionen wird nun Pattern Clustering, das zentrale Problem dieser Arbeit, definiert.

> Pattern Clustering
> **Eingabe:** Eine Eingabematrix $M \in \Sigma^{n \times m}$, eine Matrix mit Patternvektoren $P \in \{\square\ ,\ \star\}^{p \times m}$ und zwei natürliche Zahlen s und k.
> **Frage:** Existiert eine konsistente Zuweisung φ von M in P, sodass die Anzahl an $\star$-Symbolen in der Ergebnismatrix M' höchstens s ist und φ k-anonym ist?

2.1 Parametrisierte Komplexitätstheorie

Die parametrisierte Komplexitätstheorie erweitert die Probleme um einen Parameter, um so Probleme genauer zu klassifizieren. Der Fokus der parametrisierten Komplexitätstheorie liegt dabei auf NP-vollständigen Problemen (NP ist die Klasse der Probleme, die von einer nichtdeterministischen Turingmaschine in polynomieller Zeit entschieden werden können [16]). Zunächst wird definiert wie ein Problem um einen Parameter erweitert wird.

Definition 5. *Ein* parametrisiertes Problem *ist eine Sprache $L \subseteq \Sigma^* \times \mathbb{N}$, wobei Σ ein beliebiges endliches Alphabet ist. Eine parametrisierte Probleminstanz ist ein geordnetes Paar $(I, e) \in \Sigma^* \times \mathbb{N}$, wobei I die eigentliche Probleminstanz ist und e der Parameter.*

Mit Hilfe des Parameters kann die Komplexität von Problemen feiner unterschieden werden. Die wichtigste parametrisierte Komplexitätsklasse ist FPT.

Symbol	Name	Beschreibung
k	Anonymität	Grad der Anonymität
Σ	Eingabealphabet	Σ ist ein endliches Alphabet
n	Anzahl Eingabezeilen	$n \in \mathbb{N}$
m	Anzahl Eingabespalten	$m \in \mathbb{N}$
M	Eingabedaten als Matrix	$M \in \Sigma^{n \times m}$
M'	Ergebnis einer Zuweisung	$M' \in (\star \cup \Sigma)^{n \times m}$
$R(M)$	Multimenge aller Eingabezeilen	$z \in R(M) \Leftrightarrow z$ ist Zeile in M
s	Kostenschranke	$s \in \mathbb{N}$
P	Matrix der Patternvektoren	$P \in \{\square\ , \star\}^{p \times m}$
p	Anzahl Patternvektoren in P	$p \in \mathbb{N}$
p_t	Anzahl Patternvektorentypen in P	$p_t \in \mathbb{N}$
q_i	i-ter Patternvektor	$1 \leq i \leq \lvert P \rvert \Rightarrow q_i \in P$
φ	Zuweisungsfunktion	$\varphi : M \to P$
$\#q_i$	Anzahl Zuweisungen	$\lvert \{v : v \in M \wedge \varphi(v) = q_i\} \rvert$

Tabelle 2.1: Wichtige in der Arbeit verwendeten Symbole und ihre Bedeutung.

Definition 6. *Ein parametrisiertes Problem $L \subseteq \Sigma^* \times \mathbb{N}$ ist in FPT, wenn für eine gegebene parametrisierte Probleminstanz (I, e) in $f(e) \cdot f_p(|I|)$ Zeit entschieden werden kann ob $(I, e) \in L$ gilt. Dabei ist f eine beliebige, nur von e abhängige, berechenbare Funktion und f_p ist ein Polynom in Abhängigkeit von $|I|$.*

Dabei wird $f(e) \cdot f_p(|I|)$ auch *FPT-Zeit* genannt. Probleme, welche in FPT liegen, werden auch *fixed-parameter tractable* genannt. Die Laufzeit wird so in einen polynomiellen Teil und einen nur von dem Parameter abhängigen exponentiellen Teil geteilt. Die Hoffnung ist, dass die gesamte Laufzeit akzeptabel ist, solange der Parameter e klein ist. Ähnlich zu der Karp-Reduktion [17] gibt es auch eine *parametrisierte Reduktion.*

Definition 7. *Seien L und L' zwei parametrisierte Probleme. Man sagt L ist* parametrisiert reduzierbar *auf L', wenn es eine Funktion R gibt, die aus einer gegebenen parametrisierten Instanz (I, e) in FPT-Zeit eine neue parametrisierte Instanz (I', e') berechnet mit den Eigenschaften:*

1. *$(I, e) \in L$ genau dann wenn $(I', e') \in L'$ und*
2. *$e' \leq f(e)$, wobei f eine beliebige berechenbare Funktion ist, die nur von e abhängt.*

Die Funktion R wird dabei parametrisierte Reduktion von L auf L' genannt. Sind L und L' zwei parametrisierte Probleme und $L \in FPT$ und es existiert eine parametrisierte Reduktion von L' auf L, so ist L' auch in FPT. Über FPT gibt es Komplexitätsklassen

die Probleme enthalten von denen vermutet wird, das diese nicht in FPT-Zeit lösbar sind. Eine solche Klasse ist W[1] [22, 13, 15]. Ist ein parametrisiertes Problem L parametrisiert reduzierbar auf ein anderes parametrisiertes Problem welches W[1]-schwer ist, so existiert wohl kein FPT-Zeit Algorithmus für L.

Ein wichtiges Konzept zum Finden guter FPT-Algorithmen ist *Kernelization*. Dabei wird versucht die Eingabeinstanz zu verkleinern und auf den "schweren" Kern, den Problemkern, zu reduzieren.

Definition 8. *Sei (I, e) eine Instanz des parametrisierten Problems L. Ein* Problemkern *ist eine kleinere Instanz (I', e') mit folgenden Eigenschaften*

1. $e' \leq e$,
2. $|I'| \leq f(e)$ *wobei f eine beliebige berechenbare Funktion ist, die nur von e abhängt, und*
3. $(I, e) \in L$ *genau dann wenn* $(I', e') \in L$.

Außerdem muss (I', e') in $g_p(|I|, e)$ Zeit berechnet werden können, wobei g_p ein beliebiges Polynom ist. Die Funktion f wird auch Größe des Problemkerns genannt.

Man sagt für ein parametrisiertes Problem L gibt es einen Problemkern, wenn es für jede Instanz $(I, e) \in \{0,1\}^* \times \mathbb{N}$ einen Problemkern gibt. Ist die Größe f des Problemkerns ein Polynom, so spricht man von einem polynomiellen Problemkern. Ein Problem ist in FPT genau dann, wenn es einen Problemkern für L gibt [10]. Schwieriger ist die Frage zu beantworten ob es für jedes Problem in FPT einen Problemkern polynomieller Größe gibt. Bodlaender et al. [4] haben gezeigt, dass es Probleme in FPT gibt, die keinen polynomiellen Problemkern haben, es sei denn coNP $\subseteq$ NP/poly. Sie haben außerdem ein Konzept entwickelt um zu zeigen, wie man die Nichtexistenz polynomieller Problemkerne zeigt. In dieser Arbeit wird nur eine leichtere Möglichkeit mittels einer Reduktion auf ein anderes Problem ohne polynomiellen Problemkern benutzt. Dazu sind folgende Definitionen notwendig. Sei (I, e) eine parametrisierte Probleminstanz, dann ist $I\#1^e$ ihre unparametrisierte Probleminstanz. Sei L ein parametrisiertes Problem, so ist $L_c = \{I\#1^e : (I, e) \in L\}$, wobei $\# \notin \Sigma$, die *unparametrisierte Version* von L.

Definition 9. *Seien L und L' zwei parametrisierte Probleme. Man sagt L ist* parametererhaltend polynomzeitreduzierbar *auf L', wenn es eine polynomzeitberechenbare Funktion $f : \{0,1\}^* \times \mathbb{N} \to \{0,1\}^* \times \mathbb{N}$ und ein Polynom $poly : \mathbb{N} \to \mathbb{N}$ gibt und für alle $I \in \{0,1\}^*$ und alle $k \in \mathbb{N}$ gilt: Wenn $f((I, k)) = (I', k')$ dann*

1. $(I, k) \in L$ *genau dann wenn* $(I', k') \in L'$ *und*
2. $k' \leq poly(k)$.

Die Funktion f wird dabei parametererhaltende Polynomzeitreduktion *von L auf L' genannt.*

Parametererhaltende Reduktionen können genutzt werden um die Nichtexistenz von polynomiellen Problemkernen zu zeigen.

Satz 1 ([5])**.** *Seien L und L' zwei parametrisierte Probleme und seien L_c und L'_c deren unparametrisierten Versionen. Seien $L_c \in$NP und L'_c NP-schwer. Gibt es eine parametererhaltende Polynomzeitreduktion von L auf L' und gibt es für L' einen polynomiellen Problemkern, so gibt es auch einen polynomiellen Problemkern für L.*

Mit Satz 1 kann man Nichtexistenzresultate bezüglich polynomiellen Problemkernen mit einfachen Mitteln übertragen.

2.2 Ergebnisse

In dieser Arbeit wird die parametrisierte Komplexität von Pattern Clustering untersucht. Es wird gezeigt das es keinen polynomiellen Problemkern für Pattern Clustering mit nur drei Spalten und mit dem Parameter p, der Anzahl an Patternvektoren gibt, es sei denn coNP/poly $\subseteq$ NP. Für den Spezialfall mit 2 Spalten $m = 2$, ohne Anonymitätsforderung $k = 1$ und ohne Kostenschranke $s = \infty$ wird gezeigt, dass Pattern Clustering NP-schwer ist. Im zweiten Teil der Arbeit werden parametrisierte Algorithmen vorgestellt. Es wird ein FPT-Algorithmus mit Parameter p, der Anzahl an Patternvektoren, für den Spezialfall $k = 1$ und $s = \infty$, gezeigt. Diese Idee wird übernommen und verfeinert um so fixed-parameter-tractability bezüglich Parameter p und den Spezialfall $s = \infty$ nachzuweisen. Abschließend wird für $k = 1$ fixed-parameter-tractability bezüglich dem kombinierten Parameter (s, p_t) gezeigt, wobei p_t die Anzahl an Patternvektortypen ist. Die Tabelle 2.2 bietet einen Überblick über die parametrisierte Komplexität von Pattern Clustering.

Spezialfall	p	(s, p_t)	n	t_{in}	m	k	$\lvert\Sigma\rvert$
-	XP*	XP*	FPT*	FPT*	NP-schwer*	NP-schwer*	NP-schwer*
$k = 1$	XP*	FPT	FPT*	FPT*	NP-schwer	NP-schwer*	NP-schwer*
$m = 2$	XP*	XP*	FPT*	FPT*	NP-schwer	NP-schwer	FPT*
$s = \infty$	FPT	-	FPT*	FPT*	NP-schwer	NP-schwer*	NP-schwer*

Tabelle 2.2: In dieser Tabelle wird die Komplexität von Pattern Clustering bezüglich verschiedener Parameter dargestellt. Die mit einem Stern gekennzeichneten Ergebnisse sind aus Bredereck et al. [9].

2.3 Kostenfunktionen

In diesem Kapitel werden einige Kostenfunktionen untersucht. Genauer gibt eine Kostenfunktion an, welche Kosten entstehen wenn eine Eingabezeile auf einen bestimmten Patternvektor zugewiesen wird. Die Kosten hängen dabei nur von dem Patternvektor ab. Die Summe über die Kosten jeder einzelnen zugewiesenen Eingabezeile sind die Gesamtkosten einer Zuweisung φ. Es wird später gezeigt, dass Pattern Clustering für jede dieser Kostenfunktionen NP-schwer bleibt.

2.3.1 Standardkostenfunktion

Bei der Standardkostenfunktion sind die Kosten die eine Eingabezeile beim Zuweisen auf einen Patternvektor q erzeugt, genau die Anzahl an $\star$-Symbolen in q. Die Standardkostenfunktion wurde bereits erwähnt. Dies ist die gleiche Kostenfunktion wie in der Definition von Pattern Clustering, wie sie auch in der Arbeit von Bredereck et al. [9], benutzt wird.

2.3.2 Binäre Kostenfunktion

Eine Kostenfunktion γ_b heißt *binär*, wenn jede Zuweisung auf einen Patternvektor Kosten 1 oder 0 verursacht. Dabei sind Patternvektoren, welche Kosten 0 verursachen von der Gestalt, dass sie nur aus $\square$-Symbolen bestehen. Jeder Patternvektor der also mindestens ein $\star$-Symbol enthält, verursacht entsprechend Kosten von genau 1 pro Zeile die auf diesen Patternvektor zugewiesen wird. Die binäre Kostenfunktion lässt sich auch durch die Standardkostenfunktion darstellen, wie folgende Beobachtung zeigt.

Beobachtung 1. *Sei $I = (M, P, k, s)$ eine* Pattern Clustering*-Instanz mit der binären Kostenfunktion. Dann kann I in eine entscheidungsäquivalente Instanz $I' = (M', P', k, s')$ mit der Standardkostenfunktion überführt werden, wobei die Anzahl an Eingabezeilen und Patternvektoren gleich bleibt.*

Beweis. Sei $I = (M, P, k, s)$ mit der binären Kostenfunktion gegeben. Es wird nun eine Instanz $I' = (M', P', k, s')$ mit der Standardkostenfunktion konstruiert. Sei q der Patternvektor aus P mit den meisten $\star$-Symbolen, sei $x(q)$ die Anzahl an $\star$-Symbolen in q. Die neue Eingabematrix ergibt sich, indem an die alte Eingabematrix $M \in \Sigma^{m \times n}$ an jede Eingabezeile $x(q)$ viele neue Spalten angehangen werden. In jeder dieser neuen Spalten steht jeweils das gleiche Symbol. Die neue Patternmatrix P' ergibt sich aus P wie folgt: sei $q_i \in P$ dann werden an q_i nun insgesamt $x(q)$ neue Spalten angehangen.

Enthält q_i mindestens ein $\star$-Symbol, so enthalten diese neuen Spalten $x(q) - x(q_i)$ $\star$-Symbole und alle anderen Spalten enthalten $\square$-Symbole. Enthält q_i keine $\star$-Symbole, so enthalten alle neuen Spalten ebenfalls $\square$-Symbole. Jeder Patternvektor in P' enthält nun entweder kein $\star$-Symbol oder genau $x(q)$. Die neue Kostenschranke s' ist gleich $s \cdot x(q)$. Die Konstruktion ist korrekt, da jede Zuweisung in der neuen Instanz genau $x(q)$ mal so viele Kosten verursacht wie in der alten Instanz. $\square$

Diese Reduktion ist eine parametrisierte Reduktion für die folgenden Parameter: Der Anzahl an Patternvekroten p, der Anzahl an Zeilen n in M, der Anzahl an Spalten m in M und k. Somit sind alle Algorithmen, die für die Standardkostenfunktion gültig sind, auch für die binäre Kostenfunktion gültig. Diese Reduktion ist keine parametrisierte Reduktion für den Parameter Anzahl an gelöschten Zeichen s, da das neue s einen Faktor enthält der gleich m, der Anzahl an Spalten, sein kann und m im Allgemeinen nicht durch eine Funktion in s beschränkt ist. Parametrisierte Algorithmen bezüglich dem Parameter s oder mit s kombinierte Parameter lassen sich also im Allgemeinen nicht so übertragen.

Pattern Clustering mit der binären Kostenfunktion lässt sich auch als die Aufgabe die Zahl nichtmodifizierter Zeilen zu maximieren beschreiben. Hinzu kommt die Nebenbedingung, dass die entstehenden Cluster Mindestgröße k haben. Außerdem sind die zu modifizierenden Zeilen konsistent auf ein Muster, also die Patternvektoren, abzubilden. Eine Anwendung ist das Zusammenstellen von Teams, sodass möglichst wenig Konflikte bei der gemeinsamen Arbeit entstehen. In der Informatik es gibt verschiedene Sachverhalte über die unterschiedliche Informatiker unterschiedlicher Meinung sein können. In diesem Beispiel sei das ob gilt P≠NP, ob man Diagonalisieren als Beweismethode anerkennt, ob Null eine natürliche Zahl ist und welches die bevorzugte Beweismethode ist. Um zu vermeiden dass ein Streit in diesen Punkten die Arbeit beeinträchtigt, könnte es sinnvoll sein die Teams so zusammen zustellen dass innerhalb der entsprechenden Teams nur geringe Konflikte auftreten. Ist man in einem Punkt unterschiedlicher Meinung, so wächst daraus nicht sofort ein Konflikt. Es sei bekannt, dass man sich in mindestens 2 Punkten einig sein muss, damit die Arbeit gut erledigt werden kann.

Für verschiedene Arbeiten sind Übereinstimmungen in verschiedenen Fragen notwendig. Zum Beispiel ist es für eine komplexitätstheoretische Arbeit notwendig sich über die P≠NP Frage einig zu sein und ob Diagonalisieren erlaubt ist. Somit bildet die Frage P≠NP mit der Frage zum Diagonalisieren eine *notwendige Übereinstimmung*. Jedes Team soll mindestens eine notwendige Übereinstimmung haben. Weitere Paare von notwendiger Übereinstimmung seien die Frage P≠NP mit der bevorzugten Beweisart und die Meinung zu Diagonalisieren mit der Frage $0 \overset{\in}{\notin} \mathbb{N}$.

Für jeden Informatiker werden in diesem Beispiel vier Werte gespeichert. Es ergebe sich folgende Matrix:

Informatiker	P≠NP	Diagonalisieren	$0 \overset{\in}{\notin} \mathbb{N}$	Beweisart
A_1	nein	ja	$\in$	Diagonalisieren
A_2	nein	ja	$\in$	Diagonalisieren
A_3	nein	ja	$\in$	Diagonalisieren
A_4	nein	ja	$\in$	Diagonalisieren
A_5	ja	nein	$\notin$	Direkt
A_6	ja	nein	$\notin$	Konstruktiv
A_7	ja	nein	$\notin$	Direkt
A_8	ja	ja	$\notin$	Induktion
A_9	ja	ja	$\notin$	Induktion
A_{10}	ja	ja	$\in$	Diagonalisieren
A_{11}	ja	ja	$\in$	Induktion

Diese Abbildung zeigt die Eingabematrix mit 11 Informatikern und deren Meinungen.

Man beachte das die erste Spalte hier nur zur besseren Darstellung dient, sie gehört nicht zur Eingabematrix. Es werden die Patternvektoren so ausgewählt, dass es in jedem Team eine notwendige Übereinstimmung gibt. Somit ergibt sich folgende Patternmatrix:

Patternmatrix P
(□, □, □, □, □,)
(⋆, □, □, ⋆, □,)
(□, □, ⋆, ⋆, □,)
(□, ⋆, ⋆, □, □,)

Der Sinn nur □-Symbole enthaltender Patternvektoren ist, dass es Teams gibt, in denen sich alle Informatiker vollständig einig sind. Mit Hilfe der Kostenschranke s kann man festlegen, wie viele Eingabezeilen auf Patternvektoren abgebildet werden die nur □-Symbole enthalten. Somit kann man mit s festlegen, wie viele Informatiker mindestens in Teams sein sollen, in denen sich alle einig sind. Sei $s = 8$, also gibt es mindestens 8 Informatiker, die nicht in solchen Teams sein müssen. Bei 11 Informatikern insgesamt ergeben sich 3 welche in solchen Teams sein müssen. Die anderen Patternvektoren erzwingen jeweils eine notwendige Übereinstimmung in jedem Team. Dies wird erreicht indem die Patternvektoren □-Symbole an jenen Stellen enthalten an denen man sich einig sein soll. Eine gute Mindestgröße eines Teams sei drei, also $k = 3$. Somit ergibt sich eine mögliche Lösung:

Team	Informatiker	P≠NP	Diagonalisieren	$0 \begin{smallmatrix}\in\\ \notin\end{smallmatrix} \mathbb{N}$	Beweisart
1	A_1	nein	ja	$\in$	Diagonalisieren
1	A_2	nein	ja	$\in$	Diagonalisieren
1	A_3	nein	ja	$\in$	Diagonalisieren
2	A_4	$\star$	ja	$\in$	$\star$
2	A_{10}	$\star$	ja	$\in$	$\star$
2	A_{11}	$\star$	ja	$\in$	$\star$
3	A_5	ja	$\star$	$\notin$	$\star$
3	A_6	ja	$\star$	$\notin$	$\star$
3	A_7	ja	$\star$	$\notin$	$\star$
3	A_8	ja	$\star$	$\notin$	$\star$
3	A_9	ja	$\star$	$\notin$	$\star$

Tabelle 2.3: Diese Tabelle zeigt eine konfliktfreie Aufteilung auf drei Teams.

2.3.3 Allgemeine Kostenfunktion

Eine Kostenfunktion γ_a heißt *allgemein*, wenn jede Zuweisung auf einen Patternvektor einen vom Benutzer gewählten Betrag an Kosten verursacht. Somit kann unter anderem modelliert werden, dass unterschiedliche Spalten unterschiedlich wertvoll sind. Dies gibt dem Ersteller der Patternmatrix eine weitere Möglichkeit die Gestalt der Lösung zu beeinflussen. Patternvektoren, welche die gleiche Anzahl an $\star$-Symbolen haben, könnten unterschiedliche Kosten bekommen, um so eine Ordnung auf den Patternvektoren zu bilden. Es wäre sogar möglich, Patternvektoren welche nur eine Spalte löschen sehr teuer zu machen, weil diese eine Spalte wichtig für den Wert der Ausgabematrix ist. Es wird ein abstraktes Beispiel vorgestellt.

Eine abstrakte Datentabelle habe 4 Spalten S_1, S_2, S_3 und S_4 absteigend sortiert nach ihrer Wichtigkeit in einer möglichen konkreten Anwendung. Die Spalten werden wie folgt bewertet: $S_1 = 45$, $S_2 = 9$, $S_3 = 3$ und $S_4 = 1$. Die Kosten eines Patternvektors ergeben sich aus der Summe der Werte der Spalten die er löscht. Der Patternvektor $(\star, \square, \square, \square)$ hätte somit Kosten 45, $(\square, \star, \star, \square)$ hätte Kosten 12 und $(\star, \star, \star, \square)$ hätte Kosten 57. Der Sinn bei dieser Kostenaufteilung ist folgender: Es muss mindestens dreimal eine Löschung einer weniger wertvollen Spalte eingespart werden um einmal ein Zeichen einer höherwertigen Spalte, bei gleichen Kosten, löschen zu können. Um bei gleichen Kosten die erste Spalte zu löschen würde es nicht einmal genügen wenn man so jeweils drei Löschungen in allen anderen Spalten einsparen könnte.

Eine andere Möglichkeit besteht darin identische Patternvektoren schrittweise mehr Kosten verursachen zulassen um so zu erreichen, dass die Lösung auf möglichst unterschiedliche Patternvektoren abbildet. Auf diese Weise könnte man fordern, dass alle Ausgabe-

zeilentypen durch einen anderen Patternvektortyp erzeugt werden bis auf einige Ausnahmen. Wieviele solcher Ausnahmen erlaubt sind wird durch die Kostenschranke s geregeln. In einem Beispiel gibt es n Eingabezeilen und für jeden gewünschten Patternvektortyp gibt es einen Patternvektor in P mit Kosten eins. Außerdem gibt es für jeden Patternvektortyp einen weiteren Patternvektor mit Kosten zwei. Nun kann mit s die Anzahl an Zuweisungen auf die teuren Patternvektoren bestimmt werden. Sei $s = n$, so gibt es in jeder Lösung keine einzige Zuweisung auf einen teuren Patternvektor. Sei $s = n + x$ so gibt es in jeder Lösung maximal x Eingabezeilen die auf einen teuren Patternvektor abgebilden werden. Somit gibt es maximal $\frac{x}{k}$ viele doppelte Patternvektoren welche von einer Lösung benutzt werden.

2.3.4 Clustering-Kostenfunktion

Eine Kostenfunktion γ_c heißt *clustering*, wenn sie nicht für jede zugewiesene Eingabezeile Kosten verursacht, sondern für jeden Patternvektor, auf den mindestens einmal abgebildet wurde, Kosten von eins verursacht. Die Clustering-Kostenfunktion minimiert somit die Anzahl an verwendeten Patternvektoren und somit die Anzahl an Ausgabezeilentypen. In dieser Arbeit wird ein FPT-Algorithmus bezüglich Parameter p, der Anzahl an Patternvektoren, vorgestellt der für die Standardkostenfunktion und den Spezialfall $s = \infty$ immer in $O(p! \cdot m \cdot n^2 \cdot t_{\text{in}}^3 \cdot \log t_{\text{in}})$ Zeit alle Lösungen findet, welche eine minimale Menge an Patternvektoren benutzen. Dieser Algorithmus kann somit genutzt werden, um Pattern Clustering mit der Clustering-Kostenfunktion zu lösen.

Man beachte, dass jede Instanz $I = (M, P, k, s)$ trivial eine ja-Instanz ist, wenn P einen Patternvektor q enthält, der nur aus $\star$-Symbolen besteht, $s > 0$ und $n \geq k$ gilt. Denn man kann in diesem Fall alle Eingabezeilen, mit Kosten eins, auf q abbilden. Weiterhin gilt ohne Beschränkung der Allgemeinheit $s < p$. In jeder Instanz in der $s \geq p$ gilt, kann $s = p$ gesetzt werden, da die Anzahl an benutzten Patternvektoren ohnehin bereits durch die Anzahl an vorhandenen Patternvektoren beschränkt ist.

Eine mögliche Anwendung für die Clustering-Kostenfunktion ist das Klassifizieren. Man hat eine Menge von Objekten mit Eigenschaften und möchte diese so Clustern, dass jedes Cluster eine Mindestgröße hat und mindestens an vorgegebenen Stellen jeweils identisch sind. Vorstellbar wäre das Klassifizieren von Krankheiten und deren mögliche Ursache. Dazu ist eine vollständige Matrix mit den Krankheiten der Patienten sowie ihrer persönlichen Daten gegeben.

Krankheit	Raucher	Beruf	Alter
Lungenkrankheit	ja	Arzt	40-50
Lungenkrankheit	ja	Chemiker	50-60
Lungenkrankheit	ja	Pädagoge	60-70
Hautkrankheit	nein	Chemiker	50-60
Hautkrankheit	ja	Chemiker	60-70
Hautkrankheit	nein	Chemiker	40-50
Herzkrankheit	nein	Chemiker	60-70
Herzkrankheit	ja	Arzt	60-70
Herzkrankheit	nein	Pädagoge	60-70

Eine Eingabematrix mit 9 Patientendaten und deren Krankheit, Beruf und Alter, sowie ob sie Raucher sind oder nicht.

Der Benutzer wählt nun Patternvektoren aus welche die Krankheit und eine weitere Spalte nicht löschen.

Patternmatrix P
($\square$, $\square$, $\star$, $\star$)
($\square$, $\star$, $\square$, $\star$)
($\square$, $\star$, $\star$, $\square$)

Krankheit	Raucher	Beruf	Alter
Lungenkrankheit	ja	$\star$	$\star$
Lungenkrankheit	ja	$\star$	$\star$
Lungenkrankheit	ja	$\star$	$\star$
Hautkrankheit	$\star$	Chemiker	$\star$
Hautkrankheit	$\star$	Chemiker	$\star$
Hautkrankheit	$\star$	Chemiker	$\star$
Herzkrankheit	$\star$	$\star$	60-70
Herzkrankheit	$\star$	$\star$	60-70
Herzkrankheit	$\star$	$\star$	60-70

Patternmatrix P und eine optimale Lösung für Pattern Clustering mit Clustering-Kostenfunktion und $k = 3$.

Für jede Krankheit wurde so ein Cluster gefunden in dem eine mögliche Ursache der Krankheit repräsentiert wird, sodass für alle anderen Krankheiten auch eine mögliche Ursache gefunden wurde.

3 Komplexität

In diesem Kapitel wird die Komplexität von Pattern Clustering untersucht. Bereits bekannt ist die NP-Schwere für den Spezialfall $m = 4$, sowie für den Spezialfall $k = 1$, Alphabetgröße $|\Sigma| = 2$ und ohne Kostenschranke $s = \infty$ [9]. Außerdem ist unter der Annahme coNP/poly $\not\subseteq$ NP bekannt, dass kein polynomieller Problemkern für Pattern Clustering bezüglich dem Parameter p existiert [9]. Dies gilt sogar für den Spezialfall $m = 4$. In diesem Kapitel wird gezeigt, dass unter diesen Voraussetzungen auch kein polynomieller Problemkern existiert, selbst wenn $m = 3$ gilt. Dies wird mit Hilfe einer parametererhaltende Polynomzeitreduktion von Set Cover gezeigt.

Im zweiten Teil dieses Kapitels wird eine Reduktion von dem NP-schweren Problem Constraint Bipartite Vertex Cover gegeben. Diese Reduktion zeigt NP-Schwere für den Spezialfall $k = 1$, $s = \infty$ und $m = 2$. Weiterhin lässt sich ihre Korrektheit, im Unterschied zu den bereits bekannten Reduktionen, durch einen vergleichsweise kurzen Beweis zeigen.

3.1 Zur Nichtexistenz polynomieller Problemkerne

In diesem Abschnitt wird eine parametererhaltende Polynomzeitreduktion von Set Cover mit dem kombinierten Parameter $(|U|, h)$ auf den Spezialfall $m = 3$ von Pattern Clustering mit dem Parameter p angegeben.

> Set Cover
> **Eingabe:** Ein endliches Universum $U = \{u_1, u_2, ...\}$, eine Menge $\mathcal{F} = \{F_i : F_i \subseteq U\}$ von Paketen, also Teilmengen von U und eine positive ganze Zahl $h \in \mathbb{N}$.
> **Frage:** Existiert eine Teilmenge $\mathcal{F}' \subseteq \mathcal{F}$ mit $\bigcup \mathcal{F}' = U$ und $|\mathcal{F}'| < h$?

Set Cover ist NP-schwer [17] und es existiert kein polynomieller Problemkern bezüglich dem kombinierten Parameter $(|U|, h)$, unter der Voraussetzung coNP $\not\subseteq$ NP/poly [12].

Satz 2. Pattern Clustering *mit $m \geq 3$ Spalten hat keinen polynomiellen Problemkern bezüglich der Anzahl an Patternvektoren p, es sei denn coNP $\subseteq$ NP/poly.*

Beweis. Der Satz wird mittels einer parametererhaltende Polynomzeitreduktion von Set Cover gezeigt. Daraus folgt die Nichtexistenz des polynomiellen Problemkerns [12]. Aus einer gegebenen parametrisierten Instanz $I_S = (U, \mathcal{F}, h)$ für Set Cover wird eine andere parametrisierte Instanz $I_P = (M, P, k, s)$ für Pattern Clustering konstruiert, sodass $I_S \in$ Set Cover $\Leftrightarrow I_P \in$ Pattern Clustering gilt. Es wird zunächst die Konstruktion beschrieben, später die Korrektheit gezeigt und am Ende wird auf die Laufzeit eingegangen.

Konstruktion: Am Ende der Konstruktion werden alle Teile der Konstruktion noch einmal in der Tabelle 3.1 übersichtlich dargestellt.

Die Eingabematrix M für I_P besteht aus drei Spalten und vier verschiedenen *Arten von Eingabezeilen*, nicht zu verwechseln mit Eingabezeilentypen. Der Unterschied zwischen Eingabezeilentypen und Arten von Eingabezeilen ist, dass in dieser Reduktion Arten von Eingabezeilen eine ähnliche Konstruktion und Funktion haben, aber im Gegensatz zu Eingabezeilentypen, nicht identisch sein müssen. Einige Eingabezeilen werden sogenannte *Einmaligkeitssymbole* $\triangle$ enthalten. Dieses $\triangle$-Symbol steht für ein Zeichen das in M nur einmal vorkommt. Somit sind zwei $\triangle$-Symbole nicht identisch, es gilt also $\triangle \neq \triangle$. Eine Stelle in der ein $\triangle$-Symbol steht muss also immer gelöscht werden, solange $k > 1$. Die Notation mit dem $\triangle$-Symbol soll es vereinfachen einmalig vorkommende Symbole darzustellen.

Die erste Art von Eingabezeilen ist die sogenannte *Universumsart* Z_u: Für jedes Element in U gibt es eine Zeile der Universumsart. Für jedes $u_i \in U$ ist die entsprechende Eingabezeile in M gleich $Z_{u_i} :=$ | $\triangle$ | $\triangle$ | i | . Es gibt $|U|$ viele solcher Zeilen. Die nächsten beiden Arten sind die *Variablenzeilenarten* $Z_\in$ und $Z_\notin$. Für jedes Element in U gibt es für jedes Paket in $\mathcal{F}$ eine Zeile der Variablenzeilenart. Sei $u_i \in U$ und $F_j \in \mathcal{F}$ und es gilt $u_i \in F_j$. Dann ist die Zeile $Z_{i \in j} :=$ | X | j | i | . Gilt $u_i \notin F_j$, dann ergibt sich die entsprechende Zeile $Z_{i \notin j} :=$ | $\triangle$ | $\triangle$ | i | . Die letzte Art von Eingabezeilen ist die *Art der Ergänzer*, bezeichnet mit Z_r. Für jedes $F_j \in \mathcal{F}$ gibt es $|\mathcal{F}|$ viele Eingabezeilen dieser Art $Z_{r_j} =$ | X | j | X | . Abschließend wird der Patternmatrix noch eine *Restezeile* für die Ergänzerart | X | $\triangle$ | X | hinzugefügt. Was es genau mit den Ergänzern und den Arten von Eingabezeilen auf sich hat, wird erklärt nachdem die Konstruktion der Patternmatrix gegeben wurde, welche nun folgt.

Die Patternmatrix P besteht ebenfalls aus 3 Spalten, und beinhaltet nur 3 Patternvektorentypen. Der erste Typ ist der sogenannte *Lösungstyp* Q_L. Es gibt h viele solcher Patternvektoren der Form $(\square, \square, \star)$. Eingabezeilen die auf solche Patternvektoren abgebildet werden sind stets von der Art $Z_\in$ oder Z_r. Später wird man an den mittleren Symbolen die Lösung $\mathcal{F}'$ ablesen können, so fern eine existiert.

Der zweite Patternvektorentyp ist der *Elementetyp* Q_E. Es gibt $|U|$ viele Patternvektoren der Form $(\star, \star, \square)$. Die Idee ist, dass auf jeden Patternvektor des Typs Q_E eine Eingabezeile der Art Z_u abgebildet wird. Um die bisher noch nicht festgelegte Anonymitätsschranke einzuhalten, müssen für jedes $u_i \in U$ alle bis auf eine Eingabezeile der Variablenzeilenarten auf Patternvektoren des Typs Q_E abgebildet werden. Es gibt für jedes u_i genau $|\mathcal{F}| + 1$ Eingabezeilen, die mit Z_{u_i} auf einen Patternvektor abgebildet werden können. Somit ergibt sich $k = |\mathcal{F}|$, da für jedes u_i eine Eingabezeile der Art $Z_\in$ übrig bleiben soll. Diese eine verbliebene Eingabezeile $Z_{i \in j}$ wird auf einen Patternvektor des Typs Q_L abgebildet. Können auf die h Patternvektoren des Typs Q_L alle $|U|$ verbliebenen Eingabezeilen des Typs $Z_{i \in j}$ konsistent abgebildet werden, so repräsentieren diese Eingabezeilen eine Lösung.

Sei A ein beliebiger Ausgabezeilentyp, der durch das Abbilden auf einen Patternvektor des Typs Q_L entstanden ist. Dann muss sichergestellt werden, dass auch A die Anonymitätsschranke k einhalten kann. Dies wird mit den Eingabezeilen von der Art der Ergänzer erreicht. Es muss nun noch ein Patternvektor für die übrig gebliebenen Ergänzer bereitgestellt werden. Dieser Patternvektor hat die Gestalt $q_r = (\square, \star, \square)$. Die Restezeile muss auf q_r abgebildet werden und stellt somit sicher, dass q_r nicht zweckentfremdet wird.

Für die Reduktion ist es wichtig, dass genau alle, bis auf $|U|$ viele, der Eingabezeile der Variablenzeilenarten auf Patternvektoren des Typs Q_E abgebildet werden. Dies wird mit Hilfe die Kostenschranke erreicht. Dazu wird ausgenutzt, dass Patternvektoren vom Typ Q_E zwei anstatt ein $\star$-Symbol enthalten und somit doppelt so teuer sind wie die anderen Patternvektorentypen. Es werden also auf jeden Patternvektoren des Typs Q_E jeweils $|\mathcal{F}|$ viele Eingabezeilen abgebildet. Es gibt $|U|$ viele solcher Patternvektoren, also werden durch diese Abbildungen Kosten in Höhe von $|U| \cdot |\mathcal{F}| \cdot 2$ verursacht. Der Faktor zwei folgt aus den zwei $\star$-Symbolen in diesen Patternvektoren. Alle anderen Eingabezeilen werden auf Patternvektoren mit Kosten eins zugewiesen. Es verbleiben $|\mathcal{F}|^2 \cdot |U| + 1$ Eingabezeilen. Die Kostenschranke ergibt sich somit als $s = |U| \cdot |\mathcal{F}| \cdot 2 + |\mathcal{F}|^2 \cdot |U| + 1$.

Tabelle 3.1 beinhaltet ein Schema für die reduzierte Instanz I_P welches nochmals alle Teile der Reduktion übersichtlich aufführt.

Korrektheit: Zu zeigen ist: $I_S \in$ Set Cover $\Leftrightarrow I_P \in$ Pattern Clustering.
$\Rightarrow$: Sei $I_S = (U, \mathcal{F}, h)$ eine Ja-Instanz und sei $\mathcal{F}' = \{F'_1, F'_2, ..., F'_h\}$ die Lösung für I_S. Ohne Beschränkung der Allgemeinheit gelte $|\mathcal{F}| = h$. Es wird jetzt eine Lösung $\varphi : M \to P$ für $I_P = (M, P, k, s)$ konstruiert. Für jedes Paket F'_j aus $\mathcal{F}'$ wird ein Patternvektor ℓ_j vom Lösungstyp Q_L reserviert. Es gibt h Patternvektoren vom Typ Q_L und $|\mathcal{F}'| \leq h$. Somit gibt es für jedes F'_j mindestens einen Patternvektor vom Typ Q_L. Es wird nun

Eingabematrix M

Art	Spalte 1	Spalte 2	Spalte 3	Bedingung	Anzahl
Z_u	$\triangle$	$\triangle$	i	$\forall u_i \in U$	$\lvert U\rvert$
$Z_\in$	X	j	i	$\forall (i,j) u_i \in F_j$	$\lvert U\rvert \cdot \lvert\mathcal{F}\rvert$
$Z_\notin$	$\triangle$	$\triangle$	i	$\forall (i,j) u_i \notin F_j$	
Z_r	X	j	X	$\lvert\mathcal{F}\rvert$ für jedes $F_j \in \mathcal{F}$	$\lvert\mathcal{F}\rvert^2$
z_r	X	$\triangle$	X	-	1

Patternmatrix P

Typ	Spalte 1	Spalte 2	Spalte 3	Anzahl
Q_L	$\square$	$\square$	$\star$	h
Q_E	$\star$	$\star$	$\square$	$\lvert U\rvert$
q_r	$\square$	$\star$	$\square$	1

$$
\begin{aligned}
s = \quad & \lvert U\rvert \cdot \lvert\mathcal{F}\rvert \cdot 2 && \text{für Patternvektoren vom Typ } Q_E \\
+ \quad & \lvert\mathcal{F}\rvert^2 \cdot \lvert U\rvert + 1 && \text{für alle anderen}
\end{aligned}
$$

$$k = \lvert\mathcal{F}\rvert$$

Tabelle 3.1: Diese Tabelle zeigt das Schema für die Konstruktion der Pattern Clustering-Instanz.

begonnen die Zuweisungen von φ zu bestimmen. Sei nun $u_i \in U$ und $u_i \in F'_j$ ein Element für das noch keine Zeile der Form $Z_{i\in x}$ auf einen Patternvektor vom Typ Q_L zugewiesen wurde. So ergibt sich $\varphi(Z_{i\in j}) = \ell_j$. Da $\mathcal{F}'$ eine Set Cover-Lösung ist, wurde für jedes $u_i \in U$ eine Zeile auf einen der h Patternvektoren vom Typ Q_L von φ zugewiesen. Man beachte, wenn $\mathcal{F}'$ nicht minimal war, so kann es Patternvektoren vom Typ Q_L geben die keine Zuweisung erhalten haben.

Es verbleiben in M nun $(\lvert\mathcal{F}\rvert - 1) \cdot \lvert U\rvert$ viele Eingabezeilen die von den Arten $Z_\in$ oder $Z_\notin$ sind und noch nicht von φ zugewiesen wurden. Für jedes $u_i \in U$ gibt es genau $\lvert\mathcal{F}\rvert - 1$ Eingabezeilen der Art $Z_\in$ oder $Z_\notin$. Diese werden zusammen mit der einen Zeile Z_{u_i} der Art Z_u auf einen Patternvektor vom Typ Q_E zugewiesen. Somit wird die Anonymitätsschranke $k = \lvert\mathcal{F}\rvert$ eingehalten. Da es $\lvert U\rvert$ viele Patternvektoren vom Typ Q_E gibt, kann für jedes $u_i \in U$ ein solcher bereit gestellt werden. Es wurden von der Abbildung φ nun insgesamt $\lvert U\rvert \cdot \lvert S\rvert$ Eingabezeilen auf Patternvektoren vom Typ Q_E zugewiesen. Die Zuweisungen auf Q_E haben insgesamt Kosten in Höhe von $\lvert U\rvert \cdot \lvert\mathcal{F}\rvert \cdot 2$ verursacht.

Für die Ausgabezeilentypen, welche durch die Zuweisungen auf die Patternvektoren vom Typ Q_L entstanden sind, ist bisher nicht gesichert, dass sie die Anonymitätsschranke k einhalten. Daher werden alle Eingabezeilen $Z_{r_j} = \boxed{X \mid j \mid X}$ der Art der Ergänzer ebenfalls auf den Patternvektor ℓ_j zugewiesen. Da die Eingabezeile Z_{r_j} genau $\lvert\mathcal{F}\rvert$ mal in M vorkommt ist die Anonymitätsschranke $k = \lvert\mathcal{F}\rvert$ damit erfüllt. Ohne Beschränkung der Allgemeinheit gilt $h < \lvert\mathcal{F}\rvert$. Somit wurde mindestens ein Block von Eingabezeilen

von der Art der Ergänzer bisher nicht von φ zugewiesen. Alle diese verbliebenen Eingabezeilen und auch die eine Restezeile z_r werden auf den Patternvektor q_r zugewiesen. Der Patternvektor q_r erhält somit die Reste der Ergänzer.

Es wurden alle Eingabezeilen auf Patternvektoren zugewiesen und dabei wurde die Konsistenz nicht verletzt. Es bleibt zu zeigen, dass φ die Kostenschranke s einhält. Es wurden genau $|U| \cdot |S|$ Eingabezeilen auf Patternvektoren vom Typ Q_E zugewiesen. Diese Zuweisungen haben insgesamt Kosten in Höhe von $|U| \cdot |\mathcal{F}| \cdot 2$ verursacht. Alle anderen Eingabezeilen wurden auf Patternvektoren vom Typ Q_L oder auf q_r zugewiesen. Diese Zuweisungen haben somit Kosten von $|\mathcal{F}|^2 \cdot |U| + 1$ verursacht. Die Gesamtkosten von φ sind somit genau $|U| \cdot |\mathcal{F}| \cdot 2 + |\mathcal{F}|^2 \cdot |U| + 1$ und somit identisch mit s. Die Kostenschranke wurde eingehalten und φ ist somit eine Lösung für Pattern Clustering.

$\Leftarrow$: Sei $I_P = (M, P, k, s)$ eine Ja-Instanz und sei φ die Lösung für I_P. Es wird jetzt eine Lösung $\mathcal{F}'$ für $I_S = (U, \mathcal{F}, h)$ konstruiert. Da φ eine Lösung ist, muss φ jede Eingabezeile der Art Z_u einem separaten Patternvektor des Typs Q_E zuweisen. Denn diese Patternvektoren sind die einzigen, welche beide $\triangle$-Symbole aus einer Eingabezeile der Art Z_u löschen. Da es genauso viele Eingabezeilen der Art Z_u gibt wie Patternvektoren vom Typ Q_E, stehen die Ausgabezeilentypen die von allen Patternvektoren vom Typ Q_E erzeugen werden fest. Damit diese Ausgabezeilentypen die Anonymitätsschranke $k = |\mathcal{F}|$ einhalten, muss φ auf jeden dieser Ausgabezeilentypen mindestens $|\mathcal{F}| - 1$ Eingabezeilen der Art $Z_\in$ oder $Z_\notin$ zuweisen. Es müssen also mindestens $|U| \cdot |\mathcal{F}|$ Eingabezeilen auf Patternvektoren vom Typ Q_E zugewiesen werden. Um die Kostenschranke einzuhalten, darf φ maximal $|U| \cdot |\mathcal{F}|$ viele Eingabezeilen auf Patternvektoren vom Typ Q_E zuweisen. Somit werden exakt $|U| \cdot |\mathcal{F}|$ Eingabezeilen auf alle Patternvektoren vom Typ Q_E zugewiesen.

Es bleibt für jedes $u_i \in U$ genau eine Eingabezeile der Art $Z_\in$ oder $Z_\notin$ welche nicht auf einen Patternvektoren vom Typ Q_E zugewiesen wird. Diese Eingabezeile ist nicht vom Typ $Z_\notin$, da von einer Lösung diese Eingabezeile nicht auf Patternvektoren vom Typ Q_L oder q_r zugewiesen werden könnte, aber nur noch solche Patternvektoren verbleiben. Die Eingabezeile z_r muss ebenfalls von jeder Lösung auf den Patternvektor q_r zugewiesen werden. Somit müssen die verbliebenen Eingabezeilen der Art $Z_\in$ auf h Patternvektoren vom Typ Q_L zugewiesen werden. Man erinnere sich, dass genau eine Zeile für jedes $u_i \in U$ verblieb. Patternvektoren vom Typ Q_L erhalten die Position des Index des Pakets, somit müssen alle Elemente auf maximal h Pakete abgebildet werden, in denen sie enthalten sind. Dies entspricht genau einer Lösung für Set Cover.

Laufzeit: Es bleibt zu zeigen das es sich um eine parametererhaltende Polynomzeitreduktion handelt. Dazu muss gezeigt werden das die Reduktion korrekt ist, in polynomieller Zeit ausführbar ist und den Parameter polynomiell erhält. Die Korrektheit

wurde bereits gezeigt. Es wird nun gezeigt, wie man die Reduktion in polynomieller Zeit ausführen kann. Für jedes $u \in U$ müssen $|\mathcal{F}|+1$ viele Eingabezeilen erstellt werden, welche sich jeweils in $O(|\mathcal{F}|)$ Zeit errechnen lassen. Dazu kommen $|\mathcal{F}|^2$ Eingabezeilen welche ebenfalls jeweils in $O(|\mathcal{F}|)$ Zeit berechnet werden können. Die Eingabematrix kann somit in polynomieller Zeit berechnet werden. Die Patternmatrix besteht aus $h+|U|+1$ Patternvektoren welche jeweils in konstanter Zeit berechnet werden können. Die Parameter s und k ergeben sich aus einer konstanten Anzahl an Multiplikationen und Additionen und sind somit auch in polynomieller Zeit berechenbar. Somit kann die Laufzeit der Reduktion durch ein Polynom in Abhängigkeit der Größe von I_S beschränkt werden. Als letztes muss gezeigt werden das der neue Parameter p beschränkt ist durch ein Polynom das nur von dem alten Parameter $(|U|, h)$ abhängt. Dies gilt trivial, denn laut Konstruktion ist $p = h + |U| + 1$. Somit handelt es sich um eine parametererhaltende Polynomzeitreduktion.

Zuletzt wird noch gezeigt, dass diese Reduktion für Pattern Clustering-Instanzen mit $m > 3$ übertragen werden kann. Sei $m \in \mathbb{N}$ nun beliebig. An jede Eingabezeile, aus der oben beschriebenen Konstruktion, werden $m-3$ viele Spalten angehängt, die alle das gleiche Symbol enthalten. An jeden Patternvektor müssen nun genauso viele, also $m-3$, Spalten angehängt werden. Diese Spalten enthalten immer ein $\square$-Symbol. Kostenschranke und Anonymitätsschranke bleiben gleich. Der Beweis für eine solche veränderte Instanz geht analog zu dem Beweis für $m = 3$. $\square$

Es bleibt die offene Frage ob Pattern Clustering bezüglich Parameter p in FPT ist oder nicht. Bekannt ist bereits das Pattern Clustering mit einem konstanten p in polynomieller Zeit lösbar ist, und somit in XP liegt [9].

3.2 Reduktion von Constraint Bipartite Vertex Cover

In diesem Abschnitt wird gezeigt, dass Pattern Clustering NP-schwer ist, selbst für den stark eingeschränkten Spezialfall mit Spaltenzahl $m = 2$, ohne Kostenschranke $s = \infty$ und $k = 1$. Dazu wird eine Reduktion von dem NP-schweren Problem Constraint Bipartite Vertex Cover angegeben [18].

> Constraint Bipartite Vertex Cover (CoBiVC)
> **Eingabe:** Ein bipartiter Graph $G = (L \dot\cup R, E)$ und zwei natürliche Zahlen k_ℓ und k_r.
> **Frage:** Gibt es zwei Knotenteilmengen $S_\ell \subseteq L$ und $S_r \subseteq R$ mit $|S_\ell| \leq k_\ell$ und $|S_r| \leq k_r$, sodass für jede Kante $\{x, y\} \in E$ gilt: $(x \in S_\ell) \vee (y \in S_r)$?

Satz 3. Pattern Clustering *mit* $m = 2$, $s = \infty$ *und* $k = 1$ *ist NP-schwer.*

Beweis. Es wird zuerst die Konstruktion beschrieben und als zweites die Korrektheit bewiesen.

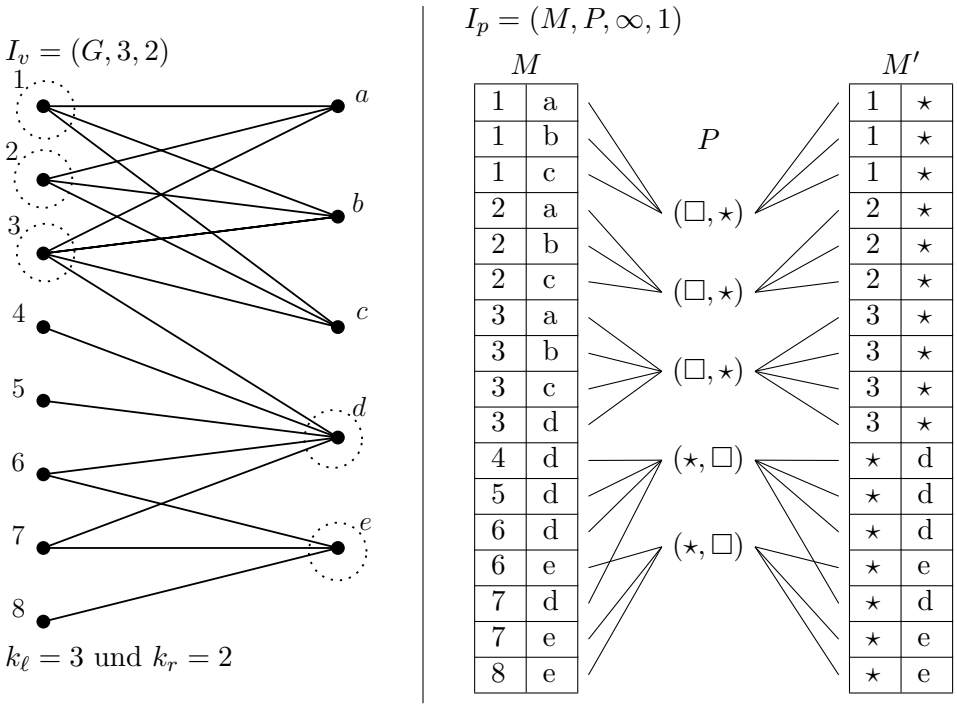
Es wird aus einer Instanz $I_V = (G, k_\ell, k_r)$ für CoBiVC eine Instanz $I_P = (M, P, s, k)$ für Pattern Clustering konstruiert. Jede Kante $\{i, j\}$ mit $i \in L$ und $j \in R$ entspricht in M einer Zeile $\boxed{i \mid j}$. Daher werden hier die Eingabezeilen auch als Kantenzeilen bezeichnet. Es gibt zwei Arten von Patternvektoren. Patternvektoren vom Typ ℓ haben die Form $(\square, \star)$. Patternvektoren vom Typ r haben die Form $(\star, \square)$. In P gibt es k_ℓ Patternvektoren vom Typ ℓ und k_r Patternvektoren vom Typ r. Abschließend sei, wie bereits erwähnt, $s = \infty$ und $k = 1$.

Damit ist die Konstruktion von I_P beschrieben. Es wird nun gezeigt, dass $I_V \in$ CoBiVC $\Leftrightarrow I_P \in$ Pattern Clustering.
$\Rightarrow$: Sei (S_ℓ, S_r) eine Lösung für I_V. Es wird eine Lösung φ für I_P angegeben die zeigt, dass $I_P \in$ Pattern Clustering. Für jeden Knoten i in S_ℓ werden alle Kantenzeilen, welche Kanten entsprechen die zu i inzident sind, auf einen Patternvektor in P vom Typ ℓ zugewiesen. Für jeden Knoten i in S_r werden alle Kantenzeilen, welche Kanten entsprechen die zu i inzident sind und noch nicht zugewiesen wurden, auf einen Patternvektor in P vom Typ r zugewiesen. Die Konsistenz der Zuweisung folgt aus der Konstruktion. Da $S_\ell \cup S_r$ in G ein Vertex Cover ist, ist jede Zeile abgedeckt und somit ist auch φ total.

$\Leftarrow$: Sei M' die Ergebnismatrix von φ. Sei $v_\ell \in M'$ eine Zeile der Gestalt $\boxed{u \mid \star}$, dann ist u in S_ℓ. Sei $v_r \in M'$ eine Zeile der Gestalt $\boxed{\star \mid w}$, dann ist w in S_ℓ. Es ist leicht zu sehen, dass $|S_\ell| \leq k_\ell, |S_r| \leq k_r$ und auch dass alle Kanten in E inzident zu einem Knoten in $S_\ell \cup S_r$ sind. Letzteres folgt aus der Totalität von φ. □

Die Abbildung 3.1 zeigt ein Beispiel für die Reduktion. In diesem Beispiel wird aus einer Instanz $I_v = (G = (L \cup R, E), k_\ell, k_r)$ für CoBiVC eine Instanz $I_p = (M, P, s, k)$ konstruiert. Dabei sei $L = \{1, 2, 3, 4, 5, 6, 7, 8\}, R = \{a, b, c, d, e\}$ und $E = \{\{1, a\}, \{1, b\}, \{1, c\}, \{2, a\}, \{2, b\}, \{2, c\}, \{3, a\}, \{3, b\}, \{3, c\}, \{3, d\}, \{4, d\}, \{5, d\}, \{6, d\}, \{6, e\}, \{7, d\}, \{7, e\}, \{8, e\}\}$. Außerdem $k_\ell = 3$ und $k_r = 2$. Eine Lösung für I_v wäre dann $S_\ell = \{1, 2, 3\}$ und $S_r = \{d, e\}$. Die Instanz I_p ergibt sich wie bereits beschrieben. Die Eingabematrix M entspricht E und P ergibt sich aus k_ℓ vielen $(\square, \star)$ und k_r vielen $(\star, \square)$ Patternvektoren. Außerdem gilt noch $k = 1$ und $s = \infty$. Auf der rechten Seite der Abbildung ist eine Ergebnismatrix für I_p. Man sieht leicht, dass gilt $I_v \in$ CoBiVC $\Leftrightarrow I_p \in$ Pattern Clustering.

Aus Satz 3 lassen sich noch einige Korollare ableiten die im Folgenden aufgelistet sind. Da in der reduzierten Instanz nur zwei verschiedene Typen von Patternvektoren benutzt werden, ergibt sich folgendes Korollar.



1	a
1	b
1	c
2	a
2	b
2	c
3	a
3	b
3	c
3	d
4	d
5	d
6	d
6	e
7	d
7	e
8	e

1	$\star$
1	$\star$
1	$\star$
2	$\star$
2	$\star$
2	$\star$
3	$\star$
3	$\star$
3	$\star$
3	$\star$
$\star$	d
$\star$	d
$\star$	d
$\star$	e
$\star$	d
$\star$	e
$\star$	e

Abbildung 3.1: Diese Abbildung zeigt links den Ursprungsgraph für CoBiVC, rechts daneben die konstruierte Instanz für Pattern Clustering mit $k = 1$ und $s = \infty$, sowie die Ergebnismatrix M'

Korollar 1. Pattern Clustering *ist mit zwei Patternvektorentypen* $p_t = 2$ *bereits NP-schwer.*

Da in für die reduzierten Instanz die Kostenschranke nicht benutzt wird, ergibt sich außerdem ein weiteres Korollar.

Korollar 2. Pattern Clustering *ist für die allgemeine, die binäre und die Clustering-Kostenfunktion NP-schwer.*

Setzt man in der reduzierten Instanz die Kosten aller Patternvektoren auf Null, so ergibt sich folgendes Korollar

Korollar 3. Pattern Clustering *ist mit der allgemeinen Kostenfunktion und* $s = 0$ *bereits NP-schwer.*

Kuo und Fuchs haben die NP-Schwere von CoBiVC mit einer Reduktion von Clique gezeigt [18]. Dabei ergibt sich, dass CoBiVC W[1]-schwer bezüglich dem Minimum von $\{k_\ell, k_r\}$ ist. Somit folgt auch dieses Korollar.

Korollar 4. Pattern Clustering *ist W[1]-schwer bezüglich der Größe des seltensten Patternvektorentyps.*

4 Algorithmen

Im vorhergehenden Kapitel wurde gezeigt das Pattern Clustering sogar für eingeschränkte Spezialfälle NP-schwer ist. Ziel dieses Kapitels ist die Entwicklung von Algorithmen zum Lösen von Pattern Clustering. Wegen der NP-Schwere besteht keine Hoffnung einen Algorithmus mit polynomieller Laufzeit zu finden. Um dennoch einen Algorithmus mit akzeptabler Laufzeit finden zu können, wird der Ansatz der parametrisierten Algorithmik verfolgt.

Zuerst wird ein FPT-Algorithmus bezüglich dem Parameter p für den NP-schweren Spezialfall $k = 1$ und $s = \infty$ vorgestellt. Dieser Algorithmus wird im zweiten Teil dieses Kapitels verbessert, sodass er für allgemeines k gültig ist. Dann wird für einige Kostenfunktionen ein partieller Problemkern gezeigt. Im letzten Teil dieses Kapitel wird ein FPT-Algorithmus bezüglich dem kombinierten Parameter (s, p_t), wobei p_t die Anzahl der Patternvektorentypen ist, für den Spezialfall $k = 1$ vorgestellt.

Zunächst einige Vorbemerkungen. Im Folgenden wird der Begriff der Permutation auf eine Patternmatrix angewandt. Sei π die Permutation einer Patternmatrix P. Das heißt, $\pi(P)$ ist eine veränderte Reihenfolge der Patternvektoren in P. Die Patternvektoren selbst werden dabei nicht verändert, sondern nur die Indizes. Der Index gibt dabei die numerische Position in der Patternmatrix an. Es wird mit $\pi(q_i)$ der i-te Patternvektor in der permutierten Patternmatrix bezeichnet.

Im Folgenden werden unter anderem Algorithmen vorgestellt, welche für den Spezialfall $k = 1$ gelten. Der Spezialfall $k = 1$ heißt, dass auf die Anonymitätsforderung verzichtet wird und eine Ausgabezeile auch einmalig in der Ausgabematrix vorkommen kann. Dieser Spezialfall kann sinnvoll sein wenn Pattern Clustering zum Clustern eingesetzt wird, da die Anzahl an Clustern auch durch die Anzahl der Patternvektoren beschränkt werden kann. Außerdem kann für den Spezialfall $k = 1$, im Gegensatz zudem allgemeinen Fall, angenommen werden, dass Eingabezeilentypen nicht geteilt werden. Es werden alle gleiche Eingabezeilen immer dem gleichen Cluster zugewiesen. Dies zeigt folgende Beobachtung, mit welcher leichter Algorithmen für den Spezialfall $k = 1$ gefunden werden können.

Beobachtung 2. *Wenn es eine Lösung φ für die* Pattern Clustering*-Instanz I gibt, so existiert auch immer eine Lösung φ', bei der alle Eingabezeilen, die vom gleichen Typ sind, auf den gleichen Patternvektor abgebildet werden.*

Beweis. Sei φ eine Lösung für $I = (M, P, s, k = 1)$ und φ bilde Eingabezeilen vom gleichen Typ auf unterschiedliche Patternvektoren ab. Es wird nun eine neue Abbildung konstruiert welche alle Eingabezeilen vom gleichen Typ auf den gleichen Patternvektor abbildet. Die neue Abbildung φ' ergibt sich aus φ indem zunächst $\varphi' = \varphi$ gesetzt und anschließend folgende Prozedur erschöpfend angewendet wird: Seien nun a, b Eingabezeilen vom gleichen Typ, $\varphi(a) = q_a$, $\varphi(b) = q_b$ und $q_a \neq q_b$. Ohne Beschränkung der Allgemeinheit sei q_a der Patternvektor mit der geringeren Anzahl an $\star$-Symbolen. Falls beide Patternvektoren gleich viele $\star$-Symbole enthalten, sei q_a derjenige mit dem kleineren Index, dies wird angenommen um sicherzustellen, dass die Prozedur terminiert. Nun setze $\varphi'(a) = \varphi'(b) = q_a$.

Wurde dieses Verfahren erschöpfend angewandt, so bildet φ' alle Eingabezeilen vom gleichen Typ auf den gleichen Patternvektor ab. Die Kosten der neuen Zuweisung φ' sind kleiner gleich als die Kosten von φ. Denn es wird jede Zuweisung, die geändert wird, durch eine Zuweisung ersetzt, die nicht mehr Kosten verursacht. Die Konsistenz und Totalität von φ' ergibt sich aus der Konsistenz und Totalität von φ. □

Beobachtung 2 kann genutzt werden um einen Algorithmus für den Spezialfall $k = 1$ zu finden. Die nächste Beobachtung vereinfacht die Suche nach Algorithmen für den Spezialfall $k = 1$ und $s = \infty$. Dieser Spezialfall ist sinnvoll, wenn man überprüfen möchte ob es für die ausgewählte Patternmatrix überhaupt eine Lösung gibt oder wenn man um jeden Preis eine hohe durchschnittliche Größe der Cluster erreichen möchte. Letzteres wird durch einen hier vorgestellten Algorithmus erreicht, welcher immer eine minimale Anzahl an Patternvektoren nutzt und somit eine minimale Anzahl an Cluster erzeugt. Für die nächste Beobachtung sind zuerst ein paar Begriffe einzuführen.

Laut Definition 3 passt eine Eingabezeile konsistent zu einem Ausgabezeilentyp, wenn sie, mit Ausnahme der $\star$-Symbole, identisch sind, siehe Definition 3. Sei $I = (M, P, s, k)$ eine Pattern Clustering-Instanz mit einer Lösung φ und sei M' die von φ erzeugte Ausgabematrix. Nun wird der Begriff der *Menge der möglichen Ausgabezeilen einer Eingabezeile* T_v^φ eingeführt. Die Menge von Ausgabezeilentypen T_v^φ enthält alle Ausgabezeilentypen aus M' die zu v konsistent passen. Formal: $T_v^\varphi = \{t_{\text{out}} : t_{\text{out}} \in R(M)' \wedge v$ passt konsistent auf $t_{\text{out}}\}$. Des Weiteren wird gesagt ein Patternvektor q erzeuge einen Ausgabezeilentypen t, wenn es in M eine Eingabezeile z gibt, sodass $\varphi(v) = q$ und t an den gleichen Stellen $\star$-Symbole hat wie q und an allen anderen Stellen identisch mit v ist. Die nun folgende Beobachtung kann genutzt werden um eine Eingabezeile v auf einen beliebigen Patternvektoren q zuzuweisen, wenn man bereits weiß, dass q einen zu v passenden Ausgabezeilentyp erzeugt.

Beobachtung 3. *Sei $I = (M, P, s = \infty, k = 1)$ eine* Pattern Clustering*-Instanz und φ eine Lösung für I. Sei T_v^φ die Menge der möglichen Ausgabezeilen für die Eingabezeile v, welche sich aus φ ergeben. So ist jede Abbildung $\varphi' : M \to P$ mit $\forall v \in R(M)$:*

$v \mapsto q$, wobei $q \in P$ ein beliebiger Patternvektor ist der einen Ausgabezeilentypen aus T_v erzeugt, ebenfalls eine Lösung für I.

Beweis. Zu zeigen ist, dass φ' konsistent ist, wenn φ konsistent ist. Denn für den Spezialfall $k = 0$ und $s = \infty$ ist jede konsistente Abbildung von M in P eine Lösung. Da sich die Menge der Ausgabezeilentypen nicht vergrößert, gibt es für jeden Ausgabezeilentyp t mindestens einen Patternvektor in P, der t erzeugen kann. Die neue Abbildung φ' bildet jede Eingabezeile v auf einen Patternvektor ab, der einen zu v passenden Ausgabezeilentyp erzeugt. Somit ist φ' konsistent. □

Mit Hilfe dieser Beobachtungen werden in den folgenden Abschnitten nun Algorithmen für diese Spezialfälle vorgestellt.

4.1 Parameter p und $k = 1$

Da der Benutzer bei Pattern Clustering Patternvektoren auswählen kann die erheblichen Einfluss auf die Art der Lösung haben, scheint die Anzahl p an Patternvektoren wohl einer der natürlichsten Parameter für dieses Problem zu sein. Eine mögliche Anwendung für eine kleine Anzahl an Patternvektoren ist es wenige große Cluster zu suchen. Bereits bekannt ist, dass Pattern Clustering bei konstantem p in polynomieller Zeit lösbar ist, also in XP bezüglich p liegt [9]. Weiterhin ist bekannt, dass es keinen polynomiellen Problemkern bezüglich p gibt siehe, Satz 2. Bredereck et al. [9] haben die Frage gestellt, ob Pattern Clustering mit Parameter p fixed-parameter tractable ist. Diese Frage wird hier, zumindest für Spezialfälle, positiv beantwortet. Für das allgemeine Problem bleibt die Frage aber offen.

Nachfolgend wird ein FPT-Algorithmus bezüglich der Anzahl an Patternvektoren p für den Spezialfall $k = 1$ und $s = \infty$ vorgestellt. Auch für diesen Spezialfall ist das Problem noch NP-schwer (siehe Satz 3). Ein FPT-Algorithmus bezüglich p ohne Einschränkungen ist bisher nicht bekannt.

Satz 4. Pattern Clustering *mit $k = 1$ und $s = \infty$ kann in $O(p!^2 \cdot m \cdot n^2)$ Zeit gelöst werden und ist somit in FPT bezüglich der Anzahl an Patternvektoren p.*

Beweis. Um den Satz zu beweisen, wird ein FPT-Algorithmus angegeben, welcher sich grob in drei Schritte gliedert.

1. Rate die richtige Permutation π von Patternvektoren, sodass die Patternvektoren absteigend nach der Anzahl ihrer Zuweisungen sortiert sind.

2. Berechne aus der Sortierung der Patternvektoren die Menge von *möglichen* Eingabezeilen, die auf die jeweiligen Patternvektoren abgebildet werden können. Diese Menge ist durch eine Funktion in p beschränkt.
3. Rate für jeden Patternvektor, aus der Menge der möglichen Ausgabezeilentypen, jeweils den richtigen.

Im ersten Schritt werden alle Permutationen der Patternvektoren P berechnet. Im zweiten Schritt wird für jede Permutation π angenommen, sie würde die Patternvektoren nach der Anzahl ihrer Zuweisungen absteigend sortieren. Das heißt es wird angenommen es gelte $\forall i, j(1 \leq i < j \leq p \Rightarrow \#\pi(q_i) \geq \#\pi(q_j))$. Somit kann angenommen werden, dass für den ersten Patternvektor $\pi(q_1)$ die meisten Eingabezeilen zugeordnet werden müssen. Der Ausgabezeilentyp, der durch $\pi(q_1)$ erzeugt wird, muss also mindestens überdurchschnittlich viele Zuweisungen erlauben. Die durchschnittliche Anzahl ist $\frac{n}{p}$ und somit auch die Mindestanzahl an Zuweisung für den ersten Patternvektor. Somit kann es maximal p viele Ausgabezeilentypen für $\pi(q_1)$ geben. Sei $T_{\pi(q_1)} = \{\tau_1, \tau_2, ..., \tau_p\}$ die Menge der möglichen Ausgabezeilentypen für $\pi(q_1)$. Das richtige Element wird geraten. Eine geratene Zahl r_1 mit $1 \leq r_1 \leq p$ gibt an welches Element aus $T_{\pi(q_1)}$ der richtige Ausgabezeilentyp ist. Jeder weitere Patternvektor $\pi(q_i)$ muss einen Ausgabezeilentyp erzeugen, auf den mindestens überdurchschnittlich viele der verbliebenen Eingabezeilen zugewiesen werden können. Davon gibt es jeweils maximal $p - i + 1$ Möglichkeiten. Welche dieser Möglichkeiten die richtige ist, wird wieder mit r_i geraten. Das Raten wird realisiert in dem einmal ein Vektor $R = (r_1, r_2, ..., r_p)$ mit $r_i \leq p - i + 1$ geraten wird. Es gibt $p!$ viele solcher Vektoren.

Es folgt nun die genaue Beschreibung des Algorithmus. Im Schritt i wird für den Patternvektor $\pi(q_i)$ die Menge der Ausgabezeilentypen berechnet, welche überdurchschnittlich viele Zuweisungen erlauben. Der r_i-te Ausgabezeilentyp wird ausgewählt und alle zu ihm passenden Eingabezeilen werden auf $\pi(q_i)$ zugewiesen. Aus Beobachtung 3 folgt, dass für diese Eingabezeilen keine anderen Patternvektoren berücksichtigt werden müssen. In einer Lösung können Eingabezeilen auf einen beliebigen passenden Patternvektor q zugewiesen werden, wenn bekannt ist, dass die Eingabezeilen auf den durch q erzeugten Ausgabezeilentypen passen. Nun werden alle Eingabezeilen, die bereits zugewiesen wurden, aus M gelöscht und der $i + 1$-ste Ausgabezeilentyp wird auf die gleiche Weise bestimmt.

Um FPT-Zeit zu erhalten bleibt noch zu zeigen, dass es für jeden i-ten Patternvektor, mit $1 \leq i \leq p$, nur $p - i + 1$ Ausgabezeilentypen existieren, die überdurchschnittlich viele Zuweisungen erlauben. Aus der Sortierung folgt: $\pi(q_i)$ ist der Patternvektor der von den verbliebenen Patternvektoren die meisten Zuweisungen erhält - also mindestens überdurchschnittlich viele. Daher kommen nur solche Ausgabezeilentypen in Frage, welche $\frac{n_i}{p-i+1}$ Zuweisungen erlauben, wobei n_i die Zahl der verbliebenen Eingabezeilen ist.

Somit gibt es für $\pi(q_i)$ nur $p-i+1$ mögliche Ausgabezeilentypen, die überdurchschnittlich viele Zuweisungen auf q_i erlauben.

Gibt es eine Lösung, so gibt es auch eine Permutation, die die Patternmatrix richtig sortiert. Für diese Permutation gibt es auch einen Vektor R der die richtigen Ausgabezeilentypen rät. Es gibt $p!$ Permutationen und für jede Permutation gibt es $p!$ Vektoren R. Außerdem müssen für jeden Patternvektor in jeder Permutation die möglichen Ausgabezeilentypen bestimmt werden. Dies ist jeweils in $O(m \cdot n^2)$ möglich, da für jede Eingabezeile nur gezählt werden muss mit wie vielen anderen Eingabezeilen sie auf den Patternvektor abgebildet werden kann und sich aus der Eingabezeile dann der Ausgabezeilentyp ergibt. Die Gesamtlaufzeit ist also in $O(p!^2 \cdot m \cdot n^2)$. □

Der Algorithmus in dem Beweis ist nur für den Spezialfall $k = 1$ und $s = \infty$ korrekt. Der Verzicht auf die Kostenschranke ist für diesen Algorithmus notwendig, da immer sofort alle passenden Eingabezeilen auf einen Patternvektor zugewiesen werden. Würde man diese sofortige Zuweisung nicht durchführen, wäre im nächsten Schritt die Eingabematrix zu groß und die Laufzeit wäre keine FPT-Zeit mehr. Der Verzicht auf die Anonymitätsforderung ist notwendig, da der Algorithmus beim festlegen eines Ausgabezeilentyps alle zu diesem Ausgabezeilentyp passenden Eingabezeilen bereits einem Patternvektor fest zuweist. Einige dieser Eingabezeilen könnten aber später notwendig sein um die Größe eines anderen Ausgabezeilentyps zu erhöhen, damit er nicht kleiner als k ist. Die Eingabezeilen können aber umsortiert werden um so die Anonymitätsforderung für alle Ausgabezeilentypen zu erfüllen. Ein solcher Algorithmus wird im nächsten Abschnitt vorgestellt.

4.2 Parameter p und $k \in \mathbb{N}$

In diesem Abschnitt wird ein FPT-Algorithmus für Pattern Clustering mit Parameter p, allgemeinem k und ohne Kostenschranke, also $s = \infty$, vorgestellt. Dieses Resultat ist allgemeiner als das Resultat aus dem Abschnitt 4.1. Allerdings stellt der Abschnitt 4.1 eine Grundidee vor, wie man Pattern Clustering lösen kann. In diesem Abschnitt wird diese Idee aufgegriffen und verfeinert, sodass der Algorithmus auch für beliebige $k \in \mathbb{N}$ angewendet werden kann. Der hier vorgestellte Algorithmus wird als Teilprozedur einen Algorithmus für das Problem Row Assignment nutzen [7, 8]. Der Algorithmus gliedert sich grob in drei Schritte.

1. Rate die richtige Permutation π der Patternvektoren.
2. Bestimme in Abhängigkeit der Reihenfolge der Patternvektoren die Ausgabezeilentypen.

3. Bestimme die Lösung φ mit Row Assignment basierend auf der Ausgabezeilentypen.

Zunächst einige Notationen: $T'_{\text{in}} = \{1, 2, ..., t_{\text{in}}\}$ ist eine Menge natürlicher Zahlen, welche die Eingabezeilentypen T_{in} repräsentiert, wobei $t_{\text{in}} = |T_{\text{in}}|$ die Anzahl an Eingabezeilentypen ist. Die Menge $T'_{\text{out}} = \{1, 2, ..., t_{\text{out}}\}$ repräsentiert die Ausgabezeilentypen T_{out}, wobei $t_{\text{out}} = |T_{\text{out}}|$ die Anzahl an Ausgabezeilentypen ist. Die *Zuweisungskosten* ω_i sind die Kosten der Zuweisung einer Zeile auf den Ausgabezeilentypen t_i. Im Fall der Standartkostenfunktion entspricht ω_i also der Anzahl an $\star$-Symbolen in t_i. Die natürliche Zahl n_j gibt die Häufigkeit an, mit welcher der Eingabezeilentyp t_j in der Eingabematrix vertreten ist. Im Folgenden wird n_j auch Größe von t_j genannt. Die Funktion $a : T'_{\text{out}} \times T'_{\text{in}} \to \{0, 1\}$ gibt an welche Eingabezeilentypen auf welche Ausgabezeilentypen passen und ermöglicht entsprechend nur solche Zuweisungen.

Es wird nun das, für den dritten Schritt wichtige, Problem Row Assignment eingeführt. Row Assignment erhält als Eingabe unter anderem die Menge der Ausgabezeilentypen und berechnet, sofern vorhanden, eine k-anonyme Zuweisung der Eingabezeilen zu den Ausgabezeilentypen. So kann Pattern Clustering auf das Finden der richtigen Ausgabezeilentypen vereinfacht werden. Nun wird Row Assignment formal definiert.

> Row Assignment
> **Eingabe:** Natürliche Zahlen $k, s, \omega_1, \omega_2, ..., \omega_{t_{\text{out}}}$ und $n_1, n_2, ..., n_{t_{\text{in}}}$ mit $\sum_{i=1}^{t_{\text{in}}} n_i = n$, außerdem eine Funktion $a : T'_{\text{out}} \times T'_{\text{in}} \to \{0, 1\}$.
> **Frage:** Existiert eine Funktion $g : T'_{\text{out}} \times T'_{\text{in}} \to \{0, 1, ..., n\}$, sodass gilt:

$$a(i,j) \cdot n \geq g(i,j) \qquad \forall i \in T'_{\text{in}} \forall j \in T'_{\text{out}} \tag{4.1}$$

$$\sum_{i=1}^{t_{\text{in}}} g(i,j) \geq k \qquad \forall j \in T'_{\text{out}} \tag{4.2}$$

$$\sum_{j=1}^{t_{\text{out}}} g(i,j) = n_i \qquad \forall i \in T'_{\text{in}} \tag{4.3}$$

$$\sum_{i=1}^{t_{\text{in}}} \sum_{j=1}^{t_{\text{out}}} g(i,j) \cdot \omega_j \leq s \tag{4.4}$$

Die Lösung g von Row Assignment bildet von natürlichen Zahlen auf natürliche Zahlen ab, sieht man die natürlichen Zahlen des Definitionsbereichs als die Indizes der Eingabezeilen und die natürlichen Zahlen des Wertebereichs als die Indizes der Ausgabezeilentypen, so kann man sich leicht den Patternvektor herleiten auf den die entsprechende Eingabezeile abgebildet wird. Somit kann man g als Zuweisung sehen. Dabei wird durch die Gleichungen und Ungleichungen sichergestellt dass g eine Lösung für Pattern Clustering ist. Ungleichung 4.1 stellt sicher, dass g nur so zuweist wie a es erlaubt. Somit

ist g konsistent, wenn a nur konsistente Zuweisungen erlaubt. Die Ungleichung 4.2 sichert das jedem Ausgabezeilentyp mindestens k Eingabezeilen zugewiesen werden. Damit Gleichung 4.3 erfüllt ist, muss jede der Eingabezeilen auf einen Ausgabezeilentyp zugewiesen wurden sein und somit muss auch g total sein. Die letzte Ungleichung 4.4 hält die Kosten der Zuweisung unter s. Dies ist bei dem hier betrachteten Spezialfall $s = \infty$ auf den ersten Blick ohne Nutzen. Allerdings kann durch die Wahl eines geeigneten s eine kostengünstigere Zuweisung mittels Row Assignment gefunden werden. Row Assignment kann in $O(t_{\text{in}}^3 \cdot \log(t_{\text{in}}))$ Zeit gelöst werden [8].

Die Eingaben für Row Assignment sind dabei: k, s und $n_1, n_2, ..., n_{t_{\text{in}}}$, welche sich leicht aus der Eingabe ablesen lassen. Außerdem das erhält Problem a und $\omega_1, \omega_2, ..., \omega_{t_{\text{out}}}$, welche sich nicht so leicht ergeben. Es wird nun angenommen die Menge der Ausgabezeilentypen wäre bekannt. Dann können $\omega_1, \omega_2, ..., \omega_{t_{\text{out}}}$ einfach aus $T_{\text{out}} = \{t_1, t_2, ...t_{t_{\text{out}}}\}$ ablesen werden, da für ω_i nur die $\star$-Symbole in t_i gezählt werden müssen. In der Menge T'_{out} sind einfach nur die natürlichen Zahlen von 1 bis t_{out} enthalten. Auch die Funktion a kann aus T_{out} berechnet werden, indem $a(t_o, t_i) = 1$ gesetzt wird genau dann, wenn t_i auf t_o passt. Also muss man nur T_{out} berechnen. Damit ergeben sich alle Eingaben für eine Row Assignment-Instanz. Diese kann dann in polynomieller Zeit gelöst werden und erzeugt somit eine Lösung für die eigentliche Pattern Clustering-Instanz. Das heißt, falls T_{out} bekannt ist, so kann eine Lösung für Pattern Clustering in polynomieller Zeit gefunden werden [8].

Im Folgenden wird nun gezeigt wie man, für den Schritt 2, eine Menge von Ausgabezeilentypen $T_{\text{out}} = \{t_1, t_2, ...t_{t_{\text{out}}}\}$ in Abhängigkeit der Reihenfolge der Patternvektoren finden kann. Dieses Verfahren ist als Pseudocode in Abbildung 4.1 dargestellt. Seien $\{q_1, q_2, ..., q_p\}$ die Patternvektoren in P und $\{z_1, z_2, ..., z_n\}$ die Zeilen in der Eingabematrix M. Eine Eingabezeile passt konsistent auf einen Ausgabezeilentyp genau dann, wenn sie sich nur an den Stellen der $\star$-Symbole unterscheiden, siehe Definition 3 und die Zeilen 23 bis 32 in Abbildung 4.1 für den entsprechenden Pseudocode. Würde die erste Zeile aus M, also z_1, auf q_1 zugewiesen, so ergibt sich der Ausgabezeilentyp t_1. Dies wird in den Zeilen 6 bis 12 ausgeführt. Nach dem Bestimmen dieses Ausgabezeilentyps werden alle Eingabezeilen, die auch auf diesen Ausgabezeilentypen passen, aus M gelöscht. Siehe dazu die Zeilen 13 bis 19. Wurde ein Patternvektor noch nicht für die Bildung eines Ausgabezeilentyps benutzt, so spricht man von einem *freien* Patternvektor. Den nächsten Ausgabezeilentyp gewinnt man nun, indem die nächste Eingabezeile z_i auf den obersten freien Patternvektor von P zugewiesen wird. Mit obersten Patternvektor ist der Patternvektor mit dem kleinsten Index gemeint. Im Anschluss werden wieder alle zu dem neuen Ausgabezeilentypen passende Eingabezeilen gelöscht. Die Reihenfolge von P hat hierbei starken Einfluss auf das gefundene T_{out} und im Allgemeinen führt die ursprüngliche Reihenfolge nicht zu dem richtigen T_{out}. Daher wird in Schritt 1 die richtige Permutation geraten, das heißt dieses Verfahren wird einfach für alle Permutationen π von P durchgeführt.

```
1: procedure GetTout(π(P), M))
2:     Matrix M' = M;
3:     Integer i = 1;
4:     Menge T_out = ∅;
5:     while M' ≠ ∅ do
6:         for j = 1 bis m do
                   ▷ In dieser Schleife wird mit Hilfe des i-ten Patternvektor der i-te
       Ausgabezeilentyp bestimmt
7:             if π(q_i)[j] = □ then
8:                 t_i[j] = M'[1][j];
9:             else
10:                t_i[j] = ⋆;
11:            end if
12:        end for
13:        T_out.add(t_i);
14:        i++;
15:        for j = 1 bis n do
               ▷ In dieser Schleife werden alle konsistent auf t_i passenden Eingabezeilen
       gelöscht
16:            if Passt(t_i, z_j) then
17:                M'.remove(z_j);
18:            end if
19:        end for
20:    end while
21: end procedure
22:
23: procedure Passt(t, z)
24:    for i = 1 bis m do
25:        if t[i] ≠ ⋆ then
26:            if t[i] ≠ z[i] then
27:                return(false);
28:            end if
29:        end if
30:    end for
31:    return(true);
32: end procedure
```

Abbildung 4.1: Pseudocode für das Verfahren zum Gewinnen von einem T_{out} in Abhängigkeit der Permutation der Patternvektoren.

Beobachtung 4. *Mit dem Verfahren, welches in Abbildung 4.1 dargestellt wird, kann die Menge von Ausgabezeilen abhängig von der Permutation von P in $O(m \cdot n^2)$ Schritten gefunden werden.*

Die Laufzeit ergibt sich wie folgt: Für jede Eingabezeile muss überprüft werden ob es bereits eine passende Ausgabezeile gibt und es gibt maximal n viele verschiedene Ausgabezeilen.

Wenn man dieses Verfahren so lange anwenden kann, dass für jede Eingabezeile ein Ausgabezeilentyp gefunden wird, so wird diese Permutation π von P eine *gute Permutation* genannt. Wenn es für das von π erzeugte T_{out} eine Lösung für Row Assignment gibt, so wird π *sehr gute Permutation* genannt.

Lemma 1. *Wenn eine Instanz für* Pattern Clustering $I = (M, P, k, s = \infty)$ *eine Lösung hat, so existiert auch eine sehr gute Permutation π von P.*

Beweis. Wenn es eine Lösung gibt, so gibt es auch immer eine Lösung, die eine minimale Anzahl an Patternvektoren benutzt. Sei P_μ eine solche minimale Menge von Patternvektoren, sodass es für $I' = (M, P_\mu, k, s = \infty)$ eine Lösung gibt. Sei φ' eine Lösung für I' und T'_{out} die Menge der Ausgabezeilentypen von φ'. Mit T_i wird die Menge an Ausgabezeilentypen bezeichnet, die auf v_i, für alle v_i mit $1 \leq i \leq n$, passen. Außerdem sei $t_{i,j} \in T_i$ der j-te Ausgabezeilentyp auf welchen v_i passt. Da P_μ minimal ist, gibt es für jeden Ausgabezeilentyp $t \in T'_{\text{out}}$ mindestens eine Eingabezeile, die nur auf t passt.

Ein Patternvektor passt zu einem Ausgabezeilentyp, wenn beide an genau den gleichen Stellen $\star$-Symbole haben. Eine sehr gute Permutation ergibt sich wie folgt: Sei q_i ein Patternvektor der zu dem Ausgabezeilentypen $t_{1,1} \in T_1$ passt. Somit ergibt sich der erste Patternvektor $\pi(q_1) = q_j$. Lösche nun die oberste Eingabezeile und alle Zeilen, die auf $t_{1,1}$ passen. Sei nun z_i der oberste Ausgabezeilentyp in der so veränderten Eingabematrix und t der oberste Ausgabezeilentyp in T_i. Sei q ein Patternvektor der zu t passt. Dann ist q der nächste Patternvektor in der Permutation. Dies wird fortgesetzt bis alle Patternvektoren in P_μ von π zugewiesen sind. Dies ist immer möglich, da es zu jedem Patternvektor in P_μ eine Eingabezeile gibt, die nur auf einen Ausgabezeilentyp passt. Wurden alle Patternvektoren zugewiesen, so sind auch keine weiteren Eingabezeilen übrig. Dies folgt aus der Tatsache das φ' eine Lösung ist.

Die Permutation π von P_μ ist eine sehr gute Permutation. Wenn man die Reihenfolge von $\pi(P_\mu)$ beibehält und die verbleibenden Patternvektoren in $P \setminus P_\mu$ in beliebiger Reihenfolge anhängt, so erhält man eine sehr gute Permutation für I. □

Satz 5. Pattern Clustering *mit $k \in \mathbb{N}$ und $s = \infty$ ist in $O(p! \cdot m \cdot n^2 \cdot t_{\text{in}}^3 \cdot \log t_{\text{in}})$ Zeit lösbar und somit in FPT bezüglich Parameter p, der Anzahl an Patternvektoren.*

M

1	2
1	1
1	1

$\pi_1(P)$
$(\star, \star)$
$(\square, \square)$

M'_1

$\star$	$\star$
$\star$	$\star$
$\star$	$\star$

$\pi_2(P)$
$(\square, \square)$
$(\star, \star)$

M'_2

1	2
$\star$	$\star$
$\star$	$\star$

M'_{opt}

$\star$	$\star$
1	1
1	1

Tabelle 4.1: Abgebildet ist eine Pattern Clustering-Instanz mit der Eingabematrix M und die zwei Permutationen der Patternmatrix mit den jeweils resultierenden Lösungen M'_1 und M'_2. Außerdem die optimale Ausgabematrix M'_{opt}.

Beweis. Sei $I = (M, P, k, s = \infty)$ eine Instanz für Pattern Clustering. Dann kann für jede der $p!$ vielen Permutationen π von P in polynomieller Zeit überprüft werden ob π eine sehr gute Permutation ist. Siehe Abbildung 4.2 für den entsprechenden Pseudocode. Diese Überprüfung muss die Ausgabezeilentypen bestimmen; dies ist in $O(m \cdot n^2)$ Schritten möglich, siehe Beobachtung 4. Dann muss die Funktion $a : T'_{\text{out}} \times T'_{\text{in}} \mapsto \{0, 1\}$ bestimmt werden, indem einfach $a(t_o, t_i) = 1$ gesetzt wird genau dann wenn t_i auf t_o passt. Dies ist in $O(m \cdot n^2)$ Schritten möglich, da für jede Eingabezeile geprüft wird auf welche Ausgabezeilentypen sie passt und es maximal n viele Ausgabezeilentypen gibt (siehe Zeilen 9 bis 13). Außerdem werden die ω_i berechnet, was in $O(m \cdot n)$ Schritten möglich ist, da einfach die $\star$-Symbole gezählt werden müssen, dies entspricht der Zeile 6. Abschließend wird mit Row Assignment überprüft, ob es eine Lösung gibt und es sich somit um eine sehr gute Permutation handelt. Sind alle Permutationen keine sehr guten Permutationen, so folgt aus Lemma 1, dass es keine Lösung für I gibt. Die Gesamtlaufzeit ist $O(p! \cdot m \cdot n^2 \cdot t_{\text{in}}^3 \cdot \log t_{\text{in}})$, wobei $O(t_{\text{in}}^3 \cdot \log t_{\text{in}})$ die Laufzeit von Row Assignment ist. Da Row Assignment für die bereits festgelegten Ausgabezeilentypen eine kostenoptimale Lösung findet, wird für jede minimale Menge von Patternvektoren $P_\mu \in P$ sodass $I' = (M, P_\mu, k, s = \infty)$ eine Lösung hat, eine kostenoptimale Lösung für I gefunden. □

Der Algorithmus aus Satz 5 ist nur korrekt für den Spezialfall $s = \infty$. Diese Einschränkung ist notwendig, da zwar Row Assignment kostenoptimal auf die Ausgabezeilentypen zuweist, aber es nicht garantiert werden kann, dass die richtigen Ausgabezeilentypen gefunden wurden. Ein möglicher Worst Case wäre zum Beispiel, wenn die oberste Eingabezeile aus M in jeder Lösung auf einen Patternvektor zugewiesen werden muss, der nur $\star$-Symbole enthält. Dann würde jedes T_{out} wie hier beschrieben berechnet, nur einen Ausgabezeilentyp enthalten und somit würde die Ausgabematrix nur $\star$-Symbole enthalten. Die Tabelle 4.1 zeigt ein Beispiel, in welchem mit dem hier beschriebenen Algorithmus keine kostenoptimale Lösung gefunden wird. Es besteht die Hoffnung die Eingabematrix M deterministisch, nach einem noch unbekannten Kriterium, zu sortieren, sodass der Algorithmus eine Optimale Lösung findet.

```
1: procedure PatternClustering(M,P, s, k)
2:     for alle π(P) do
3:         T_out = GetTout(π(P), M);                ▷ siehe Abbildung 4.1 für GetTout
4:         Menge Ω = ∅;          ▷ In Ω werden zur besseren Übersicht die ω_i gesammelt
5:         for alle t_i ∈ T_out do
6:             ω_i = Anzahl⋆(t_i);          ▷ ω_i entspricht der Anzahl an ⋆-Symbolen in t_i
7:             Ω.add(ω_i);
8:         end for
9:         for alle t_i ∈ T_out do
10:            for alle z_j ∈ T_in do
11:                a[i][j] = Passt(t_i, z_j);                ▷ siehe Abbildung 4.1 für Passt
12:            end for
13:        end for
14:        if RowAssignment(M, T_in, T_out, Ω, a) then
15:            return(true);
16:        end if
17:    end for
18:    return(false);
19: end procedure
```

Abbildung 4.2: Pseudocode zu dem Algorithmus aus dem Beweis von Satz 5

Der Algorithmus aus Satz 5 findet immer eine Lösung die nur das Minimum an Patternvektoren benötigt. Somit ist dieser Algorithmus kostenoptimal für die Clustering-Kostenfunktion. Man erinnere sich: Die Clustering-Kostenfunktion fragt nach einer Lösung die maximal s Ausgabecluster beziehungsweise Ausgabezeilentypen erzeugt. Somit ist s eine obere Schranke für die benutzten Patternvektoren in einer Lösung. Man kann den Algorithmus so abändern, dass sich die Laufzeit speziell für die Clustering-Kostenfunktion verbessern lässt. Für die Clustering-Kostenfunktion gilt stets $s \leq p$

Korollar 5. Pattern Clustering *mit der Clustering-Kostenfunktion kann man lösen in* $O(\binom{p}{s} \cdot s! \cdot m \cdot n^2 \cdot t_{\text{in}}^3 \cdot \log t_{\text{in}})$ *Zeit und ist somit FPT bezüglich dem Parameter* p.

Beweis. Zunächst wird für $s \leq p$ gezeigt, dass stets $p! \geq \binom{p}{s} \cdot s!$ gilt.

$$\begin{aligned} & p! \geq \binom{p}{s} \cdot s! \\ \Leftrightarrow \quad & p! \geq \frac{p!}{s! \cdot (p-s)!} \cdot s! \\ \Leftrightarrow \quad & p! \geq \frac{p!}{(p-s)!} \\ \Leftrightarrow \quad & (p-s)! \geq 1 \end{aligned}$$

Nun werden die Änderungen des Algorithmus beschrieben: Anstatt alle Permutationen zu überprüfen, werden lediglich s viele Patternvektoren geraten, welche in der Lösung vorkommen. Man kann $\binom{p}{s}$ viele Mengen der Größe s aus P auswählen. Nun müssen nur noch für die jeweils s vielen ausgewählten Patternvektoren alle $s!$ vielen Permutationen überprüft werden. Die Laufzeit für die Überprüfung ändert sich dabei nicht, es wird also $O(m \cdot n^2 \cdot t_{\text{in}}^3 \cdot \log t_{\text{in}})$ Zeit benötigt um eine Permutation zu überprüfen. Wenn es für eine Menge von Patternvektoren der Größe s eine Lösung gibt, so wird sie auch gefunden. Dies folgt aus Lemma 1. Gibt es eine Lösung, die weniger als s viele Patternvektoren benutzt, so wird diese auch gefunden. Denn dann gibt es eine Obermenge dieser Patternvektoren, welche unter anderen diese wenigen richtigen Patternvektoren sehr gut permutiert. □

4.3 Partieller Problemkern für spezielle Kostenfunktionen

In diesem Abschnitt werden partielle Problemkerne [2] für Pattern Clustering mit den Kostenfunktionen aus Abschnitt 2.3 vorgestellt. Im Gegensatz zu einem Problemkern, beschränkt ein partieller Problemkern nicht die gesamte Eingabegröße durch eine Funktion abhängig von dem Parameter, sondern nur eine oder mehrere Dimensionen. In dieser Arbeit wird die Anzahl an Spalten m durch eine Funktion von der Anzahl an Patternvektoren p begrenzt. Ein partieller Problemkern wird durch die erschöpfende Anwendung von *Datenreduktionsregeln* erreicht. Eine Datenreduktionsregel reduziert in polynomieller Zeit die Größe der Eingabe. Eine Datenreduktionsregel R heißt *korrekt* wenn für jede Instanz I auf die R angewandt werden kann gilt, dass die reduzierte Instanz entscheidungsäquivalent zu I ist.

Sei $I = (M, P, k, s)$ eine Instanz für Pattern Clustering dabei sei die Patternmatrix $P \in \{\star, \square\}^{p \times m}$. In diesem Abschnitt wird gezeigt, wie man für die allgemeine Kostenfunktion, die binäre Kostenfunktion und die Clustering-Kostenfunktion Spalten ver-

schmelzen kann, sodass $m \leq 2^p$ gilt und man somit einen partiellen Kernel erhält. Dazu einige Notationen.

Notation: Die i-te Spalte mit $1 \leq i \leq m$ von M wird mit v_i^M bezeichnet, analog die i-te Spalte von P mit v_i^P. Zwei Spalten v_i^P und v_j^P sind in P *identisch* wenn sie in jeder Zeile in P identisch sind. Mit $R(M)$ wird die Multimenge aller Eingabezeilen bezeichnet, wie bereits definiert wurde. Zwei Spalten v_i^M und v_j^M werden *verschmolzen*, indem v_j^M und v_j^P gelöscht werden und in jeder Zeile $z_\ell \in R(M)$ an der Stelle i das Symbol $M[i][\ell]$ ersetzt wird durch $M[\ell][i] \circ M[\ell][j]$. Dabei ist $\circ$ die Konkatenation.

Zuerst wird die Datenreduktionsregel *Regel 1* vorgestellt, danach wird gezeigt, dass sie zu einem partiellen Problemkern führt, abschließend wird gezeigt, dass sie für die Kostenfunktionen aus Abschnitt 2.3 korrekt ist.

Regel 1: Sind v_i^P und v_j^P identisch in P, so verschmelze v_i^M und v_j^M.

Lemma 2. *Sei $I = (M, P, k, s)$ mit $M \in \Sigma^{n \times m}$ und $P \in \{\star, \square\}^{p \times m}$. Sei weiterhin die Instanz $I_r = (M', P', k, s)$, jene Instanz die sich nach erschöpfender Anwendung von Regel 1 auf I ergibt. Sei $M' \in \Sigma'^{n \times m'}$, dann gilt $m' \leq 2^p$.*

Beweis. Dieses Lemma wird mit einem Widerspruchsbeweis bewiesen. Sei $m' > 2^p$. Da die neue Patternmatrix $P' \in \{\star, \square\}^{p \times m'}$ eine Matrix über einen binären Alphabet ist, kann es in P nur 2^p viele verschiedene Spalten geben. Es müssen also 2 Spalten identisch sein, also kann Regel 1 angewandt werden, dies ist ein Widerspruch zu der Aussage Regel 1 wurde erschöpfend angewandt. □

Nun wird die Korrektheit für die drei Kostenfunktionen aus Abschnitt 2.3 gezeigt.

Lemma 3. *Regel 1 ist korrekt für* Pattern Clustering *mit der binären Kostenfunktion.*

Beweis. Sei $I = (M, P, k, s)$ eine Instanz von Pattern Clustering für die Regel 1 angewendet werden kann und sei $I' = (M', P', k, s)$ die Instanz die sich nach der Anwendung der Regel 1 ergibt. Zu zeigen ist: Es gibt eine Lösung φ für I genau dann, wenn eine Lösung φ' für I' existiert. Bei der Anwendung der Regel wird die Reihenfolge der Zeilen in P und M nicht verändert. Es behalten also alle Eingabezeilen und Patternvektoren ihre alten Indizes. Eine Lösung φ' für I' bildet *indexerhaltend* zu φ ab, das heißt $\varphi'(z_i') = q_j' \Leftrightarrow \varphi(z_i) = q_j$ mit $z_i' \in M'$, $q_j' \in P'$, $z_i \in M$ und $q_j \in P$. Nachfolgend wird gezeigt, dass die Zuweisung φ' konsistent ist genau dann wenn φ konsistent ist.

Seien ohne Beschränkung der Allgemeinheit v_1^P und v_2^P die beiden Spalten auf welche Regel 1 angewandt wurde. Da $v_1^P = v_2^P$ gilt, wurden bei jeder Abbildung von φ entweder die ersten beiden Symbole in jeder Eingabezeile gelöscht oder keine der beiden. Daher ist φ' konsistent genau dann wenn gilt $M'[i][1] = M'[j][1] \Leftrightarrow (M[i][1] = M[j][1] \wedge M[i][2] = M[j][2])$. Dies folgt aus der Eineindeutigkeit der Konkatenation.

Die Zuweisung φ' ist k-anonym genau dann wenn φ k-anonym ist, dies folgt aus der Konstruktion. Es bleibt zu zeigen, dass die Kosten der Zuweisungsfunktionen gleich sind. Bei der binären Kostenfunktion kostet eine Zuweisung auf einen Patternvektor immer gleich viel, es sei den er enthält nur □-Symbole. Beim Verschmelzen kann aus einem Patternvektor ohne ⋆-Symbol kein Patternvektor entstehen, welcher mindestens ein ⋆-Symbol enthält. Es können auch nicht alle ⋆-Symbole aus einem Patternvektor beim verschmelzen gelöscht werden. Da immer zwei ⋆-Symbole gebraucht werden um ein ⋆-Symbol zu löschen. □

Korollar 6. *Regel 1 ist korrekt für* Pattern Clustering *mit der allgemeinen Kostenfunktion.*

Beweis. Sei $I = (M, P, k, s)$ eine Instanz von Pattern Clustering für die Regel 1 angewendet werden kann, sei $I' = (M', P', k, s)$ die Instanz die sich nach der Anwendung der Regel 1 ergibt. Zu zeigen ist: Es gibt eine Lösung φ für I genau dann, wenn eine Lösung φ' für I' existiert. Dabei wird φ' genauso konstruiert wie in dem Beweis zu Lemma 3. Auch die Argumentation für die Konsistenz ergibt sich analog. Es bleibt zu zeigen, dass die Kosten der beiden Zuweisungsfunktionen gleich sind. Da die Kosten für jede Zuweisung bei jedem Patternvektor nicht verändert wurden und jeder Patternvektor genauso viele Zuweisungen in φ' erhält wie in φ, ist die Gleichheit der Kosten gegeben. □

Korollar 7. *Regel 1 ist korrekt für* Pattern Clustering *mit der Clustering-Kostenfunktion.*

Beweis. Sei $I = (M, P, k, s)$ eine Instanz von Pattern Clustering für die Regel 1 angewendet werden kann, sei $I' = (M', P', k, s)$ die Instanz die sich nach der Anwendung der Regel 1 ergibt. Zu zeigen ist: Es gibt eine Lösung φ für I genau dann, wenn eine Lösung φ' für I' existiert. Dabei wird φ' genauso konstruiert wie in dem Beweis zu Lemma 3. Auch die Argumentation für die Konsistenz ergibt sich analog. Es bleibt zu zeigen, dass die Kosten der beiden Zuweisungsfunktionen gleich sind. Da nur Kosten für jeden Patternvektor auf den mindestens eine Zeile zugewiesen wurde und φ' auf genauso viele Patternvektoren zuweist wie φ, ist die Gleichheit der Kosten gegeben. □

Aus dem Lemma 2 und Lemma 3, beziehungsweise den Korollaren 6 und 7, ergibt sich folgender Satz.

Satz 6. *1. Für* Pattern Clustering *mit der binären Kostenfunktion gibt es immer eine Entscheidungsäquivalente Instanz, welche weniger als 2^p Spalten hat, wobei p die Anzahl an Patternvektoren ist. Diese Instanz kann in polynomieller Zeit berechnet werden.*

2. Für Pattern Clustering *mit der allgemeinen Kostenfunktion gibt es immer eine Entscheidungsäquivalente Instanz, welche weniger als 2^p Spalten hat. Diese Instanz kann in polynomieller Zeit berechnet werden.*

3. Für Pattern Clustering *mit der Clustering-Kostenfunktion gibt es immer eine Entscheidungsäquivalente Instanz, welche weniger als 2^p Spalten hat. Diese Instanz kann in polynomieller Zeit berechnet werden.*

4.4 Parameter s

In diesem Abschnitt wird eine Idee für einen FPT-Algorithmus bezüglich dem Parameter s gezeigt. Man erinnere sich; s ist die Kostenschranke und p_t ist die Anzahl an Patternvektorentypen. Diese Idee wird im zweiten Teil angewandt um einen FPT-Algorithmus bezüglich dem kombinierten Parameter (s, p_t) für Pattern Clustering mit der Standardkostenfunktion und dem Spezialfall $k = 1$ zu zeigen. Ob sich die Idee auch für den nicht kombinierten Parameter s anwenden lässt ist bisher nur eine Vermutung. In diesem Abschnitt wird der Begriff des *kostenlosen* Patternvektors benutzt. Ein kostenloser Patternvektor ist ein Patternvektor ohne $\star$-Symbole. Ein *kostenpflichtiger* Patternvektor oder kurz *Bezahlvektor* ist ein Patternvektor, der nicht kostenlos ist.

Beobachtung 5. *Gibt es keinen kostenlosen Patternvektor, so hat die Instanz I maximal s Eingabezeilen oder I besitzt keine Lösung.*

Beweis. Jede Eingabezeile muss auf irgendeinen Patternvektor abgebildet werden und jeder Patternvektor löscht mindestens ein Zeichen, daher hat jede totale Zuweisung mindestens Kosten n, wobei n die Anzahl an Eingabezeilen ist. Somit ist $n \leq s$. □

Mit einem ähnlichen Argument folgt trivial die nächste Beobachtung.

Beobachtung 6. *Gibt es für eine Instanz eine Lösung φ mit Kosten höchstens s, so bildet φ maximal auf s viele Bezahlvektoren ab.*

Im Folgenden wird mit $\#g$ die Anzahl an kostenlosen Patternvektoren bezeichnet und mit $\#t$ wird die Anzahl an unterschiedlichen Eingabezeilentypen in einer Instanz I. Gelte $\#g \geq \#t$ und $k = 1$, so existiert eine einfache Lösung, denn jeder Eingabezeilentyp kann auf einen kostenlosen Patternvektor abgebildet werden. Daher kann angenommen werden, dass $\#t - \#g > 0$ ist. Man beachte: Für jede Ja-Instanz gilt $\#t - \#g \leq s$, dies folgt aus Beobachtung 5.

Beobachtung 7. *Für jede* Pattern Clustering*-Instanz I mit $k = 1$ und $\#t - \#g > 0$ gilt. Gibt es eine Funktion φ die $\#t - \#g$ viele Eingabezeilentypen vollständig in die Menge der Bezahlvektoren abbildet, dann ist I eine Ja-Instanz.*

Beweis. Die verbliebenen $\#g$ vielen Eingabezeilentypen können vollständig auf die $\#g$ vielen kostenlosen Patternvektoren abgebildet werden. □

Im Folgenden wird es das Ziel sein, mit Hilfe einer so genannten Kostenmenge $\#t - \#g$ viele Eingabezeilentypen mit Kosten s vollständig in die Menge der Bezahlvektoren abzubilden. Wenn dies in FPT-Zeit getan werden kann, so ist das eigentliche Problem auch in FPT-Zeit gelöst. Dazu wird im Folgenden das neue Konzept der *Kostenmenge* vorgestellt. Eine Kostenmenge H ist eine Multimenge von geordneten Paaren $h = (c, \ell)$ mit $c \in \mathbb{N}$ und $\ell \in \mathbb{N}^s$. Das geordnete Paar h wird *Kostenpaar* genannt. Es repräsentiert im Vektor ℓ die Größen von unbekannten Eingabezeilentypen und die Kosten c, die diese beim Abbilden auf einen ebenfalls noch unbekannten Patternvektor q in einer Zuweisung verursachen. Der Vektor $\ell = (\ell_1, \ell_2, ...\ell_s)$ enthält außerdem noch die Information, wieviele Eingabezeilenzeilentypen auf q abgebildet werden. Dies ist die Anzahl an Nichtnulleinträgen in ℓ. Die ℓ_i seien stets absteigend sortiert.

Im Folgenden wird stets angenommen, dass $k = 1$ gilt. Somit kann Beobachtung 5 benutzt werden. Daher kann weiter angenommen werden, dass Eingabezeilentypen stets komplett auf einen Patternvektor abgebildet werden. Somit kann man an ℓ ablesen wieviele Eingabezeilen insgesamt auf q abgebildet werden. Dies entspricht genau der Summe $x = \sum_{i=1}^{s} \ell_i$. Somit ist $\frac{c}{x}$ gleich der Anzahl an $\star$-Symbolen in q. Bei jeder Kostenmenge einer Lösung ist c stets so bestimmt, dass $\frac{c}{x}$ eine ganze Zahl ist. Hier wird nochmal kurz zusammengefasst welche Informationen ein Kostenpaar $h = (c, \ell = (\ell_1, ..., \ell_s))$ enthält:

1. Die Anzahl $\star$-Symbole in dem Patternvektor q und die Gesamtkosten der Abbildung auf q,
2. die Anzahl an verschiedenen Eingabezeilentypen und jeweils deren Größe und
3. die Gesamtzahl an Eingabezeilen, die auf q abgebildet werden.

Eine *Realisierung* r für ein Kostenpaar (c, ℓ) ist ein Patternvektor q und eine Menge von Eingabezeilentypen, welche die Größen haben, die von ℓ vorgegeben werden, und auf q

mit exakt Kosten c abgebildet werden können. Eine *realisierende Abbildung* für eine Kostenmenge ist eine Menge von Realisierungen für Kostenpaare, sodass jedes Kostenpaar konfliktfrei zu den anderen realisiert wird. Konfliktfrei heißt dabei, dass jede Eingabezeile und jeder Patternvektor nur einmal benutzt wird. Mit der Kostenmenge H ist es nun möglich Restriktionen an realisierende Abbildungen zu stellen. Diese Restriktionen umfassen die Anzahl an Eingabezeilentypen und die Kosten, die die Abbildung ϕ verursacht. Somit kann nun untersucht werden, ob es eine Abbildung ϕ gibt, die mit Kosten s eine Anzahl von $\#t - \#g$ vielen Eingabezeilentypen vollständig in die Menge der Bezahlvektoren abbildet. Zunächst wird ϕ mit Hilfe von H auf die Menge der Bezahlvektoren beschränkt. Dies geschieht indem für jedes $h = (c, \ell) \in H$ jeweils $c > 0$ gelten soll. Als nächstes werden die Gesamtkosten beschränkt. Dies passiert einfach mit:

$$s \geq \sum_{i=1}^{|H|} c_i$$

wobei $h_i = (c_i, l_i)$ das i-te Kostenpaar ist. Zuletzt wird die Anzahl der Eingabezeilentypen festgelegt. Die Anzahl aller Nichtnulleinträgen in allen Kostenpaaren muss zusammen exakt[1] $\#t - \#g$ sein. Eine solche Kostenmenge wird *restriktiv* genannt.

Nun wird gezeigt, dass es maximal $f(s)$ viele verschiedene solcher restriktiven Kostenmengen gibt: Eine solche Kostenmenge hat maximal s Kostenpaare. Jedes Kostenpaar besteht aus $s + 1$ Elementen. Jedes dieser Elemente kann jeweils nur $s + 1$ verschiedene Werte annehmen. Daher ist die Kardinalität der Menge aller restriktiven Kostenmengen durch eine Funktion in s beschränkt. Aus Beobachtung 7 folgt für $k = 1$, dass es eine Lösung φ gibt, wenn es eine realisierende Abbildung ϕ gibt die H realisiert. Gibt es für I eine Lösung, so gibt es eine Kostenmenge für die es eine realisierende Abbildung gibt. Sei H eine solche Kostenmenge. Nun wird beschrieben, wie eine realisierende Abbildung für eine Kostenmenge gefunden werden kann. Eine Realisierung für ein Kostenpaar (c, ℓ) ist ein Patternvektor q und eine Menge von Eingabezeilentypen, welche die Größen haben die von ℓ vorgegeben werden und auf q mit exakt Kosten c abgebildet werden können.

Ziel ist es nun, eine realisierende Abbildung ϕ zu finden. Dazu wird nun eine Menge von Realisierungen für alle Kostenpaare gesammelt und in dieser Menge wird nach einer konfliktfreien realisierenden Abbildung gesucht. Diese Menge soll dabei nur so groß sein, dass man Brute-Force Methoden anwenden kann um eine realisierende Abbildung zu finden. Diese Menge nennen wir *Realisierungskandidatenmenge* oder *Realisierungsmenge*. Im folgenden Abschnitt wird eine Realisierungsmenge vorgestellt.

[1] Gibt es eine Lösung die mit Kosten maximal s mehr als $\#t - \#g$ Eingabezeilentypen komplett auf Bezahlvektoren zuweist, so gibt es auch immer eine Lösung die exakt $\#t - \#g$ Eingabezeilentypen mit Kosten maximal s auf Bezahlvektoren zuweist. Dies folgt aus $k = 1$ und der damit verbundenen beliebigen Größe der Ausgabezeilentypen.

Parameter (s, p_t). Da es bisher nicht gelungen ist eine Realisierungsmenge zu finden die nur durch eine Funktion in s beschränkt ist, wird nur eine vorgestellt die durch eine Funktion in s und p_t beschränkt ist, wobei p_t die Anzahl an Patternvektortypen ist. Zunächst wird hierzu das Konzept der Realisierungstabelle eingeführt. Im Anschluss wird eine zu große Realisierungstabelle vorgestellt. Später wird erklärt, wie man sie durch eine Funktion in s und p_t beschränken kann.

Eine *Realisierungstabelle* $T_{h,t}$ für ein Kostenpaar h und einen Patternvektortyp t ist wie folgt aufgebaut. Es gibt $1 + |\ell|$ viele Spalten. Für jedes ℓ_i gibt es eine Spalte L_i und die zusätzliche Spalte entspricht einem Ausgabezeilentyp. In den Zeilen L_i stehen jeweils Eingabezeilentypen der Größe ℓ_i, die zu dem Ausgabezeilentyp b passen. Ein Eingabezeilentyp passt zu einem Ausgabezeilentyp b, wenn sie sich nur an der Stelle der $\star$-Symbole in b unterscheiden. Der Ausgabezeilentyp muss wiederum zu dem Patternvektorentyp t passen, das heißt beide müssen an den gleichen Stellen $\star$-Symbole haben. Für jeden Ausgabezeilentyp werden mehrere Zeilen mit Eingabezeilentypen gesammelt. Zeilen die den gleichen Ausgabezeilentyp haben, werden *Zeilenblock* genannt. Ein Zeilenblock ist *absolut vollständig* wenn in jeder Spalte L_i alle zu b passenden Eingabezeilentypen der Größe ℓ_i stehen. Eine Tabelle $T_{h,t}$ ist *absolut vollständig* wenn alle möglichen Ausgabezeilentypen des entsprechende Zeilenblock absolut vollständig sind. Ein Ausgabezeilentyp ist für h und t *möglich*, wenn es für jedes ℓ_i mindestens einen Eingabezeilentyp gibt. Gibt es für ein ℓ_i weniger passende Eingabezeilentypen als für ℓ_j, dann sind einige Zeilen in Spalte L_i leer. Die erste Spalte enthält die Ausgabezeilentypen. Eine Zeile, in der alle Spalten außer der ersten Spalte leer sind, gibt es in einer Realisierungstabelle nicht.

Beobachtung 8. *Eine absolut vollständige Realisierungstabelle $T_{h,t}$ für h und t hat folgende Eigenschaften.*

- *Sind ℓ_i und ℓ_j nummerisch gleich, so enthält auch die Spalte L_i die gleichen Eingabezeilentypen wie Spalte L_j.*
- *Um eine Realisierung für h zu finden, können folgendermaßen paarweise unterschiedliche Eingabezeilentypen aus $T_{h,t}$ kombiniert werden. Aus jeder Spalte muss genau ein Eingabezeilentyp ausgewählt werden und alle ausgewählten Eingabezeilentypen müssen zu dem gleichen Zeilenblock gehören.*
- *In $T_{h,t}$ sind unterschiedliche Zeilenblöcke stets disjunkt in ihren Eingabezeilentypen.*
- *Die Größe von $T_{h,t}$ ist im Allgemeinen nicht durch eine Funktion in p_t und s beschränkt.*

Im Folgenden werden wir zeigen wie man eine solche Realisierungstabelle reduziert wird, mit dem Ziel ihre Größe durch eine Funktion in p_t und s zu beschränken. Eine solche reduzierte Realisierungstabelle wird *s-vollständig* genannt. Gesucht ist nun eine s-vollständige Realisierungstabelle mit der man die Instanz lösen kann.

Definition 10. *Eine reduzierte Realisierungstabelle $T_{h,t}$ heißt* s-vollständig *wenn sie folgende Modifikationen erfüllt:*

1. *Anstatt aller möglichen Ausgabezeilentypen werden zunächst nur solche Ausgabezeilentypen in die Realisierungstabelle aufgenommen, für die es mindestens eine konfliktfreie Realisierung gibt. Das heißt es gibt wenigstens eine Kombination mit jeweils einem Eingabezeilentyp aus jeder Spalte, die außerdem paarweise verschieden sind.*
2. *Für jede Realisierungstabelle $T_{h,t}$ werden nur maximal s verschiedene Zeilenblöcke gesammelt. Existieren weniger als s Zeilenblöcke, so werden alle in $T_{h,t}$ aufgenommen.*
3. *Für jedem Zeilenblock gibt es in jeder Spalte maximal s verschiedene Eingabezeilentypen. Existieren weniger als s Eingabezeilentypen in einer Spalte, so werden in dieser Spalte alle aufgenommen.*

Eine Realisierungsmenge A, die für jedes $h \in H$ und jeden Patternvektortypen t in P eine solche s-vollständig Realisierungstabelle enthält, wird auch s-vollständig genannt. Die Realisierungsmenge A ist durch eine Funktion in s und p_t beschränkt. Man beachte zunächst es gilt stets $|H| \leq s$. Es gibt $|H| \cdot p_t$ viele Realisierungstabellen $T_{h,t}$, und jede hat $s + 1$ Spalten und maximal s^2 viele Zeilen (s Zeilenblöcke mit je s Zeilen). Die Tabellen 4.2 bis 4.6 zeigen ein Beispiel für eine absolut vollständige Realisierungsmenge und für eine s-vollständige Realisierungstabelle.

Spalte 1	1	2	3	4	1	2	3	4	5	6	1	2	3	4	5	6	0
Spalte 2	a	a	a	a	b	b	b	b	b	b	c	c	c	c	c	c	o
Anzahl	1	1	1	1	2	2	2	2	2	2	2	2	2	2	2	2	3

Tabelle 4.2: Transponierte Eingabematrix M. Die ersten beiden Zeilen entsprechen den Spalten von M und geben die Eingabezeilentypen an, die dritte Zeile gibt die Häufigkeit des entsprechenden Typs an.

Patternmatrix P

$(\square, \star)$	$\times\ 1$
$(\star, \square)$	$\times\ 1$
$(\star, \star)$	$\times\ 1$
$(\square, \square)$	$\times\ 12$

$s = 10$, $\#t = 17$,
$\#g = 12$, $\#t - \#g = 5$

Kostenmenge H

c	ℓ
2	(2)
2	(1, 1)
6	(1, 2)

Tabelle 4.3: Hier ist die Patternmatrix abgebildet, ebenfalls mit der Anzahl des entsprechen Patternvektorentyps, weiterhin abgebildet sind sonstige Eingabeparameter und eine Kostenmenge für die es eine realisierende Abbildung ϕ gibt. Es wurde auf die Darstellung der Nulleinträge in den ℓ's verzichtet.

$h_1 = (2, (2)) \implies$ der Patternvektor enthält ein $\star$-Symbol

$t_1 = (\square, \star)$

T_{h_1,t_1}

	b	ℓ_1
$b_1 =$	1 ⋆	1 b
		1 c
$b_2 =$	2 ⋆	2 b
		2 c
$b_3 =$	3 ⋆	3 b
		3 c
$b_4 =$	4 ⋆	4 b
		4 c
$b_5 =$	5 ⋆	5 b
		5 c
$b_6 =$	6 ⋆	6 b
		6 c

$t_2 = (\star, \square)$

T_{h_1,t_2}

	b	ℓ_1
$b_1 =$	⋆ b	1 b
		2 b
		3 b
		4 b
		5 b
		6 b
$b_2 =$	⋆ c	1 c
		2 c
		3 c
		4 c
		5 c
		6 c

Tabelle 4.4: Aus h_1 ergibt sich die Anzahl an $\star$-Symbolen, da ein Eingabezeilentyp der Größe zwei auf einen Patternvektor q mit Kosten zwei abgebildet werden soll. Also muss q ein $\star$-Symbol enthalten. Es gibt zwei Patternvektorentypen in P die ein $\star$-Symbol enthalten. Daher ergeben sich zwei Realisierungstabellen. Man beachte, dass $\boxed{0}\boxed{\star}$ in T_{h_1,t_1} kein zulässiger Ausgabezeilentyp ist, da die Eingabezeile nicht die Größe zwei hat, ebenso sind $\boxed{\star}\boxed{a}$ und $\boxed{\star}\boxed{o}$ in T_{h_1,t_2}.

$h_2 = (2, (1, 1)) \implies$ der Patternvektor enthält ein $\star$-Symbol

$t_1 = (\square, \star)$

T_{h_2,t_1}

	b	ℓ_1	ℓ_2
~~$b_1 =$~~	~~1 ⋆~~	~~1 a~~	~~1 a~~
~~$b_2 =$~~	~~2 ⋆~~	~~2 a~~	~~2 a~~
~~$b_3 =$~~	~~3 ⋆~~	~~3 a~~	~~3 a~~
~~$b_4 =$~~	~~4 ⋆~~	~~4 a~~	~~4 a~~

$t_2 = (\star, \square)$

T_{h_2,t_2}

	b	ℓ_1	ℓ_2
$b_1 =$	⋆ a	1 a	1 a
		2 a	2 a
		3 a	3 a
		4 a	4 a

Tabelle 4.5: Aus h_2 ergibt sich wieder die Anzahl an $\star$-Symbolen und es resultieren zwei Realisierungstabellen. Alle Zeilen in T_{h_2,t_1} sind aber nicht in der reduzierten s-vollständigen Realisierungstabelle enthalten, da sie gegen das Kriterium (1) aus Definition 10 verstoßen.

$h_3 = (6, (1, 2)) \implies$ der Patternvektor enthält zwei $\star$-Symbole

$t_3 = (\star, \star)$

T_{h_3,t_3}

	b	ℓ_1	ℓ_2
$b_1 =$	⋆ ⋆	1 a	1 b
		2 a	1 c
		3 a	2 b
		4 a	2 c
			3 b
			3 c
			4 b
			4 c
			5 b
			5 c
			~~6 b~~
			~~6 c~~

Tabelle 4.6: Die Eingabezeilentypen 6 b und 6 c sind nicht in der s-vollständigen Realisierungstabelle, da es bereits s andere Eingabezeilentypen gibt.

Lemma 4. *Wenn es für die Kostenmenge H eine realisierende Abbildung gibt, so gibt es auch in der s-vollständigen Realisierungsmenge A für H eine realisierende Abbildung.*

Beweis. Sei $\phi = \{r_1, r_2, r_3...\}$ die realisierende Abbildung für H. Man sortiere die Elemente aus R so, dass alle r_j mit $j \in \mathbb{N}$ und $i \leq j \leq |R|$ nicht in A sind und alle r_j mit

$j < i$ in A sind. Somit ist r_i die Realisierung mit dem kleinsten Index, die nicht in H ist. Sei h das Kostenpaar welches von r_i realisiert wurde und sei $r_i = (q_i, l_i)$ und q_i sei vom Typ t. Es wird nun gezeigt, dass in der Realisierungstabelle $T_{h,t}$ für h und t eine konfliktfreie Realisierung existiert. Es ist bekannt, dass r_i nicht in $T_{h,t}$ ist. Also sind in $T_{h,t}$ nicht alle möglichen Realisierungen. Dafür gibt es zwei Möglichkeiten.

Erstens: Ist $T_{h,t}$ mit Ausgabezeilentypen voll (enthält also exakt s Ausgabezeilentypen), so gibt es mindestens einen Ausgabezeilentyp mit der h realisiert werden kann und der nicht im Konflikt mit einer Realisierung r_j mit $j < i$ steht. Denn alle r_j mit $j < i$ können zusammen maximal s Eingabezeilentypen benutzen. Man beachte außerdem, dass die Zeilenblöcke in $T_{h,t}$ jeweils disjunkt in der Eingabezeilen sind. Daher muss es einen Ausgabezeilentyp in $T_{h,t}$ geben der nicht zu den Realisierungen r_j mit $j < i$ im Konflikt steht.

Zweitens: Die zweite Möglichkeit warum r_i nicht in $T_{h,t}$ ist, ist dass es einen vollen Ausgabezeilentyp b gibt. Das heißt es gibt einen Zeilenblock, in dem mindestens ein Eingabezeilentyp e von r_i nicht in dem Zeilenblock ist. Sei dieser Eingabezeilentyp von der Größe ℓ_i. Dann gibt es in der Spalte L_i genau s verschiedene Eingabezeilentypen. Das heißt mindestens ein anderer Eingabezeilentyp kann an den Platz von e treten und so h realisieren ohne dass sich der Ausgabezeilentyp ändert. Gibt es mehrere fehlende Eingabezeilentypen, kann diese Argumentation wiederholt angewandt werden.

Eine weitere Möglichkeit für das Fehlen einer Realisierung in A gibt es nicht. Jetzt ist r_i realisiert und i wird um Eins erhöht, bis alle Kostenpaare realisiert wurden. Somit ist die Existenz einer Realisierung in A bewiesen.

□

Satz 7. Pattern Clustering *mit $k = 1$ kann in $O(s^{6s+5} \cdot p_t^{s+1} \cdot m \cdot n)$ Zeit gelöst werden und ist somit in FPT bezüglich dem kombinierten Parameter (s, p_t) und in XP bezüglich Parameter s, wobei s die Kostenschranke und p_t die Anzahl Patternvektortypen ist.*

Beweis. Sei I eine Instanz von Pattern Clustering mit $k = 1$. Gibt es eine Lösung φ für I, so gibt es auch eine Kostenmenge H für diese Lösung. Für H gibt es eine s-vollständige Realisierungsmenge, in der laut Lemma 4 eine realisierende Abbildung gefunden werden kann. Die Menge aller H für I ist durch eine Funktion in s beschränkt, wie im folgenden gezeigt wird. Da die tatsächlichen Kosten von φ nicht bekannt sind, muss für jedes $s' \leq s$ eine Kostenmenge geraten werden. Jede Kostenmenge hat maximal s' Kostenpaare $h = (c, \ell)$, wobei die Summe aller c gleich s' ist. Es gibt $O(B_{s'})$ viele Möglichkeiten s' als Summe von natürlichen Zahlen darzustellen, dabei ist $B_{s'}$ die s'-te Bellsche Zahl. Der Vektor ℓ enthält natürliche Zahlen, deren Summe gleich c ist, also gibt es für jedes c insgesamt $O(B_c)$ verschiedene solcher Vektoren ℓ. Im folgenden

werden B_c und $B_{s'}$ jeweils durch B_s nach oben abgeschätzt. Es gilt $B_s \in O(s^s)$, somit gibt es $O(s \cdot s^s \cdot s^s)$ viele verschiedene Kostenmengen. Für jede Kostenmenge gilt $|H| \leq (s+1) \cdot s$. Die s-vollständige Realisierungstabelle für H kann in $O(s \cdot p_t \cdot s \cdot s^2 \cdot n \cdot m)$ Zeit gefunden werden, da für jedes der maximal s Kostenpaare in H für jeden Patternvektortyp maximal s Zeilenblöcke und für jeden Zeilenblock maximal s^2 viele zueinander passende Eingabezeilentypen gefunden werden müssen. Die Größe jeder s-vollständige Realisierungstabelle liegt in $O(s \cdot p_t \cdot s \cdot s^2)$, denn für jedes der maximal s Kostenpaare werden, für jeden der p_t Patternvektoren, maximal s viele Zeilenblöcke mit jeweils maximal s^2 vielen Eingabezeilentypen gesammelt. Für jedes der maximal s vielen Kostenpaare muss eine Realisierung aus der s-vollständigen Realisierungstabelle gefunden werden. Somit gibt es $O((s \cdot p_t \cdot s \cdot s^2)^s)$ in Frage kommende Realisierungen, die alle auf Konfliktfreiheit überprüft werden. Die Gesamtlaufzeit ergibt sich somit als $O((s \cdot s^s \cdot s^s) \cdot (s \cdot p_t \cdot s \cdot s^2 \cdot n \cdot m) \cdot ((s \cdot p_t \cdot s \cdot s^2)^s))$, vereinfacht also $O(s^{6s+5} \cdot p_t^{s+1} \cdot m \cdot n)$. Somit kann die Instanz in FPT-Zeit entschieden werden. □

Die Laufzeit des hier vorgestellten Algorithmus ist jenseits von praktikablen Anwendungen. Außerdem ist der Parameter s in der Regel nicht klein. Es handelt sich also vorrangig um ein Klassifizierungsresultat. Die Zugehörigkeit von Pattern Clustering zu FPT bezüglich Parameter s und p ist bereits bekannt [9]. Parameter p_t ist aber ein echt stärkerer Parameter als p. Denn Pattern Clustering ist NP-schwer bei konstantem p_t. Dies folgt aus Korollar 1. Allerdings ist Pattern Clustering bezüglich p in XP [9]. Der hier gezeigte Algorithmus ist nur korrekt für den Spezialfall $k = 1$. Diese Einschränkung ist notwendig, da bei beliebigen k die Eingabezeilentypen möglicherweise auf mehrere Patternvektoren verteilt werden. Die Möglichkeit des Aufteilens von Eingabezeilentypen wäre aber mit diesem Ansatz nicht in FPT-Zeit realisierbar. Vor allem da möglicherweise auch Eingabezeilentypen geteilt werden müssten, welche eigentlich auf kostenlose Patternvektoren abgebildet werden. Davon kann es sehr viele geben. Eine Abdeckung aller Möglichkeiten wäre nicht durch eine Funktion in (s, p_t) beschränkt. Die Tabelle 4.7 zeigt ein Beispiel bei dem auf das Aufteilen von Eingabezeilentypen notwendig ist um die optimale Lösung zu finden.

M

1	2	1
2	2	3
1	3	3
1	2	3
1	2	3
1	2	3

P

$(\square, \square, \star)$
$(\square, \star, \square)$
$(\star, \square, \square)$
$(\star, \star, \star)$
$k = 2$

M'_1

1	2	$\star$
1	2	$\star$
1	2	$\star$
1	2	$\star$
$\star$	$\star$	$\star$
$\star$	$\star$	$\star$

M'_{opt}

1	2	$\star$
1	2	$\star$
1	$\star$	3
1	$\star$	3
$\star$	2	3
$\star$	2	3

Tabelle 4.7: Abgebildet ist links eine PATTERN CLUSTERING-Instanz mit der Eingabematrix M, die Patternmatrix P und $k = 2$. Daneben ist eine Ausgabematrix M'_1, ohne aufteilen der Eingabezeilentypen, und die optimale Ausgabematrix M'_{opt} zu sehen.

5 Ausblick

Pattern Clustering ist ein Problem mit vielen potentiellen Anwendungen. Die wohl wichtigste Anwendung ist das durch den Benutzer geführte Anonymisieren. Dies ermöglicht das Vermeiden von unerwünschten Löschungen in der Ausgabematrix. Den vielfältigen Anwendungen steht die hohe Komplexität des Problems gegenüber. Es wurde mit einer neuen und einfacheren Reduktion gezeigt, dass das Problem selbst ohne Anonymitätsforderung, ohne Kostenschranke und nur zwei Spalten in der Eingabematrix NP-schwer ist. In dieser Arbeit wurde die parametrisierte Komplexität untersucht. Für den Spezialfall $k = 1$ wurde fixed-parameter-tractability bezüglich dem kombinierten Parameter (s, p_t), Anzahl Löschungen und Anzahl der unterschiedlichen Patternvektorentypen gezeigt. Es besteht die Vermutung, dass die Idee des in Abschnitt 4.4 gezeigten Algorithmus sich modifizieren lässt, um so für den Spezialfall $k = 1$ fixed-parameter-tractability bezüglich des Parameters s alleine nachzuweisen. In einer typischen Pattern Clustering-Instanz ist die Anzahl an gelöschten Zeichen s meistens nicht klein. Dies macht s zu einem tendenziell schlechten Parameter. Die Anzahl der Patternvektoren p wird von dem Benutzer selber gewählt und kann so nach Anwendung entsprechend klein sein. Dies macht p zu einem interessanten Parameter. In dieser Arbeit wurde, für den Spezialfall $s = \infty$, fixed-parameter-tractability bezüglich des Parameters p nachgewiesen. Es bleibt die offene und interessante Frage, ob Pattern Clustering fixed-parameter-tractable bezüglich des Parameters p ist.

Um die Vermutungen über die Stärken und Schwächen der Parameter zu bestätigen oder ihnen zu widersprechen, würden sich empirische Untersuchungen der Datenlage in Real-World-Instanzen anbieten. Eine solche Untersuchung könnte auch bisher unbekannte Strukturen in den Daten aufzeigen und so zu neuen interessanten Parametern führen. Für erste experimentelle Versuche würde sich der Algorithmus aus Abschnitt 4.2 anbieten. Es wird vermutet, dass die Average-Case Laufzeit deutlich besser als die angegebene Worst-Case Laufzeit ist und dieser Algorithmus, bei einer guten Implementierung, Lösungen in akzeptabler Zeit findet.

Neben dem parametrisierten Ansatz bieten sich andere algorithmische Methoden an, um Pattern Clustering zu lösen. Die im Abschnitt 4.1 und 4.2 vorgestellten Algorithmen für den Parameter p haben jeweils im ersten Schritt eine Permutation der Patternvektoren geraten. Daher liegt die Vermutung nahe, ein randomisierender Algorithmus könnte in diesen Fällen eine bessere Laufzeit bieten. Ein weiterer Umstand der

für Randomisierung spricht ist die Tatsache, dass es für jeden Patternvektor q, der von einer Lösung benutzt wird, mindestens k Eingabezeilen existieren, die mit q den richtigen Ausgabezeilentyp erzeugen.

In vielen typischen Anwendungen für das Anonymisieren sind die Eingabeinstanzen groß. Bei einer großen vorhandenen Datenmenge kann möglicherweise auf die Forderung nach einer optimalen Lösung verzichtet werden. Somit würde sich approximierende Algorithmen anbieten, welche zwar keine optimale Lösung garantieren, aber eine polynomielle Laufzeit haben.

Pattern Clustering bietet wohl ein weites Feld von Anwendungsmöglichkeiten und ist somit ein lohnenswertes Forschungsgebiet.

Literaturverzeichnis

[1] Gagan Aggarwal, Rina Panigrahy, Tomás Feder, Dilys Thomas, Krishnaram Kenthapadi, Samir Khuller, and An Zhu. Achieving anonymity via clustering. *ACM Transactions on Algorithms*, 6(3):49:1–49:19, 2010. 5

[2] Nadja Betzler, Jiong Guo, Christian Komusiewicz, and Rolf Niedermeier. Average parameterization and partial kernelization for computing medians. *J. Comput. Syst. Sci.*, 77(4):774–789, 2011. 40

[3] Jeremiah Blocki and Ryan Williams. Resolving the complexity of some data privacy problems. *37th International Colloquium on Automata, Languages and Programming (ICALP'10)*, 6199:393–404, 2010. 5

[4] Hans L. Bodlaender, Rodney G. Downey, Michael R. Fellows, and Danny Hermelin. On problems without polynomial kernels. *Journal of Computer and System Sciences*, 75(8):423–434, 2009. 12

[5] Hans L. Bodlaender, Stéphan Thomassé, and Anders Yeo. Kernel bounds for disjoint cycles and disjoint paths. *Theoretical Computer Science*, 412(35):4570–4578, 2011. 13

[6] Paola Bonizzoni, Gianluca Della Vedova, and Riccardo Dondi. Anonymizing binary and small tables is hard to approximate. *Journal of Combinatorial Optimization*, 22(1):97–119, 2011. 5

[7] Paola Bonizzoni, Gianluca Della Vedova, Riccardo Dondi, and Yuri Pirola. Parameterized complexity of k-anonymity: Hardness and tractability. In *Proceedings of the 21st International Workshop On Combinatorial Algorithms (IWOCA'10)*, volume 6460 of *Lecture Notes in Computer Science*, pages 242–255. Springer, 2010. 5, 33

[8] Robert Bredereck, André Nichterlein, Rolf Niedermeier, and Geevarghese Philip. The effect of homogeneity on the complexity of k-anonymity. In *Proceedings of the 18th International Symposium on Fundamentals of Computation Theory (FCT'11)*, volume 6914 of *Lecture Notes in Computer Science*, pages 53–64. Springer, 2011. 5, 33, 35

[9] Robert Bredereck, André Nichterlein, Rolf Niedermeier, and Geevarghese Philip. Pattern-guided data anonymization and clustering. In *Proceedings of the 36th Mathematical Foundations of Computer Science (MFCS'11)*, volume 6907 of *Lecture*

Notes in Computer Science, pages 182–193. Springer, 2011. 2, 4, 5, 13, 14, 20, 25, 31, 51

[10] Liming Cai, Jianer Chen, Rodney G. Downey, and Michael R. Fellows. Advice classes of parameterized tractability. *Annals of Pure and Applied Logic*, 84(1):119–138, 1997. 12

[11] Alina Campan, Traian Marius Truta, and Nicholas Cooper. User-controlled generalization boundaries for p-sensitive k-anonymity. In *Proceedings of the 25th Symposium on Applied Computing (SAC'10)*, pages 1103–1104. ACM, 2010. 5

[12] Michael Dom, Daniel Lokshtanov, and Saket Saurabh. Incompressibility through colors and IDs. In *Proceedings of the 36th International Colloquium on Automata, Languages and Programming ICALP(09)*, volume 5555 of *Lecture Notes in Computer Science*, pages 378–389. Springer, 2009. 20, 21

[13] Rod G. Downey and Michael R. Fellows. *Parameterized Complexity*. Springer, 1999. 12

[14] Patricia A. Evans, Todd Wareham, and Rhonda Chaytor. Fixed-parameter tractability of anonymizing data by suppressing entries. *Journal of Combinatorial Optimization*, 18(4):362–375, 2009. 5

[15] Jörg Flum and Martin Grohe. *Parameterized Complexity Theory*. Springer, 2006. 12

[16] Michael R. Garey and David S. Johnson. *Computers and Intractability; A Guide to the Theory of NP-Completeness*. W. H. Freeman & Co., New York, NY, USA, 1990. 10

[17] Richard M. Karp. Reducibility among combinatorial problems. In *Complexity of Computer Computations*, The IBM Research Symposia Series, pages 85–103. Plenum Press, New York, 1972. 11, 20

[18] Sy-Yen Kuo and W. Kent Fuchs. Efficient spare allocation for reconfigurable arrays. *IEEE Design and Test of Computers*, 4:24–31, 1987. 25, 27

[19] Ashwin Machanavajjhala, Daniel Kifer, Johannes Gehrke, and Muthuramakrishnan Venkitasubramaniam. ℓ-diversity: Privacy beyond k-anonymity. *Transactions on Knowledge Discovery from Data*, 1(1):3:1–3:52, 2007. 5

[20] Adam Meyerson and Ryan Williams. On the complexity of optimal k-anonymity. In *PODS*, pages 223–228. ACM, 2004. 5

[21] John Miller, Alina Campan, and Traian Marius Truta. Constrained k-anonymity: Privacy with generalization boundaries, 2008. 5

[22] Rolf Niedermeier. *Invitation to Fixed-Parameter Algorithms*. Oxford University Press, 2006. 12

[23] Latanya Sweeney. Achieving k-anonymity privacy protection using generalization and suppression. *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, 10(5):571–588, 2002. 5

[24] Latanya Sweeney. k-anonymity: A model for protecting privacy. *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, 10(5):557–570, 2002. 5

[25] Traian Marius Truta and Bindu Vinay. Privacy protection: p-sensitive k-anonymity property. In *Proceedings of the 22nd International Conference on Data Engineering Workshops (ICDE'06)*, page 94. IEEE Computer Society, 2006. 5

Selbständigkeitserklärung

Hiermit erkläre ich, dass ich die Diplomarbeit selbständig verfasst habe und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe. Alle Stellen der Arbeit, die wörtlich oder sinngemäß aus Veröffentlichungen oder aus anderweitigen fremden Äußerungen entnommen wurden, sind als solche kenntlich gemacht. Ferner erkläre ich, dass die Arbeit noch nicht in einem anderen Studiengang als Prüfungsleistung verwendet wurde.

Jena, den 16. Dezember 2011 Thomas Köhler