Beweisbar Gesetzmäßigkeiten in Daten finden und ausnutzen[1]

Stefan Neumann[2]

Abstract: Im letzten Jahrzehnt gab es immensen Fortschritt und Wachstum in den Bereichen der Künstlichen Intelligenz und des Maschinellen Lernens. Ermöglicht wurde diese Entwicklung durch spezialisierte neue Hardware, die immer größere Verfügbarkeit von Daten und Durchbrüche bei der Entwicklung von Algorithmen, die *Gesetzmäßigkeiten in Daten finden und ausnutzen*. Obwohl wir uns in der Praxis täglich vom großen Erfolg dieser Algorithmen überzeugen können, ist unser theoretisches Verständnis von ihnen jedoch weiterhin eingeschränkt. Allerdings wären formale Garantien für diese Algorithmen wünschenswert, weil sie wichtige Einblicke in die Stärken und die Grenzen dieser Algorithmen bieten. Diese Dissertation verkleinert die Kluft zwischen Theorie und Praxis, indem wir Algorithmen enwickeln, die *beweisbar* Gesetzmäßigkeiten in Daten finden und ausnutzen.

1 Einleitung

In den letzten Jahren gab es enorme Fortschritte auf den Gebieten der Künstlichen Intelligenz, des Data Mining und des Maschinellen Lernens. Dies hat sowohl in der Industrie als auch im akademischen Bereich zum immensen Wachstum dieser Bereiche geführt. Treiber dieser Entwicklung waren spezialisierte neue Hardware, die immer größere Verfügbarkeit von Daten und Durchbrüche bei der Entwicklung von Algorithmen, die *Gesetzmäßigkeiten in Daten finden und ausnutzen*.

Viele in der Praxis eingesetzte Algorithmen sind allerdings *Heuristiken*, also Algorithmen ohne mathematisch beweisbare Qualitätsgarantien. Daher können wir uns zwar vom Erfolg dieser Algorithmen in Anwendungen überzeugen, aber unser theoretisches Verständnis von ihnen und den zugrunde liegenden Probleme der Informatik bleibt beschränkt. Man kann sagen, dass der Fortschritt bei der Entwicklung von praktischen Algorithmen in den letzten Jahren so rasant war, dass sich die Kluft zwischen dem, was praktisch möglich ist, und unserem theoretischen Verständnis stark vergrößert hat.

Dennoch ist ein solides theoretisches Verständnis der praktischen Methoden höchst wünschenswert: Die rigorose mathematische Analyse eines Algorithmus liefert wichtige Einblicke in dessen Stärken und deckt zudem seine Grenzen auf. Formale Garantien für einen Algorithmus helfen uns daher, dessen Qualität zu bewerten und sie geben uns uns ein besseres Verständnis dafür, wie viel Vertrauen wir in seine Ausgabe haben können. Letzteres ist besonders wichtig, wenn die Daten sensible Attribute enthalten (etwa Alter oder Geschlecht) oder wenn die Ausgabe des Algorithmus für kritische Entscheidungen verwendet wird (zum Beispiel um zu entscheiden, ob ein Patient eine Krankheit hat oder nicht).

[1] Englischer Titel der Dissertation: “Provably Finding and Exploiting Patterns in Data” [Ne20]

[2] Königliche Technische Hochschule (KTH), Stockholm, Schweden, neum@kth.se

Es ist jedoch kein Zufall, dass nur wenige praktische Algorithmen theoretische Garantien besitzen. Viele der Probleme, die von diesen Algorithmen gelöst werden, sind NP-schwer und oft sogar NP-schwer zu approximieren. Da die Algorithmen in der Praxis aber gute Ergebnisse liefern, suggeriert dies, dass die klassische Worst-Case-Analyse, die üblicherweise bei der Analyse von Algorithmen eingesetzt wird, für diese Klasse von Problemen zu pessimistisch ist: Es ist zwar möglich, künstliche Eingaben zu entwickeln, an denen ein Algorithmus scheitert, aber in der Praxis kommt dies nie vor, weil die praktischen Eingabedaten viel mehr Struktur und *Gesetzmäßigkeiten* (engl. "pattern", auf deutsch auch "Muster") enthalten als die künstlich erzeugten Eingaben. Um also Algorithmen mit theoretischen Garantien zu erhalten, werden wir *über die Worst-Case-Analyse hinaus* gehen und dabei die Gesetzmäßigkeiten *finden und ausnutzen*.

In dieser Dissertation präsentieren wir Algorithmen, die *beweisbar* Gesetzmäßigkeiten in Daten finden und ausnutzen. Dadurch wird die Kluft zwischen dem, was praktisch möglich ist, und unserem theoretischen Verständnis verringert. Die Ergebnisse dieser Arbeit können in die folgenden Kategorien eingeteilt werden:

1. *Gesetzmäßigkeiten nachweislich finden:* Wir entwickeln Algorithmen, die beweisbar eine Menge von versteckten Clustern in bipartiten Zufallsgraphen finden. Dieses Problem muss etwa bei der Analyse von Online-Shopping-Daten gelöst werden, wo man Gruppen ("Cluster") von Produkten finden möchte, die häufig zusammen gekauft werden. Wir präsentieren den ersten Algorithmus, der nachweislich *winzige* versteckte Cluster findet. Dieses Resultat liefert eine theoretische Begründung für den Erfolg von Heuristiken, die in der Praxis angewendet werden. Weiterhin präsentieren wir den ersten Datenstromalgorithmus, der einen bipartiten Graphen nur zwei Mal sequentiell liest und dabei eine Menge von versteckten Clustern findet. Dieser Algorithmus skaliert auf viel größere Datensätze als vorhandene Methoden.

2. *Komplexität des Findens von Gesetzmäßigkeiten:* Wir betrachten häufig verwendete Subroutinen von Data Mining-Algorithmen und erhalten neue Härteresultate für deren Komplexität. Für das approximative Zählen der Dreiecke in einem Graphen und für die Approximation der Häufigkeit von Itemsets in Transaktionsdatenbanken zeigen wir, dass existierende Algorithmen nicht signifikant verbessert werden können, außer gängige Hypothesen der Komplexitätstheorie sind falsch. Außerdem präsentieren wir eine Hierarchie der Enumerationskomplexität von mehreren Maximal Frequent Pattern Mining-Problemen und wir bestimmen eine Bedingung, unter der diese Hierarchie zusammenbricht.

3. *Gesetzmäßigkeiten nachweislich ausnutzen:* Wir formulieren ein Modell, das die Gesetzmäßigkeiten von Kommunikationsanfragen in Rechenzentren abbildet, und entwickeln Algorithmen, die diese Gesetzmäßigkeiten in den Daten ausnutzen und damit den Datenverkehr im Rechenzentrum reduzieren. Wir beweisen, dass der kompetitive Faktor unserer Algorithmen asymptotisch optimal ist. Wir zeigen außerdem, dass verteilte Union-Find-Datenstrukturen mit unserem Modell implementiert werden können. Unsere Algorithmen basieren auf einer neuen Technik, die effiziente ganzzahlige lineare Programmierung ("integer linear programing", ILP) mit der manuellen Berechnung optimaler ILP-Lösungen kombiniert.

Um unsere Ergebnisse zu erhalten, verwenden wir die sogenannte Beyond-Worst-Case-Analyse, die Annahmen an die Eingabedaten macht, um die Eigenschaften zu modellieren, die man von Daten aus der Praxis erwarten würde. Zum Beispiel analysieren wir die Algorithmen aus Punkt 1 für Zufallsgraphen statt für beliebige Graphen. Für die Ergebnisse von Punkt 3 präsentieren wir ein Modell für die Gesetzmäßigkeiten der Kommunikationsanfragen in einem Rechenzentrum. Damit erzielen wir nachweislich bessere Resultate als die bestmöglichen, die keine Annahme an die Kommunikationsanfragen machen. Die Härteresultate aus Punkt 2 werden unter Verwendung der klassischen Worst-Case-Analyse hergeleitet, aber sie dienen als Motivation, diese Probleme in Zukunft mit Hilfe der Beyond-Worst-Case-Analyse zu untersuchen: Unsere hergeleiteten unteren Grenzen sind *scharf* ("tight"), sie zeigen also, dass die Laufzeiten der vorhandenen Algorithmen bezüglich der Worst-Case-Analyse nicht signifikant verbessert werden können. Folglich ist weiterer Fortschritt bei diesen Problemen nur dann möglich, wenn man gewisse Gesetzmäßigkeiten bei den Daten annimmt und diese in der Analyse ausnützt.

2 Gesetzmäßigkeiten nachweislich finden

Ein beliebtes Problem im Unüberwachten Lernen ("unsupervised learning") ist das Finden von Gesetzmäßigkeit in bipartiten Graphen. Dieses Problem ist auch bekannt als *Biclustering* oder *Co-Clustering* und wird seit den 1970er Jahren untersucht [Ha72]. Das Problem wird in verschiedenen Bereichen der Informatik studiert, etwa in Data Mining [Dh01, Mi08], im Maschinellen Lernen [LCX15, ZA19] und der Bioinformatik [MO04].

In diesem Problem ist die Eingabe ein bipartiter Graph $G = (U \cup V, E)$ und die Gesetzmäßigkeiten sind Cluster $U_1, \ldots, U_k \subseteq U$ und $V_1, \ldots, V_k \subseteq V$, sodass jeder *Bicluster* (U_i, V_i) bestimmte Kriterien erfüllt. Beliebte Kriterien sind, dass die Bicluster (U_i, V_i) Subgraphen induzieren, die entweder einen vollständigen bipartiten Graphen ("Biclique") bilden [Or77, GGM04] oder "viele" Kanten relativ zur globalen Dichte ("density") des Graphen enthalten [ZA19, RPG16] (siehe unten für das Problem, das wir untersuchen).

In Anwendungen stehen die beiden Seiten des bipartiten Graphen für Objekte von verschiedenen Typen und eine Kante kennzeichnet die Interaktion der entsprechende Objekte. Beispielsweise kann man Daten von Online-Shops analysieren, indem man die Knoten auf der linke Seite U von G mit den Kunden des Online-Shops identifiziert und die Knoten auf der rechten Seite V von G mit Produkten identifiziert. Eine Kante (u, v) kennzeichnet, dass Kunde u das Produkt v gekauft hat. Jeder Bicluster (U_i, V_i) entspricht daher einer Gruppe von Produkten V_i, die von ähnlichen Kunden U_i gekauft werden.

Reale Datensätze weisen häufig zwei entscheidende Eigenschaften auf: Auf der einen Seite des bipartiten Graphen sind die Knotengrade beschränkt und auf der anderen Seite des bipartiten Graphen enthalten die Cluster V_j nur sehr wenige Knoten. In der Praxis sind diese Annahme etwa im obigen Online-Shop-Beispiel erfüllt: Fast alle Kunden kaufen höchstens einige Hundert verschiedene Produkte, daher ist der Grad der Knoten in U beschränkt. Außerdem wird in Experimenten oft beobachtet, dass die Produkt-Cluster V_j meist aus weniger als 50 Produkten bestehen. Typische solche Produkt-Cluster sind die

7 Harry Potter-Bücher oder die 23 Filme aus dem Marvel Cinematic Universe. Daher sind die Cluster V_j viel kleiner als V, denn V enthält häufig Millionen Produkte. Man kann also sagen, dass die Cluster V_j *winzig* sind im Vergleich zur Größe von V.

Zwei der größten Herausforderungen in diesem Bereich waren folgende: (1) Während viele praktische Algorithmen winzige Cluster V_j finden können, konnten die besten theoretischen Ergebnisse nur das Finden von mittelgroßen Clustern garantieren (Einzelheiten siehe unten). Können wir diese Lücke schließen? (2) Moderne praktische Algorithmen liefern sehr gute Ergebnisse für Graphen mit Tausenden Knoten, aber sie skalieren nicht auf Graphen mit mehreren Hunderttausend oder gar Millionen Knoten. Können wir effizientere Algorithmen erhalten, indem wir ausnutzen, dass die Graphen in der Praxis sehr dünnbesetzt sind? *Wir beantworten beide Fragen positiv.*

Das Argument für Zufallsgraphen. Die Tatsache, dass frühere Algorithmen für diese Probleme meist Heuristiken waren, ist kein Zufall. Tatsächlich sind die meisten Probleme zum Finden von Clustern in bipartiten Graphen NP-schwer wenn man *Worst-Case-Eingaben* betrachtet [Or77]. Zudem gibt es starke untere Schranken für die Laufzeit von Approximationsalgorithmen [CIK16].

Um diese Härteergebnisse zu umgehen, betrachten wir ein gängiges Zufallsgraphenmodell (siehe unten für die formale Definition). Dieses Modell wurde in verschiedenen Wissenschaftsbereichen studiert, vom Maschinellem Lernen über die Theoretische Informatik bis hin zu Mathematik und Physik [Ab18]. Dank diesem Modell kann man Algorithmen entwickeln, die garantieren, dass sie eine Menge von *versteckten* Clustern $U_1,\dots,U_k$ und $V_1,\dots,V_k$ finden. Darüber hinaus wurden verschiedene Heuristiken (ohne theoretische Garantien) mit Hilfe von ähnlichen Modellen entwickelt [RPG16, Ru17] und diese Algorithmen liefern hervorragende Ergebnisse in der Praxis. Das deutet darauf hin, dass die Annahmen des Zufallsgraphenmodells die Realität gut abbilden.

Die formale Definition des Zufallsgraphenmodell ist wie folgt. Sei $G = (U \cup V, E)$ ein bipartiter Graph mit k *versteckten* Clustern $U_1,\dots,U_k \subseteq U$ und $V_1,\dots,V_k \subseteq V$ auf jeder Seite des Graphen. Ferner seien $p,q \in [0,1]$ Wahrscheinlichkeiten mit $p > q$. Wir nehmen an, dass Kanten (u,v) zwischen Knoten $u \in U_i$ und $v \in V_i$ mit Wahrscheinlichkeit p eingefügt werden und Kanten (u,v) mit $u \in U_i$ und $v \in V_j$ mit $i \neq j$ mit Wahrscheinlichkeit q eingefügt werden. Dadurch gibt es "relativ viele" Kanten zwischen den Knoten jedes Biclusters (U_i,V_i) im Vergleich zur globalen Dichte des Graphen, wo es "relativ wenige" Kanten gibt. Das Problem ist nun wie folgt: Bei Eingabe eines Zufallsgraphen, der entsprechend der oben beschriebenen Verteilung erzeugt wurde, muss der Algorithmus die verstecken Cluster $U_1,\dots,U_k$ und $V_1,\dots,V_k$ finden.

Winzige Cluster finden. Entsprechend unserer Argumentation oben sind die rechten Cluster V_j in der Praxis winzig. Während praktische Algorithmen höchst erfolgreich bei der Identifizierung dieser Cluster sind, konnten vorhandene Algorithmen mit theoretischen Garantien [Mc01, LCX15] nur mittelgroße Cluster der Größe $|V_j| = \Omega(\sqrt{n})$ finden, wobei $n = |V|$ die Anzahl der Knoten ist. Dies ist in dem oben beschriebenen praktischen Szenario unrealistisch (etwa für $|V| \geq 10^6$ erhalten wir $\sqrt{|V|} \geq 10^3$). Daher gibt es eine große Kluft zwischen den praktischen Ergebnissen und ihrer theoretischen Begründung.

Wir schließen diese Lücke zwischen Theorie und Praxis, indem wir einen praktischen Algorithmus präsentieren, der beweisbar winzige Cluster findet. Für alle $\varepsilon > 0$ zeigen wir, dass man Cluster V_j der Größe $|V_j| = O(n^\varepsilon)$ finden kann, wenn einige Bedingungen an die Paramater p, q und die Größe der linken Cluster U_i erfüllt sind. Vorherige Algorithmen benötigen $\varepsilon \geq \frac{1}{2}$. Die Garantien gelten auch dann, wenn der Graph extrem dünnbesetzt ist und der Grad jedes Knoten nur polylogarithmisch in der Gesamtzahl der Knoten ist.

Darüber hinaus zeigen Experimente, dass unser Algorithmus auf künstlich erzeugten Daten bessere Ergebnisse erzielt als existierende Heuristiken und dass die in realen Daten gefundenen Cluster von hoher Qualität ist. Somit kombiniert der Algorithmus theoretische Garantien mit praktischer Leistung.

Effiziente Datenstromalgorithmen. Eine zweite wichtige Frage in diesem Bereich ist es, vorhandene Methoden skalierbarer zu machen. Wir untersuchen dieses Problem im Datenstrommodell, bei dem die Eingabe ein Datenstrom der linken Knoten $u \in U$ zusammen mit all ihren inzidenten Kanten ist. Das Ziel ist, einen Algorithmus zu entwickeln, der den Graphen nur wenige Male sequentiell liest und der zudem wenig Speicher benötigt.

Wir präsentieren den ersten Datenstromalgorithmus ("streaming algorithm"), der den Graphen *nur einmal* sequentiell liest und dabei die rechten Cluster V_i findet. Der Algorithmus ist extrem speichereffizient: Der verwendete Speicherplatz ist lediglich $O(ks \log m)$, wobei $m = |U|$ die Anzahl der Knoten im Datenstrom ist, k ist die Anzahl der Cluster und s ist eine obere Schranke für die Knotengrade der Knoten in U (oben hatten wir argumentiert, dass s in realen Datensätzen klein ist, etwa weil Kunden in Online-Shops nur wenige Hundert Produkte kaufen). Damit ist die Speicherplatznutzung nur einen $O(\log m)$-Faktor höher als nötig, um überhaupt die Cluster V_i zu speichern (dies würde $O(ks)$ Speicher benötigen). Für eine Version des Algorithmus beweisen wir, dass er den informationstheoretisch optimalen Speicherplatz verwendet und zudem die versteckten Cluster findet, wenn der Graph anhand des Zufallsgraphenmodells erzeugt wurde. Nach einem zweiten sequentiellen Lesens des Datenstroms kann der Algorithmus außerdem die linken Cluster finden.

Zudem erweitern wir den Algorithmus, sodass er Boolesche Matrixfaktorisierungen (BMF) berechnen kann, was ein wichtiges Problem in den Bereichen des Data Mining [Mi08, HPM18] und des Maschinellen Lernens [RPG16, Ru17] ist. Es ist zudem der erste Datenstromalgorithmus für BMF, den es jemals gab.

In Experimenten mit realen Datensätzen ist der Algorithmus um Größenordnungen schneller und speichereffizienter als ein klassischer Algorithmus, der nicht im Datenstrommodell arbeitet. Insbesondere verwendet unser Algorithmus in keinem Datensatz mehr als 500 MB RAM, selbst wenn die Graphen Millionen von Knoten und Kanten enthalten. Weiterhin hat er hat die wünschenswerte Eigenschaft, dass seine Laufzeit linear in der Anzahl der Kanten des Graphen skaliert. Zudem findet der Algorithmus Cluster von hoher Qualität, die innerhalb von Faktor 2 von der Qualität des klassischen Algorithmus liegen.

3 Die Komplexität des Findens von Gesetzmäßigkeiten

Als nächstes entwickeln wir ein besseres Verständnis der Komplexität von fundamentalen Problemen im Zusammenhang mit der Suche nach Gesetzmäßigkeiten in Daten.

Komplexität der Approximation von Häufigkeiten. Einige der am häufigsten verwendeten Subroutinen in Data Mining-Algorithmen dienen der Berechnung der *Häufigkeit* von Gesetzmäßigkeiten. Wir fokussieren uns auf zwei wichtige Subroutinen: Das Zählen von Dreiecken in Graphen und die Bestimmung von Häufigkeiten in Transaktionsdatenbanken.

Wenn etwa die Eingabe ein Graph G ist und die gesuchte Gesetzmäßigkeit ein Dreieck ist, ist das Ziel, die Anzahl von Dreiecken $\#\Delta(G)$ in G zu zählen. Das Zählen von Dreiecken in Graphen ist von großer Wichtigkeit bei der Analyse von sozialen Netzwerken und in der Bioinformatik bei der Analyse von chemischen Strukturen.

In Transaktionsdatenbanken ist das Ziel, die Häufigkeit von *Itemsets* (Elementemengen) zu berechnen. Formal ist dieses Problem wie folgt definiert: Sei $[d] = \{1, \dots, d\}$ eine Menge von Elementen ("items"), dann ist ein *Itemset* eine Teilmenge von $[d]$. Eine $m \times d$ *Transaktionsdatenbank* $\mathscr{D}$ ist eine (Multi-)Menge $\mathscr{D} = \{T_1, \dots, T_m\}$, wobei jedes T_i ein Itemset über $[d]$ ist. Wir nennen $\#\mathsf{supp}(T) = |\{i : T \subseteq T_i\}|$ die *Häufigkeit* ("support") des Itemset T, $\#\mathsf{supp}(T)$ ist also die Anzahl der *Transaktionen* $T_i \in \mathscr{D}$ mit $T \subseteq T_i$. In der Praxis tritt dieses Problem in Online-Shops auf: Wenn die Elemente in $[d]$ die Produkte eines Online-Shops sind, dann entspricht eine Transaktion T_i den von einem Kunden gekauften Produkten und $\#\mathsf{supp}(T)$ zählt die Anzahl der Kunden, die die Produkte in T gekauft haben.

Obwohl $\#\mathsf{supp}(T)$ und $\#\Delta(G)$ in der Praxis häufig berechnet werden müssen, werden die Subroutinen zur Berechnung dieser Größen in der Praxis weiterhin als langsam betrachtet. Wenn wir also die Berechnung von $\#\mathsf{supp}(T)$ oder $\#\Delta(G)$ beschleunigen könnten, würde dies viele praktische Algorithmen beschleunigen. Leider implizieren gängige Komplexitätshypothesen, dass Algorithmen die *exakten* Werte von $\#\mathsf{supp}(T)$ und $\#\Delta(G)$ nicht wesentlich schneller berechnen können als mittels erschöpfender Suche [Wi05]. Da die exakte Berechnung von $\#\mathsf{supp}(T)$ und $\#\Delta(G)$ häufig zu langsam ist, greifen einige Algorithmen auf die *approximative* Berechnung dieser Größen zurück, um eine schnellere Laufzeit zu erhalten [MTV94, Li16, RU14]. Nun ist es eine natürliche Frage, wie schnell diese approximative Berechnung durchgeführt werden kann.

Daher untersuchen wir die feinkörnige ("fine-grained") Komplexität der *Approximation* von $\#\mathsf{supp}(T)$ und $\#\Delta(G)$. Feinkörnige Komplexität [Wi18] ist ein relativ neuer Bereich in der Komplexitätstheorie, der darauf abzielt, scharfe untere Schranken für die Laufzeit von Problemen aus der Informatik zu beweisen. Diese unteren Schranken werden mit Hilfe von Hypothesen bewiesen, die stärker sind als die Standardannahme $\mathsf{P} \neq \mathsf{NP}$. Wir betrachten *Lückenversionen* ("gap version") von $\#\mathsf{supp}(T)$ und $\#\Delta(G)$. Für einen gegebenen Schwellenwert ("threshold") K muss entschieden werden, ob $\#\mathsf{supp}(T) \geq K$ oder $\#\mathsf{supp}(T) \leq K/3$. Falls $K/3 < \#\mathsf{supp}(T) < K$, darf der Algorithmus irgendeine Antwort zurückgeben. Dieses Problem wurde bereits in der Vergangenheit aus der Perspektive von oberen Schranken untersucht [MTV94, Li16, RU14], da man effiziente Subroutinen für diese Probleme verwenden kann, um vorhandene Algorithmen zu beschleunigen.

Für die Lückenversion von $\#\mathsf{supp}(T)$ erhalten wir eine scharfe untere Schranke, die wir nun formal beschreiben. Angenommen die Eingabe sind eine Transaktionsdatenbank mit m Transaktionen und ein Schwellenwert K. Ein einfacher Algorithmus zur Lösung der Lückenversion von $\#\mathsf{supp}(T)$ wählt zufällig $O((m/K)\log m)$ Transaktionen und approximiert anhand dieser Zufallsstichprobe $\#\mathsf{supp}(T)$. Wir zeigen, dass dieser Algorithmus nicht wesentlich verbessert werden kann: Sofern eine gängigen Komplexitätshypothese korrekt ist, kann die Lückenversion von $\#\mathsf{supp}(T)$ nicht wesentlich schneller gelöst werden kann als in Zeit[3] $(m/K)^{1-o(1)}$. Damit liefern wir eine vollständige Erklärung der Komplexität der Lückenversion von $\#\mathsf{supp}(T)$ bis auf $m^{o(1)}$-Faktoren.

Wir erhalten scharfe untere Schranken für die Lückenversion von $\#\Delta(G)$, bei der wir Dreiecke in einem Graphen mit n Knoten zählen. In der Lückenversion von $\#\Delta(G)$ müssen wir entscheiden, ob $\#\Delta(G) \geq K$ oder $\#\Delta(G) \leq K/3$. Wir zeigen, dass ein Algorithmus, der die Lückenversion von $\#\Delta(G)$ schneller als in Zeit $(n^3/K)^{1-o(1)}$ entscheidet und keine schnelle Matrixmultiplikation verwendet, zu einem Durchbruch im Bereich der kombinatorischen Algorithmen führt. Weiterhin zeigen wir, dass unsere untere Schranke für die Lückenversion von $\#\Delta(G)$ untere Schranken für die approximative Berechnung wichtiger Graphmetriken impliziert. Zum Beispiel beweisen wir untere Schranken für die approximative Berechnung des Clustering-Koeffizienten oder der Transitivität eines Graphen.

Wir präsentieren scharfe obere und untere Schranken für eine Lückenversion von SAT, bei der wir entscheiden müssen, ob eine SAT-Formel mindestens K erfüllende Zuweisungen besitzt *oder gar keine*. Wir zeigen unter Annahme einer gängigen Komplexitätshypothese, dass dieses Problem nicht schneller als in Zeit $(2^n/K)^{1-o(1)}$ gelöst werden kann.

Auflisten maximaler Gesetzmäßigkeiten. Eine weitere häufig verwendete Subroutine in Data Mining-Algorithmen ist es, alle *maximalen* Gesetzmäßigkeiten zu finden. Für eine Transaktionsdatenbank $\mathscr{D}$ und einen Schwellenwert K heißt ein Itemset T *häufig*, wenn $\#\mathsf{supp}(T) \geq K$; andernfalls heißt es *selten*. Außerdem ist T *maximal häufig*, wenn T häufig ist und für alle Itemsets T' mit $T \subsetneq T'$ gilt, dass T' selten ist. Nun besteht das Problem darin, alle maximal häufigen Itemsets auszugeben. Dieses Problem wurde auch für andere Datentypen untersucht, bei denen die Datenbanken aus Graphen oder Sequenzen bestehen.

Beim Auflisten aller maximal häufigen Itemsets ("maximal frequent pattern mining") müssen alle maximal häufigen Itemsets berechnet und ausgegeben werden. Da es potentiell exponentiell (in $|\mathscr{D}|$) viele maximal häufige Itemsets gibt und die Ausgabe viel größer als die Eingabe werden kann, ist es dienlich, die Komplexität dieses Problems nicht nur in der Größe der Eingabe zu messen, sondern in der Größe der Eingabe *und Ausgabe* [KK14]. Dies führt zu Aufzählungs- und Erweiterbarkeitsproblemen. Ein typisches Problem in diesem Bereich wäre beispielsweise, eine gegebene Menge von maximal häufigen Itemsets zu erweitern: gegeben eine Menge von maximal häufigen Itemsets, berechne ein weiteres maximal häufiges Itemset, das nicht in der Eingabemenge enthalten ist, oder gebe aus, dass kein solches maximal häufiges Itemset existiert. Solche Probleme werden im Bereich der *Aufzählungskomplexität* ("enumeration complexity") studiert [JPY88].

[3] Wenn ein Problem "nicht schneller als in Zeit $T^{\alpha-o(1)}$" gelöst werden kann, entspricht dies der folgenden formalen Aussage: "Für alle $\varepsilon > 0$ gibt es ist keinen Algorithmus mit Laufzeit $O(T^{\alpha-\varepsilon})$." Mit anderen Worten: Es gibt keinen Algorithmus mit einer polynomiell schnelleren Laufzeit als $O(T^{\alpha})$.

Wir entwickeln eine *Hierarchie* der Aufzählungskomplexität von Data Mining-Problemen für verschiedene Datentypen. Die Datentypen, die wir berücksichtigen, umfassen Transaktions-, Sequenz- und Graphdatenbanken von verschiedene Graphklassen (von Bäumen bis zu allgemeinen Graphen). Wir zeigen, dass die Hierarchie zusammenbricht (also alle Probleme die gleiche Komplexität haben), wenn wir das Problem leicht verallgemeinern und die Menge der erlaubten Itemsets leicht einschränken dürfen.

4 Gesetzmäßigkeit nachweislich ausnutzen

Wir betrachten zudem Gesetzmäßigkeiten von Kommunikationsanfragen in Rechenzentren. In der Praxis kann empirisch gezeigt werden, dass es diese Gesetzmäßigkeiten gibt und dass Rechenzentren, die sich an diese Gesetzmäßigkeit anpassen, geringere Kosten haben [Ha14]. Daher entwickeln wir Algorithmen, die die Gesetzmäßigkeiten *ausnutzen*.

Wir entwickeln ein formales Modell, das die Struktur der Gesetzmäßigkeit in Rechenzentren erfasst. Wir nehmen an, dass das Rechenzentrum ℓ Server enthält, auf die n Workloads verteilt sind. Zu Beginn ist jeder Workload einem Server zugewiesen und jeder Server hat eine Kapazität, um $(1+\varepsilon)k$ Workloads zu speichern, wobei $k = n/\ell$ und $\varepsilon > 0$ eine Konstante ist. Anschließend beginnen die Workloads mittels Kommunikationsanfragen über das Netzwerk zu kommunizieren. Diese Folge von Kommunikationsanfragen erhält der Algorithmus während seiner Laufzeit (“online”).

Unsere Annahme für die Kommunikationsanfragen ist, dass sie *kleine Gesetzmäßigkeiten induzieren*: Wir betrachten die Workloads als Knoten eines Graphen und die Kommunikationsanfragen als Kanten. Unser Modell geht davon aus, dass, nachdem wir alle Kommunikationsanfragen bearbeitet haben, die Zusammenhangskomponenten im resultierenden Graphen höchstens k Knoten enthalten (es kann also jede Komponente auf einem einzelnen Server gespeichert werden). Diese Zusammenhangskomponenten sind die *Gesetzmäßigkeiten* der Workloads, die durch die Kommunikationsanfragen induziert werden. Weiterhin gehen wir davon aus, dass nach Abschluss der Kommunikationsanfragen alle Knoten aus derselben Komponente auf demselben Server gespeichert sein müssen.

Die Kosten für Kommunikationsanfragen und für das Verschieben von Workloads sind wie folgt. Wenn die kommunizierenden Workloads dem selben Server zugewiesen sind, hat diese Kommunikationsanfrage Kosten 0; andernfalls betragen die Kosten 1. Der Algorithmus kann sich jederzeit entscheiden, Workloads zwischen den Servern zu verschieben, wobei die Kapazitätsbeschränkungen der Server eingehalten werden müssen; für jedes Verschieben muss der Algorithmus $\alpha > 1$ bezahlen. Wir analysieren den Algorithmus mittels einer *Kompetitivitätsanalyse* (“competitive analysis”): wir teilen die von unserem Algorithmus gezahlten Kosten durch die Kosten des optimalen Offline-Algorithmus, der alle Kommunikationsanfragen im Voraus kennt und minimale Kosten hat.

Für dieses Modell entwickeln wir Algorithmen und zeigen, dass ihre Kompetitivität (bis auf konstante Faktoren) optimal ist. Das Hauptergebnis ist ein *randomisierter* Algorithmus mit Kompetitivität (“competitive ratio”) $O(\log \ell + \log k)$. Wir beweisen zudem eine passende untere Schranke von $\Omega(\log \ell + \log k)$ für randomisierte Algorithmen. Darüber hinaus

erhalten wir einen *deterministischen* Algorithmus mit Kompetitivität $O(\ell \log k)$ und eine passende untere Schranke von $\Omega(\ell \log k)$. Als Anwendung unserer Algorithmen zeigen wir, dass sie zur Implementierung von verteilten Union-Find-Datenstrukturen mit beinahe optimaler Netzwerkkommunikation (bis auf $O(1)$-Faktoren) verwendet werden können.

Ohne die Annahme an die Gesetzmäßigkeiten hätte der bestmögliche deterministische Algorithmus eine Kompetitivität von $\Omega(k)$ für $\ell = O(1)$ [Av19]. Dies zeigt, dass das *Ausnutzen* der Gesetzmäßigkeiten notwendig ist, um bessere Algorithmen zu erhalten.

Um diese Ergebnisse zu erzielen, entwickeln wir eine neue Technik, die effiziente ganzzahlige lineare Programmierung ("integer linear programming", ILP) mit der manuellen Wartung optimaler ILP-Lösungen kombiniert: Wenn neue Kommunikationsanfragen eingehen, verwenden wir ein ILP, um die Workloads den Servern zuzuweisen (dies ähnelt klassischen Algorithmen für Prozess-Scheduling). Wir können jedoch unsere Garantien nicht erreichen, wenn wir bei jeder Kommunikationsanfrage das ILP neu lösen. Stattdessen identifizieren wir bestimmte Typen von Kommunikationsanfragen, nach denen wir manuell eine optimale ILP-Lösung erhalten können ohne einen Workload zu verschieben.

Literaturverzeichnis

[Ab18] Abbe, Emmanuel: Community Detection and Stochastic Block Models: Recent Developments. J. Mach. Learn. Res., 18(177):1–86, 2018.

[Av19] Avin, Chen; Bienkowski, Marcin; Loukas, Andreas; Pacut, Maciej; Schmid, Stefan: Dynamic Balanced Graph Partitioning. In: SIAM J. Discrete Math. 2019.

[CIK16] Chandran, L. Sunil; Issac, Davis; Karrenbauer, Andreas: On the Parameterized Complexity of Biclique Cover and Partition. In: IPEC. S. 11:1–11:13, 2016.

[Dh01] Dhillon, Inderjit S.: Co-clustering documents and words using bipartite spectral graph partitioning. In: KDD. S. 269–274, 2001.

[GGM04] Geerts, Floris; Goethals, Bart; Mielikäinen, Taneli: Tiling Databases. In: DS. S. 278–289, 2004.

[Ha72] Hartigan, John A.: Direct Clustering of a Data Matrix. J. Am. Stat. Assoc., 67(337):123–129, 1972.

[Ha14] Hamedazimi, Navid; Qazi, Zafar; Gupta, Himanshu; Sekar, Vyas; Das, Samir R; Longtin, Jon P; Shah, Himanshu; Tanwer, Ashish: Firefly: A reconfigurable wireless data center fabric using free-space optics. In: SIGCOMM. Jgg. 44, S. 319–330, 2014.

[HPM18] Hess, Sibylle; Piatkowski, Nico; Morik, Katharina: The Trustworthy Pal: Controlling the False Discovery Rate in Boolean Matrix Factorization. In: SDM. S. 405–413, 2018.

[JPY88] Johnson, David S.; Papadimitriou, Christos H.; Yannakakis, Mihalis: On Generating All Maximal Independent Sets. Inf. Process. Lett., 27(3):119–123, 1988.

[KK14] Kimelfeld, Benny; Kolaitis, Phokion G.: The Complexity of Mining Maximal Frequent Subgraphs. ACM Trans. Database Syst., 39(4):32:1–32:33, 2014.

[LCX15] Lim, Shiau Hong; Chen, Yudong; Xu, Huan: A Convex Optimization Framework for Bi-Clustering. In: ICML. S. 1679–1688, 2015.

[Li16] Liberty, Edo; Mitzenmacher, Michael; Thaler, Justin; Ullman, Jonathan: Space Lower Bounds for Itemset Frequency Sketches. In: PODS. S. 441–454, 2016.

[Mc01] McSherry, Frank: Spectral Partitioning of Random Graphs. In: FOCS. S. 529–537, 2001.

[Mi08] Miettinen, Pauli; Mielikäinen, Taneli; Gionis, Aristides; Das, Gautam; Mannila, Heikki: The Discrete Basis Problem. IEEE Trans. Knowl. Data Eng., 20(10):1348–1362, 2008.

[MO04] Madeira, Sara C.; Oliveira, Arlindo L.: Biclustering Algorithms for Biological Data Analysis: A Survey. IEEE/ACM Trans. Comput. Biology Bioinform., 1(1):24–45, 2004.

[MTV94] Mannila, Heikki; Toivonen, Hannu; Verkamo, A. Inkeri: Efficient Algorithms for Discovering Association Rules. In: AAAI Workshops. Technical Report WS-94-03. S. 181–192, 1994.

[Ne18] Neumann, Stefan: Bipartite Stochastic Block Models with Tiny Clusters. In: NeurIPS. S. 3871–3881, 2018. Ein extended abstract von diesem Paper erschien bei der INFORMATIK 2019 unter dem Titel "Finding Tiny Clusters in Bipartite Graphs" in der Session *Best of Data Science Made in Germany, Austria and Switzerland*.

[Ne20] Neumann, Stefan: Provably Finding and Exploiting Patterns in Data. Dissertation, University of Vienna, 2020.

[Or77] Orlin, James: Contentment in graph theory: covering graphs with cliques. Indagationes Mathematicae, 80(5):406–424, 1977.

[RPG16] Ravanbakhsh, Siamak; Póczos, Barnabás; Greiner, Russell: Boolean Matrix Factorization and Noisy Completion via Message Passing. In: ICML. S. 945–954, 2016.

[RU14] Riondato, Matteo; Upfal, Eli: Efficient Discovery of Association Rules and Frequent Itemsets through Sampling with Tight Performance Guarantees. ACM Trans. Knowl. Discov., 8(4):20:1–20:32, 2014.

[Ru17] Rukat, Tammo; Holmes, Christopher C.; Titsias, Michalis K.; Yau, Christopher: Bayesian Boolean Matrix Factorisation. In: ICML. S. 2969–2978, 2017.

[Wi05] Williams, Ryan: A new algorithm for optimal 2-constraint satisfaction and its implications. Theor. Comput. Sci., 348(2-3):357–365, 2005.

[Wi18] Williams, Virginia Vassilevska: On some fine-grained questions in algorithms and complexity. In: Proc. ICM. 2018.

[ZA19] Zhou, Zhixin; Amini, Arash A.: Analysis of spectral clustering algorithms for community detection: the general bipartite setting. J. Mach. Learn. Res., 20:47:1–47:47, 2019.

Stefan Neumann ist Postdoktorand an der Königlichen Technischen Hochschule (KTH) in Stockholm, Schweden, in der Gruppe von ACM Fellow Aristides Gionis. Er promovierte an Universität Wien unter der Betreuung von ACM Fellow Monika Henzinger; während dieser Zeit besuchte er ACM Fellow Eli Upfal für sechs Monate an der Brown University in Providence, RI, USA. Für seine Dissertation hat er den *Award of Excellence* von der österreichischen Bundesregierung erhalten. Sein Paper [Ne18] wurde bei der INFORMATIK 2019 als *Best of Data Science Made in Germany, Austria and Switzerland* präsentiert. Vor seiner Promotion erhielt er einen Master in Informatik von der Universität des Saarlandes und dem Max Planck Institut für Informatik und einen Bachelor in Mathematik von der Universität Jena.