Masterarbeit

Cluster-Minimierung in Geometrischen Graphen

Jakob Geiger

Abgabedatum: 29. Juni 2020
Betreuer: Prof. Dr. Alexander Wolff

Julius-Maximilians-Universität Würzburg
Lehrstuhl für Informatik I
Algorithmen, Komplexität und wissensbasierte Systeme

Inhaltsverzeichnis

1 Einleitung

Geographische Daten sind häufig komplex – zu komplex, um von einem Betrachter leicht gelesen und interpretiert werden zu können. Aus diesem Sachverhalt entsteht die Notwendigkeit, solche Datenmengen in sinnvoller Weise so zu vereinfachen, dass die visuelle Komplexität auf ein erträgliches Maß reduziert wird, ohne jedoch zu viele Informationen zu verlieren. Ein beliebter Ansatz ist das sogenannte *Clustering* von Datenpunkten. Dabei werden diese zu mehreren übergeordneten Entitäten, den sogenannten *Clustern*, zusammengefasst. In der Literatur finden sich viele verschiedene Definitionen des Begriffs Cluster, eine Tatsache, die von Estivill-Castro [EC02] ausführlich diskutiert wurde. Hier ist damit die Zusammenfassung einer Anzahl an Objekten zu einer gemeinsamen Menge gemeint.

Als ein Durchbruch auf dem Gebiet des Clusterings gilt der *k-Means*-Ansatz, der erstmals von MacQueen [Mac67] beschrieben wurde. Dabei wird eine Menge S an geometrischen, also mit Koordinaten versehenen, Punkten auf eine feste Anzahl k von Clustern aufgeteilt. Dabei soll die aggregierte quadratische Abweichung aller Punkte zum Schwerpunkt des entsprechenden Clusters minimiert werden, es wird also ein besonderes Augenmerk auf die physische Dichte der Cluster gelegt.

Obwohl dieser Algorithmus wegweisend für das Feld der Datenanalyse war und ist, birgt er dennoch einige Probleme. Beispielsweise wird bereits im Vorfeld der Parameter k gewählt, obwohl der zu bearbeitende Datensatz möglicherweise ein „natürliches“ Clustering zulässt, die aber nicht genau k Cluster enthält. Deshalb wurden seit der Veröffentlichung des Ansatzes zahlreiche Verbesserungen sowie neuartige Verfahren entwickelt, um das Clustering-Problem zu lösen, wie zum Beispiel *k-Means++* [AV07] oder *DBSCAN* [EKSX96]. Viele dieser Algorithmen sind ebenfalls auf die Identifikation möglichst dichter Cluster fokussiert.

Gerade im genannten Kontext geographischer Daten ist die physische Nähe zweier Objekte häufig nicht aussagekräftig genug um ein sinnvolles Clustering zu finden. Wir betrachten daher Daten, die *Kategorien* zugeordnet werden. Die Definition dieser Kategorien ist zunächst abstrakt, spezifische Begriffe werden je nach Anwendung individuell eingeführt. Betrachtet man beispielsweise eine Stadtkarte und legt als Punktmenge die dort vorhandenen Geschäfte fest, würden die Koordinaten die physischen Standorte dieser repräsentieren. Als Kategorien könnte man die Art der Geschäfte, wie Restaurant oder Friseur, festlegen. Dies erlaubt eine differenziertere Betrachtung der vorliegenden Daten. Es erscheint sinnvoll, nur Datenpunkte, die der gleichen Kategorie angehören, in einem gemeinsamen Cluster zuzulassen. In der Literatur finden sich einige Resultate zu diesem Thema, die in Abschnitt 1.1 diskutiert werden.

In der Algorithmischen Geometrie ist es üblich, statt von Kategorien von *Farben* zu sprechen. Betrachten wir also das Problem, eine mit k Farben versehene Punktemenge

in einfarbige Cluster aufzuteilen. Es bietet sich an, die Nähe zweier Punkte über Kanten zu modellieren, die Grundmenge also als die Knoten eines Graphen $G = (V, E)$ aufzufassen. Da die Kantenmenge E ausdrücken soll, welche dieser Knoten eine vergleichsweise große Nähe zueinander aufweisen, ist es sinnvoll, sie aus bestehenden Algorithmen abzuleiten, die entwickelt wurden, um genau dies zu erreichen. Ein bekanntes Beispiel für einen solchen Algorithmus ist die sogenannte *Delaunay-Triangulierung*. Eine Delaunay-Triangulierung einer Menge P von Punkten in der Ebene ist eine Triangulierung, in der für alle enthaltenen Dreiecke gilt, dass der *Umkreis* des Dreiecks, also derjenige Kreis, auf dem alle drei Eckpunkte des Dreiecks liegen, in seinem Inneren keine Punkte aus P enthält. Dabei ergibt sich stets ein zusammenhängender planarer Graph. Deshalb genügt es als letzten Schritt einfach alle Kanten zwischen unterschiedlich gefärbten Knoten zu löschen um ein Clustering zu erhalten.

Dieser Ansatz bietet eine einfache Methode, gefärbte Punkte in der Ebene zu clustern, er liefert allerdings tendenziell Ergebnisse, die keine sehr hohe Dichte aufweisen. Um hier Abhilfe zu schaffen ist es nötig, den zugrundeliegenden Graphen mit mehr Kanten auszustatten, insbesondere ist die Voraussetzung der Planarität zu restriktiv. Um jedoch die visuelle Komplexität durch sich überlappende Cluster zu vermeiden, ist es auch weiterhin sinnvoll, Kantenkreuzungen in der Ausgabe zu verbieten. Es ergibt sich also die folgende Problemdefinition:

Cluster-Minimierung. Sei $G = (V, E)$ ein geometrischer Graph. Finde einen Teilgraphen $H = (V, E')$ von G mit $E' \subseteq E$, so dass sich keine zwei Kanten in E' kreuzen und die Anzahl der Zusammenhangskomponenten von H minimiert wird.

Dies ist eine Verallgemeinerung des beschriebenen Problems, da in der Definition keine Farben genannt werden. Es ist jedoch leicht zu sehen, dass das obige Problem durch Cluster-Minimierung gelöst werden kann, indem alle Kanten zwischen unterschiedlich gefärbten Knoten aus G gelöscht werden. Das Problem Cluster-Minimierung bildet den theoretischen Kern dieser Arbeit, folglich wird ihm das größte Augenmerk zuteil.

Es bleibt eine Methode zu finden, aus einer Punktmenge einen geeigneten Graphen zu generieren. Wie bereits erwähnt existieren Algorithmen, die genutzt werden können, um genau dies zu erreichen. In dieser Arbeit fokussieren wir uns auf die sogenannten *β-Skelette*. Für eine Menge P von Punkten in der Ebene ist das β-Skelett ein geometrischer Graph, dessen Kantenmenge von einem Parameter β abhängt. Dabei gilt, dass bei schrumpfendem β die Kantendichte wächst. Eine formale Definition der Skelette findet sich in Abschnitt 2.4. Experimentelle Resultate bezüglich der kombinierten Anwendung von β-Skeletten und verschiedenen Heuristiken für Cluster-Minimierung wurden bereits in dem zu dieser Arbeit passenden Masterpraktikum [Gei20] behandelt, daher bildet dieses Thema nur einen kleinen Teil der vorliegenden Arbeit.

1.1 Stand der Forschung

Das Problem Cluster-Minimierung ist bis heute weitgehend unerforscht. In dieser Form wurde es zuerst von Akitaya et al. [HAC+19] formuliert, jedoch nicht erschöpfend diskutiert. An gleicher Stelle finden sich einige verwandte Probleme. Die Autoren betrachteten die Fragestellung eingeschränkt auf 1-planare Graphen, in diesem Fall ist eine optimale Lösung effizient durch einen einfachen Greedy-Algorithmus erreichbar. Weiterhin wurde dort das Problem *Kanten-Maximierung* betrachtet. Dabei handelt es sich um eine eng mit der Cluster-Minimierung verknüpfte Fragestellung. Es werden die gleichen Graphen betrachtet, jedoch wird, anstatt die Anzahl der Zusammenhangskomponenten in der Ausgabe zu minimieren, die Menge der ausgewählten Kanten maximiert. Kanten-Maximierung ist effizient lösbar für Graphen mit maximal zwei Farben.

Bereg et al. [BFK+15] betrachteten die Zerlegung einer Punktmenge in k Farbklassen und untersuchten das Problem, in einer solchen Zerlegung einen Steinerbaum mit minimaler Kantenlänge für jede Farbklasse zu finden, so dass sich keine zwei Bäume kreuzen. Sie liefern ein PTAS für Graphen mit $k = 2$ und Approximationsalgorithmen für größere k. Kindermann et al. [KKR+18] betrachteten dieses Problem für gewöhnliche Spannbäume und zeigten, dass es in diesem Fall, außer für einige Spezialfälle, NP-vollständig ist.

Ein gefärbter Graph wird *Support* genannt, wenn jede Farbklasse einen zusammenhängenden Subgraphen induziert. Mit der Frage nach der Identifikation eines Supports mit minimaler Kantenlänge für zweifarbige Punktmengen beschäftigten sich Hurtado et al. [HKvK+18] und zeigten, dass sich das Problem als Schnitt zweier Matroide darstellen lässt. Akitaya et al. [ALT20] bewiesen, dass dieses Problem für Graphen mit mindestens drei Farben NP-schwer ist. Castermans et al. [CvM+19] betrachteten die Suche nach planaren Supports mit minimaler Kantenlänge und zeigten die NP-Schwere für Graphen mit zwei Farben.

Knauer et al. [KSSW07] untersuchten das Problem, für ungefärbte geometrische Graphen Spannbäume mit minimalen Kantenkreuzungen zu finden, und bewiesen, dass dieses sogar NP-schwer zu approximieren ist. Bereg et al. [BJYZ11] betrachteten kreuzungsminimale Spannbäume für zweifarbige Punktmengen in der Ebene. Dieses Problem ist effizient lösbar für den Fall, dass keine drei Punkte kollinear sind. Sind kollineare Punkte erlaubt, ist die Entscheidung, ob kreuzungsfreie Spannbäume für die beiden Farben existieren, NP-vollständig.

1.2 Eigener Beitrag

Um die Grenzen des effizient Möglichen abzustecken, wird zunächst in Abschnitt 3 ein Beweis über die NP-Vollständigkeit von Cluster-Minimierung geführt. Um zu zeigen, dass Cluster-Minimierung NP-schwer ist, wird eine Reduktion von *Planar Independent Set* durchgeführt, einem bekannten NP-schweren Problem [GJ79]. Zu diesem Zweck beschreiben wir ein Verfahren, in dem aus einem planaren Graphen, für den das Independent-Set-Problem gelöst werden soll, ein geometrischer Graph konstruiert wird. Dabei wird der planare Graph zunächst in eine Repräsentation als L-Form-Schnittgraph überführt,

aus dieser konstruieren wir einen äquivalenten Segment-Schnittgraphen. Dessen Segmente werden wiederum in die Kanten des gewünschten geometrischen Graphen übersetzt. Weiterhin argumentieren wir, dass mit der gleichen Konstruktion auch das Problem Kanten-Maximierung NP-schwer ist.

Im Anschluss daran werden in Kapitel 4 mehrere Heuristiken für Cluster-Minimierung diskutiert. Eine davon, ein einfacher Greedy-Ansatz, fand bereits bei Akitaya et al. [HAC+19] Erwähnung. Neben ihrer Motivation und ihrer Formulierung als Algorithmus werden die Heuristiken auch auf ihre Laufzeit hin untersucht.

In Kapitel 5 wird die Approximationsgüte der Heuristiken thematisiert. Wir konstruieren für jede Heuristik eine Graphfamilie, auf der Lösungen der Heuristik um mehr als einen konstanten Faktor von einer optimalen Lösung abweichen. Weiterhin zeigen wir im Zuge eines Vergleichs zwischen zwei der Heuristiken die Existenz einer Graphenfamilie, für die eine der Heuristiken Ergebnisse liefert, die um einen linearen Faktor besser sind als die von der anderen Heuristik ausgegebenen.

In Kapitel 6 wird ein ganzzahliges lineares Programm formuliert, welches Cluster-Minimierung exakt löst. Dieses bedient sich der Modellierung des Problems als Flussnetzwerk. Obwohl kein Verfahren zur effizienten Lösung von ganzzahligen linearen Programmen bekannt ist, lassen sich mit ihrer Hilfe doch viele praxisrelevante Probleme bei moderater Instanzgröße in annehmbarer Zeit lösen. Das in dieser Arbeit vorgestellte ganzzahlige lineare Programm wurde im zugehörigen Masterpraktikum [Gei20] auf realen Datensätzen getestet und lieferte auf Instanzen mit bis zu 100 Knoten innerhalb weniger Minuten die korrekte Lösung.

2 Grundlagen

In diesem Kapitel sollen einige Konzepte und Begrifflichkeiten erläutert werden, die zum Verständnis der vorliegenden Arbeit unerlässlich sind.

2.1 Graphentheoretische Konzepte

Grundlegend für das Thema der Arbeit sind die *geometrischen Graphen*. Für gewöhnlich werden in der Informatik Graphen über eine Menge von Knoten und eine Menge von Kanten repräsentiert. Dies ist eine abstrakte Darstellung, die zunächst keine Rückschlüsse auf eine mögliche Zeichnung zulässt. Im Gegensatz dazu liegt einem geometrischen Graphen eine Beschreibung durch geometrische Objekte zugrunde. Zum Beispiel ist ein Graph, dessen Knoten Punkte in der Ebene sind und dessen Kanten der geradlinigen Verbindung zwischen zwei Punkten entsprechen, ein geometrischer Graph. Diese Darstellung liegt den Problemen in dieser Arbeit zu Grunde und ist für gewöhnlich gemeint, wenn wir von geometrischen Graphen sprechen. Allerdings sind die Konzepte und Algorithmen, die im Verlauf dieser Arbeit besprochen werden, mit wenig Aufwand auf andere Formen von geometrischen Graphen erweiterbar. So sind zum Beispiel die in Kapitel 4 besprochenen Heuristiken ohne weiteres auf geometrische Graphen mit nichtgeradlinigen Kanten anwendbar, nur die Berechnung der Kantenkreuzungen erfolgt auf andere Weise.

Wie Akitaya et al. [HAC+19] bewiesen, lässt sich Cluster-Minimierung für 1-planare Graphen exakt und effizient lösen. Dieses Resultat bildet eine Komponente einer Heuristik aus Kapitel 4, daher soll der Begriff der *1-Planarität* hier definiert werden.

Für einen Graphen $G = (V, E)$ bezeichnen wir eine Abbildung $\Gamma : V \cup E \to \mathbb{R}^2$ als eine *Zeichnung*, wenn Γ auf V eingeschränkt injektiv ist und für jede Kante $\{u, v\} \in E$ gilt, dass $\Gamma(\{u, v\})$ eine Kurve mit Endpunkten $\Gamma(u)$ und $\Gamma(v)$ ist. Ein regulärer Graph heißt *planar*, wenn er eine Zeichnung ohne Kantenkreuzungen besitzt. Die 1-Planarität bildet eine Abschwächung dieses Konzepts. Ein Graph heißt 1-planar, wenn er eine Zeichnung besitzt, in der jede Kante höchstens eine andere Kante schneidet. Diese Definition bezieht sich, wie der Begriff der Planarität, auf die abstrakte Repräsentation eines Graphen. Im Kontext von geometrischen Graphen, insbesondere den in dieser Arbeit hauptsächlich betrachteten euklidischen Graphen, ist die Zeichnung eines Graphen bereits vorgegeben, die theoretischen Konzepte von Planarität und 1-Planarität sind also nicht ohne weiteres übertragbar. Die entsprechenden Begriffe sind *eben* beziehungsweise *1-eben*. Wir nennen einen geometrischen Graphen eben oder 1-eben, wenn in der entsprechenden Zeichnung jede Kante ungekreuzt beziehungsweise höchstens einmal gekreuzt ist.

Ebenfalls zentral für diese Arbeit ist das Konzept der *Cluster*. Wie in der Einleitung bereits erwähnt finden sich in der Literatur viele verschiedene Definitionen des Begriffs.

Jede dieser Definitionen bezeichnet im Grunde jedoch mit Clustering das Zusammenfassen mehrerer Datenpunkte zu einer übergeordneten Entität. Im Kontext des Problems Cluster-Minimierung verstehen wir unter einem Cluster eine Zusammenhangskomponente eines kreuzungsfreien geometrischen Graphen.

2.2 Modellierung

Ein wichtiges Werkzeug zur Modellierung abstrakter Probleme sind die sogenannten *Flussnetzwerke*. Da wir diese in Kapitel 6 zur Herleitung eines exakten Algorithmus für Cluster-Minimierung nutzen, soll der Begriff hier kurz erklärt werden. Als Netzwerk bezeichnen wir einen gerichteten Graphen $G = (V, E)$ mit zwei ausgezeichneten Knoten, der Quelle s und der Senke t. Weiterhin wird jeder Kante aus E eine nichtnegative *Kapazität* zugewiesen. Von einem *Fluss* spricht man, wenn jeder Kante zusätzlich ein nichtnegativer *Flusswert* zugewiesen wird. Dabei muss gelten, dass für keine Kante der Flusswert die Kapazität übersteigt. Weiterhin muss für jeden Knoten außer Quelle und Senke gelten, dass der Betrag des Zuflusses dem des Abflusses entspricht. Diese grundlegende Definition lässt sich erweitern, indem zum Beispiel jeder Kante eine Mindestkapazität zugewiesen wird. Für die Modellierung von Cluster-Minimierung in Kapitel 6 verwenden wir eine Abwandlung der Flussnetzwerke, in der mehrere Quellen und Senken erlaubt sind.

Ein weiteres Hilfsmittel zur Modellierung stellen die *linearen Programme* (kurz: LP) dar. Ein LP besteht aus mehreren Komponenten, den Variablen, den Nebenbedingungen und der Zielfunktion. Die Variablen repräsentieren dabei Sachverhalte des Problems, das über das LP modelliert wird, während die Nebenbedingungen das Verhalten der Variablen regulieren. Die Zielfunktion weist schließlich einer Belegung der Variablen einen Wert zu. Das Ziel eines linearen Programms ist es üblicherweise, eine Variablenbelegung zu finden, welche die Zielfunktion entweder maximiert oder minimiert. Viele Probleme lassen sich über lineare Programmierung modellieren, reguläre LPs sind außerdem effizient lösbar. Für diese Arbeit von großer Bedeutung sind die *ganzzahligen linearen Programme* (ILPs). Diese unterscheiden sich von regulären LPs durch die Einschränkung einiger oder aller Variablen auf ganzzahlige Wertebereiche. Für ILPs ist, im Gegensatz zu regulären linearen Programmen, kein effizientes Lösungsverfahren bekannt. Dies deckt sich mit der Beobachtung, dass viele NP-schwere Probleme als ILP darstellbar sind. Die Konstruktion eines exakten Algorithmus für Cluster-Minimierung in Kapitel 6 erfolgt über die Modellierung des Problems als ganzzahliges lineares Programm.

2.3 Union-Find

Ein zentraler Bestandteil der in Kapitel 4 vorgestellten Heuristiken ist die *Union-Find*-Datenstruktur. Diese soll daher an dieser Stelle kurz beschrieben werden.

Union-Find ist eine von Galler und Fischer [GF64] im Jahre 1964 erstmals beschriebene Datenstruktur. Mit ihr lässt dich die Partition einer Menge in disjunkte Teilmengen verwalten. Dazu wird aus jeder gespeicherten Menge eines seiner Elemente, das *kanonische Element*, als Repräsentant gewählt. Dieses wird genutzt, um die Unterscheidung der

Teilmengen effizient durchführen zu können.

Union-Find stellt die drei Operationen MakeSet, Union und Find zur Verfügung. MakeSet(x) erzeugt eine neue Teilmenge, die zunächst nur das Element x enthält. Dieses wird dementsprechend als Repräsentant der Menge gewählt. Union(x, y) identifiziert die Repräsentanten der Teilmengen, die x beziehungsweise y enthalten. Anschließend werden diese Mengen unter einem gemeinsamen kanonischen Element zusammengefasst. Die Wahl des neuen Repräsentanten hängt dabei von der Implementierung ab, es wird jedoch stets eines der kanonischen Elemente der beiden vereinigten Mengen selektiert. Find(x) liefert das kanonische Element derjenigen Menge, zu der x gehört.

Abgesehen von der Tatsache, dass die Funktionalität von Union-Find nahezu kongruent zu den Anforderungen des Cluster-Minimierungs-Problems ist, bietet sich diese Datenstruktur auch durch ihre höchst kompetitive Effizienz an. Wie Tarjan und van Leeuwen [TvL84] zeigten, benötigt Union-Find, bei korrekter Implementierung, für $n-1$ Union- und m Find-Operationen eine Laufzeit von $O(n + m\alpha(n))$. Dabei bezeichnet $\alpha(n)$ die *Inverse Ackermann-Funktion*. Diese wächst extrem langsam, tatsächlich liefert sie für Eingaben $< 10^{80}$, was in etwa der geschätzten Anzahl an Atomen im beobachtbaren Universum entspricht, nur Werte, die kleiner als 5 sind. Daher kann sie für alle praktischen Anwendungen als konstant angenommen werden. Friedman und Saks [FS89] bewiesen, dass diese Laufzeit für alle Datenstrukturen, die die geforderten Operationen bereitstellen, optimal ist. Die MakeSet-Operation ist in konstanter Zeit realisierbar.

2.4 β-Skelette

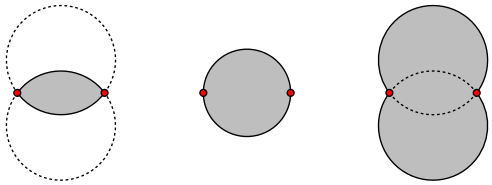
In diesem Abschnitt soll das Konzept der β-Skelette näher beschrieben werden. β-Skelette wurden zuerst von Kirkpatrick und Radke [KR85] als eine Variante der von Edelsbrunner et al. [EKS83] diskutierten α-*Formen* definiert, welche wiederum mit den Konzepten der konvexen Hülle und der Delaunay-Triangulierung verwandt sind.

Zur Konstruktion eines β-Skeletts werden zwei Komponenten benötigt, nämlich eine Menge P von Punkten in der Ebene und ein Parameter β. Zunächst muss in Abhängigkeit von β ein Winkelmaß θ berechnet werden, dies geschieht über die folgende Formel.

$$\theta = \begin{cases} \sin^{-1}(\frac{1}{\beta}) & \text{für } \beta \geq 1 \\ \pi - \sin^{-1}(\beta) & \text{für } \beta \leq 1 \end{cases}$$

Nach Konstruktion gilt, dass θ bei steigendem β schrumpft. Definiere für zwei Punkte p, q die Menge R_{pq} mit $R_{pq} = \{r \mid \angle prq > \theta\}$. Für $\beta \leq 1$ nimmt R_{pq} die Form des Schnitts zweier Kreisscheiben mit Radius $d(p,q)/\beta$ an. Für $\beta \geq 1$ ergibt sich hingegen die Vereinigung zweier Kreisscheiben mit Radius $\beta \cdot d(p,q)$. Abbildung 2.1 zeigt die verschiedenen Ausprägungen.

Eine Kante $\{p, q\}$ ist genau dann in der Kantenmenge des β-Skeletts enthalten, wenn die Menge R_{pq} keinen Punkt aus V enthält. Die grau gefärbten Flächen in Abbildung 2.1 bilden also „verbotene“ Bereiche. Das bei steigendem β monotone Wachstum dieser Areale impliziert, dass die Kantenmenge eines β-Skeletts in diesem Fall monoton schrumpft.



(a) $\beta < 1$ (b) $\beta = 1$ (c) $\beta > 1$

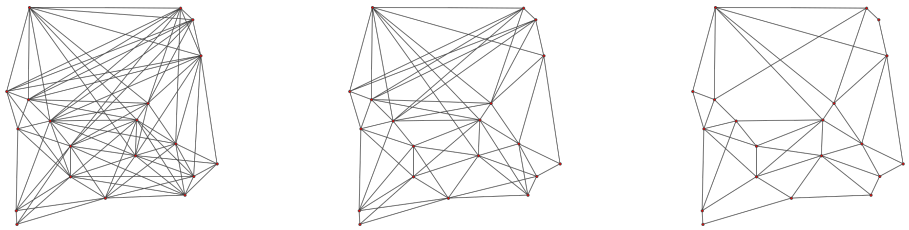
Abb. 2.1: Die Menge R_{pq} für verschiedene Werte von β.

Jedes Skelett ist also ein Subgraph aller Skelette mit kleinerem β. Abbildung 2.2 zeigt die Skelette auf einer Punktmenge bei verschiedenen β-Werten.

Da für $\beta \leq 0$ die Bereiche ausarten würden, ist es sinnvoll, die Domäne von β auf positive Werte zu beschränken. Damit ergibt sich die formale Definition der β-Skelette.

Definition (β-Skelett)**.** Sei V eine Punktmenge in der Ebene. Für $\beta \in (0, \infty)$ ist das β-Skelett für V definiert als der Graph $G = (V, E)$ mit $E = \{\{p, q\} \mid V \cap R_{pq} = \emptyset\}$.

Bemerkung. Für $\beta = 1$ ergibt sich der sogenannte *Gabriel-Graph*. Da dieser planar ist, sind folglich alle β-Skelette für $\beta \geq 1$ planar.



(a) $\beta = 0{,}5$ (b) $\beta = 0{,}7$ (c) $\beta = 0{,}9$

Abb. 2.2: β-Skelette für verschiedene Werte von β.

3 Komplexität

In diesem Abschnitt soll ein Beweis über die NP-Vollständigkeit von Cluster-Minimierung geführt werden. Dieser besteht, wie alle Beweise dieser Art, aus zwei Teilen. Zunächst soll argumentiert werden, dass es sich bei Cluster-Minimierung tatsächlich um ein Problem aus der Komplexitätsklasse NP handelt. Im Anschluss muss der Nachweis erbracht werden, dass zusätzlich die Bedingung der NP-Schwere erfüllt wird. Zu diesem Zweck soll das bekannte Problem *Planar Independent Set* auf Cluster-Minimierung reduziert werden.

3.1 Grundlegendes und Definitionen

Zunächst soll das Problem *Planar Independent Set* formal korrekt definiert werden. Dabei orientieren wir uns an der Darstellung des regulären Independent-Set-Problems, wie sie bei Garey und Johnson [GJ79] zu finden ist.

Definition (Planar Independent Set)**.** Sei $G = (V, E)$ ein planarer Graph und $k \in \mathbb{N}^+$. Entscheide, ob G eine unabhängige Menge mit Mächtigkeit mindestens k enthält.

Da isolierte Knoten die Lösung des Problems offensichtlich nicht erschweren, beschränken wir uns im Folgenden auf zusammenhängende planare Graphen. Dies bietet den Vorteil, dass im Kontext von Laufzeitanalysen die codierte Eingabe einer Probleminstanz jeden Knoten explizit enthalten muss und ihre Größe damit linear in der Anzahl der Knoten von unten beschränkt ist.

Um die angestrebte Reduktion einfacher zu gestalten, nutzen wir statt des bereits definierten Cluster-Minimierungs-Problems eine Variante desselben, in der keine Optimierung gesucht wird, sondern lediglich die Frage nach der Existenz einer Lösung mit einer bestimmten Größe gestellt wird.

Definition (Cluster-Anzahl)**.** Sei $G = (V, E)$ ein geometrischer Graph und $k \in \mathbb{N}^+$. Entscheide, ob G eine kantenkreuzungsfreie Partitionierung in höchstens k Zusammenhangskomponenten zulässt.

Wie ebenfalls bei Garey und Johnson [GJ79] nachzulesen ist, ist die Umformulierung eines Optimierungsproblems in ein Entscheidungsproblem zum Zweck einer Beweisführung durchaus üblich, da die NP-Vollständigkeit einer Variante stets auch die NP-Vollständigkeit der anderen impliziert. Die geläufigen Argumente dafür sind auch hier gültig. Angenommen, ein Algorithmus könnte Cluster-Anzahl lösen. Dann ließe sich mit seiner Hilfe leicht eine Methode zur Lösung von Cluster-Minimierung entwerfen. Zum Beispiel kann mit linear vielen Aufrufen des Algorithmus die Größe eines optimalen Clusterings gefunden werden. Im Anschluss kann für eine Kante e überprüft werden, ob sie

diesem Clustering angehört, indem man den ursprünglichen Algorithmus für G/e aufruft und überprüft, ob dadurch die Größe einer optimalen Lösung wächst. Auch dies benötigt nur linear viele Aufrufe. Umgekehrt kann natürlich über das Finden eines optimalen Clusterings entschieden werden, ob ein Clustering einer bestimmten Größe existiert. Es ist also klar, dass es für den angestrebten Beweis genügt, das Problem Cluster-Anzahl zu betrachten.

3.2 Cluster-Minimierung ist NP-vollständig

Zunächst zeigen wir die Zugehörigkeit von Cluster-Anzahl zur Komplexitätsklasse NP.

Lemma 3.1. *Cluster-Anzahl ist in NP.*

Beweis. Die Zugehörigkeit von Cluster-Anzahl zu NP ergibt sich über ein einfaches Argument. Betrachte das folgende Vorgehen. Für jede Kante in E entscheide nichtdeterministisch, ob sie in die Lösung aufgenommen wird oder nicht. Dann überprüfe, ob die Lösung planar ist und höchstens k Zusammenhangskomponenten besitzt und gib die Antwort aus. Dies ist ein NP-Algorithmus. □

Wir wenden uns nun dem Beweis der NP-Schwere von Cluster-Anzahl zu. Dafür wollen wir eine Instanz von Planar Independent Set in einen Graphen verwandeln, in dem jede Kante genau einen Knoten aus der ursprünglichen Instanz repräsentiert. Diese Kanten sollen sich genau dann schneiden, wenn die korrespondierenden Knoten in der ursprünglichen Instanz adjazent sind. Wie wir später zeigen werden, ergibt sich dadurch eine gültige Reduktion von Planar Independent Set auf Cluster-Anzahl. Damit eine Reduktion für eine Argumentation über NP-Schwere geeignet ist, muss sie in Polynomialzeit durchführbar sein. Auch über diesen Umstand argumentieren wir im Folgenden.

Die angesprochene Umwandlung erfolgt nicht direkt, sondern über mehrere Phasen. Zunächst konstruieren wir aus dem übergebenen planaren Graphen einen äquivalenten *L-Form-Schnittgraphen*. Ein L-Form-Schnittgraph ist ein Graph, bei dem jeder Knoten durch eine geometrische Figur repräsentiert wird, die aus einem vertikalen und einem horizontalen Segment besteht. Dabei schließt das horizontale Segment am unteren Ende des vertikalen Segments an und verläuft nach rechts. Beide Strecken müssen eine positive Länge haben. Zwei Knoten sind genau dann adjazent, wenn sich die beiden sie repräsentierenden Figuren schneiden. Zwei Knoten dürfen höchstens einen gemeinsamen Punkt haben. In Abbildung 3.1 sind ein planarer Graph sowie ein entsprechender L-Form-Schnittgraph dargestellt. Gonçalves et al. [GIP18] haben gezeigt, dass jeder planare Graph eine solche Darstellung besitzt. Wir orientieren uns an der Methodik dieser Autoren um zu zeigen, dass sich diese Darstellung in polynomieller Zeit konstruieren lässt.

Lemma 3.2. *Aus einem planaren Graphen $G = (V, E)$ lässt sich in polynomieller Zeit ein äquivalenter L-Form-Schnittgraph konstruieren.*

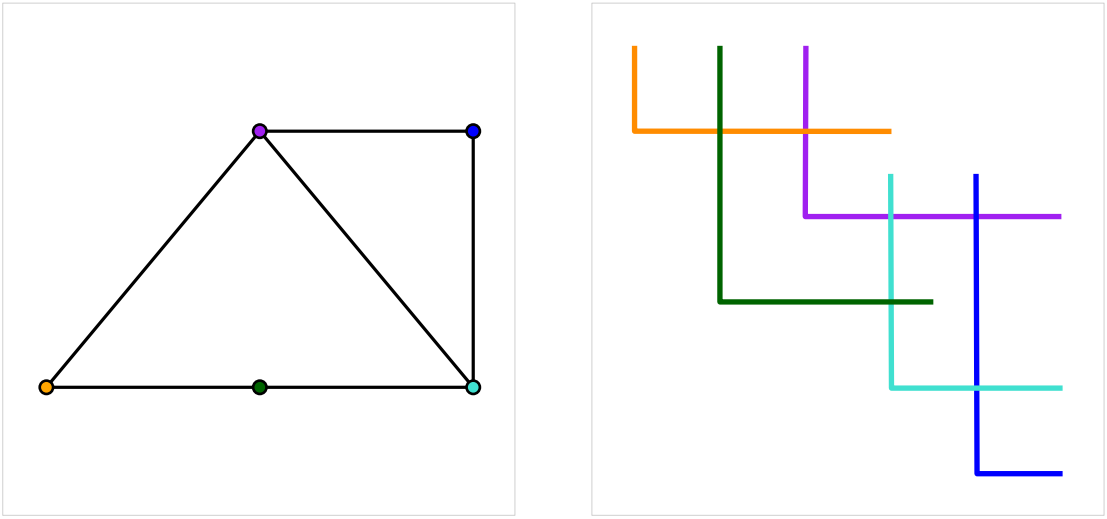


Abb. 3.1: Ein planarer Graph und eine äquivalente L-Darstellung.

Beweis. Der Konstruktion der Autoren folgend muss zunächst eine Triangulierung $T = (V', E')$ konstruiert werden, so dass G ein induzierter Subgraph von T ist. Insbesondere darf keine Kante aus $E' \setminus E$ zwischen zwei Knoten aus V verlaufen, neue Kanten müssen also stets an neue Knoten anschließen. Chalopin et al. [CGO10] zeigten, dass eine solche Triangulierung für planare Graphen stets existiert, die dort angegebene Konstruktion lässt sich leicht in polynomieller Zeit durchführen. Die genannten Eigenschaften der Triangulierung sind unabdingbar, da sich zwar Knoten und dazu adjazente Kanten leicht aus einer L-Repräsentation entfernen lassen, Kanten zwischen bestehenden Knoten jedoch nicht.

Ausgehend von dieser Triangulierung definieren die Autoren eine schrittweise Dekonstruktion, aus der sich dann wiederum die L-Darstellung ergibt. Dabei gehört der aktuell betrachtete Knoten genau einer von drei Kategorien an, für die sich die Platzierung der jeweiligen Figur unterscheidet. Sowohl der Test auf die Zugehörigkeit zu einer Kategorie als auch die drei verschiedenen Konstruktionswege sind in polynomieller Zeit durchführbar.

Zum Schluss sind noch alle Knoten aus $V' \setminus V$ zu entfernen, ein Schritt, der offensichtlich ebenfalls in Polynomialzeit durchführbar ist. □

Bemerkung. Die L-Repräsentation eines Graphen mit n Knoten passt in ein Gitter mit Länge und Breite n. [GIP18]

Ausgehend von dem soeben konstruierten L-Form-Schnittgraphen wird nun eine weitere Zwischenstufe angestrebt, nämlich ein *Segment-Schnittgraph*. Die Definition eines Segment-Schnittgraphen ist analog zu der eines L-Form-Schnittgraphen, allerdings handelt es sich bei den verwendeten geometrischen Objekten um Liniensegmente mit beliebiger Ausrichtung. Abbildung 3.2 zeigt einen L-Form-Schnittgraphen sowie einen äquivalenten Segmentgraphen.

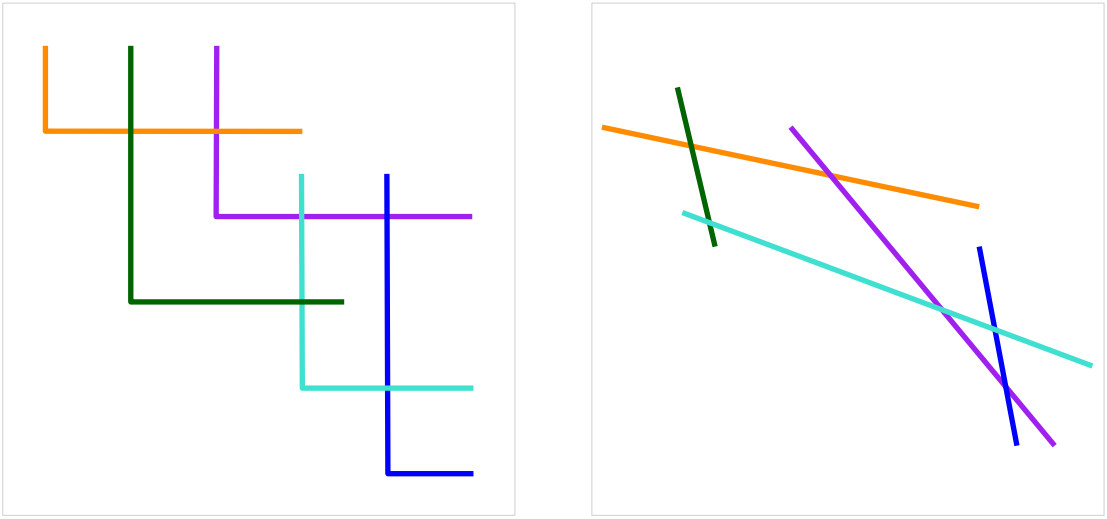


Abb. 3.2: Ein L-Form-Schnittgraph und ein äquivalenter Segment-Schnittgraph.

Eine berühmte Vermutung, die Scheinerman in seiner Doktorarbeit [Sch84] aufstellte, besagt, dass jeder planare Graph eine Repräsentation als Segment-Schnittgraph besitzt. Fünfundzwanzig Jahr später konnte die These von Chalopin und Gonçalves [CG09] bewiesen werden. Für diese Anwendung unabdingbar ist jedoch vor allem ein Resultat von Biedl [Bie20], nach dem jeder Graph, der eine L-Repräsentation besitzt, und damit insbesondere jeder planare Graph, auf einem Gitter mit Länge und Breite 4^n als Segment-Schnittgraph darstellbar ist. Biedl stellt auch eine einfache Konstruktion vor, mit der man einen L-Form-Schnittgraphen in einen äquivalenten Segment-Schnittgraphen umwandeln kann. Resultate und Verfahren aus jenem Aufsatz erlauben es uns, das folgende Lemma zu beweisen.

Lemma 3.3. *Aus einem planaren Graphen $G = (V, E)$ lässt sich in polynomieller Zeit ein äquivalenter Segment-Schnittgraph konstruieren.*

Beweis. Sei $G = (V, E)$ ein planarer Graph. Die Konstruktion von Biedl nutzt als Ausgangspunkt die L-Repräsentation eines Graphen. Wir gehen daher im Folgenden, in Übereinstimmung mit Lemma 3.2, von einer solchen Repräsentation aus. Der erste Schritt besteht darin, die vorliegende Repräsentation auf ein Gitter mit Länge und Breite $2n$ anzupassen, und zwar so, dass in beide Richtungen jede Koordinate genau einmal belegt ist. Biedl nennt in ihrer Arbeit Argumente, nach denen dies problemlos möglich ist. Anschließend wird die Darstellung um 180° rotiert, es ergeben sich also T-Formen. Seien $(l, t), (r, b)$ mit $l, b, r, t \in \{0, \ldots, 2n-1\}$ die Koordinaten der Endpunkte einer solchen Form. Dann ergeben sich die Koordinaten der Endpunkte des korrespondieren Segments durch $(2^l, 2^t - 2^{t-(r-l)})$ und $(2^r - 2^{r-(t-b)}, 2^b)$. Die Berechnung dieser Koordinaten ist offensichtlich in Polynomialzeit möglich, insbesondere wird nur polynomiell viel Speicherplatz benötigt. □

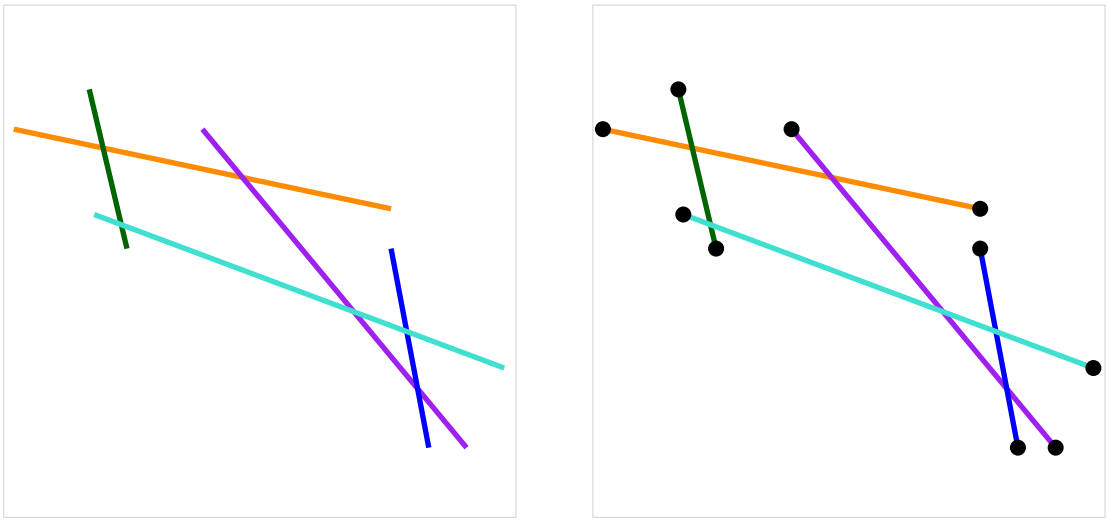


Abb. 3.3: Die Konstruktion des Graphen aus Definition 3.4.

Bemerkung. Aus der Konstruktion ergibt sich, dass keine zwei Segmente einen gemeinsamen Endpunkt haben können.

Wir haben gezeigt, dass ein planarer Graph in polynomieller Zeit in einen äquivalenten Segment-Schnittgraphen verwandelt werden kann. Auf Grundlage einer solchen Darstellung vollziehen wir nun die letzte benötigte Transformation.

Definition 3.4. Sei S ein Segment-Schnittgraph, der aus den Segmenten $s_i = [p_i, q_i]$, $i \in \{1, \ldots, n\}$ besteht. Dann ist der Graph $G^t = (V^t, E^t)$ definiert über

$$V^t = \{p_i \mid i \in \{1, \ldots, n\}\} \cup \{q_i \mid i \in \{1, \ldots, n\}\}$$
$$E^t = \{\{p_i, q_i\} \mid i \in \{1, \ldots, n\}\}$$

Die Endpunkte aller Segmente aus S bilden also die Knotenmenge V^t von G^t. Zwei Knoten in V^t sind genau dann adjazent, wenn sie die Endpunkte des gleichen Segments in S sind. Diese Konstruktion ist offensichtlich in Polynomialzeit durchführbar.

Es bleibt zu zeigen, dass ein auf diese Weise konstruierter Graph die gewünschte Reduktion leistet.

Lemma 3.5. *Sei $G = (V, E)$ ein planarer Graph und $k \in \mathbb{N}$. Dann gilt: (G, k) ist eine Ja-Instanz von* Planar Independent Set *genau dann, wenn $(G^t, 2|V| - k)$ eine Ja-Instanz von* Cluster-Anzahl *ist.*

Beweis. Nach Konstruktion der Segment-Schnittdarstellung von G korrespondiert jede Kante aus E^t zu genau einem Knoten aus V und zwei Kanten aus E^t schneiden sich genau dann, wenn die entsprechenden Knoten in G adjazent sind. Eine unabhängige Menge in G induziert also eine Menge von untereinander ungekreuzten Kanten in G^t und umgekehrt.

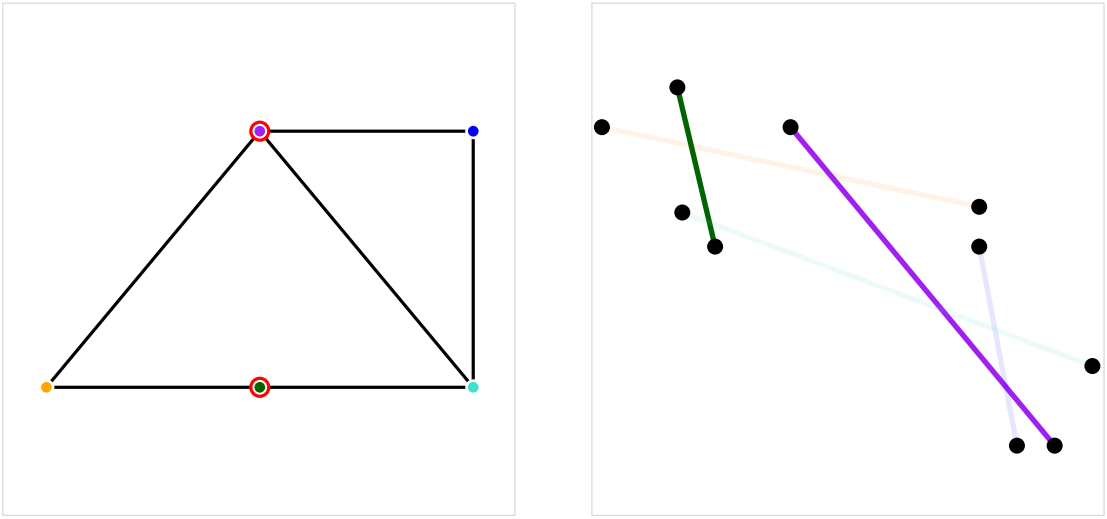


Abb. 3.4: Eine unabhängige Menge und das entsprechende Clustering.

Jeder Knoten in G^t ist Endpunkt genau einer Kante. Die maximale Anzahl an Zusammenhangskomponenten in einer Lösung für Cluster-Anzahl ist $|V^t| = 2|V|$, und jede selektierte Kante verringert die Anzahl der Cluster um genau 1.

Sei nun U eine unabhängige Menge mit Größe k in G. Selektiert man in G^t die entsprechenden Kanten, ergibt sich eine gültige Lösung mit genau $2|V| - k$ Clustern. Existiert umgekehrt in G^t eine Lösung mit Größe $2|V| - k$ müssen in dieser genau k Kanten selektiert sein. Diese Kanten induzieren eine unabhängige Menge mit Größe k in G. Abbildung 3.4 illustriert den Zusammenhang der Probleminstanzen. □

Wir haben nun gezeigt, dass sich Planar Independent Set in polynomieller Zeit auf Cluster-Anzahl reduzieren lässt. Da Cluster-Anzahl und Cluster-Minimierung aus komplexitätstheoretischer Sicht äquivalent sind, ergibt sich damit der folgende Satz.

Satz 3.6. *Cluster-Minimierung ist NP-vollständig.*

3.3 Kanten-Maximierung ist NP-vollständig

In der Einleitung wurde das von Akitaya et al. [HAC+19] im Zusammenhang mit Cluster-Minimierung formulierte Problem *Kanten-Maximierung* angesprochen. Dieses gestaltet sich wie folgt.

Definition (Kanten-Maximierung)**.** Sei $G = (V, E)$ ein geometrischer Graph. Finde einen Subgraphen $H = (V, E')$ mit $E' \subseteq E$, so dass E' keine Kantenkreuzungen enthält und $|E'|$ maximiert wird.

Es ist leicht zu sehen, dass in einem 1-regulären Graphen die Anzahl an Kanten genau dann maximiert wird, wenn die Anzahl an Clustern minimiert wird. Dies entspricht

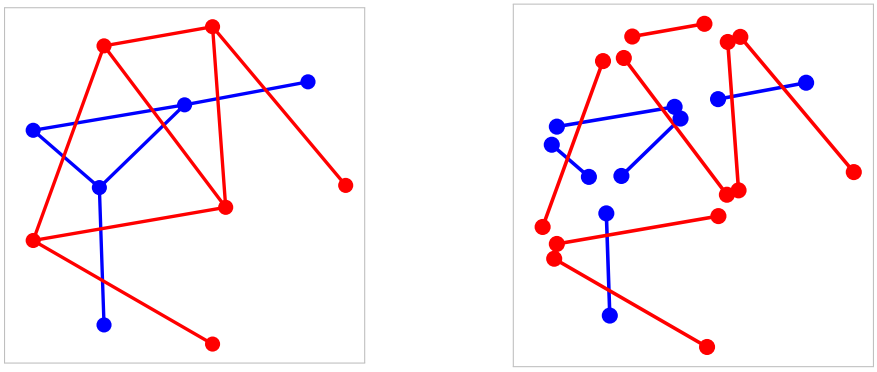


Abb. 3.5: Eine Instanz von Kanten-Maximierung und ihre Reduktion auf Cluster-Minimierung.

der Argumentation aus dem Beweis von Lemma 3.5. Folglich kann mittels Kanten-Maximierung eine nach Definition 3.4 konstruierte Probleminstanz optimal gelöst werden, die beschriebene Reduktion von Planar Independent Set ist also auch für Kanten-Maximierung gültig. Da das Problem von einem einfachen NP-Algorithmus gelöst werden kann, ergibt sich das folgende Korollar.

Korollar 3.7. *Kanten-Maximierung ist NP-vollständig.*

Bemerkung. Der komplexitätstheoretischen Äquivalenz von Cluster-Minimierung und Kanten-Maximierung entsprechend lässt sich Kanten-Maximierung durch eine einfache Konstruktion auf Cluster-Minimierung reduzieren. Dafür muss ein gegebener geometrischer Graph in einen 1-regulären Graphen verwandelt werden, indem jede Kante ein Stück verkürzt wird und an ihren Enden neue Knoten eingefügt werden. Dies muss so durchgeführt werden, dass keine Kantenkreuzungen verloren gehen. Mit der bekannten Argumentation wird für einen solchen Graphen die Anzahl der selektierten Kanten genau dann maximiert, wenn die Anzahl der Cluster minimiert wird. Abbildung 3.5 illustriert dieses Vorgehen.

4 Heuristiken

Aufgrund der in Kapitel 3 bewiesenen NP-Vollständigkeit von Cluster-Minimierung ist die Existenz eines exakten, effizienten Algorithmus unwahrscheinlich. Daher sollen in diesem Kapitel einige Heuristiken zur Approximation dieses Problems vorgestellt sowie auf ihre Laufzeit untersucht werden.

4.1 Greedy

Wie in vielen Arbeiten bildet auch hier ein einfacher Greedy-Algorithmus den Ausgangspunkt der Suche nach einer möglichst guten Heuristik. Es soll also iterativ die jeweils lokal optimale Kante selektiert werden. Dabei ist die Wahl eines Auswahlkriteriums nicht sofort offensichtlich, da jede Kante die aktuelle Clusteranzahl höchstens um eins verringern kann und daher potenziell viele Kanten die Lösung lokal auf die gleiche Weise beeinflussen. Stattdessen fokussieren wir uns auf Kanten, deren Selektion den weiteren Verlauf des Algorithmus möglichst wenig negativ beeinflussen. Da die Selektion einer Kante stets das Löschen aller sie schneidenden Kanten beinhaltet, liegt es nahe, immer diejenige Kante zu wählen, deren Kreuzungszahl zum jeweiligen Zeitpunkt minimal ist. Außerdem ist es sinnvoll, Kanten, die innerhalb bereits bestehender Cluster verlaufen und somit kein Potenzial zur Verbesserung der Lösung tragen, bei der Berechnung der Kreuzungszahlen zu ignorieren. Zur Verwaltung der aktuellen Cluster verwenden wir die in Abschnitt 2.3 vorgestellte Datenstruktur Union-Find. Aus diesen Überlegungen ergibt sich der folgende Algorithmus 1.

Laufzeit Die Laufzeit des Greedy-Algorithmus setzt sich aus drei großen Bausteinen zusammen: Der Berechnung der Kantenkreuzungen, der Auswahl der minimal gekreuzten Kanten und der Benutzung von Union-Find.

Balaban [Bal95] stellte 1995 einen Algorithmus vor, der für m Segmente mit insgesamt k Schnitten in $O(k + m \log m)$ alle Kreuzungen berechnet und ausgibt. Diese Laufzeit entspricht, wie Chazelle und Edelsbrunner [CE92] zeigten, dem Optimum und ist eine Verbesserung gegenüber der Komplexität des geläufigen *Bentley-Ottmann*-Algorithmus.

Zur Auswahl der aktuell minimal gekreuzten Kante bietet sich die Verwendung einer Datenstruktur an, die eine Prioritätswarteschlange realisiert. Dabei verwenden wir Kanten als Elemente und weisen ihnen ihre jeweilige Kreuzungszahl als Schlüssel zu. Für den Greedy-Algorithmus besonders wichtig sind die Operationen Remove, ExtractMin und DecreaseKey. Daher suchen wir nach einer Datenstruktur, die diese möglichst effizient bereitstellt. Eine solche Realisierung einer Prioritätswarteschlange ist der *Fibonacci-Heap*. Fibonacci-Heaps wurden erstmals von Fredman und Tarjan [FT87] be-

Algorithmus 1 : Greedy Cluster Minimierung(Graph $G = (V, E)$)

Eingabe : Graph $G = (V, E)$
Ausgabe : $E' \subseteq E$, so dass E' keine Kreuzungen enthält und möglichst wenige Zusammenhangskomponenten in $G' = (V, E')$ verbleiben

```
1  E' ← ∅
2  foreach v ∈ V do MakeSet(v)
3  foreach e = (v, w) aus E ohne Kreuzungen do
4      Verschiebe e von E nach E'
5      Union(Find(v),Find(w))
6  while E ≠ ∅ do
7      foreach e = (v, w) aus E do
8          if Find(v) = Find(w) then
9              Lösche e
10     Wähle beliebige Kante e = (v, w) mit minimaler Kreuzungszahl
11     if find(v) ≠ find(w) then
12         Verschiebe e von E nach E' und lösche alle Kanten aus E, die e kreuzen
13         Union(Find(v),Find(w))
14     else
15         Lösche e.
16 return E'
```

schrieben und bieten amortisierte Laufzeiten von $O(\log n)$ für Remove und ExtractMin sowie $O(1)$ für DecreaseKey. Weiterhin kann die Initialisierung in linearer Zeit durchgeführt werden. Es bleibt zu diskutieren, wie oft die Operationen jeweils verwendet werden. Offensichtlich wird jede Kante vom Greedy-Algorithmus genau einmal aus der Datenstruktur entfernt, entweder über Remove oder über ExtractMin. Für m Kanten ergibt sich folglich eine Laufzeit von $O(m \log m)$. Wird eine Kante entfernt, müssen die Schlüsselwerte der sie kreuzenden Kanten aktualisiert werden. Die Anzahl dieser Operationen ist jedoch beschränkt durch die Anzahl der Kreuzungen k. Die obere Laufzeitschranke $O(k)$ folgt sofort. Insgesamt tragen die Operationen des Fibonacci-Heaps also eine Laufzeit von $O(k + m \log m)$ bei.

Als letztes betrachten wir die Zeitkomplexität, die durch Verwendung von Union-Find entsteht. Wie in Abschnitt 2.3 erwähnt benötigt Union-Find zur Abarbeitung von $n - 1$ Union- und m Find-Operationen eine Laufzeit von $O(n + m\alpha(n))$. Es ist leicht zu sehen, dass für jede Kante die Aufrufe der Funktionen Union und Find mit dem Entfernen der Kante zusammenfallen. Beide Operationen müssen, für eine Kantenanzahl m, also höchstens $O(m)$ mal durchgeführt werden. Mit den Kosten für die Initialisierung, die sich bei n Knoten auf $O(n)$ belaufen, ergibt sich für die Verwendung von Union-Find eine Laufzeit von $O(n + m + m\alpha(m)) \subseteq O(n + m\alpha(m))$.

Aus der Analyse der Laufzeiten der drei Hauptbestandteile ergibt sich mit der Domi-

nanz von $O(m \log m)$ über $O(m\alpha(m))$ die Zeitkomplexität des Greedy-Algorithmus.

Lemma 4.1. *Für eine Instanz mit n Knoten, m Kanten und k Kantenkreuzungen hat der Greedy-Algorithmus eine Laufzeit von $O(n + k + m \log m)$.*

Bemerkung. Für einen Graphen ohne isolierte Knoten gilt $O(n) \subseteq O(m)$, damit erreicht der Greedy-Algorithmus für derartige Eingaben eine Laufzeit von $O(k + m \log m)$. Es ist anzunehmen, dass diese Laufzeit unter allen Algorithmen, die gültige Lösungen produzieren, optimal ist, da diese Zeit bereits benötigt wird, um sämtliche Kantenkreuzungen zu berechnen.

4.2 Reverse Greedy

Wir betrachten nun eine Variante des Greedy-Algorithmus, den *Reverse-Greedy*-Algorithmus. Auch hier werden, dem Gedanken eines Greedy-Ansatzes entsprechend, iterativ Entscheidungen getroffen, die zum jeweiligen Zeitpunkt den meisten Erfolg versprechen. Im Gegensatz zu dem im vorherigen Abschnitt besprochenen Algorithmus wird beim Reverse-Greedy-Vorgehen jedoch nicht die aktuell minimal gekreuzte Kante selektiert, sondern die jeweils maximal gekreuzt Kante gelöscht. Dieses Verfahren entspringt ähnlichen Überlegungen wie denen, die dem Greedy-Algorithmus zugrunde liegen. Statt jedoch diejenige Kante zu wählen, die den weiteren Verlauf am wenigsten negativ beeinflusst, wird hier der umgekehrte Ansatz verfolgt. Es wird also stets die Kante ausgeschlossen, deren Präsenz am stärksten mit dem restlichen Verfahren interferiert. Wird auf solche Weise die letzte Kreuzung einer Kante entfernt, kann diese bedenkenlos selektiert werden. Dadurch können Cluster entstehen, bevor alle Kantenkreuzungen entfernt sind. Daher ist es auch hier sinnvoll, nach jedem Schritt alle unselektierten Kanten innerhalb eines bestehenden Clusters zu löschen, damit diese den Auswahlprozess nicht mehr beeinflussen. Mit der erneuten Nutzung von Union-Find formulieren wir den folgenden Algorithmus 2.

Laufzeit Die Laufzeit des Reverse-Greedy-Algorithmus stimmt in weiten Teilen mit der des Greedy-Ansatzes überein. Auch hier zerfällt die Zeitkomplexität in drei Hauptteile, nämlich die Berechnung der Kantenkreuzungen, die Identifizierung einer maximal gekreuzten Kante und die Verwendung von Union-Find.

Während die Berechnung der Kantenkreuzungen und die Benutzung von Union-Find mit ähnlichen Argumenten wie für den Greedy-Algorithmus die gleiche asymptotische Laufzeit beitragen wie oben, bedarf die Identifizierung einer maximal gekreuzten Kante genauerer Analyse.

Für den Reverse-Greedy-Algorithmus würden wir bei Verwendung eines Fibonacci-Heaps die Funktion ExtractMax benötigen, die jedoch nicht effizient unterstützt wird. Die Benutzung eines Fibonacci-Heaps scheidet also aus. Stattdessen nutzen wir an dieser Stelle einen binären Suchbaum, der, bei richtiger Implementierung, die Operationen ExtractMax, Remove und DecreaseKey in logarithmischer Zeit zur Verfügung stellt. Auch hier muss jede der m Kanten genau einmal aus der Datenstruktur entfernt werden, es ergibt sich eine Laufzeit von $O(m \log m)$. Für die Behandlung der k Kantenkreuzungen

Algorithmus 2 : Reverse Greedy Cluster Minimierung(Graph $G = (V, E)$)

Eingabe : Graph $G = (V, E)$
Ausgabe : $E' \subseteq E$, so dass E' keine Kreuzungen enthält und möglichst wenige Zusammenhangskomponenten in $G' = (V, E')$ verbleiben

```
1  E' ← ∅
2  foreach v ∈ V do MakeSet(v)
3  while E ≠ ∅ do
4      foreach e = (v, w) aus E ohne Kreuzungen do
5          Verschiebe e von E nach E'
6          Union(Find(v),Find(w))
7      foreach e = (v, w) aus E do
8          if Find(v) = Find(w) then
9              Lösche e
10     Lösche beliebige Kante mit maximaler Kreuzungszahl
11 return E'
```

müssen k DecreaseKey-Operationen durchgeführt werden, hierfür muss eine Laufzeit von $O(k \log k)$ aufgewandt werden.

Aus diesen Überlegungen leiten wir die Laufzeit des Reverse-Greedy-Algorithmus ab.

Lemma 4.2. *Für eine Instanz mit n Knoten, m Kanten und k Kantenkreuzungen hat der Reverse-Greedy-Algorithmus eine Laufzeit von $O(n + k \log k + m \log m)$.*

4.3 Reverse 1-plane Greedy

Akitaya et al. [HAC+19] zeigten, dass sich Cluster-Minimierung für 1-planare Graphen effizient lösen lässt. Sie argumentierten, dass ein einfacher Greedy-Algorithmus das Problem bereits exakt löst. Mithilfe dieses Resultats formulieren wir eine Variation des Reverse-Greedy-Algorithmus, den *Reverse-1-plane-Greedy*-Algorithmus (kurz: R1PG). Bei dieser gehen wir zunächst vor wie beim Reverse-Greedy-Algorithmus. Statt jedoch so lange Kanten zu löschen, bis der übrige Graph planar ist, stoppen wir bereits, sobald 1-Planarität erreicht wird. Die übrige Instanz lösen wir exakt.

Dieser Algorithmus wurde mit der Hoffnung entwickelt, die Resultate des Reverse-Greedy-Algorithmus zu verbessern. Tatsächlich sind die beiden Verfahren jedoch äquivalent. Dies ist über folgendes Argument zu erklären.

Bis zum Erreichen der 1-Planarität verhalten sich die Algorithmen Reverse-Greedy und R1PG identisch. Erst im weiteren Vorgehen unterscheiden sie sich. Während der Reverse-Greedy-Algorithmus weiterhin iterativ aktuell maximal gekreuzte Kanten entfernt, nutzt R1PG das Verfahren von Akitaya et al. [HAC+19], um die übrige Instanz exakt zu lösen. Dieses Verfahren entspricht einem einfachen Greedy-Ansatz, es wird also iterativ eine aktuell minimal gekreuzte Kante selektiert. Da jede verbliebene Kante jedoch höchstens

Algorithmus 3 : Reverse 1-plane Greedy Cluster Minimierung(Graph $G = (V, E)$)

Eingabe : Graph $G = (V, E)$
Ausgabe : $E' \subseteq E$, so dass E' keine Kreuzungen enthält und möglichst wenige Zusammenhangskomponenten in $G' = (V, E')$ verbleiben

```
1  E' ← ∅
2  foreach v ∈ V do MakeSet(v)
3  while G ist nicht 1-planar do
4      foreach e = (v, w) aus E ohne Kreuzungen do
5          Verschiebe e von E nach E'
6          Union(Find(v),Find(w))
7      foreach e = (v, w) aus E do
8          if Find(v) = Find(w) then
9              Lösche e
10     Lösche beliebige Kante mit maximaler Kreuzungszahl
11 Löse übrige Instanz exakt.
12 return E'
```

einmal gekreuzt ist, spielt es keine Rolle, ob eine solche Kante selektiert wird, oder ob stattdessen eine Kante gelöscht und die sie kreuzende Kante selektiert wird. Damit liefern beide Algorithmen, bis auf potenzielle Unterschiede bei der zufälligen Wahl von Kanten, das gleiche Ergebnis. In der restlichen Arbeit wird daher der R1PG-Algorithmus nicht weiter betrachtet.

5 Theoretische Analyse der Algorithmen

In diesem Kapitel widmen wir uns der Frage nach der Performanz der vorgestellten Heuristiken. Dabei fokussieren wir uns auf die Differenz zwischen einer optimalen und einer von den Algorithmen gefunden Lösung. Dafür konstruieren wir eine Familie von Probleminstanzen, auf der sowohl der Greedy- als auch der Reverse-Greedy-Ansatz beliebig schlechte Ergebnisse liefern. Weiterhin betrachten wir ein Beispiel für eine Graphenfamilie, für die mit der Greedy-Heuristik Lösungen konstanter Größe gefunden werden, mit dem Reverse-Greedy-Algorithmus jedoch nur Lösungen mit einer linearen Anzahl an Clustern.

5.1 Greedy

Wir wenden uns zunächst dem Greedy-Algorithmus zu. Um seine theoretischen Grenzen auszuloten, formulieren wir einen Satz, der die Existenz eines konstanten Approximationsfaktors ausschließt.

Satz 5.1. *Es existiert eine Familie von Graphen $(G_k)_{k\in\mathbb{N}^+}$, für die eine optimale Lösung von Cluster-Minimierung konstante Größe hat, der Greedy-Algorithmus jedoch nur Lösungen mit linearer Größe findet.*

Der Beweis dieser Aussage erfolgt, indem zunächst eine Konstruktion einer solchen Familie von Graphen angegeben wird und im Anschluss die genannten Eigenschaften gezeigt werden.

Definition 5.2. Sei $k \in \mathbb{N}^+$. Fixiere die Punkte $u_k = (k, k+1)$ und $v_k = (k+1, k)$. Definiere V_k wie folgt:

$$\begin{aligned} X_k &:= \{x_i = (i, 0) \mid i \in 0, \ldots, k\} \\ Y_k &:= \{y_i = (0, i) \mid i \in 1, \ldots, k\} \\ V_k &:= X_k \cup Y_k \cup \{u_k, v_k\} \end{aligned}$$

Man beachte, dass $|Y_k| = |X_k| - 1$ gilt. Die Kantenmengen E_k setzen sich wie folgt zusammen:

$$\begin{aligned} A_k &:= \{\{x_i, u\} \mid i \in 0, \ldots, k\} \\ B_k &:= \{\{y_i, v\} \mid i \in 1, \ldots, k\} \\ C_k &:= \{\{x_i, x_{i+1}\} \mid i \in 0, \ldots, k-1\} \\ E_k &:= A_k \cup B_k \cup C_k \end{aligned}$$

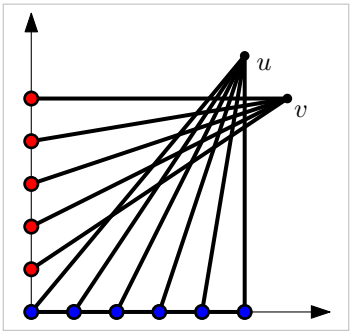


Abb. 5.1: Der nach Definition 5.2 konstruierte Graph G_5.

Es werden also alle Punkte aus X_k mit u_k verbunden (A_k), alle Punkte aus Y_k sind zu v_k adjazent (B_k). Weiterhin bilden die Knoten aus X_k einen Pfad (C_k). Die Graphen G_k ergeben sich über $G_k = (V_k, E_k)$. Abbildung 5.1 zeigt den Graphen G_5.

Lemma 5.3. *Für jeden Graphen $G_k, k \in \mathbb{N}^+$, der nach Definition 5.2 konstruiert wurde, hat die optimale Lösung für Cluster-Minimierung Größe 3.*

Beweis. Selektiere alle Kanten aus B_k sowie alle Kanten aus C_k. Es ergeben sich die Cluster X_k,$Y_k \cup \{v\}$ und $\{u\}$. □

Lemma 5.4. *Für jeden Graphen $G_k, k \in \mathbb{N}^+$, der nach Definition 5.2 konstruiert wurde, gibt der Greedy-Algorithmus eine Lösung der Größe $k + 2$ aus.*

Beweis. Der Greedy-Algorithmus selektiert zunächst alle Kanten aus C_k, da diese ungekreuzt sind. Es ergibt sich das vorläufige Cluster X_k. Da nun weder ungekreuzte Kanten noch Kanten innerhalb bestehender Cluster verbleiben, wählt der Algorithmus eine Kante mit minimaler Kreuzungszahl. Jede Kante aus A_k kreuzt jede Kante aus B_k und umgekehrt. Da $|A_k| = k + 1 = |B_k| + 1$ gilt, wird eine beliebige Kante e aus A_k selektiert. Alle Kanten aus B_k werden gelöscht, da sie e schneiden. Die verbleibenden Kanten aus A_k sind nun ungekreuzt und verlaufen innerhalb eines bereits bestehenden Clusters, der Algorithmus terminiert also. Die ausgegebenen Cluster sind $X_k \cup \{u\}$, $\{v\}$, sowie $\{y_i\}$ für $i \in 1, \ldots, k$. □

Über Lemma 5.3 und Lemma 5.4 ergibt sich die Aussage von Satz 5.1.

5.2 Reverse Greedy

Betrachten wir nun die Performanz des Reverse-Greedy-Algorithmus. Auch für diesen gilt ein Analogon zu Satz 5.1.

Satz 5.5. *Es existiert eine Familie von Graphen $(G_k)_{k \in \mathbb{N}^+}$, für die eine optimale Lösung von Cluster-Minimierung konstante Größe hat, der Reverse-Greedy-Algorithmus jedoch nur Lösungen mit linearer Größe findet.*

Der Beweis dieses Satzes wird dem vorherigen Abschnitt entsprechend geführt. Dafür verwenden wir die Konstruktion aus Definition 5.2.

Lemma 5.6. *Für jeden Graphen $G_k, k \in \mathbb{N}^+$, der nach Definition 5.2 konstruiert wurde, gibt der Reverse-Greedy-Algorithmus eine Lösung der Größe $k+2$ aus.*

Beweis. Dem Vorgehen des Greedy-Algorithmus folgend selektiert der Reverse-Greedy-Algorithmus zunächst alle ungekreuzten Kanten, also alle Kanten aus C_k. Da nun keine ungekreuzten Kanten und keine Kanten innerhalb bestehender Cluster in E_k verbleiben, löscht der Algorithmus im nächsten Schritt eine beliebige maximal gekreuzte Kante. Analog zur Argumentation im Beweis von Lemma 5.4 ist jede Kante aus A_k genau k mal gekreuzt, jede Kante aus B_k weist eine Kreuzungszahl von $k+1$ auf. Dementsprechend wird eine beliebige Kante $e \in B_k$ vom Algorithmus gelöscht. Da e jede Kante aus A_k, aber keine Kante aus B_k kreuzt, ist die nächste maximal gekreuzte Kante wiederum in B_k zu finden. Offensichtlich werden nun iterativ alle Kanten aus B_k gelöscht. Es verbleiben die Kanten aus A_k, die alle ungekreuzt sind und daher selektiert werden. Im Anschluss terminiert der Algorithmus, die ausgegebenen Cluster entsprechen dem Ergebnis des regulären Greedy-Algorithmus. □

5.3 Vergleich zwischen den Algorithmen

Wir haben gezeigt, dass sowohl der Greedy-Algorithmus als auch der Reverse-Greedy-Algorithmus im schlechtesten Fall um einen linearen Faktor von der optimalen Lösung abweichen. Nun wollen wir die theoretische Performanz der beiden Verfahren miteinander vergleichen. Dazu konstruieren wir eine Familie von Problemen, für die der Greedy-Algorithmus Lösungen mit konstanter Größe findet, die vom Reverse-Greedy-Algorithmus gefundenen Lösungen jedoch lineare Größe haben. Dabei nutzen wir aus, dass beim Reverse-Greedy-Ansatz zunächst eine maximal gekreuzte Kante gelöscht wird, indem wir einen Graphen wählen, dessen optimale Lösungen eine solche Kante enthalten müssen.

Satz 5.7. *Es existiert eine Familie $(G_k)_{k\in\mathbb{N}}$ von geometrischen Graphen, so dass der Greedy-Algorithmus angewendet auf G_k eine Lösung konstanter Größe liefert, während der Reverse-Greedy-Algorithmus eine Lösung linearer Größe zurückgibt.*

Wir beschreiben die Konstruktion dieser Graphenfamilie anhand ihrer Darstellung in Abbildung 5.2.

Sei $k \in \mathbb{N}$. Wir beginnen zunächst mit zwei waagrechten, parallelen Pfaden der Länge $k+3$, also mit jeweils $k+4$ Knoten. Wir bezeichnen den oberen Pfad mit A_k und den unteren Pfad mit A'_k. Anschließend platzieren wir über jeder Kante aus A_k und unter jeder Kante aus A'_k einen Knoten und verbinden die jeweils gegenüberliegenden Knoten mit Kanten. Diese Knoten fassen wir in den beiden Mengen B_k und B'_k zusammen. Diese beiden Mengen haben nach Konstruktion jeweils die Mächtigkeit $k+3$. Oberhalb von B_k und unterhalb von B'_k fügen wir wiederum die zu A_k und A'_k parallelen Pfade C_k und C'_k mit Länge $2k+3$ ein. Die Knoten aus B_k und B'_k werden mit den Knoten aus C_k und

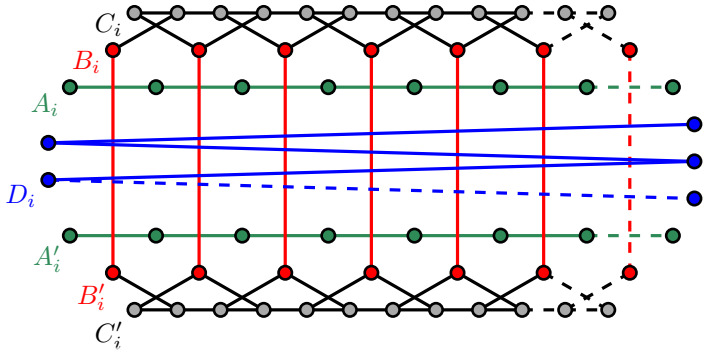


Abb. 5.2: Die Konstruktion der Graphenfamilie aus Satz 5.5.

C'_k wie in Abbildung 5.2 verbunden. Zum Schluss erstellen wir noch einen Pfad D_k der Länge k. Dessen Knoten platzieren wir so, dass jede seiner Kanten alle Kanten zwischen Knoten aus B_k und B'_k schneidet.

Abbildung 5.2 zeigt den Graphen G_4 sowie in gestichelten Linien seine Erweiterung zum Graphen G_5.

Lemma 5.8. *Sei $k \in \mathbb{N}$. Angewendet auf den Graphen G_k findet der Greedy-Algorithmus eine Lösung mit Größe 7 für Cluster-Minimierung.*

Beweis. Der Greedy-Algorithmus wählt zunächst alle ungekreuzten Kanten, also die Kanten der Pfade C_k und C'_k, aus. Im verbleibenden Graphen ist die minimale Kreuzungszahl unter allen Kanten 1. Der Algorithmus wählt willkürliche eine von zwei Möglichkeiten. Entweder wird eine Kante auf einem der Pfade A_k und A'_k selektiert, oder eine derjenigen Kanten, die die Mengen B_k und B'_k mit den Pfaden C_k und C'_k verbinden. In beiden Fällen schneidet die selektierte Kante e genau eine andere Kante. Im ersten Fall ist dies eine Kante, die zwischen den Mengen B_k und B'_k verläuft. Nach Entfernung dieser ist die zu e korrespondierende Kante in A_k beziehungsweise A'_k ungekreuzt und wird als nächstes selektiert. Im zweiten Fall erfüllt die kreuzende Kante die gleiche Funktion wie e.

Es ist leicht zu sehen, dass auf diese Weise iterativ alle Kanten aus A_k und A'_k sowie ein Teil der Verbindungskanten zwischen B_k und C_k beziehungsweise B'_k und C'_k selektiert werden. Die Selektion dieser Verbindungskanten hat zur Folge, dass, bis auf jeweils eine Ausnahme, alle Knoten aus B_k und B'_k mit den Pfaden C_k beziehungsweise C'_k verbunden sind. Alle Kanten zwischen den Mengen B_k und B'_k werden bei diesem Verfahren gelöscht. Da nun alle Kanten des Pfades D_k ungekreuzt sind, werden sie selektiert. Im Anschluss terminiert der Algorithmus.

Die Pfade A_k, A_k und D_k repräsentieren jeweils ein Cluster. Weiterhin werden durch die Pfade C_k und C'_k sowie den angeschlossenen Knoten Cluster gebildet. Jeweils ein Knoten aus B_k und B'_k verbleiben in eigenen Clustern. □

Bemerkung. Da iterativ alle Kanten zwischen B_k und B'_k gelöscht werden, sinkt die Kreuzungszahl der Kanten in D_k. Es ist also möglich, dass Teile dieser Kanten vor den

letzten unselektierten Kanten aus A_k und A'_k ausgewählt werden. Dies ändert jedoch nichts an der ausgegebenen Lösung.

Lemma 5.9. *Sei $k \in \mathbb{N}$. Angewendet auf den Graphen G_k findet der Reverse-Greedy-Algorithmus eine Lösung mit Größe $k+7$ für Cluster-Minimierung.*

Beweis. Wie beim regulären Greedy-Algorithmus werden zunächst die Kanten der Pfade C_k und C'_k selektiert. Da nun weder ungekreuzte Kanten noch Kanten innerhalb bestehender Cluster verbleiben, wird eine maximal gekreuzte Kante gelöscht.

Die Kanten des Pfads D_k sind genau $k+3$ mal gekreuzt, die Kanten zwischen den Mengen B_k und B'_k genau $k+2$ mal. Es wird also eine Kante aus D_k gelöscht. Da die Kreuzungszahl der verbliebenen Kanten aus D_k dadurch nicht reduziert wird und keine ungekreuzten Kanten entstehen, werden alle Kanten aus D_k iterativ gelöscht.

Im Anschluss sind die Verbindungskanten zwischen den Mengen B_k und B'_k zweifach gekreuzt. Eine davon wird also gelöscht. Die beiden sie schneidenden Kanten aus den Mengen A_k und A'_k sind nun ungekreuzt und werden selektiert. Dieses Verfahren wird iterativ fortgesetzt.

Als letztes werden iterativ Kanten zwischen B_k und C_k sowie B'_k und C'_k gelöscht beziehungsweise selektiert.

Die ausgegebenen Cluster entsprechen denen, die vom Greedy-Algorithmus gefunden werden, allerdings bildet hier jeder der $k+1$ Knoten des Pfades D_k ein eigenes Cluster. Es folgt die Aussage des Lemmas. □

Mit Lemma 5.8 und Lemma 5.9 ergibt sich die Aussage von Satz 5.7.

Bemerkung. Trotz des Resultats aus Satz 5.7 hat der Reverse-Greedy-Algorithmus in Experimenten tendenziell bessere Ergebnisse geliefert als der Greedy-Ansatz. [Gei20]

6 Exakter Algorithmus

In diesem Kapitel formulieren wir ein ganzzahliges lineares Programm, welches Cluster-Minimierung exakt löst. Dieses beruht auf der Modellierung des Problems als Flussnetzwerk. Die Knoten und Kanten des Netzwerks entsprechen dabei den Knoten und Kanten des eingegebenen Graphen. Die Grundidee ist dabei, jeden Knoten entweder als Quelle oder als Senke zu betrachten. Dabei soll jedes Cluster genau eine Senke enthalten. Ist ein Knoten eine Quelle, erzeugt er einen Nettoausfluss von genau 1. Eine Senke nimmt den erzeugten Fluss aller Quellen des entsprechenden Clusters auf. Dabei dürfen nur selektierte Kanten Fluss transportieren. Die Anzahl an Clustern wird schließlich minimiert, wenn die Anzahl an Senken minimiert wird.

Da die Berechnung der Kantenkreuzungen die Komplexität des ILP nicht erhöht, gehen wir im Folgenden davon aus, dass die Kantenkreuzungen bekannt sind. Die Entkopplung ihrer Berechnung vom ILP ermöglicht es außerdem, das Programm auch für nicht geradlinige Graphen zu benutzen.

6.1 Variable

Wir beginnen mit der Beschreibung der verwendeten Variablen. Zentral für das Problem Cluster-Minimierung ist die Selektion von Kanten. Dementsprechend benötigen wir Variablen, die repräsentieren, ob eine Kante $e = \{u, v\}$ selektiert wurde. Wir führen also binäre Variablen x_{uv} ein, denen genau dann der Wert 1 zugewiesen wird, wenn die entsprechende Kante selektiert wird. Dabei ist zu beachten, dass eine Kante zwar ungerichtet ist, die Indizes einer Variable jedoch nicht ohne Weiteres vertauscht werden können. Daher korrespondiert jede Kante $e = \{u, v\}$ zu zwei stets gleichwertigen Variablen x_{uv} und x_{vu}.

Weiterhin benötigen wir Variablen, die für jeden Knoten seinen Status als Quelle beziehungsweise Senke festlegen. Wir führen also für jeden Knoten v die beiden binären Variablen s_v und t_v ein, die genau dann den Wert 1 erhalten, wenn v eine Quelle beziehungsweise Senke ist. Offensichtlich kann nur genau eine der beiden Variablen gesetzt sein.

Als nächstes definieren wir Variablen, die den Status der Cluster beschreiben. Da im Vorfeld nicht klar ist, wie viele Cluster eine potenzielle Lösung enthalten wird, ist nicht sofort klar, wie viele Variablen zu definieren sind. Da allerdings die Anzahl an Clustern in einer Lösung nicht die Anzahl an Knoten im eingegebenen Graphen überschreiten kann, orientieren wir uns an dieser oberen Schranke. Wir legen also eine der Knotenanzahl entsprechende Menge an potenziellen Clustern fest und weisen jedem Cluster c davon eine binäre Variable a_c zu. Diese soll genau dann den Wert 1 annehmen, wenn das entsprechende Cluster in der Lösung mindestens einen Knoten enthält. An der Anzahl

der gesetzten Variablen a_c lässt sich also die Größe einer gefundenen Lösung ableiten. Weiterhin setzen wir Knoten und Cluster zueinander in Beziehung, indem für jede Kombination eines Knotens v und eines Clusters c eine Variable t_{vc} eingeführt wird. Diese zeigt an, ob v die eindeutige Senke im Cluster c darstellt.

Schlussendlich werden noch Variablen f_{uv} benötigt, die für jedes Knotenpaar u, v den dazwischen verlaufenden Fluss repräsentieren. Diese definieren wir nur für adjazente Knoten. Auch hier ist zu beachten, dass für jede Kante $\{u, v\}$ sowohl die Variable f_{uv} als auch die Variable f_{vu} existiert. Da kein negativer Fluss existieren kann, müssen alle Variablen f nichtnegative Werte annehmen.

Zusammenfassend ergeben sich für unser ganzzahliges lineares Programm bei Eingabe eines Graphen $G = (V, E)$ mit Knotenanzahl $|V| = n$ die folgenden Variablen.

$$
\begin{aligned}
x_{uv} &\in \{0,1\} && \forall \{u,v\} \in E \\
a_c &\in \{0,1\} && \forall c \in \{1,\dots,n\} \\
t_{vc} &\in \{0,1\} && \forall v \in V, c \in \{1,\dots,n\} \\
t_v &\in \{0,1\} && \forall v \in V \\
s_v &\in \{0,1\} && \forall v \in V \\
f_{uv} &\geq 0 && \forall \{u,v\} \in E
\end{aligned}
$$

6.2 Nebenbedingungen

Wir wenden uns nun der Beschreibung der Nebenbedingungen zu. Zunächst definieren wir die Regeln, die das Verhalten der Kanten beschreiben. Wie bereits erwähnt korrespondiert die Selektion einer Kante $\{u, v\}$ mit zwei Variablen, x_{uv} und x_{vu}. Es ist offensichtlich, dass diese stets den gleichen Wert annehmen müssen.

$$x_{uv} = x_{vu} \qquad \forall u, v \in V$$

Weiterhin darf von einem Paar gekreuzter Kanten höchstens eine selektiert werden.

$$x_{ab} + x_{cd} \leq 1 \qquad \text{für gekreuzte Kanten } \{a, b\} \text{ und } \{c, d\}$$

Das Verhalten der Knoten ist über die folgenden Bedingungen geregelt. Nach Konstruktion ist jeder Knoten entweder eine Quelle oder eine Senke.

$$s_v = 1 - t_v \qquad \forall v \in V$$

Ein Knoten ist genau dann eine Senke, wenn er die eindeutige Senke eines Clusters ist. Ein Knoten kann die Senke höchstens eines Clusters sein.

$$t_v = \sum_{c \in \{1,\dots,n\}} t_{vc} \leq 1 \qquad \forall v \in V$$

Ein Cluster hat genau dann mindestens einen Knoten, wenn es eine dedizierte Senke hat. Ein Cluster kann höchstens eine Senke enthalten.

$$\sum_{v \in V} t_{vc} = a_c \leq 1 \qquad \forall c \in \{1, \ldots, n\}$$

Als letztes betrachten wir die Bedingungen, die das Verhalten des Flusses beschreiben. Wie bereits erwähnt dürfen nur selektierte Kanten Fluss transportieren. Der Parameter n ist an dieser Stelle gewählt, weil er eine natürliche obere Schranke für den Durchlauf jeder Kante bildet.

$$f_{uv} \leq n \cdot x_{uv} \qquad \forall \{u, v\} \in E$$

Die letzte Regel beschränkt den Nettozu- beziehungsweise abfluss für jeden Knoten. Wie bereits erwähnt muss jede Quelle einen Fluss von 1 erzeugen. Das Verhalten der Senken ist weniger eindeutig geregelt, klar ist jedoch, dass sie keinen positiven Nettoabfluss erzeugen dürfen. Eine Senke, die der einzige Knoten eines Clusters ist, erreicht einen ausgeglichenen Fluss. Da der gesamte Fluss des Netzwerks durch die Anzahl der Knoten von oben beschränkt ist, ergeben sich die folgenden Grenzen.

$$s_v - n \cdot t_v \leq \sum_{\{u,v\} \in E} (f_{vu} - f_{uv}) \leq s_v \qquad \forall v \in V$$

6.3 Ein ILP für Cluster-Minimierung

Mithilfe der oben beschriebenen Variablen und Nebenbedingungen sind wir nun bereit, das ganzzahlige lineare Programm zu formulieren. Durch das Ziel, eine Belegung mit möglichst wenigen Clustern zu finden ergibt sich die Zielfunktion wie folgt.

$$\text{minimiere} \sum_{c \in \{1,\ldots,n\}} a_c$$

Unter den Nebenbedingungen

$$\begin{aligned}
x_{uv} &= x_{vu} && \forall u, v \in V \\
x_{ab} + x_{cd} &\leq 1 && \text{für gekreuzte Kanten } \{a, b\} \text{ und } \{c, d\} \\
s_v &= 1 - t_v && \forall v \in V \\
t_v &= \sum_{c \in \{1,\ldots,n\}} t_{vc} \leq 1 && \forall v \in V \\
\sum_{v \in V} t_{vc} &= a_c \leq 1 && \forall c \in \{1, \ldots, n\} \\
f_{uv} &\leq n \cdot x_{uv} && \forall \{u, v\} \in E \\
s_v - n \cdot t_v &\leq \sum_{\{u,v\} \in E} (f_{vu} - f_{uv}) \leq s_v && \forall v \in V
\end{aligned}$$

Mit den Variablen

$$
\begin{aligned}
x_{uv} &\in \{0,1\} && \forall \{u,v\} \in E \\
a_c &\in \{0,1\} && \forall c \in \{1,\dots,n\} \\
t_{vc} &\in \{0,1\} && \forall v \in V, c \in \{1,\dots,n\} \\
t_v &\in \{0,1\} && \forall v \in V \\
s_v &\in \{0,1\} && \forall v \in V \\
f_{uv} &\geq 0 && \forall \{u,v\} \in E
\end{aligned}
$$

Korrektheit Wir zeigen die korrekte Arbeitsweise des ganzzahligen linearen Programms. Sei $G = (V, E)$ ein Graph, auf dem Cluster-Minimierung ausgeführt werden soll. Wir argumentieren zunächst, dass eine potenzielle Lösung zu einer korrekten Belegung des ILP mit gleicher Größe korrespondiert. Im Anschluss zeigen wir, dass eine vom ILP gefundene Lösung eine korrekte Lösung von Cluster-Minimierung mit gleicher Clusteranzahl induziert.

Sei zu einer Lösung von Cluster-Minimierung E' die Menge der selektierten Kanten und C die Menge der resultierenden Cluster. Wir selektieren im ILP alle Kanten eines minimalen Spannwalds S von $G' = (V, E')$. Offensichtlich entsprechen die Zusammenhangskomponenten von S genau den Clustern in C. Wähle zu jedem Cluster aus C einen beliebigen Knoten aus diesem Cluster als Senke und bestimme diesen als Wurzel des entsprechenden Spannbaums. Dann ist jeder Knoten entweder eine Senke oder über genau einen Pfad mit einer Senke verbunden. Weise jeder selektierten Kante einen Flussdurchlauf zu, der der Größe des dort befestigten Teilbaums entspricht. Mit beliebiger Indizierung der Cluster ergibt sich so eine korrekte Variablenbelegung des ILP. Weiterhin entspricht der Wert der Zielfunktion $|C|$.

Nehmen wir andererseits eine Lösung des ILP mit Wert k an. Wir fassen alle darin selektierten Kanten in der Menge E' zusammen. Nach Konstruktion enthält E' kein Paar gekreuzter Kanten. Weiterhin ist jeder Knoten in $G' = (V, E')$ mit genau einer Senke verbunden oder selbst eine Senke. Da die Lösung des ILP den Wert k hat und damit genau k Senken enthält, ist G' eine valide Lösung für Cluster-Minimierung und enthält genau k Cluster.

7 Fazit und Ausblick

In dieser Arbeit wurde das Problem betrachtet, zu einem gegebenen geometrischen Graphen einen planaren Subgraphen mit einer minimalen Anzahl an Zusammenhangskomponenten zu finden. In Kapitel 3 wurde die NP-Vollständigkeit von Cluster-Minimierung bewiesen. Die Kapitel 4 und 5 beschäftigten sich mit der Definition und Analyse mehrerer Heuristiken für dieses Problem. Für jede vorgeschlagene Heuristik wurde jedoch die Existenz eines konstanten Approximationsfaktors ausgeschlossen. In Kapitel 6 wurde durch Modellierung als ganzzahliges lineares Programm ein exakter Algorithmus für das Problem Cluster-Minimierung hergeleitet.

Aus den Ergebnissen dieser Arbeit ergeben sich weitere interessante Fragestellungen. Die Abwesenheit konstanter Approximationsfaktoren für alle Heuristiken werfen die Frage nach Verbesserungspotenzial für diese Algorithmen auf. Eine mögliche Verbesserung des Greedy-Algorithmus, die bei der Forschung für diese Arbeit entwickelt wurde, besteht darin, Kanten, die zwischen den gleichen bestehenden Clustern verlaufen, bei der Berechnung der Kantenkreuzungen zusammenzufassen. Für diese Modifikation wurde jedoch auch eine Graphenfamilie entdeckt, die einen konstanten Approximationsfaktor ausschließt. Abbildung 7.1 skizziert deren Konstruktion. Die Frage, ob ein Algorithmus existiert, der Cluster-Minimierung bis auf einen konstanten Faktor approximiert, bleibt also weiterhin offen.

Das Problem Kanten-Minimierung bleibt weitgehend unerforscht. Zwar konnte in dieser Arbeit die NP-Vollständigkeit gezeigt werden, doch auch hier stellt sich die Frage nach einer Approximation mit konstantem Faktor.

In Anlehnung an die Motivation des Problems Cluster-Minimierung über das Clustering von geographischen Daten können weitere, eher praxisorientierte Ansätze formuliert werden. Wie im Algorithmus DBSCAN kann es zum Beispiel sinnvoll sein, Datenpunkte,

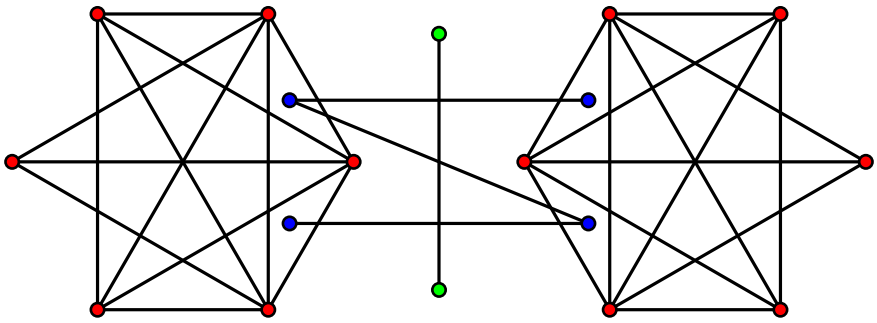


Abb. 7.1: Ein Vertreter der Graphenfamilie, welche einen konstanten Approximationsfaktor für den verbesserten Greedy-Algorithmus ausschließt.

die sich nicht leicht Clustern zuordnen lassen, als Rauschen zu klassifizieren und von der Betrachtung auszuschließen. Eine Erweiterung dieses Ansatzes wäre es, jeden Datenpunkt mit einer „Wichtigkeit“ zu versehen und diese in die Klassifikation als Rauschen einfließen zu lassen. Im Beispiel aus der Einleitung, in dem die Geschäfte einer Stadt als Datenpunkte dienen, könnte zum Beispiel die Verkaufsfläche als Wichtigkeit verstanden werden. Eine weitere Möglichkeit wäre es, die Häufigkeit, mit der Geschäfte einer Kategorie auftreten, in die Bewertung einfließen zu lassen.

Literaturverzeichnis

[ALT20] Hugo A. Akitaya, Maarten Löffler und Csaba D. Tóth: Multi-colored spanning graphs. *Theoretical Computer Science*, 2020. https://doi.org/10.1016/j.tcs.2020.04.022.

[AV07] David Arthur und Sergei Vassilvitskii: K-Means++: The advantages of careful seeding. In: *Proceedings of the Eighteenth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA'07)*, Seite 1027–1035, 2007.

[Bal95] Ivan J. Balaban: An optimal algorithm for finding segments intersections. In: *Proceedings of the 11th Annual ACM Symposium on Computational Geometry*, Seiten 211–219, 1995. https://doi.org/10.1145/220279.220302.

[BFK+15] Sergey Bereg, Krzysztof Fleszar, Philipp Kindermann, Sergey Pupyrev, Joachim Spoerhase und Alexander Wolff: Colored non-crossing Euclidean Steiner forest. In: Khaled M. Elbassioni und Kazuhisa Makino (Herausgeber): *Proc. 26th International Symposium on Algorithms and Computation (ISAAC 2015)*, Band 9472 der Reihe *Lecture Notes in Computer Science*, Seiten 429–441. Springer, 2015. https://doi.org/10.1007/978-3-662-48971-0_37.

[Bie20] Therese Biedl: Segment representations with small resolution. *Information Processing Letters*, 153(105868), 2020. https://doi.org/10.1016/j.ipl.2019.105868.

[BJYZ11] Sergey Bereg, Minghui Jiang, Boting Yang und Binhai Zhu: On the red/blue spanning tree problem. *Theoretical Computer Science*, 412(23):2459–2467, 2011. https://doi.org/10.1016/j.tcs.2010.10.038.

[CE92] Bernard Chazelle und Herbert Edelsbrunner: An optimal algorithm for intersecting line segments in the plane. *J. ACM*, 39(1):1–54, 1992. https://doi.org/10.1145/147508.147511.

[CG09] Jérémie Chalopin und Daniel Gonçalves: Every planar graph is the intersection graph of segments in the plane. In: *Proceedings of the 41st Annual ACM Symposium on Theory of Computing (STOC'09)*, Seiten 631–638, 2009. https://doi.org/10.1145/1536414.1536500.

[CGO10] Jérémie Chalopin, Daniel Gonçalves und Pascal Ochem: Planar graphs have 1-string representations. *Discrete & Computational Geometry*, 43:626–647, 2010. https://doi.org/10.1007/s00454-009-9196-9.

[CvM+19] Thom Castermans, Mereke van Garderen, Wouter Meulemans, Martin Nöllenburg und Xiaoru Yuan: Short plane supports for spatial hypergraphs. *Journal of Graph Algorithms and Applications*, 23(3):463–498, 2019.

[EC02] Vladimir Estivill-Castro: Why so many clustering algorithms: a position paper. *SIGKDD Explor. Newsl*, 4(1):65–75, 2002. https://doi.org/10.1145/568574.568575.

[EKS83] Herbert Edelsbrunner, David Kirkpatrick und Raimund Seidel: On the shape of a set of points in the plane. *IEEE Transactions on Information Theory*, 29(4):551–559, 1983.

[EKSX96] Martin Ester, Hans Peter Kriegel, Jörg Sander und Xiaowei Xu: A density-based algorithm for discovering clusters in large spatial databases with noise. In: *Proceedings of the 2nd International Conference on Knowledge Discovery and Data Mining (KDD'96)*, Seiten 226–231. AAAI Press, 1996.

[FS89] Michael Fredman und Michael Saks: The cell probe complexity of dynamic data structures. In: *Proceedings of the 21st Annual ACM Symposium on Theory of Computing (STOC'89)*, Seiten 345–354, 1989. https://doi.org/10.1145/73007.73040.

[FT87] Michael L. Fredman und Robert Endre Tarjan: Fibonacci Heaps and their uses in improved network optimization algorithms. *J. ACM*, 34(3):596–615, 1987. https://doi.org/10.1145/28869.28874.

[Gei20] Jakob Geiger: Geometrische Cluster, 2020. http://www1.pub.informatik.uni-wuerzburg.de/pub/theses/2020-geiger-praktikum.pdf.

[GF64] Bernard A. Galler und Michael J. Fisher: An improved equivalence algorithm. *Commun. ACM*, 7(5):301–303, 1964. https://doi.org/10.1145/364099.364331.

[GIP18] Daniel Gonçalves, Lucas Isenmann und Claire Pennarun: Planar graphs as L-intersection or L-contact graphs. In: Artur Czumaj (Herausgeber): *Proceedings of the Annual ACM-SIAM Symposium on Discrete Algorithms (SODA'18)*, Seiten 172–184, 2018. https://doi.org/10.1137/1.9781611975031.12.

[GJ79] Michael R. Garey und David S. Johnson: *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman & Co., USA, 1979.

[HAC+19] Jan Henrik Haunert, Hugo Akitaya, Sabine Cornelsen, Philipp Kindermann, Tamara Mchedlidze, Martin Nöllenburg, Yoshio Okamoto und Alexander Wolff: Clustering colored points in the plane. In: Sara Irina Fabrikant, Silvia Miksch und Alexander Wolff (Herausgeber): *Visual Analytics for Sets*

over Time and Space (Dagstuhl Seminar 19192), Band 9 der Reihe *Dagstuhl Reports*, Seiten 53–56. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2019. http://drops.dagstuhl.de/opus/volltexte/2019/11380/.

[HKvK+18] Ferran Hurtado, Matias Korman, Marc J. van Kreveld, Maarten Löffler, Vera Sacristán, Akiyoshi Shioura, Rodrigo I. Silveira, Bettina Speckmann und Takeshi Tokuyama: Colored spanning graphs for set visualization. *Comput. Geom.*, 68:262–276, 2018. https://doi.org/10.1016/j.comgeo.2017.06.006.

[KKR+18] Philipp Kindermann, Boris Klemz, Ignaz Rutter, Patrick Schnider und André Schulz: The partition spanning forest problem. In: Wolfgang Mulzer (Herausgeber): *Proc. 34th European Workshop on Computational Geometry (EuroCG'18)*. Berlin, 2018.

[KR85] David G. Kirkpatrick und John D. Radke: A framework for computational morphology. In: Godfried T. Toussaint (Herausgeber): *Computational Geometry*, Band 2 der Reihe *Machine Intelligence and Pattern Recognition*, Seiten 217–248. North-Holland, 1985. https://doi.org/10.1016/B978-0-444-87806-9.50013-X.

[KSSW07] Christian Knauer, Étienne Schramm, Andreas Spillner und Alexander Wolff: Configurations with few crossings in topological graphs. *Computational Geometry*, 37(2):104–114, 2007. https://doi.org/10.1016/j.comgeo.2006.06.001.

[Mac67] James MacQueen: Some methods for classification and analysis of multivariate observations. In: *Proceedings of the 5th Berkeley Symposium on Mathematical Statistics and Probability, Volume 1: Statistics*, Seiten 281–297, Berkeley, Calif., 1967. University of California Press. https://projecteuclid.org/euclid.bsmsp/1200512992.

[Sch84] Edward Scheinerman: *Intersection Classes and Multiple Intersection Parameters of Graphs*. Princeton University, 1984. https://books.google.de/books?id=KpQEGwAACAAJ.

[TvL84] Robert E. Tarjan und Jan van Leeuwen: Worst-case analysis of Set Union algorithms. *J. ACM*, 31(2):245–281, 1984. https://doi.org/10.1145/62.2160.

Erklärung

Hiermit versichere ich die vorliegende Abschlussarbeit selbstständig verfasst zu haben, keine anderen als die angegebenen Quellen und Hilfsmittel benutzt zu haben, und die Arbeit bisher oder gleichzeitig keiner anderen Prüfungsbehörde unter Erlangung eines akademischen Grades vorgelegt zu haben.

Würzburg, den 29. Juni 2020

.....
Jakob Geiger