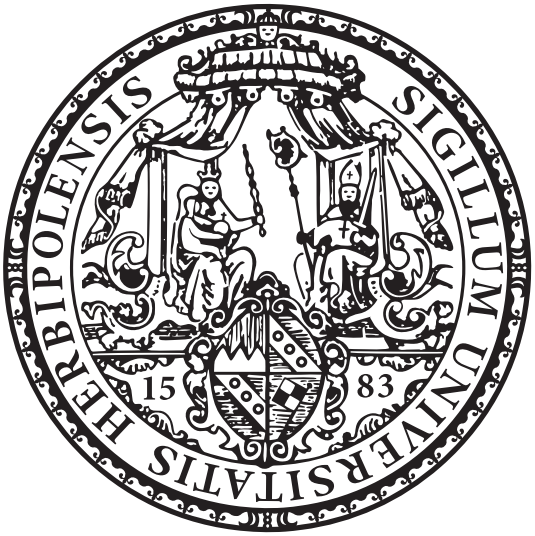
Masterarbeit

Über das Färben verallgemeinerter Intervallgraphen

Florian Mittelstädt

Abgabedatum:	11. Mai 2022
Betreuer:	Prof. Dr. Alexander Wolff
	Dr. Joachim Spoerhase
	Johannes Zink, M. Sc.



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

Zusammenfassung

Für eine Menge von Intervallen ist der korrespondierende Intervallgraph eine Repräsentation der Schnitte zwischen den Intervallen. Wir untersuchen eine verallgemeinerte Variante des Graphen-Färbeproblems auf Intervallgraphen mit gerichteten und ungerichteten Kanten. Dabei weisen wir den Knoten Zahlen aus einer möglichst kleinen Menge zu. Adjazenten Knoten werden unterschiedliche Zahlen zugewiesen. Zusätzlich gilt für eine gerichtete Kante (u, v), dass v eine größere Zahl als u zugewiesen ist. Wir zeigen, wie der bekannte Greedy-Ansatz zum Färben von Intervallgraphen auf unser Problem übertragen werden kann. Als Hauptresultat beweisen wir, dass dieser Algorithmus in unserem speziellen Färbeproblem stets eine Färbung mit minimaler Anzahl an Farben erzeugt. Diese Verallgemeinerung des Färbens formalisiert ein Teilproblem, das beim automatisierten Zeichnen von Kabelplänen für elektronische Maschinen auftritt. Wir zeigen mit unserem Resultat, dass beim orthogonalen Zeichnen von Kabelplänen stets eine Zeichnung mit minimaler Höhe generiert wird. Die Laufzeit unseres Algorithmus ist optimal.

Abstract

A graph that corresponds to a set of intervals and their overlaps is called an interval graph. We study a generalized variant of the graph coloring problem for interval graphs, where some edges may be directed while others are undirected. We assign each node a number from as small a set as possible. Adjacent nodes must be assigned different colors. Additionally, for a directed edge (u, v), the number assigned to v must be greater than that of u. We show how to adapt the well-known greedy coloring approach for interval graphs to solve our problem. As our main result, we prove that this algorithm always produces a coloring with a minimal number of colors for our specific problem. This generalization of coloring formalizes a sub-problem that occurs when drawing cable plans of electronic machines. With our result, we show that drawing a cable plan orthogonally yields a drawing of minimal height. The runtime of our algorithm is optimal.

Inhaltsverzeichnis

1 Einleitung

Die Intervallgraphen sind eine Unterklasse der ungerichteten Graphen, die so zu einer Menge von Intervallen korrespondieren, dass jeder Knoten ein Intervall repräsentiert und zwei Knoten genau dann durch eine Kante verbunden sind, wenn ihre Intervalle sich schneiden. Unter einer *Knotenfärbung* eines Graphen versteht man eine Zuordnung aller Knoten zu Farben, sodass zwei benachbarte Knoten stets unterschiedlich gefärbt sind. Um bei einer Färbung die Farben nach „Größe“ unterscheiden zu können (Farbe „x“ ist größer als Farbe „y“), verwendet man natürliche Zahlen.

Die Berechnung einer Färbung in Intervallgraphen kann mit einer bekannten Greedy-Strategie optimal gelöst werden: *Betrachte die Intervalle in aufsteigender Reihenfolge ihrer linken Endpunkte und färbe jedes Intervall in der niedrigsten, zulässigen Farbe.* Dieser Algorithmus liefert einen intuitiven Beweis, dass Intervallgraphen perfekt sind [Wes01].

Ein *gemischter* Graph besitzt sowohl gerichtete als auch ungerichtete Kanten. Korrespondiert ein gemischter Graph zu einer Menge von Intervallen, so spricht man von einem gemischten Intervallgraphen. Diese stellen eine Verallgemeinerung der üblicherweise ungerichteten Graphenklasse dar.

Für eine Färbung eines gemischten Graphen gilt, wie im ungerichteten Graph, dass durch eine ungerichtete Kante verbundenen Knoten unterschiedliche Farben zugeordnet sind. Für zwei durch eine gerichtete Kante verbundene Knoten wird zwischen zwei Formen von gerichteter Färbung unterschieden: Bei der *nicht-strikten* Färbung gilt: Die Farbe des Knoten, von dem die Kante ausgeht, ist höchstens so groß wie die Farbe des Knotens, zu dem die Kante gerichtet ist. Beim *strikten* Färben gemischter Graphen ist die Farbe des ausgehenden Knotens kleiner als die des Knotens, zu dem die Kante gerichtet ist. Sotskov [Sot20] hat eine historische Revue dieser Typen von Färbeproblemen veröffentlicht. Darin identifiziert er drei wesentliche Probleme in Zusammenhang mit dem Färben gemischter Graphen, nämlich das Existenzproblem: *Existiert eine gültige Färbung des vorliegenden Graphen?*, das Optimierungsproblem: *Wie berechnet man eine Färbung mit der kleinstmöglichen Anzahl an Farben?* und das Aufzählungsproblem: *Wie zählt man alle gültigen Färbungen eines Graphen auf?*.

Die früheste bekannte Formulierung des nicht-strikten Färbeproblems in gemischten Graphen stammt von Sotskov und Tanaev [ST76]. Darin zeigen sie, dass ein gemischter Graph genau dann nicht-strikt gefärbt werden kann, wenn sein gerichteter Untergraph keinen Kreis enthält, in dem zwei Knoten vorkommen, die im ungerichteten Untergraph benachbart sind. Die erste bekannte Erwähnung des strikten Färbeproblems in gemischten Graphen stammt von Hansen, Kuplinksy und de Werra [HKdW97]. Dort wurden sogenannte *Orientierungen* der Kantenmenge untersucht. Dabei handelt es sich um Graphen, die aus der Ersetzung einer Teilmenge der ungerichteten Kanten durch gerichtete

Kanten entstehen. Offenbar besitzt ein solcher Graph nur dann eine strikte Färbung, wenn eine vollständige Orientierung seiner Kanten existiert, die keinen gerichteten Kreis aufweist. Auch existiert in einem solchen Graph eine Färbung mit maximal k unterschiedlichen Farben genau dann, wenn eine vollständige Orientierung existiert, so dass der resultierende, gerichtete Graph keinen gerichteten Kreis und keinen gerichteten Pfad aus mehr als k Knoten enthält.

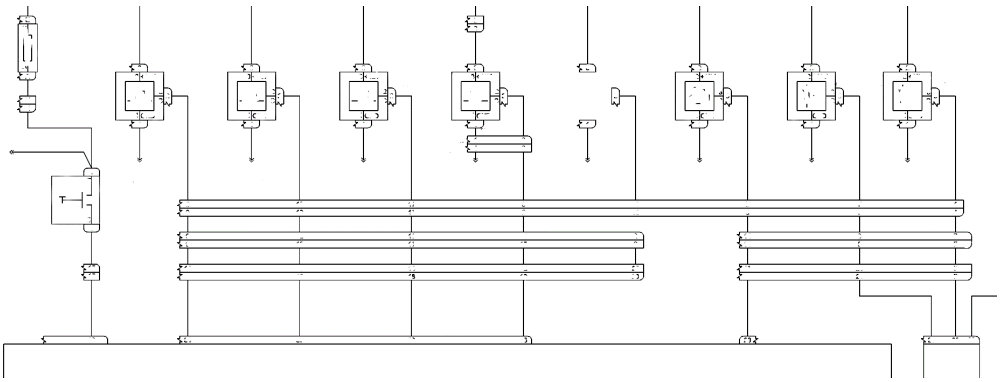


Abb. 1.1: Ausschnitt aus einem handgezeichneten Schaltplan. Beschriftungen wurden bewusst entfernt [ZWBW22]

Zink et al. [ZWBW22] haben ein Problem vorgestellt, das mit dem Auffinden einer strikten Färbung eines gemischten Intervallgraphen zusammenhängt. Sie haben untersucht, wie man das Zeichnen von Schaltplänen komplexer Maschinen automatisiert. Ein Schaltplan einer solchen Maschine ist eine Abbildung in der Ebene, in der die elektronischen Komponenten sowie die sie verbindenden Leitungen dargestellt sind. In modernen, industriell verwendeten Maschinen kommen komplexe elektronische, mechanische oder hydraulische Systeme zur Anwendung. Zeichnungen, die diese Systeme schematisch darstellen, können den Konstrukteuren beim Analysieren vorhandener Maschinen helfen, dem Wartungspersonal bei der Reparatur dienen oder beim Auffinden von Fehlerquellen helfen. Abbildung 1.1 zeigt einen Ausschnitt aus dem elektrischen Schaltplan einer Maschine. Dieser wurde manuell erstellt und zeigt die elektronischen Komponenten und elektrischen Verbindungen. Durch die zunehmende technische Komplexität der Anlagen ist das manuelle Erstellen solcher Zeichnungen zu aufwändig und zeitintensiv geworden. Zink et al. [ZWBW22] haben für das Problem, einen solchen Schaltplan automatisiert zu zeichnen, einen mehrstufigen Plan entworfen, der sich am Algorithmus von Sugiyama [STT81] orientiert. Innerhalb dieser algorithmischen Lösung ist ein Teilproblem, die Leitungen so anzuordnen, dass die Anzahl an Überschneidungen zwischen den Leitungen klein ist und dabei die Zeichnung kompakt bleibt. Wir betrachten dazu eine abstrakte Verallgemeinerung solcher Schaltpläne, die auf Brückner zurückgeht [Brü21]. Dabei ergibt sich aus dem Anordnen der Leitungen ein Problem, das ein Spezialfall des strikten Färbens eines gemischten Intervallgraphen ist.

Eigener Beitrag Im Rahmen dieser Arbeit wollen wir gemischte Intervallgraphen untersuchen. Ein konkretes Problem, bei dem zu einer Menge von Intervallen sowohl gerichtete, als auch ungerichtete Kanten im korrespondieren Graph vorkommen, wird in Kapitel 2 vorgestellt. Dabei wollen wir einen Kabelplan im orthogonalen Stil zeichnen. Die Kanten in dieser Zeichnung werden durch Intervalle charakterisiert, denen wir nach bestimmten Regeln y-Koordinaten zuweisen. Das entstehende Problem ist äquivalent zum strikten Färben eines gemischten Intervallgraphen. Wir wollen möglichst kompakte Zeichnungen berechnen und sind daher am Optimierungsproblem interessiert: *Wie berechnet man eine Färbung mit minimaler Anzahl an Farben?* In Kapitel 3 verallgemeinern wir die bekannte Greedy-Strategie zum Färben von Intervallgraphen auf unser Problem. Dazu geben wir eine Implementierung an, mit der in Laufzeit $\mathcal{O}(n^2)$ eine gültige Lösung berechnet wird. In Kapitel 4 verbessern wir die Laufzeit dieser Implementierung auf $\mathcal{O}(n \log n)$ und zeigen, dass diese Laufzeitschranke scharf ist. Schließlich zeigen wir in Kapitel 5, dass unser Algorithmus stets eine Lösung mit minimaler Farbanzahl für das strikte Graphenfärbe-Problem berechnet. In Kapitel 6 gehen wir auf weiterführende Forschungsfragen ein.

2 Grundlagen

Wir wollen ein Färbeproblem auf Intervallgraphen lösen, das beim Erstellen orthogonaler Zeichnungen von Schaltplänen Anwendung findet. Das Problem tritt auf, wenn wir den Leitungen, die in dem Schaltplan als Streckenzug aus vertikalen und horizontalen Geradensegmenten dargestellt werden, y-Koordinaten zuordnen. Wir nennen das Problem Layer-Assignment. Wir beginnen damit, den Schaltplan so zu abstrahieren, dass wir auf einer Eingabemenge, die bestimmten Eigenschaften genügt, arbeiten. Danach formalisieren wir unser Problem und definieren, was eine gültige Lösung ist. Anschließend gehen wir auf die Grundlagen der Graphen ein, und definieren alle relevanten Begriffe. Anschließend widmen wir uns in Kapitel 3 einem Lösungsansatz, der auf dem bekannten Greedy-Algorithmus zum Färben von Intervallgraphen basiert und der bereits von Zink et al. [ZWBW22] verwendet wurde. Dazu geben eine konkrete Implementierung an. Wir verbessern in Kapitel 4 die Laufzeitschranke dieses Algorithmus gegenüber der naiven Implementierung und zeigen, dass diese optimal ist. Schließlich zeigen wir in Kapitel 5, dass dieser Algorithmus optimal ist.

2.1 Abstraktion des Schaltplans

Wir wollen für eine komplexe Maschine einen Kabelplan zeichnen, der die elektrischen Leitungen zwischen den elektronischen Komponenten der Maschine abbildet. Die Zeichnung soll im *orthogonalen Stil* gezeichnet sein. Orthogonal impliziert, dass die Leitungen als achsparallele *Strecken* gezeichnet werden, die sich ausschließlich in rechten Winkeln treffen. In unserem Fall heißt das konkret: Leitungen werden als Abfolge horizontaler und vertikaler Strecken dargestellt.

Definition 2.1. *Sei* $\mathbb{R}^2$ *die* Ebene. *Eine* kartesische Koordinate *ist ein Tupel* $(x, y) \in \mathbb{R}^2$. *Eine* Strecke *in der Ebene ist ein Koordinatenpaar* $((x_1, y_1), (x_2, y_2))$. *Wenn* $x_1 = x_2$ *gilt, so nennen wir die Strecke* vertikal. *Gilt* $y_1 = y_2$*, so heißt die Strecke* horizontal.

Eine Leitung in einem Kabelplan wird als y-monotone Folge von Strecken gezeichnet. Zunächst sei eine Leitung nur durch ein Koordinatenpaar $(\mathit{start}, \mathit{ziel})$ mit $\mathit{start} = (x_{\text{start}}, y_{\text{start}})$ und $\mathit{ziel} = (x_{\text{ziel}}, y_{\text{ziel}})$ gegeben. Diese definieren die Koordinaten der elektrischen Anschlüsse, zwischen denen eine Leitung verläuft. Wir treffen folgenden Annahme: Die Startpunkte aller Leitungen liegen auf einer gemeinsamen horizontalen Geraden, die durch $y_{\text{oben}} \in \mathbb{R}$ verläuft. Wir nennen diese den *oberen Rand der Zeichnung.* Analog dazu liegen auch alle Zielpunkte auf einer gemeinsamen horizontalen Geraden. Diese verläuft durch $y_{\text{unten}} \in \mathbb{R}$ mit $y_{\text{unten}} < y_{\text{oben}}$ und heißt *unterer Rand der Zeichnung.* Da y_{oben} und y_{unten} global in der Zeichnung festgelegt sind, gilt für alle Leitungen: $y_{\text{start}} = y_{\text{oben}}$

und $y_{\text{ziel}} = y_{\text{unten}}$. Entsprechend können wir eine Leitung zwischen y_{start} und y_{ziel} nur noch durch ihre x-Koordinaten angeben. Dabei sagen wir, die Leitung ist *rechtsgerichtet*, wenn ihr Zielpunkt weiter rechts liegt als ihr Startpunkt. Symmetrisch heißt die Leitung *linksgerichtet*, wenn ihr Zielpunkt weiter links liegt als ihr Startpunkt.

Definition 2.2. *Sei y_{oben} der obere und y_{unten} der untere Rand einer Zeichnung, mit $y_{\text{oben}} > y_{\text{unten}}$ und $y_{\text{oben}}, y_{\text{unten}} \in \mathbb{R}$. Eine Leitung ist eine y-monotone Folge von achsenparallelen Strecken zwischen einem* Startpunkt *$start = (x_{\text{start}}, y_{\text{start}})$ und einem* Zielpunkt *$ziel = (x_{\text{ziel}}, y_{\text{ziel}})$ mit $x_{\text{start}}, x_{\text{ziel}} \in \mathbb{R}$. Da für alle Leitungen $y_{\text{start}} = y_{\text{oben}}$ und $y_{\text{ziel}} = y_{\text{unten}}$ gilt, schreiben wir die Leitung kurz als $(x_{\text{start}}, x_{\text{ziel}})$. Die Leitung heißt* rechtsgerichtet*, wenn $x_{\text{start}} < x_{\text{ziel}}$ gilt. Wir nennen die Leitung* linksgerichtet*, wenn gilt: $x_{\text{start}} > x_{\text{ziel}}$.*

Wir sagen, die Leitung *verläuft nach rechts* bzw. *verläuft nach links*, wenn sie rechts- respektive linksgerichtet ist. Die Leitungen sollen als Folge von Strecken gezeichnet werden. Weil es sich um eine Zeichnung in Orthogonal-Darstellung handelt, definieren wir für eine Leitung die Begriffe *horizontales* und *vertikales Segment*. Diese sind die Strecken, aus denen sich die Darstellung einer Leitung zusammensetzt.

Definition 2.3. *Sei $(x_{\text{start}}, x_{\text{ziel}})$ eine Leitung und $y_{\text{hor}} \in \mathbb{N}$ eine y-Koordinate mit $y_{\text{ziel}} < y_{\text{hor}} < y_{\text{start}}$. Wir nennen die Strecke $hor = ((x_{\text{start}}, y_{\text{hor}}), (x_{\text{ziel}}, y_{\text{hor}}))$ das* horizontale Segment *der Leitung. Die Strecken $vert_{\text{start}} = ((x_{\text{start}}, y_{\text{start}}), (x_{\text{start}}, y_{\text{hor}}))$ respektive $vert_{\text{ziel}} = ((x_{\text{ziel}}, y_{\text{hor}}), (x_{\text{ziel}}, y_{\text{ziel}}))$ heißen oberes und unteres* vertikales Segment *der Leitung.*

Um einen Kabelplan zu zeichnen, wird zunächst für jede Leitung y_{hor} algorithmisch festgelegt. Dann werden hor, $vert_{\text{start}}$ und $vert_{\text{ziel}}$ gezeichnet. Den entstehenden Streckenzug nennen wir *orthogonale Darstellung* der Leitung.

Definition 2.4. *Sei $(x_{\text{start}}, x_{\text{ziel}})$ eine Leitung und $y_{\text{hor}} \in \mathbb{N}$, sodass $y_{\text{ziel}} < y_{\text{hor}} < y_{\text{start}}$ gilt. Wir nennen $(vert_{\text{start}}, y_{hor}, vert_{\text{ziel}})$ die* orthogonale Darstellung *von $(x_{\text{start}}, x_{\text{ziel}})$.*

Zunächst handeln wir einige Spezialfälle ab. Sei $(x_{\text{start}}, x_{\text{ziel}})$ eine Leitung und sei $x_{\text{start}} = x_{\text{ziel}}$. In diesem Fall besitzt hor keine Ausdehnung in horizontaler Richtung. Die Leitung kann dann durch ein einziges vertikales Segment dargestellt werden, welches durch x_{start} verläuft. Wir sind an den Leitungen interessiert, in deren Darstellung ein horizontales Segmenten vorkommt. Daher lassen wir die Leitungen, für die $x_{\text{start}} = x_{\text{ziel}}$ gilt, außen vor. Sie können in einem letzten Schritt in die fertigen Zeichnung eingefügt werden.

Abbildung 2.1 zeigt weitere Spezialfälle. Haben zwei Leitungen, die in die selbe Richtung verlaufen, identische Start- oder Zielpunkte, so würden die vertikalen Segmente dieser beiden Leitungen sich in der orthogonalen Darstellung, wie in (I) und (II) zu sehen, überlappen. Wir schließen diesen Fall aus, da zwei Leitungen nicht den selben Anschluss in einer elektronischen Komponente belegen können. Ist hingegen $(x_{\text{start}}, x_{\text{ziel}})$ eine linksgerichtete Leitung und $(x'_{\text{start}}, x'_{\text{ziel}})$ eine rechtsgerichtete, so könnte $x_{\text{start}} = x'_{\text{ziel}}$ oder $x_{\text{ziel}} = x'_{\text{start}}$ gelten. In diesem Fall würden beide Leitungen nicht die selben Anschlüsse

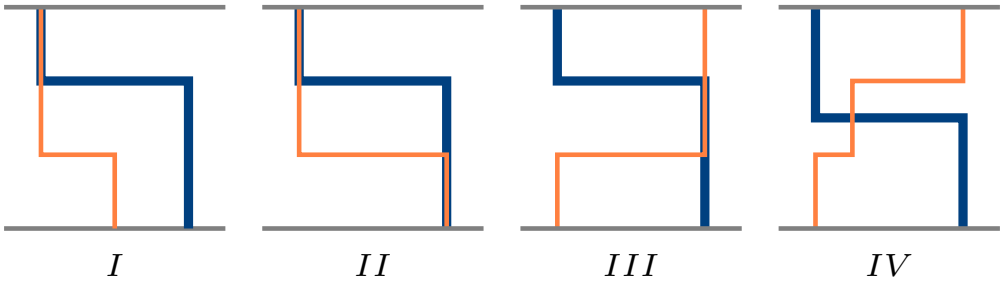


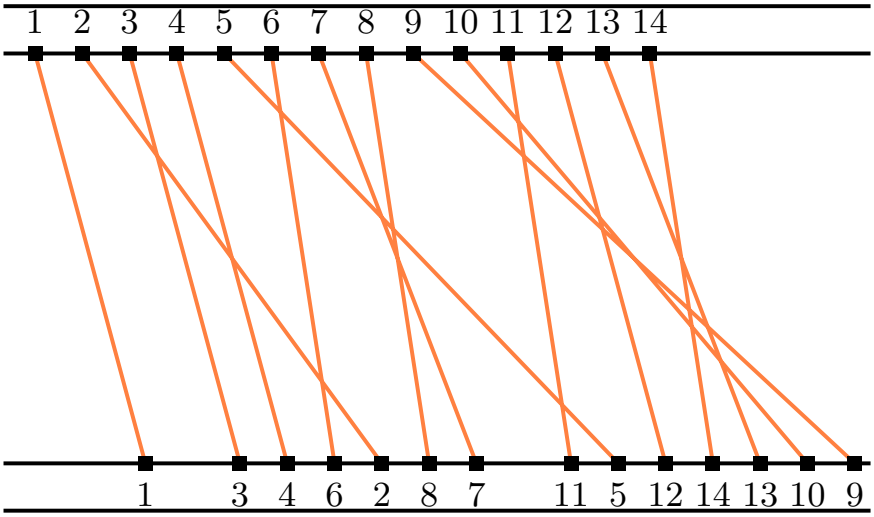
Abb. 2.1: (I) und (II) sind Spezialfälle der Eingabe, die wir ausschließen. (III) zeigt einen Spezialfall, der vorkommen kann. (IV) zeigt für diesen Fall eine Darstellung einer Leitung mit zwei horizontalen Segmenten.

belegen. Es käme jedoch, wie in (III) abgebildet, zu einer Überschneidung eines der vertikalen Segmente. Zink et al. haben diesen Fall behandelt, indem sie diejenigen Leitungen, deren Zielpunkte gleich dem Startpunkt einer in der Zeichnung tiefer platzierten Leitung sind, durch zwei horizontale Segmente dargestellt haben [ZWBW22]. Dies ist in (IV) zu sehen.

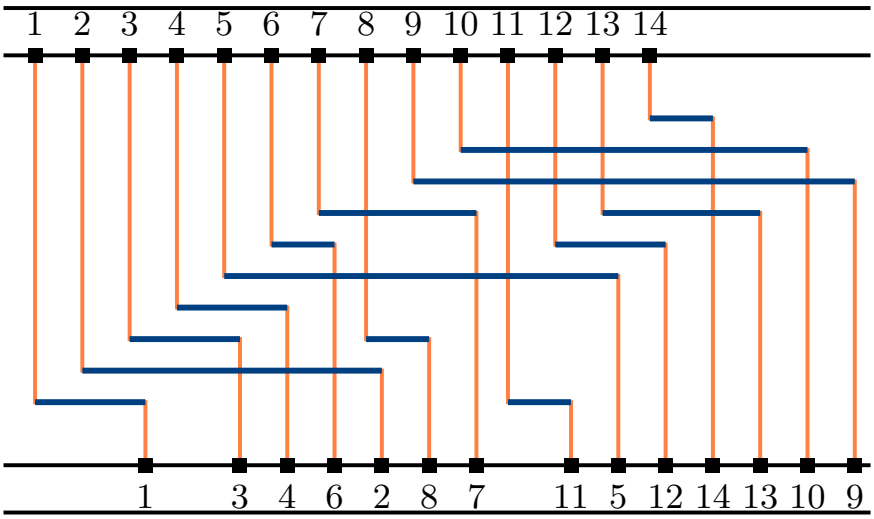
Wir schlagen stattdessen eine Vorverarbeitung der Eingabe vor. Sei $\Delta x_{\min}$ die kleinste, von null verschiedene Distanz entlang der x-Achse von zwei Start- oder Zielpunkten. Wenn wir sämtliche Zielpunkte um $1/2 \cdot \Delta x_{\min}$ relativ zu den Startpunkten nach rechts verschieben, so sind in der resultierenden Instanz die x-Koordinaten aller Startpunkte verschieden von den x-Koordinaten aller Zielpunkten. Dabei bleibt die Richtung jeder Leitung erhalten. Da außerdem bereits ausgeschlossen wurde, dass zwei Leitungen den selben Anschluss belegen, gelte fortan, dass alle x-Koordinaten der Start- und Zielpunkte paarweise verschieden sind.

Wir beschränken uns zunächst auf den Fall, dass nur rechtsgerichtete Leitungen vorkommen. Der linksgerichtete Fall lässt sich analog lösen. Zu allen Resultaten existiert ein Beweis, der symmetrisch zum rechtsgerichteten Fall ist. In Abbildung 2.2 sieht man einen möglichen Kabelplan. Dabei zeigt (a) eine Menge von rechtsgerichteten Leitungen, die als Geraden zwischen Start- und Zielpunkt einer Leitung gezeichnet wurden. Die x-Koordinaten der Start- und Zielpunkte sind paarweise verschieden. In (b) wurden diese Leitungen im orthogonalen Stil gemäß Definition 2.4 aus zwei vertikalen und einer horizontalen Strecke gezeichnet. Das horizontale Segment jeder Leitung ist in Blau hervorgehoben. Besonders die Leitungen weiter rechts in (a) sind durch die vielen Überschneidungen und teils kleinen Winkel zwischen den Geraden unübersichtlich. In (b) sind die Leitungen deutlich besser voneinander zu unterscheiden.

Wir haben definiert, was die orthogonale Darstellung einer Leitung ist und die Eingabe auf paarweise verschiedene x-Koordinaten der Start- und Zielpunkte beschränkt. Im kommenden Kapitel wollen wir unser Problem genauer formalisieren.



(a) Leitungen, dargestellt als Geraden zwischen Start- und Zielpunkt. Alle x-Koordinaten sind paarweise verschieden. Die Zeichnung enthält Bereiche, die wegen vieler Überschneidungen unübersichtlich sind.



(b) Hier sind die Leitungen orthogonal gezeichnet. Die horizintalen Segmente sind blau hervorgehoben. Der Kabelplan ist lesbarer. Die Abstände zwischen einzelnen Leitungen sind gleichmäßiger.

Abb. 2.2: Zwei mögliche Darstellungen eines Kabelplans mit rechtsgerichteten Leitungen. Die Nummerierung ist aufsteigend, nach x-Koordinaten der Startpunkte.

2.2 Eingabe und Begriffe

In Definition 2.3 wurde y_{hor} als Koordinate für das Zeichnen des horizontalen Segments einer Leitung eingeführt. Da der obere und untere Rand der Zeichnung festgelegt sind, wird die orthogonale Darstellung einer Leitung nur noch durch ihre x-Koordinaten $(x_{\text{start}}, x_{\text{ziel}})$ und y_{hor} definiert. Bei dem Problem, das wir betrachten, geht es nun darum, für jede Leitung y_{hor} so festzulegen, dass in der Zeichnung keine Überschneidungen zwischen horizontalen Segmenten vorkommen. Wir gehen davon aus, dass alle horizontalen Segmente zu rechtsgerichteten Leitungen gehören. Ein solches Segment kann als abgeschlossenes Intervall zwischen einem *linken Endpunkt* l und einem *rechten Endpunkt* r aufgefasst werden. Da wir von rechtsgerichteten Leitungen ausgehen, ist $x_{\text{start}} = l$ der linke und $x_{\text{ziel}} = r$ der rechte Endpunkt.

Definition 2.5. *Ein abgeschlossenes* Intervall $e = [l, r] \subseteq \mathbb{R}$ *ist eine Teilmenge der reellen Zahlen und enthält alle* $x \in \mathbb{R}$ *mit* $l \leq x \leq r$. *Wenn* $(x_{\text{start}}, x_{\text{ziel}})$ *eine Leitung ist, dann ist das Intervall* $[l, r]$ *mit* $l = \min\{x_{\text{start}}, x_{\text{ziel}}\}$ *und* $r = \max\{x_{\text{start}}, x_{\text{ziel}}\}$ *eine* Intervall-Repräsentation *der Leitung.*

Wenn e ein Intervall ist, das eine Leitung repräsentiert, dann schreiben wir $l(e)$ bzw. $r(e)$ und meinen den linken respektive rechten Endpunkt von e. Zwei Intervalle e und e' *überlappen*, wenn ihre Schnittmenge nicht leer ist. Dies ist genau dann der Fall, wenn sie wenigstens eine gemeinsame x-Koordinate besitzen. Wir sprechen von einer *vollständigen* Überlappung, wenn e komplett in e' enthalten ist. In diesem Fall ist e' nach links wie auch nach rechts länger als e. Wir sprechen von einer *teilweisen* Überlappung, wenn e und e' überlappen und e weiter links beginnt und endet als e'.

Definition 2.6. *Zwei Intervalle* e, e' *überlappen, wenn* $e \cap e' \neq \emptyset$ *gilt. Die Überlappung heißt* vollständig*, wenn* $l(e) < l(e')$ *und* $r(e) > r(e')$ *gilt. Sie heißt* teilweise*, wenn* $l(e) < l(e')$ *und* $r(e) < r(e')$ *gilt.*

Eine Lösung L einer Instanz ist eine gültige Zuordnung der Intervalle zu y-Koordinaten, sodass die Leitungen, deren horizontale Segmente die Intervalle repräsentieren, ohne Überschneidungen gezeichnet werden können. Wir führen dafür den Begriff *Ebene* ein. Jedes Intervall in einer Lösung wird einer ganzzahligen Ebene zugeordnet. Die Ebene entspricht in der Zeichnung der y-Koordinate des horizontalen Segments y_{hor}.

Definition 2.7. *Sei* $(x_{\text{start}}, x_{\text{ziel}})$ *eine Leitung und* $e = [x_{\text{start}}, x_{\text{ziel}}]$ *das Intervall, welches die Leitung repräsentiert. Eine* Ebene $\varphi(e) \in \mathbb{N}$ *ist eine natürliche Zahl, die* e *zugeordnet wird, sodass* $y_{\text{hor}} = \varphi(e)$ *gilt.*

Wir wollen allen Intervallen eine Ebene zuordnen. Dabei gilt, dass Intervalle, die sich überlappen, also für die $e \cap e' \neq \emptyset$ gilt, nicht auf die selbe Ebene gezeichnet werden dürfen. In einem Intervallgraphen, in dem für jedes Paar an Intervallen e, e' mit $e \cap e' \neq \emptyset$ eine ungerichtete Kante existiert, stellt eine gültige Färbung auch eine Zuordnung der horizontalen Segmente zu Ebenen dar. Wir wollen jedoch auch die in der Zeichnung entstehenden Überschneidungen klein halten. Betrachten wir Abbildung 2.3: Sind zwei

(a) Zwei Intervalle mit teilweiser Überlappung. Wird das weiter rechts befindliche Intervall unten gemalt, kommt es zu zwei Kreuzungen, die beide vermeidbar sind.

(b) Zwei Intervalle mit vollständiger Überlappung. Die Reihenfolge der beiden Intervalle ist frei, da eine Kreuzung in beiden Fällen auftritt.

Abb. 2.3: Überschneidung bei teilweiser und vollständiger Überlappung

Intervalle e und e' vollständig überlappend, so kommt es stets zu einer Kreuzung zwischen dem vertikalen Segment von e und dem horizontalen Segment von e', ganz gleich, ob e über, oder unter e' gezeichnet wird. Überlappen zwei Intervalle jedoch teilweise sieht man, dass es zu zwei Kreuzungen zwischen den orthogonalen Darstellungen von e und e' kommt, wenn beide Intervalle zu rechtsgerichteten Leitungen gehören und das weiter links befindliche Intervall oben liegt. Weist man dem weiter rechts befindlichen Intervall dagegen eine höhere Ebene zu, so schneiden sich die Leitungen in der Zeichnung nicht. Wir verbieten diese Form der Überlappung daher und sagen, dass folgendes gilt: Für teilweise überlappende, rechtsgerichtete Intervalle e und e', wobei $e < e'$ (sprich: e befindet sich weiter links), muss gelten: $\varphi(e) < \varphi(e')$. Oder einfach, dass das rechte der beiden Intervalle oben liegt.

Definition 2.8. *Sei I eine Menge von Intervallen, die rechtsgerichtete Leitungen repräsentieren. Eine Lösung L ist eine Funktion $\varphi : I \to \mathbb{N}$, die allen $e \in I$ eine* Ebene *$\varphi(e) \in \mathbb{N}$ zuordnet, sodass für Intervalle $e, e' \in I$ mit $e \cap e' \neq \emptyset$ folgende Bedingungen erfüllt sind:*

1. $\varphi(e) \neq \varphi(e')$
2. $l(e) < l(e') \wedge r(e) < r(e') \Rightarrow \varphi(e) < \varphi(e')$

Wenn zwei Intervalle e, e' teilweise überlappen, und e liegt weiter links als e', dann sagen wir: *e ist gerichtet zu e'*. Überlappen e, e' vollständig, so sagen wir: *e ist ungerichtet* zu e'. Mit den eingeführten Begriffen können wir nun die Problemstellung definieren. Wir suchen für eine Menge von horizontalen Segmenten eine gültige Zuweisung zu den Ebenen, sodass die Bedingungen aus Definition 2.8 erfüllt sind. Wir bezeichnen die Anzahl an Ebenen in einer Lösung als *Höhe* von L.

Definition 2.9. *Wir erhalten als Eingabe eine Menge I an Intervallen gemäß Definition 2.5. Dabei gilt für alle $e, e' \in I$ mit $e \neq e'$: $l(e) \neq l(e')$, $r(e) \neq r(e')$, $r(e) \neq r(e')$ und $l(e) \neq r(e')$. Beim Layer-Assignment-Problem suchen wir nach einer Lösung L laut Definition 2.8.*

Symmetrisch zu dieser Definition gilt für zwei linksgerichtete Leitungen, dass das weiter links befindliche Intervall oben liegen muss. Man kann so eine Lösung und das damit

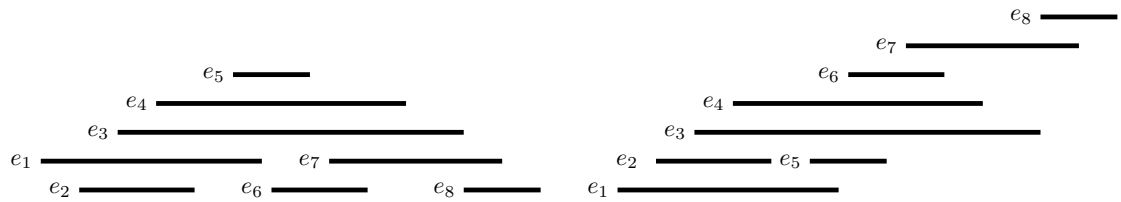


Abb. 2.4: Eine Beispiel-Instanz (links) und eine dazugehörige, gültige Lösung (rechts). Die Intervalle sind aufsteigend nach ihren linken Endpunkten nummeriert. Die Lösung auf sieben Ebenen ist hier optimal.

verbundene Problem für linksgerichtete Leitungen definieren. Die Zuordnung zu den Ebenen gemäß Definition 2.8 impliziert eine eindeutige *Reihenfolge* zwischen teilweise überlappenden Intervallen, bedeutet jedoch nicht, dass diese auf konsekutiven Ebenen liegen müssen.

Definition 2.10. *Zwei Intervalle $e, e' \in I$ haben eine* Reihenfolge*, wenn eine Teilmenge $I' \subseteq I$ und eine Sortierung $(e_1, e_2, \ldots, e_k)$ von $I' \cup \{e, e'\}$ existiert, sodass für jedes Paar an Intervallen e_i, e_{i+1} mit $i \in [1, k-1]$ gilt:*

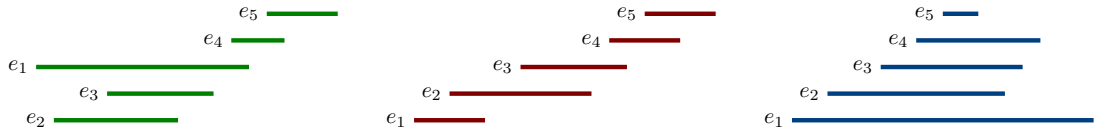
$$e_i \cap e_{i+1} \neq \emptyset \wedge l(e_i) < l(e_{i+1}) \wedge r(e_i) < r(e_{i+1})$$

Wir betrachten nun in Abbildung 2.4 ein konkretes Beispiel einer Instanz und eine dazugehörige, gültige Lösung. Die Intervalle in der Abbildung sind aufsteigend nummeriert nach ihren linken Endpunkten. Intervall e_1 ist das am weitesten links liegende und wird auf die unterste Ebene platziert. Intervall e_2 wird von e_1 vollständig überlappt, darf also sowohl darüber als auch darunter platziert werden. Wir platzieren es auf der zweiten Ebene. Da e_5 keine Überlappung mit e_2 hat, darf es ebenso auf ebene zwei liegen. Sowohl e_3 als auch e_4 müssen oberhalb von e_2 liegen. Wir platzieren sie auf Ebenen drei und vier. Die Intervalle e_5, e_6, e_7, e_8 bilden eine Reihenfolge. Da außerdem e_6 und e_7 sich mit e_3 und e_4 schneiden, platzieren wir sie auf Ebenen fünf bis sieben. Die gezeigte Lösung zeichnet die acht Intervalle auf sieben Ebenen, was hier optimal ist.

Wir sind an einer kompakten Zeichnung interessiert. Dies erfordert, dass wir stets nach einer Lösung mit wenigen Ebenen suchen. Es soll also keine leeren Ebenen oder Lücken geben. Wir definieren den Begriff *Stapel*. Wir erinnern uns, dass in einer gültigen Lösung jedem Intervall e eine Ebene $\varphi(e) \in \mathbb{N}$ zugewiesen ist. Unter einem Stapel verstehen wir eine Menge von Intervallen auf konsekutiven Ebenen, sodass jedes Intervall in dem Stapel eine Überlappung mit den Intervallen des Stapels darüber und darunter hat.

Definition 2.11. *Ein Stapel über k Ebenen $S \subseteq L$ ist eine Teilmenge von k Intervallen in L auf konsekutiven Ebenen, sodass für jedes Paar $e, e' \in S$ mit $\varphi(e) = \varphi(e') + 1$ gilt: $e \cap e' \neq \emptyset$. Der Stapel heißt* einfach*, wenn $\bigcap S \neq \emptyset$ gilt. Wir nennen S* gerichtet*, wenn aus $\varphi(e) = \varphi(e') + 1$ stets folgt: e und e' überlappen teilweise. Falls in S kein Paar e, e' mit teilweiser Überlappung existiert, so heißt S* ungerichtet*.*

Es gilt also laut Definition 2.11 in S je eine paarweise Überlappung aufeinander folgender Intervalle. Da ein Stapel über *konsekutive* Ebenen definiert ist, gibt es in S



(a) Ein Stapel, der weder einfach ist, noch gerichtet. Intervalle überlappen vollständig oder teilweise.

(b) Ein gerichteter Stapel. Es liegt eine teilweise Überlappung von Intervallen auf konsekutiven Ebenen vor.

(c) Ein einfacher Stapel. Es liegt eine paarweise Überlappung aller Intervalle des Stapels vor.

Abb. 2.5: Verschiedene Typen von Stapel.

keine leeren Ebenen. Es handelt sich um einen *einfachen Stapel*, wenn alle $e \in S$ eine gemeinsame x-Koordinate enthalten. Wir sprechen von einem *gerichteten Stapel*, wenn zwei aufeinander folgende Intervalle in S stets teilweise überlappen. Dann bildet der Stapel eine Reihenfolge. Wir sprechen von einem *ungerichteten* Stapel, wenn alle $e \in S$ paarweise vollständig überlappen.

Beobachtung 2.12. *Jeder ungerichtete Stapel ist ein einfacher Stapel.*

Beweis. Sei S ein ungerichteter Stapel aus k Intervallen in L. Wenn $(e_1, \ldots, e_k)$ eine aufsteigende Sortierung der Intervalle in S nach ihren linken Endpunkten ist, dann gilt für jedes Paar e_i, e_j mit $i < j$: $l(e_i) < l(e_j)$. Daraus folgt wegen der vollständigen Überlappung, dass auch $r(e_i) > r(e_j)$ gilt. In diesem Fall ist stets e_{i+1} vollständig in e_i enthalten. Daraus folgt: e_k ist in den Intervallen $e_1 \ldots e_{k-1}$ enthalten. □

Laut Beobachtung 2.12 gilt für jeden ungerichteten Stapel S, dass alle Intervalle in S eine gemeinsame x-Koordinate besitzen. Damit ist jeder ungerichtete Stapel auch ein einfacher Stapel. In Abbildung 2.5 sind verschiedene Typen von Stapeln abgebildet. In (a) sieht man einen Stapel, der nicht optimal gelöst wurde. Intervalle e_2 und e_3 sind beide ungerichtet zu Intervall e_1. Da e_2 und e_3 außerdem keine Schnittmenge mit e_4 und e_5 haben, dürfen sie oberhalb von e_1 und neben e_4 und e_5 platziert werden. Somit existiert eine Lösung, die nur drei Ebenen benötigt. Die Intervalle in (b) bilden eine Reihenfolge. Der gezeigte Stapel ist auf fünf Ebenen optimal gelöst. Der Stapel in (c) ist ein einfacher Stapel, da alle Intervalle e_1 bis e_5 eine gemeinsame x-Koordinate besitzen.

Bevor wir unseren Beitrag zur Lösung des Problem vorstellen, benötigen wir noch einige Grundlagen. Probleme auf einer Menge von Intervallen zu Lösen findet breite Anwendung in der Informatik und über die Disziplin hinaus. So werden unter Anderem etwa in der Wirtschaft beim *scheduling* (engl. für Terminplanung) Tätigkeiten oder Prozesse, die ein zeitliches Intervall einnehmen, so terminiert, dass möglichst viele dieser Intervalle konfliktfrei stattfinden können. Umgekehrt werden bei der *resource allocation* (engl. für Ressourcenallokation) wirtschaftliche Ressourcen (dies können z. B. Arbeitskräfte, Maschinen oder Werkzeuge sein) entsprechend terminlicher Vorgaben für bestimmte Zeiträume verplant, um deren Nutzungszeit maximal auszuschöpfen und Leerlauf-Zeiten zu vermeiden [BNBYF$^+$01]. In der Biologie kommen Intervalle beispielsweise beim *DNA-mapping* zum Einsatz, wo sie bei der Visualisierung von Mutationen in der feineren

Struktur des Erbgutes dienen [Ben59]. Das Zuweisen von Ebenen zu den Intervallen ist äquivalent zum Färben eines gemischten Graphen (Satz 5.3). Daher beginnen wir mit diesen und erklären alle relevanten Konzepte und Definitionen. Anschließend betrachten wir das Graphen-Färben und nennen relevante Resultate aus der Literatur.

2.3 Graphen

Ein *Graph* G ist eine Datenstruktur mit einer *Knotenmenge* und einer *Kantenmenge*. Die Knoten repräsentieren Objekte, die im Graphen verwaltet werden. Die Beziehungen zwischen den Knoten stellen wir mithilfe der Kanten des Graphen dar. Wir schreiben für die Knotenmenge V. Eine *ungerichtete Kante* ist eine gegenseitige Beziehung $\{u, v\}$ zwischen Knoten $u, v \in V$. Für eine Menge von ungerichteten Kanten schreiben wir E. Eine *gerichtete Kante* ist eine einseitige Beziehung (u, v) eines Knoten $u \in V$ zu einem Knoten $v \in V$. Für die Menge von gerichteten Kanten schreiben wir $\vec{E}$.

Definition 2.13. *Ein Graph G ist ein Tupel aus einer Knotenmenge V und einer Kantenmenge. Die Menge der ungerichteten Kanten in G heißt E. Eine Kante $\{u, v\}$ heißt* ungerichtete Kante*. Wir nennen $G = (V, E)$ einen* ungerichteten Graph*. Eine Kante (u, v) heißt* gerichtete Kante *und die Menge der gerichteten Kanten heißt $\vec{E}$. Wir nennen $G = (V, \vec{E})$* gerichteter Graph*. Enthält G sowohl gerichtete wie auch ungerichtete Kanten, so schreiben wir G als Tripel $G = (V, \vec{E}, E)$ und nennen G* gemischter Graph*.*

Wenn zwei Knoten $u, v \in V$ durch eine Kante verbunden sind, so sind sie *benachbart* oder *adjazent*. Wir nennen die Menge von Knoten, mit denen ein $v \in V$ eine Kante hat, die *Nachbarschaft* von v und schreiben $N(v)$. Die Nachbarschaft von $N(v)$, vereinigt mit v, nennen wir *abgeschlossene Nachbarschaft*.

Definition 2.14. *Zwei Knoten $u, v \in V$ heißen* benachbart *oder* adjazent*, wenn $\{u, v\} \in E$ oder $(u, v) \in \vec{E}$ gilt. Für einen Knoten $v \in V$ heißt $N(v) \subseteq V$* Nachbarschaft *von v und es gilt: $v' \in N(v) \iff \{v, v'\} \in E$ oder $(v, v') \in \vec{E}$ oder $(v', v) \in \vec{E}$. Wir nennen $N(v) \cup v$* abgeschlossene Nachbarschaft *von v.*

Eine Kante (v, v) bzw. $\{v, v\}$ mit $v \in V$ heißt *Selbstkante*. Wenn eine ungerichtete Kante $\{u, v\}$ mehrfach in E vorkommt, oder wenn die gerichteten Kanten (u, v) und (v, u) beide in $\vec{E}$ vorkommen, nennen wir sie *Mehrfachkante*. Die Graphen, in denen weder Selbst- noch Mehrfachkanten vorkommen, heißen *einfache Graphen*. Wir beschränken uns im Folgenden ausschließlich auf einfache Graphen.

Ein *Intervallgraph* wird aus einer Menge von Intervallen I so konstruiert, dass I genau die Menge von Knoten V ist. Eine Kante $e = \{u, v\} \in E$ existiert genau dann, wenn die Intervalle u, v sich überlappen. Für diese Klasse an Graphen existieren zahlreiche Resultate. Lekkerkerker und Boland präsentierten 1962 ihre Charakterisierung der Intervallgraphen, wonach ein Graph ein Intervallgraph ist, genau dann, wenn er kein asteroidales Tripel (siehe Definition 2.18) enthält und ein Saitengraph (laut Definition 2.17) ist [LB62]. Dann, im Jahre 1964 haben Gilmore und Hoffman eine alternative Charakterisierung bewiesen. Demnach ist ein Graph genau dann ein Intervallgraph, wenn jeder

Kreis der Länge 4 eine diagonale Kante besitzt und wenn in seinem Komplementgraphen jeder ungerade Kreis eine Saite hat [GH64].

Definition 2.15. *Ein Intervallgraph $G = (V, E)$ zu einer Menge von Intervallen I ist ein Graph, sodass $V = I$ und $E = \{\{u, v\} : u, v \in I \wedge u \cap v \neq \emptyset\}$ gilt.*

Wir können also einfach zu einer gegebenen Menge von Intervallen den zugehörigen Graphen konstruieren. Der umgekehrte Fall, ob ein Graph zu einer Menge von Intervallen korrespondiert, wurde von Lekkerkerker und Boland 1962 charakterisiert. Zunächst brauchen wir einige Begriffe:

Definition 2.16. *Ein* Kreis $C \subseteq V$ *der Länge k ist eine Folge von Knoten $(v_1, v_2, \ldots, v_k)$, sodass für jedes Paar $v_i, v_{i+1} \in C$ mit $i \in [1, k-1]$ die Kante $\{v_i, v_{i+1}\}$ sowie die Kante $\{v_1, v_k\}$ existiert.*

Ein *Untergraph* von G beschreibt eine beliebige Teilmenge der Knoten und Kanten eines Graphen. Ein *induzierter* Untergraph hingegen beschreibt eine beliebige Teilmenge der Knoten, jedoch unter Beibehaltung aller Kanten zwischen Knoten des Untergraphen. Eine *Saite* für einen Kreis der Länge ≥ 4 ist eine Kante, die nicht zum Kreis gehört und die zwei Knoten des Kreises verbindet. Wenn in G jeder Kreis, der mindestens Länge 4 hat, eine Saite aufweist, nennen wir G *Saitengraph.*

Definition 2.17. *Für einen Graphen G und einen Kreis $C \subseteq V$ der Länge ≥ 4 ist eine* Saite *eine Kante $e \in E$, die zwei nicht-konsekutive Knoten in C verbindet. Wir nennen G einen Saitengraphen, wenn jeder induzierte Kreis der Länge ≥ 4 in G eine Saite besitzt.*

Drei Knoten $v_1, v_2, v_3 \in V$ bilden ein sogenanntes *asteroidales Tripel* (kurz AT), wenn v_1, v_2, v_3 so verbunden sind, dass zwischen jedem Paar von ihnen ein Pfad existiert, der den übrigen Knoten und seine Nachbarn meidet. Abbildung 2.6 zeigt die vollständige Liste der in Intervallgraphen verbotenen Untergraphen. Dabei ist (III_n) ein induzierter Kreis der Länge ≥ 4. Die übrigen Untergraphen stellen die möglichen asteroidalen Tripel dar. Keiner der abgebildeten Graphen korrespondiert zu einer Menge von Intervallen.

Definition 2.18. *Drei Knoten $v_1, v_2, v_3 \in V$ heißen* asteroidales Tripel *oder AT, genau dann, wenn Pfade P_1, P_2, P_3 existieren, sodass P_i mit $i \in \{1, 2, 3\}$ die Knoten v_j, v_k mit $j, k \in \{1, 2, 3\} \setminus i$ und $j \neq k$ enthält und $N(v_i) \cup v_i \notin P_i$ gilt. Ein Graph heißt* AT-frei*, wenn er kein AT enthält.*

Wir behaupten nun, dass ein Intervallgraph ein Saitengraph ist. Das heißt, dass in einem Graphen, der zu einer Menge von Intervallen korrespondiert, jeder induzierte Kreis, der aus mindestens vier Knoten besteht, eine Saite hat. Der Beweis stammt in ausführlicher Form aus [GH64].

Lemma 2.19. *[GH64] Intervallgraphen sind Saitengraphen.*

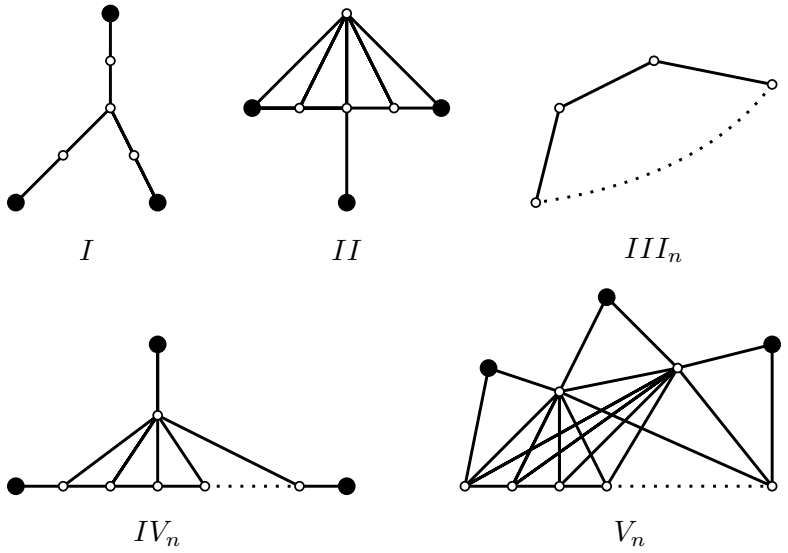


Abb. 2.6: [LB62] Verbotene induzierte Untergraphen in Intervallgraphen. Die schwarzen Knoten sind jeweils asteroidale Tripel.

Beweis. Gegeben einen Kreis C der Länge ≥ 4 sowie drei Intervalle $u, v, w \in C$, die konsekutiv in C vorkommen. Ohne Beschränkung der Allgemeingültigkeit seien $u \cap v \neq \emptyset$ und $v \cap w \neq \emptyset$ sowie $u \cap w = \emptyset$. Da u, w disjunkt sind, existiert eine x-Koordinate, die in v enthalten ist, nicht aber in u und w. Um einen Kreis der Länge ≥ 4 durch u, v, w zu Konstruieren, muss ein Intervall z existieren, sodass $u \cap z \neq \emptyset$ und $w \cap z \neq \emptyset$ gelten. In diesem Fall muss z eine Koordinate enthalten, die bereits in v enthalten ist. Daraus folgt, dass es die Kante $\{v, z\} \notin C$ gibt und C hat eine Saite. □

Satz 2.20. *[LB62] G ist ein Intervallgraph $\iff$ G ist ein Saitengraph und AT-frei.*

Das obige Resultat von Lekkerkerker und Boland wurde 1964 von Gilmore und Hoffman um eine weitere Charakterisierung ergänzt. Klären wir dazu zunächst, was unter einem *Komplementgraphen* zu verstehen ist. Ein Graph G und sein Komplement $\overline{G}$ verwenden die selbe Knotenmenge V. Eine Kante $\{u, v\}$ mit $u, v \in V$ ist in $\overline{G}$, genau dann, wenn $\{u, v\}$ *nicht* in G ist.

Definition 2.21. *Zu einem Graphen $G = (V, E)$ ist der* Komplementgraph *definiert als $\overline{G} = (V, \overline{E})$ und eine Kante $\{u, v\}$ mit $u, v \in V$ und $u \neq v$ ist in $\overline{E}$ $\iff$ $\{u, v\} \notin E$.*

Satz 2.22. *[GH64] G ein Intervallgraph $\iff$ Jeder Kreis der Länge 4 in G und jeder Kreis ungerader Länge in $\overline{G}$ besitzt eine Saite.*

Wir haben die Intervallgraphen kennengelernt und zwei alternative Charakterisierungen aus der Literatur vorgestellt. Im folgenden Abschnitt wollen wir das Färben von Intervallgraphen behandeln. Dies wird später unsere Grundlage für den Beweis von Satz 5.21.

2.4 Färben

Unter Färben von Graphen versteht man eine Zuordnung von Farben (in der Regel werden natürliche Zahlen verwendet) zu den Knoten. In einer gültigen Färbung sind zwei adjazenten Knoten stets unterschiedliche Farben zugeordnet. Das Graphen-Färbe-Problem fragt nach einer Färbung mit minimaler Anzahl an Farben. Das parametrisierte k-Färbe-Problem fragt, ob es eine gültige Färbung mit höchstens k Farben gibt.

Definition 2.23. *Eine* Färbung *eines ungerichteten Graphen $G = (V, E)$ ist eine Funktion* $\text{col} : V \to \mathbb{N}$ *sodass für Knoten $u, v \in V$ gilt: $\{u, v\} \in E \Rightarrow \text{col}(u) \neq \text{col}(v)$. Ist G ein gerichteter oder gemischter Graph, so gilt außerdem: $(u, v) \in \vec{E} \Rightarrow \text{col}(u) < \text{col}(v)$*

Definition 2.23 entspricht einer *strikten* Färbung gemischter oder gerichteter Graphen. Wir beschränken uns im folgenden Ausschließlich auf das strikte Färben. Eine *Clique* $\mathcal{C}$ ist eine Teilmenge von Knoten aus V, die vollständig verbunden ist. Das heißt, dass für jedes Paar $v_i, v_j \in \mathcal{C}, i \neq j$ die Kante $\{v_i, v_j\}$ existiert. Wir bezeichnen die Kardinalität der größten Clique $|\mathcal{C}_{max}|$ in einem Graphen G als *Cliquenzahl* $\omega(G)$. Da in einer Clique jeder Knoten benachbart zu allen anderen ist, können nie zwei Knoten, die einer Clique angehören, der selben Farbe zugeordnet werden. Daraus wird auch klar: Ein Graph G lässt sich niemals mit weniger Farben einfärben, als die Größe seiner größten Clique. Die Mindestanzahl an Farben, die es braucht, um einen Graphen gültig zu Färben, heißt *Färbezahl* $\chi(G)$.

Definition 2.24. *In einem ungerichteten Graph G ist eine* Clique $\mathcal{C} \subseteq V$ *ist eine Teilmenge der Knoten von G, sodass für alle $u, v \in \mathcal{C}$ mit $u \neq v$ gilt: $\{u, v\} \in E$. Ist G ein gerichteter oder gemischter Graph, so gilt für alle $u, v \in \mathcal{C}$ mit $u \neq v$: $\{u, v\} \in E$ oder $(u, v) \in \vec{E}$. Wenn $\mathcal{C}_{\max}$ eine größte Clique in G ist nennen wir $\omega(G) = |\mathcal{C}_{\max}|$ die* Cliquenzahl *von G.*

Definition 2.25. *Die* Färbezahl $\chi(G)$ *eines Graphen ist die Mindestanzahl an Farben, sodass G gültig gefärbt werden kann.*

Da in einer gültigen Färbung zwei adjazente Knoten unterschiedlicher Farbe sein müssen, kann eine Clique der Größe k nicht mit weniger als k Farben eingefärbt werden. Wir halten dies in einer Beobachtung fest:

Beobachtung 2.26. $\chi(G) \geq \omega(G)$

Da Intervallgraphen eine Teilmenge der Graphen bilden, die einige besondere Eigenschaften haben (siehe Satz 2.20 und 2.22), gelten für sie auch spezielle Resultate. Generell stellt die Cliquenzahl eine untere Schranke für die Anzahl an benötigten Farben dar. Bei den Intervallgraphen ist es tatsächlich so, dass die Färbezahl mit der Cliquenzahl gleich ist. Beweise für diese Behauptung finden sich unter Anderem in Lehrbüchern, beispielsweise in Douglas West's *Introduction to Graph Theory* [Wes01]. Wir zeigen einen leicht abgeänderten Beweis dieser Behauptung.

Definition 2.27. *Ein Graph G heißt* perfekt *wenn gilt: $\chi(G) = \omega(G)$.*

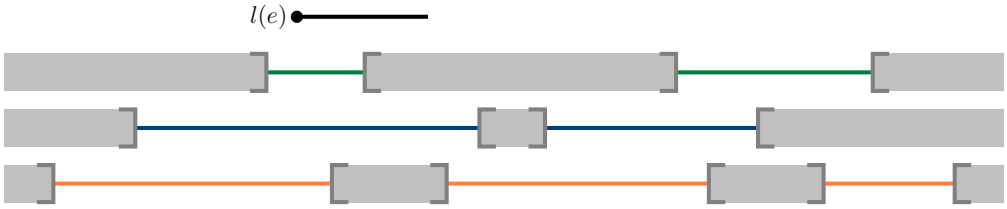


Abb. 2.7: Färbung eines Intervallgraphen. Wird ein Intervall in einer noch nicht vergebenen Farbe gefärbt, so liegt sein linker Endpunkt im Komplement aller grauen Bereiche.

Satz 2.28. *Intervallgraphen sind perfekt.*

Beweis. Wir beweisen die Behauptung durch Konstruktion einer gültigen k-Färbung. Zu Beginn sei $i = 1$ die niedrigste Farbe. Sei e das Intervall mit dem kleinsten linken Endpunkt. Wir setzen $\text{col}(e) = i$. Wir nennen ein Intervall e' *Folgeintervall* von e, wenn $e \cap e' = \emptyset$ gilt und es existiert kein e'' mit $e \cap e'' = \emptyset$ und $l(e'') < l(e')$. Sei nun e das zuletzt gefärbte Intervall und e' das Folgeintervall von e. Wir setzen $\text{col}(e') = \text{col}(e)$. Existiert kein Folgeintervall und es gibt noch ungefärbte Intervalle, so erhöhen wir i auf $i + 1$ und beginnen erneut bei dem Intervall mit dem kleinsten linken Endpunkt.

Nun gilt offensichtlich Folgendes: In jedem Schritt liegt eine gültige Färbung der bisher verarbeiteten Intervalle vor, da nie zwei Intervalle, die sich schneiden, in der selben Farbe eingefärbt wurden. Sei $k - 1$ die höchste Farbe, die wir bisher vergeben haben und e ein Intervall, das in k gefärbt werden soll. Dann gilt: $l(e)$ liegt weder weiter links als das erste, noch weiter rechts als das letzte in jeder Farbe gefärbte Intervall. Sonst wäre es in einem früheren Schritt gefärbt worden. Nun kann $l(e)$ aber auch nicht in der Lücke zwischen zwei Intervallen e' und e'' mit $\text{col}(e') = \text{col}(e'')$ und $l(e') < l(e'')$ liegen. Sonst wäre e Folgeintervall von e' und es wäre $\text{col}(e) = \text{col}(e')$.

Also muss $l(e)$ im Komplement aller Lücken zwischen Intervallen und innerhalb der ersten und letzten in einer Farbe gefärbten Intervalle liegen. Dies ist offensichtlich eine nicht-leere Schnittmenge von $k - 1$ Intervallen. Daraus folgt: Der korrespondierende Intervallgraph besitzt eine k-Clique. □

Abbildung 2.7 verdeutlicht den obigen Beweis: Das schwarze Intervall liegt mit seinem linken Endpunkt weder vor dem jeweils ersten, noch nach dem letzten in einer früheren Farbe eingefärbten Intervall. Diese Bereiche sind grau markiert. Auch die Lücken zwischen einem Intervall und seinem Folgeintervall sind grau. Das Komplement über die Vereinigung aller grauen Bereiche ist stets ein nicht-leerer Schnitt über k Intervalle, wenn k Farben bereits erschöpfend vergeben wurden.

Ein Beispiel für eine Konstruktion eines Graphen mit $\chi(G) > \omega(G)$ gibt es in Abbildung 2.8 zu sehen. Hier wurden zwei Cliquen der Größe 4 so verbunden, dass der Knoten 5 zu Knoten in vier unterschiedlichen Farben benachbart ist, ohne dass eine Clique mit Größe 5 entsteht. Es existiert keine gültige Färbung des Graphen mit weniger als fünf Farben. Der dargestellte Graph ist kein Intervallgraph, da unter Anderem der Kreis $(5, 4, 2, 3, 5)$ keine Saite hat. Darunter ist eine Menge von Intervallen zu sehen, deren Intervallgraph ein Obergraph von G ist. Die fehlenden Saiten $\{2, 5\}$ und $\{5, 6\}$ sind in

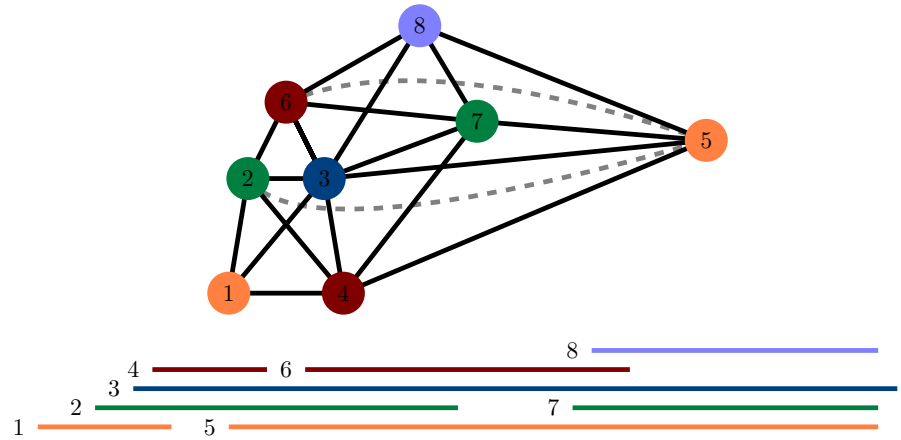


Abb. 2.8: Ein Graph mit $\chi(G) = 5$ und $\omega(G) = 4$. Fehlende Saiten sind in Grau dargestellt. Der Graph, um die Saiten $\{2, 5\}$ und $\{6, 5\}$ erweitert, korrespondiert zu den Intervallen 1 bis 8.

Grau dargestellt. Der um die fehlenden Saiten erweiterte Graph enthält mit $\{3, 6, 7, 8, 5\}$ eine Clique der Größe 5. Die Graphen, für die Satz 2.28 hält, werden *perfekte Graphen* genannt.

Die *starke Vermutung zu den perfekten Graphen* ist eine 1961 von Claude Berge geäußerte Vermutung, der nach die perfekten Graphen folgender Maßen charakterisiert sind: Ein Graph ist perfekt, genau dann, wenn weder der Graph noch sein Komplement einen induzierten Kreis ungerader Länge besitzen [Ber61]. Diese Vermutung wurde erst 2006 von Chudnovsky et al. durch den *starken Satz zu den perfekte Graphen* bewiesen [CRST06]. Brückner [Brü21] konnte mit Satz 2.30 zeigen, dass der erweiterte Konfliktgraph (siehe Abschnitt 5.1) perfekt ist. Wir werden in Kapitel 5 einen alternativen Beweis dieses Resultats als Korollar zu unserem Hauptresultat erhalten.

Definition 2.29. *Ein Graph G ist ein* Berge-Graph*, genau dann, wenn jeder induzierte Kreis ungerader Länge größer* 3 *in G sowie seinem Komplement $\overline{G}$ eine Saite besitzt.*

Satz 2.30. *[CRST06] Ein Graph G ist ein* Berge-Graph*, genau dann, wenn G perfekt ist.*

Wir haben in Kapitel 1 über Kabelpläne gesprochen, die wir algorithmisch zeichnen wollen. Dazu haben wir das Layer-Assignment-Problem formuliert, welches eine Äquivalenz zum Färben eines gemischten Intervallgraphen aufweist (siehe Satz 5.3). Im vorangegangenen Kapitel haben wir die graphentheoretischen Grundlagen kennengelernt, auf die wir in Kapitel 5 zurückgreifen. Im folgenden Abschnitt untersuchen wir einen Lösungsansatz für das Layer-Assignment-Problem.

3 Greedy-Verfahren

Wir wollen eine Lösung für das Layer-Assignment-Problem (siehe Definition 2.9) berechnen. Dazu betrachten wir ein simples, iteratives Platzierungsverfahren, das von Zink et al. [ZWBW22] verwendet wurde, um die Intervalle den Ebenen zuzuordnen. Dabei wird in jedem Berechnungsschritt ein Intervall betrachtet, und dieses in der bereits berechneten Teillösung auf die tiefste, zulässige Ebene platziert. Dies entspricht der Lösungsstrategie zum Färben von Intervallgraphen, wie sie in der Literatur vorkommt [Wes01]. In jedem Schritt wird ein bestimmtes Kriterium optimiert. Hier ist dies die Ebene, auf der ein Intervall platziert wird. Eine auf diese Weise einmal getroffene Entscheidung wird im späteren Verlauf der Berechnung nicht mehr revidiert. Damit zählt das Verfahren zu den *Greedy-Algorithmen* (von engl. für *gierig*). Wir wollen dieses Verfahren zum Lösen von Layer-Assignment anpassen und eine konkrete Implementierung besprechen. Wir nennen den Algorithmus Greedy-Layer-Assignment.

Sei I die Eingabemenge und L die gültige Teil-Lösung, die nach Platzieren einer gewissen Anzahl an Intervallen aus I entstanden ist, und $e \in I \setminus L$ ein noch nicht in die Lösung platziertes Intervall aus der Eingabe. Dann ist eine Ebene $\varphi(e)$ *zulässig*, wenn für alle $e' \in L$, die zu e gerichtet sind, $\varphi(e') < \varphi(e)$ gilt und wenn für alle $e' \in L$ mit $\varphi(e') = \varphi(e)$ gilt: $e' \cap e = \emptyset$. Dies folgt direkt aus Definition 2.8.

Beobachtung 3.1. *Sei L eine gültige Teil-Lösung von I und $e \in I \setminus L$ ein Intervall, das noch nicht platziert wurde. Dann ist $\varphi(e)$* zulässig*, wenn für alle $e' \in L$ zwei Bedingungen erfüllt sind:*

1. *e' ist gerichtet zu $e \Rightarrow \varphi(e') < \varphi(e)$*
2. *$\varphi(e') = \varphi(e) \Rightarrow e' \cap e = \emptyset$*

Wenn wir die Intervalle in aufsteigender Reihenfolge ihrer linken Endpunkte verarbeiten ist sichergestellt, dass in jedem Schritt alle bereits platzierten Intervalle weiter links beginnen als das im aktuellen Schritt der Berechnung betrachtete. Dies ist wichtig, da laut Beobachtung 3.1 eine Ebene nur zulässig ist, wenn alle Intervalle, die bereits platziert wurden, und die gerichtet sind zu einem noch nicht platzierten Intervall, darunter liegen. Angenommen, im Verlauf der Berechnung wird ein Intervall e betrachtet, und ein Intervall e' mit $l(e) < l(e')$ ist bereits einer Ebene zugeordnet. Wenn e' zu e gerichtet ist, dann müssten die Zuordnung von e' revidiert werden, wenn unterhalb von e' keine zulässige Ebene existiert. Aus diesem Grund wollen wir in jedem Schritt, dass jedes bereits platzierte Intervall weiter links beginnt als alle noch zu verarbeitenden.

In jedem Schritt wird dann das jeweils zu verarbeitende Intervall $e \in I \setminus L$ auf die niedrigste zulässige Ebene in der Teillösung L platziert. Daraus folgt auch, dass L zu

jedem Zeitpunkt während der Berechnung eine gültige Lösung für die bereits verarbeiteten Intervalle ist. Beachte, dass wegen der Sortierung für jedes Paar an Intervallen $e \in I \setminus L$ und $e' \in L$ gilt: $l(e') < l(e)$. Folglich hängt die Art der Überlappung nur noch von e und $r(e')$ ab.

Beobachtung 3.2. *Sei I eine Menge von Intervallen, L eine Teillösung für I, $e \in I \setminus L$ und $e' \in L$. Dann hängt die Überlappung von e, e' nur noch von e und $r(e')$ ab.*

Beweis. Es gilt $l(e') < l(e)$, da e' vor e verarbeitet wurde. Laut Definition 2.6 unterscheiden wir bei $e \cap e' \neq \emptyset$ zwischen vollständiger und teilweiser Überlappung. Wenn $r(e') < l(e)$ gilt, dann ist offensichtlich $e \cap e' = \emptyset$. Ist hingegen $r(e') > l(r)$, so gilt $e \cap e' \neq \emptyset$. Dann ist entweder $r(e') > r(e)$ oder $r(e') < r(e)$. Im ersten Fall ist die Überlappung vollständig, im zweiten teilweise. □

Wenn L eine Teillösung mit m Ebenen ist, und $e \in I \setminus L$ soll platziert werden, dann ist $m + 1$ offenbar eine zulässige Ebene. Denn dann gilt für alle $e' \in L$, die e schneiden, und damit insbesondere für alle e', die zu e gerichtet sind: $\varphi(e') < m + 1$.

Beobachtung 3.3. *Sei L eine Teillösung mit m Ebenen und $e \in I \setminus L$ ein Intervall. Dann ist $\varphi(e) = m + 1$ eine zulässige Ebene.*

Soll e in die Teillösung platziert werden, und es existiert keine zulässige Ebene kleiner $m + 1$, so ist $\varphi(e) = m + 1$ die *niedrigste zulässige Ebene* für e. Werden die Ebenen von L nun in absteigender Reihenfolge, beginnend bei m, betrachtet und die oberste Ebene, auf der sich ein zu e gerichtetes Intervall befindet sei k, so gilt für jede zulässige Ebene: $\varphi(e) > k$. Wir nennen ein höchstes, zu e gerichtetes Intervall in L *Auflieger* $a(e)$ und $\varphi(a(e))$ die *Aufliegerebene* von e (vgl. Definition 5.4).

Definition 3.4. *Sei L eine Teillösung und $e \in I \setminus L$ ein Intervall. Wenn $a(e) \in L$ das höchste, zu e gerichtete Intervall in L ist, dann heißt $\varphi(a(e))$* Aufliegerebene *von e.*

Existiert zu einem $e \in I \setminus L$ kein Auflieger, so gilt für alle zulässigen Ebenen offenbar: $\varphi(e) > 0$, wenn 1 die niedrigste Ebene ist, die wir vergeben. Folglich gehen wir fortan davon aus, dass für ein Intervall, das keinen Auflieger hat, $\varphi(a(e)) = 0$ gilt.

Beobachtung 3.5. *Sei L eine Teillösung und $e \in I \setminus L$ ein Intervall. Für jede zulässigen Ebene $\varphi(e)$ gilt: $\varphi(e) > \varphi(a(e))$.*

Wenn $\varphi(a(e))$ die Aufliegerebene für e ist und wir eine Ebenen k' mit $\varphi(a(e)) < k' < m + 1$ finden, sodass kein $e' \in L$ mit $\varphi(e') = k'$ sich mit e schneidet, so ist k' eine zulässige Ebene. Insbesondere ist k' eine niedrigere zulässige Ebene als $m+1$. Wir können beim abwärts Iterieren durch die Ebenen von L die niedrigste zulässige Ebene unter den bereits betrachteten Ebenen aufrechterhalten. Erreichen wir die Aufliegerebene, so ist die niedrigste bisher gefundene zulässige Ebene minimal unter allen zulässigen Ebenen.

Behauptung 3.6. *Sei L eine Teillösung mit Höhe m und $e \in I \setminus L$. Weiter sei $\varphi(a(e))$ die Aufliegerebene von e. Wenn wir in konstanter Zeit für jede Ebene k mit $\varphi(a(e)) < k < m + 1$ testen können, ob sie zulässig ist, dann finden wir die niedrigste zulässige Ebene in $\mathcal{O}(m)$ Schritten.*

In Beobachtung 3.2 haben wir gesehen, dass wir beim Platzieren von e nur den rechten Endpunkt eines Intervalls in L mit e vergleichen müssen. Wir können sogar noch einen Schritt weiter gehen: Auf jeder Ebene der Teillösung ist nur der am weitesten rechts liegende rechte Endpunkt interessant.

Lemma 3.7. *Sei L eine Teillösung, $e \in I \setminus L$ ein Intervall und k eine Ebene in L. Wenn $e' = \max\{r(e') \mid \varphi(e') = k, e' \in L\}$ gilt, dann ist $e \cap e'' = \emptyset$ für alle $e'' \neq e'$ mit $\varphi(e'') = k$ und $e'' \in L$.*

Beweis. Angenommen es existiert ein $e'' \in L$ mit $\varphi(e'') = k$, sodass $e \cap e'' \neq \emptyset$ gilt. Da wegen der Sortierung der Eingabe $l(e'') < l(e)$ gilt, muss $r(e'') > l(e)$ gelten. Sonst würden e und e'' nicht überlappen. Nun ist aber auch $l(e') < l(e)$. Ebenfalls wegen der Sortierung der Eingabe. Dann muss $r(e'') > l(e')$ sein und es gilt: $e' \cap e'' \neq \emptyset$. In diesem Fall ist L keine gültige Teillösung. Daraus folgt, dass in einer gültigen Lösung $e'' \cap e = \emptyset$ gilt. □

Wegen Lemma 3.7 ist also klar: Wenn die Intervalle aufsteigend nach linken Endpunkten sortiert verarbeitet werden, und ein Intervall e soll in die Teillösung L platziert werden, dann ist e in jedem Fall disjunkt zu allen Intervallen einer Ebene, außer dem am weitesten rechts befindlichen. Aus diesem Grund genügt es, e nur mit dem „rechtesten“ rechten Endpunkt einer jeden Ebene zu vergleichen. Wir bezeichnen diesen mit $\max_{r(k)}$, wenn $k \in \mathbb{N}$ eine Ebene der Teillösung ist. Mit Lemma 3.7 können wir nun Behauptung 3.6 beweisen.

Definition 3.8. *Sei $k \in \mathbb{N}$ eine Ebene in einer Teillösung L. Dann ist $\max_{r(k)} = \max\{r(e) \mid \varphi(e) = k, e \in L\}$ der rechteste rechte Endpunkt eines Intervalls auf Ebene k.*

Lemma 3.9. *Sei L eine Teillösung mit Höhe m und $e \in I \setminus L$. Wir können in $\mathcal{O}(m)$ Schritten die niedrigste zulässige Ebene $\varphi(e)$ finden.*

Beweis. Aus Lemma 3.7 folgt, dass auf jeder der m Ebenen in L nur $\max_{r(k)}$ benötigt wird, um eine Überschneidung mit e zu testen. Dies ist mit konstantem Aufwand möglich. Daraus folgt, dass wir in höchstens $\mathcal{O}(m)$ Schritten die niedrigste zulässige Ebene für e finden. □

Gehen wir nun auf eine mögliche Implementierung von Greedy-Layer-Assignment ein. Dazu benötigen wir Datenstrukturen, die uns das testen jeder Ebene mit konstantem Rechenaufwand erlauben. Eine mögliche Datenstruktur *Solution* zum Verwalten von L kann wie folgt aussehen: Eine iterierbare Sammlung von Ebenen, wobei eine Ebene eine Sammlung von Intervallen ist. Nach Initialisierung enthält *Solution* nur eine leere Ebene an der Stelle 1. Eine Anfrage *Solution*$[k]$ gibt Ebene k wieder, wenn $k \in \mathbb{N}$ ist und die Lösung mindestens k Ebenen hat. Jeder Ebene in *Solution* kann mit $Add(e)$ ein Intervall e zugefügt werden. Wird mit *Solution*$[k].Add(e)$ ein Intervall zugefügt, und die Lösung hat nur $m < k$ Ebenen, so wird eine leere Ebene an der Stelle $m + 1$ zugefügt. Eine Ebene kann als Zusatzinformation $\max_{r(k)}$ enthalten. Diese lässt sich, wann immer ein Intervall dieser Ebene zugeordnet wird, aktualisieren und mit einem Befehl $MaxR()$

abrufen. Dabei ist $\max_{r(k)}$ stets der rechte Endpunkt des zuletzt zugefügten Intervalls. Eine Anfrage *Solution*$[k]$.*maxR*() gibt $\max_{r(k)}$ wieder. Um die berechnete Lösung ausgeben zu lassen, kann beispielsweise jede Ebene eine Liste ihrer enthaltenen Intervalle ausgegeben. Um während der Berechnung die jeweils aktuelle niedrigste Ebene zu speichern, verwenden wir eine Variable *candidate*. Diese wird zu Beginn jedes Schritts auf $m+1$ gesetzt, wenn m die aktuelle Höhe der Lösung ist. Sie wird aktualisiert, wenn im Verlauf der Berechnung eine zulässige Ebene kleiner $m+1$ gefunden wird.

Eine Unterroutine von Greedy-Layer-Assignment ist *Intersect*(e, k). Sie erhält ein Intervall e und ein $k \in \mathbb{N}$. Nun ruft *Intersect* die Methode *Solution*$[k]$.*maxR*() auf und vergleicht e mit $\max_{r(k)}$. Dabei gibt *Intersect* als Rückgabewert das Schlüsselwort *none* zurück, wenn $l(e) > \max_{r(k)}$ gilt. Wenn nicht, und es gilt $r(e) > \max_{r(k)}$, wird *partial* zurückgegeben. Schlägt auch dies fehl, wird die Anfrage schließlich mit *total* quittiert. Der Rückgabewert von *Intersect* wird in einer Variable *intersection* gespeichert.

Algorithmus 1: Greedy-Layer-Assignment(I^*)

Eingabe: Aufsteigend nach linken Endpunkten sortierte Liste an Intervallen I^*
Ausgabe: *Solution* L, in der jedes e aus I^* einer Ebene zugewiesen ist

```
1  L = new Solution()
2  foreach e in I* do
3      candidate = L.size() + 1 // L.size() ist Anzahl der Ebenen in L
4      for i = L.size() downto 1 do
5          intersection = Intersect(e, i)
6          if intersection == partial then
7              break
8          else if intersection == none then
9              candidate = i
10     L[candidate].Add(e)
11 return L
```

3.1 Korrektheit

Sei $(e_1, e_2, \ldots, e_n)$ die aufsteigend nach linken Endpunkten sortierte Eingabemenge. Zu Beginn von Schritt i sind die Intervalle e_1 bis e_{i-1} mit den $i-1$ kleinsten x-Koordinaten ihrer linken Endpunkte bereits verarbeitet und *Solution* sei eine gültige Teillösung gemäß Definition 2.8 mit Höhe m. Wir betrachten e_i und *candidate* wird auf $m+1$ gesetzt. Existiert keine zulässige Ebene $k \leq m$ für e_i in *Solution*, so ist $m+1$ offensichtlich die niedrigste zulässige Ebene. Dann wird *Solution* um eine Ebene auf $m+1$ vergrößert und e_i dort platziert.

Anschließend iteriert der Algorithmus abwärts durch die Ebenen. Die Laufvariable i wird zu Beginn jedes Rechenschritts auf die Höhe der Teillösung gesetzt und bis 1 runtergezählt. Auf jeder Ebene, die durchlaufen wird, testet der Algorithmus mit *Intersect*,

welche Art von Überlappung e_i mit Intervallen auf dieser Ebene hat. Wegen Lemma 3.7 genügt es hier, nur $\max_{r(k)}$ zu betrachten. Nun gibt es drei mögliche Rückgabewerte für *Intersect*: *total*, *partial* und *none*. Ist die Überlappung *partial*, so befindet sich auf Ebene i Intervall e', das zu e_i gerichtet ist. Der Algorithmus beendet den aktuellen Rechenschritt und platziert e_i auf der Ebene *candidate*. Wenn *candidate* eine zulässige Ebene ist, dann auch *die niedrigste zulässge Ebene*, da in jeder gültigen Lösung $\varphi(e') < \varphi(e_i)$ gelten muss und der Rechenschritt bei der ersten Ebene, für die *partial* zurückgegeben wird, die Schleife verlässt. Da wir die Ebenen abwärts iterieren, ist dies offensichtlich $\varphi(a(e))$.

Gibt *Intersect* als Rückgabewert *none* zurück, so befindet sich auf Ebene i kein Intervall, das e_i überlappt. In diesem Fall wird *candidate* auf i gesetzt. Wenn kein e' mit $\varphi(e') > i$ existiert, das zu e gerichtet ist, dann ist i eine zulässige Ebene für e_i. Den Fall, dass ein zu e_i gerichtetes Intervall e' mit $\varphi(e') > i$ existiert, fängt der Algorithmus bereits mit *partial* ab. Da im Falle *none* kein e' mit $\varphi(e') = i$ und $e_i \cap e' \neq \emptyset$ existiert, ist i eine zulässige Ebene. Da i nur bis 1 runtergezählt wird, endet die Schleife in jedem Fall nach diesem Schritt. Dann wird e auf der Ebene *candidate* platziert.

Bleibt noch der Fall, dass weder *partial* noch *none* zurückgegeben werden. In diesem Fall liegt zwischen e_i und einem e' auf Ebene i eine vollständige Überlappung vor. Dann ist i keine zulässige Ebene für e_i. Der Algorithmus führt in diesem Fall die Schleife fort und *candidate* bleibt unverändert. Auch dies ist korrekt, da *candidate* dann noch immer eine zulässige Ebene ist, aber noch Ebenen unterhalb von i existieren können, die zulässig sind. Unter der Annahme, dass wir eine Menge von rechtsgerichteten Intervallen, aufsteigend nach linken Endpunkten sortiert erhalten, ist Greedy-Layer-Assignment korrekt.

Ein konkretes Beispiel eines Berechnungsschrittes in einer Teillösung mit $m = 7$, also bislang sieben Ebenen, sieht man in Abbildung 3.1. Wenn e_i platziert werden soll, beginnt der Algorithmus, indem er $\varphi(e_i) = m + 1 = 8$ vormerkt. Finden wir keine Ebene kleiner 8, auf der e_i liegen darf, so wird e_i dort platziert. Der Algorithmus findet auf Ebenen 7, 6 und 5 Intervalle, die zu e_i ungerichtet sind. In diesem Fall ändert sich an $\varphi(e_i)$ nichts. Auf Ebene 4 befindet sich kein Intervall, das e_i überlappt und $\varphi(e_i) = 4$ wird notiert. Auf Ebene 3 befindet sich erneut ein zu e_i ungerichtetes Intervall und $\varphi(e_i)$ bleibt unverändert. Auf Ebene 2 befindet sich Intervall e', welches gerichtet zu e_i ist und der Algorithmus terminiert mit $\varphi(e_i) = 4$.

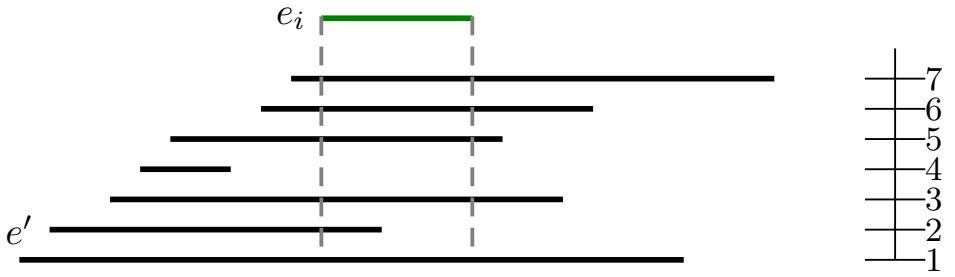


Abb. 3.1: Ein konkretes Beispiel eines Berechnungsschritts in Greedy-Layer-Assignment.

Der Algorithmus lässt sich leicht anpassen, um auch für eine Menge linksgerichteter

Leitungen die Intervalle korrekt zu platzieren. Dazu ist lediglich notwendig, dass jede Ebene auch den linken Endpunkt des zuletzt zugefügten Intervalls auf dieser Ebene zurückgibt. Die Untermethode *Intersect* ist dann so zu erweitern, dass vollständige und teilweise Überlappung allgemein gemäß Definition 2.6 getestet werden. Dann ist leicht zu sehen, dass GREEDY-LAYER-ASSIGNMENT auch den linksgerichteten Fall korrekt löst, wenn die Eingabe *absteigend nach rechten Endpunkten* sortiert ist.

3.2 Analyse

Zur Vorverarbeitung einer Menge von n Intervallen kann ein Sortier-Algorithmus mit Laufzeit in $\mathcal{O}(n \log n)$ verwendet werden. Die Lösung kann als Liste und jede Ebene darin ebenfalls als Liste verwaltet werden. Die Methoden *Intersect*, *Add* und *MaxR* haben jeweils konstanten Rechenaufwand. Die Intervalle können mit ihren linken und rechten Endpunkten als zwei Gleitkommazahlen gespeichert werden. Man kann auch eine *Normierung* der Eingabe durchführen, wie sie von Brückner [Brü21] vorgeschlagen wurde. Dabei wird jeder Endpunkt $\text{end}(e)$ mit $e \in I$ und $\text{end} \in \{l, r\}$ durch eine normierte Koordinate end_{norm} ersetzt, sodass aus $\text{end}(e) < \text{end}(e')$ mit $e, e' \in I$ stets folgt: $\text{end}_{\text{norm}}(e) < \text{end}_{\text{norm}}(e')$. Da dabei für jedes Paar an Endpunkten aus der Eingabe ihre Reihenfolge erhalten bleibt, entspricht eine Lösung der normierten Instanz auch einer Lösung der ursprünglichen Eingabe. Der Platzbedarf für die Lösung ist dann in $\mathcal{O}(n)$, wenn n die Größe der Eingabemenge ist. Algorithmus 1 verwendet nur drei Variablen: Die Laufvariable i, *intersection* und *candidate*, was nur konstanten zusätzlichen Speicher erfordert. Für eine Ebene die Art der Überlappung zu testen ist in konstanter Zeit möglich. Wegen Lemma 3.7 können wir die niedrigste zulässige Ebene in $\mathcal{O}(n)$ finden, da eine Lösung bis zu n viele Ebenen haben kann. Für n Intervalle die niedrigste zulässige Ebene zu finden ist dann in $\mathcal{O}(n^2)$ Zeit möglich. Damit ergibt sich ein Gesamtaufwand für GREEDY-LAYER-ASSIGNMENT von $\mathcal{O}(n \log n) + \mathcal{O}(n^2) = \mathcal{O}(n^2)$ Zeit und $\mathcal{O}(n)$ Platz.

Satz 3.10. *Das LAYER-ASSIGNMENT-PROBLEM kann in $\mathcal{O}(n^2)$ Zeit und $\mathcal{O}(n)$ Platz gelöst werden, wenn $n = |I|$ die Größer der Eingabe ist.*

Wir wollen das LAYER-ASSIGNMENT-PROBLEM schneller lösen und zeigen im folgenden Kapitel, wie mit spezialisierten Datenstrukturen die niedrigste, gültige Ebene in jedem Schritt der Berechnung, mit logarithmischem Zeitaufwand ermittelt werden kann. Zuerst stellen wir eine geeignete Datenstruktur vor, dann passen wir diese für unsere Zwecke an, um effizient ein Intervall in einer Teillösung zu platzieren.

4 Laufzeitverbesserung

Algorithmus 1 besteht im Grunde aus zwei geschachtelten Schleifen. In der äußeren Schleife werden die Intervalle der Eingabemenge in aufsteigender Reihenfolge ihrer linken Endpunkte verarbeitet. In der inneren Schleife wird das gerade betrachtete Intervall in der bisher entstandenen Teillösung platziert. Dabei wird für jedes $e \in I \setminus L$ die niedrigste zulässige Ebene in L gesucht (siehe Beobachtung 3.1). Wir wollen die Laufzeit für Greedy-Layer-Assignment verbessern und behaupten, dass wir die niedrigste zulässige Ebene in $\mathcal{O}(\log n)$ Zeit finden können. Wir bewerkstelligen dies mit zwei augmentierten Rot-Schwarz-Bäumen (kurz RS-Baum), in denen die Teillösung verwaltet wird.

Sei e das zu platzierende Intervall und L eine Teillösung mit Höhe m. Wir nennen eine Ebene k *frei*, wenn $\max_{r(k)} < l(e)$ gilt. Aus Lemma 3.7 wissen wir, dass auf jeder Ebene nur $\max_{r(k)}$ benötigt wird, um eine Überlappung mit e zu testen. Folglich kann auf einer Ebene mit $\max_{r(k)} < l(e)$ kein Intervall in L eine Überlappung mit e haben. Und wegen Beobachtung 3.5 wissen wir: Nur die Ebenen oberhalb von $\varphi(a(e))$ kommen als zulässig in Frage. Wir teilen die Suche nach der niedrigsten zulässigen Ebene in zwei Anfragen auf, die wir effizient beantworten können:

Definition 4.1. *Sei L eine Teillösung mit Höhe m, $k \in \mathbb{N}$ und $e \notin L$ das zu platzierende Intervall. Wir berechnen die niedrigste zulässige Ebene für e mit zwei Anfragen:*

- *Finde Aufliegerebene* $\varphi(a(e)) = \max\{k \in [0, m] \mid l(e) < \max_{r(k)} < r(e)\}$.
- *Berechne* $\min\{k' \in [\varphi(a(e)) + 1, m + 1] \mid \max_{r(k')} < l(e)\}$.

Wir verwenden je einen spezialisierten RS-Baum mit unterschiedlichen Augmenten, für jede der zwei Anfragen aus Definition 4.1. Wollen wir ein $e \in I \setminus L$ platzieren, so benötigen wir $\varphi(a(e))$, um die niedrigste zulässige Ebene zu finden. Daher widmen wir uns zunächst der Berechnung von $\varphi(a(e))$. Danach verwenden wir $\varphi(a(e))$, um in einem zweiten RS-Baum die niedrigste zulässige Ebene zu finden.

4.1 Balancierte Bäume

Wir schlagen zur Verbesserung der Laufzeit von Greedy-Layer-Assignment einen *binären Suchbaum* oder kurz *Baum* vor. Dieser ist eine Datenstruktur, in der *Knoten* verwaltet werden. In jedem Knoten ist ein *Schlüssel* gespeichert. Jedem Knoten werden bis zu zwei *Kindknoten* und ein *Elternknoten* zugeordnet. Der Baum besitzt genau einen Knoten genannt *Wurzel*, der nicht Kind eines anderen Knoten ist.

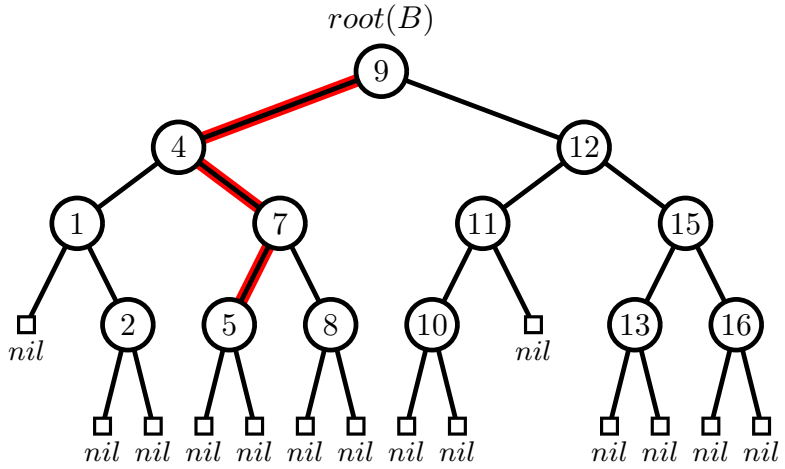


Abb. 4.1: Ein Baum. In Rot ein längster Wurzel-Blatt-Pfad, der bei einer erfolglosen Anfrage für $key = 6$ abgeschritten wird.

Definition 4.2. *Ein* Baum B *ist eine Datenstruktur, in der eine Menge von* Knoten *verwaltet wird. Ein Knoten* $v \in B$ *ist ein Quadrupel* $(key, parent, left, right)$ *mit* $key \in \mathbb{R}$ *und* $parent, left, right \in B$ *mit* $parent(v) \neq left(v) \neq right(v) \neq v$*. Dabei gilt stets:* $parent(v') = v \iff \text{rel}(v) = v'$ *mit* $\text{rel} \in \{left, right\}$*. Weiter besitzt* B genau einen *Knoten* $root \in B$*, sodass kein* $v \in B \setminus root(B)$ *existiert mit* $right(v) = root(B)$ *oder* $left(v) = root(B)$*. Es gibt keine zwei verschiedenen Knoten* $v, v' \in B$*, sodass* $\text{rel}(v) = \text{rel}(v')$ *gilt mit* $\text{rel} \in \{left, right\}$*.*

Wenn $v, v' \in B$ verschiedene Knoten sind, sodass $left(v) = v'$ oder $right(v) = v'$, dann nennen wir v' *linkes* respektive *rechtes Kind* von v. Umgekehrt heißt dann v *Elternknoten* von v'. Es kann sein, dass einem Knoten kein Kind zugewiesen ist. Ebenso ist per Definition 4.2 $root(B)$ kein Elternknoten zugewiesen. Für diesen Fall besitzt B einen Platzhalter *nil*. Wir schreiben $\text{rel}(v) = nil$ mit $\text{rel} \in \{parent, left, right\}$. Wir sagen, ein $v \in B$ ist ein *Blatt*, wenn gilt: $right(v) = left(v) = nil$.

Ein *Pfad* $w = (v_1, v_2, \ldots, v_k)$ der Länge k in einem Baum ist eine Abfolge von Knoten $v_i \in B$ mit $i \in \mathbb{N}$ und $i \in [1, k]$, sodass für Knoten v_i, v_{i+1} entweder $right(v_i) = v_{i+1}$ oder $left(v_i) = v_{i+1}$ gilt. Der Pfad *verläuft nach rechts*, wenn $right(v_i) = v_{i+1}$ gilt, andernfalls verläuft er *nach links*. Wenn $v_i, v_j \in w$ und $i < j$ gilt, so nennen wir v_i *Vorfahr* von v_j und v_j *Nachfahr* von v_i. Die Länge eines längsten Pfades von $root(B)$ zu einem Blatt heißt *Höhe* $h(B)$. Alle $v \in B \setminus root(B)$ sind Nachfahr von $root(B)$ und $root(B)$ besitzt keine Vorfahren. Alle $v \in B \setminus root(B)$ heißen *innere Knoten* von B. Ein *Teilbaum* $B' \subseteq B$ ist eine *zusammenhängende* Teilmenge der Knoten aus B. Zusammenhängend heißt hier, dass ein $v \in B'$ existiert, sodass alle $v' \in B' \setminus v$ Nachfahr von v sind. In diesem Fall ist v die Wurzel von B' und wir schreiben $root(B') = v$. Ist $right(v)$ die Wurzel eines B', so sprechen wir vom *rechten Teilbaum* von v, andernfalls vom *linken Teilbaum* von v. Offensichtlich ist $root(B)$ Vorfahr aller inneren Knoten.

Die Knoten in B unterliegen einer Ordnung. Für jeden $v \in B$ gilt: Die Schlüssel aller

Knoten im linken Teilbaum von v sind höchstens so groß wie die Schlüssel aller Knoten in jedem rechten Teilbaum von v.

Definition 4.3. *Ein Baum B heißt* geordnet*, wenn für alle $v \in B$ gilt: $key(left(v)) \leq key(right(v))$.*

Lemma 4.4. *Sei $x \in \mathbb{R}$. Man kann in $\mathcal{O}(h(B))$ Vergleichen testen, ob ein $v \in B$ existiert mit $key(v) = x$.*

Beweis. Sei B ein geordneter Baum laut Definition 4.3 und v_{root} die Wurzel von B. Wenn $key(v_{\text{root}}) = x$ gilt, so ist die Anfrage offenbar beantwortet. Andernfalls gilt entweder $key(v_{\text{root}}) < x$ oder $key(v_{\text{root}}) > x$. Ohne Beschränkung der Gültigkeit sei $key(v_{\text{root}}) < x$. Wegen Definition 4.3 gilt für alle Knoten im linken Teilbaum unter v_{root}, dass ihr Schlüssel höchstens so groß ist wie die Schlüssel aller Knoten im rechten Teilbaum. Daraus folgt, dass im linken Teilbaum kein Knoten existieren kann, für den $key(v) = x$ gilt. In diesem Fall wird die Anfrage im rechten Teilbaum fortgesetzt. Verfährt man hier rekursiv, so bildet die Anfrage einen Pfad durch B, dessen Länge durch $h(B)$ beschränkt ist. Stößt man dabei auf den gesuchten Knoten oder ein Blatt, so ist die Anfrage beantwortet. Da man in jedem Schritt höchstens eine konstante Anzahl von Vergleichen benötigt, liegt der Rechenaufwand für eine Anfrage in $\mathcal{O}(1) \cdot h(B) \in \mathcal{O}(h(B))$. □

Betrachten wir das Beispiel in Abbildung 4.3: Hier wurde eine Anfrage nach einem Knoten mit $key = 6$ gestellt. Der Pfad, den man bei so einer Anfrage abschreitet, ist hier in Rot hervorgehoben. Die Effizienz einer Anfrage hängt von der Höhe des Baumes ab. Es existieren Bäume, die eine garantierte Höhe, relativ zur Anzahl der Knoten im Baum aufweisen. Diese werden *selbstbalancierend* genannt. Beispiele für Bäume, die diese Eigenschaft implementieren und bei Einfüge- bzw. Löschoperationen aufrechterhalten, sind unter Anderem RS-Baum, AVL-Baum (benannt nach Adelson-Velskii und Landis) sowie *bounded balance-* oder $BB[\alpha]$-Baum [AVL62, NR73, Bay72].

Der RS-Baum ist ein höhenbalancierter binärer Suchbaum, der auf Bayer[Bay72] zurückgeht. Dort wird eine Baumstruktur, genannt *symmetrischer, binärer B-Baum* beschrieben, deren Knoten durch eine von zwei möglichen Typen von gerichteten Kanten verbunden sind: Vertikale δ- und horizontale ρ-Kanten (Anmerkung: Diese gerichteten Kanten entsprechen unserer Zuordnung als Kindknoten. Eine gerichtete Kante (u, v) bedeutet: u ist Elternknoten von v, v ist Kindknoten von u). Die Regeln, nach denen diese Bäume balanciert werden, lauten wie folgt: Auf allen Wurzel-Blatt-Pfaden befindet sich die gleiche Anzahl an δ-Kanten. Alle Knoten, die keine Blätter sind, haben zwei Kinder. Es gibt keine zwei aufeinanderfolgenden ρ-Kanten. Bayer bewies, dass sein Baum eine garantierte Höhe von höchstens $2 \cdot \log_2(n + 2) - 2$ hat, wenn n die Anzahl an Knoten in B ist. Die RS-Bäume, wie sie heute in Lehrbüchern vorkommen, verwenden anstelle der zwei Typen von Kanten die Farben Rot und Schwarz, welche den Knoten zugeordnet werden. Diese Definition, die unter Anderem in Cormen et al. [CLRS09] verwendet wird, geht auf Guibas und Sedgewick [GS78] zurück.

Lemma 4.5. *[CLRS09, Lemma 13.1] Ein Rot-Schwarz-Baum mit n inneren Knoten hat Höhe $h \leq 2 \cdot \log_2(n + 1)$ und implementiert Suchen, Einfügen und Löschen eines beliebigen Knoten in $\mathcal{O}(\log n)$.*

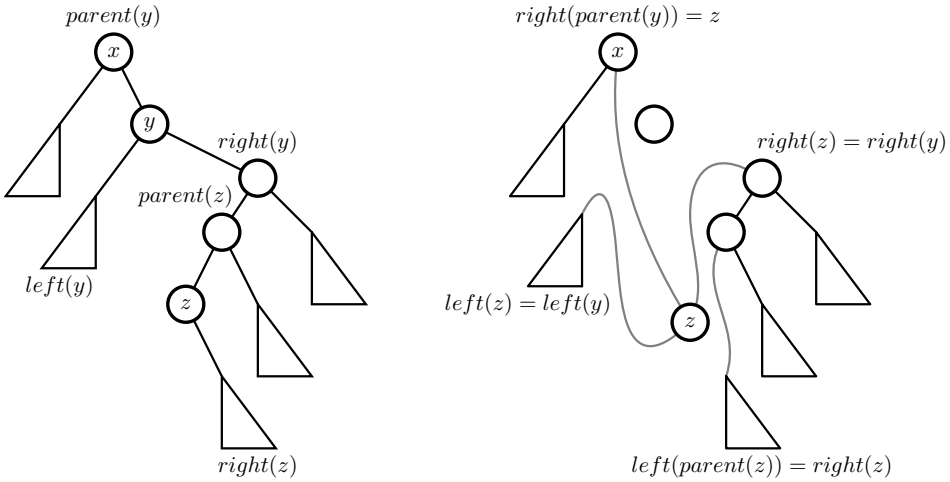


Abb. 4.2: Ein Baum, aus dem y gelöscht werden soll. Dabei sei y rechtes Kind seines Elternknotens x und z der Knoten mit dem nächst-größeren Schlüssel als y. Nach der Neuzuordnung der Eltern- und Kindknoten kann y aus dem Baum gelöscht werden. Leere *nil*-Knoten wurden weggelassen und Teilbäume als Dreiecke stilisiert.

Verwendet man einen RS-Baum, so ist auch nach wiederholten Einfüge- und Löschoperationen die Höhe des Baumes durch $\mathcal{O}(\log n)$ beschränkt. Wir werden in den folgenden Abschnitten zeigen, wie ein Baum mit leichten Anpassungen unsere Anfragen beantworten kann. Dabei gehen wir im Folgenden stets davon aus, dass die verwendete Datenstruktur ein selbstbalancierender Baum ist und die Eigenschaft aus Lemma 4.5 erfüllt. Abbildung 4.2 zeigt, wie RS-Bäume eine Löschoperation durchführen. Dabei werden stets nur eine konstante Anzahl an Operationen benötigt, um die balancierte Eigenschaft aufrecht zu erhalten.

4.1.1 Bereichsabfrage in binären Suchbäumen

Wir wollen die Laufzeit zum Lösen von Greedy-Layer-Assignment verbessern. Der RS-Baum stellt eine geeignete Datenstruktur dar, um dieses Ziel zu erreichen. Wir haben den Rechenaufwand, die niedrigste zulässige Ebene für ein Intervall $e \in I \setminus L$ zu berechnen, in zwei Anfragen aufgeteilt (siehe Definition 4.1): Berechne $\varphi(a(e))$ und finde die niedrigste freie Ebene, die größer als $\varphi(a(e))$ ist. Wir berechnen zuerst $\varphi(a(e))$. Danach lässt sich mit wenig Aufwand zeigen, dass ein leicht angepasster Baum auch die zweite Anfrage effizient beantworten kann. Sei L eine Teillösung mit m Ebenen und B ein Baum, sodass jede Ebene in L durch genau einen Knoten in B repräsentiert wird. Offenbar gilt dann $n = m$, wenn n die Anzahl an Knoten in B ist. Wir wollen den Baum nach $\max_{r(k)}$ ordnen. Sei $i \in B$ derjenige Knoten, der Ebene i repräsentiert. Dann gilt: $key(i) = \max_{r(i)}$. Wir nennen diesen Baum $B_{\max(R)}$

Definition 4.6. *Sei L eine Teillösung und $\{1, 2, \ldots, m\}$ die Ebenen in L. Dann ist $B_{\max(R)}$ ein RS-Baum mit den Knoten $\{1, 2 \ldots, m\}$ und für jeden $v \in B$ gilt: $key(v) = \max_{r(v)}$.*

In $B_{\max(R)}$ gilt wegen Definition 4.3 für alle $i \in B_{\max(R)}$, dass alle Knoten im linken Teilbaum von i zu niedrigeren Ebenen gehören als alle Knoten im rechten Teilbaum von i. Dies ist kein Widerspruch zur Ordnung des Baumes, da jede Ebene durch genau einen Knoten repräsentiert wird. Eine *Bereichsabfrage* in einem Baum ist eine Suche nach allen Teilbäumen die Knoten enthalten, deren Schlüssel innerhalb eines *Suchintervalls* $[x_{\min}, x_{\max}]$ mit $x \in \mathbb{R}$ liegen.

Definition 4.7. *Sei* $[x_{\min}, x_{\max}]$ *mit* $x \in \mathbb{R}$ *ein* Suchintervall *und* B *ein RS-Baum. Eine* Bereichsabfrage *an* B *sucht nach zwei Wurzel-Blatt-Pfaden* $p = (v_1, v_2, \dots v_k)$ *und* $p' = (v'_1, v'_2, \dots v'_k)$ *mit* $v_1 = v'_1 = root(B)$ *und* v_k, v'_k *sind Blätter. Für zwei Knoten* $v_i, v_{i+1} \in p$ *gilt:* $v_{i+1} = left(v_i)$*, wenn* $x_{\min} \leq key(v_i)$ *und* $v_{i+1} = right(v_i)$*, sonst. Für zwei Knoten* $v'_i, v'_{i+1} \in p'$ *gilt analog:* $v'_{i+1} = right(v'_i)$*, wenn* $x_{\max} \geq key(v'_i)$ *und* $v'_{i+1} = left(v'_i)$*, sonst.*

Lemma 4.8. *Seien* p, p' *Wurzel-Blatt-Pfade gemäß Definition 4.7 in* B *und es gelte* $v_i = v'_i$ *mit* $v_i \in p, v'_i \in p'$*. Dann ist* $v'_{i+1} = v_{i+1}$ *mit* $v'_{i+1} \in p, v_{i+1} \in p'$*, wenn* $key(v_i) \notin [x_{\min}, x_{\max}]$ *gilt.*

Beweis. Wir zeigen die Behauptung durch Induktion. Zu Beginn einer Suchanfrage gilt $v_1 = v'_1 = root(B)$. Dies entspricht genau der Definition und erfüllt die Behauptung. Sei nun $p = (v_1, \dots v_l, v_{l+1}, \dots, v_k)$ und $p' = (v_1, \dots v_l, v'_{l+1}, \dots, v'_k)$. Dann sind die Suchpfade p, p' identisch auf den ersten l Knoten in B. Nun ist noch zu zeigen, dass aus $key(v_l) \notin [x_{\min}, x_{\max}]$ stets folgt: $v_{l+1} = v'_{l+1}$. Entweder ist $key(v_l) < x_{\min} < x_{\max}$. Dann gilt $v_{l+1} = right(v_l) = v'_{l+1}$. Oder es ist $x_{\min} < x_{\max} < key(v_l)$. Dann gilt $v_{l+1} = left(v_l) = v'_{l+1}$ und die Behauptung ist bewiesen. □

Wegen Lemma 4.8 sind die zwei Pfade p, p' offenbar identisch, bis zu einem höchsten Knoten $v_i \in p \cap p'$ mit $x_{\min} \leq key(v_i) \leq x_{\max}$. Wir nennen diesen Knoten v_{split}. Die beiden Suchpfade der Bereichsabfrage unterscheiden sich demnach erst unterhalb von v_{split}. Wir schreiben $v_{\min}$ für $v_k \in p$ und $v_{\max}$ für $v'_k \in p'$. Die Pfade $p_{\min} = (v_{\text{split}}, \dots v_{\min}) \subseteq p$ und $p_{\max} = (v_{\text{split}}, \dots v_{\max}) \subseteq p'$ heißen fortan *linker* respektive *rechter Suchpfad* einer Bereichsabfrage. Wegen Lemma 4.5 können wir effizient in B nach v_{split}, $v_{\max}$ und $v_{\min}$ suchen. Wir behaupten, dass der Rechenaufwand für eine Bereichsabfrage beschränkt ist durch $2 \cdot h(B)$ und somit in $\mathcal{O}(\log n)$ liegt.

Lemma 4.9. *Eine Bereichsabfrage kann in* $\mathcal{O}(2 \cdot h(B)) \in \mathcal{O}(\log n)$ *beantwortet werden, wenn* B *ein RS-Baum ist.*

Beweis. Offenbar ist die Länge von $p_{\min}$ respektive $p_{\max}$ gleich der Höhe des Baumes, wenn $v_{\text{split}} = root(B)$ gilt. Dann ist unsere Behauptung erfüllt. Falls v_{split} verschieden der Wurzel des Baumes ist, so muss offenbar zuerst v_{split} gefunden werden. Entsprechend setzt sich die Bereichsanfrage dann aus drei Teilen zusammen: Finde v_{split}, finde $v_{\min}$ und finde $v_{\max}$. Der Aufwand, v_{split} zu finden ist offenbar beschränkt durch $h(B)$. Die Länge der Pfade $p_{\min}$ und $p_{\max}$ ist dann beschränkt durch $h(B)$ minus die Länge des Suchpfades $p_{\text{split}} = (root(B), \dots, v_{\text{split}})$. Sei $|p|$ die Länge eines Suchpfades in B. Dann gilt: $|p_{\min}| + |p_{\max}| + |p_{\text{split}}| \leq 2 \cdot (h(B) - |p_{\text{split}}|) + |p_{\text{split}}| \leq 2 \cdot h(B)$ und die Behauptung ist bewiesen. □

Wir wollen nun noch zeigen, dass alle Knoten, die rechts von $p_{\min}$ sowie links von $p_{\max}$ in B liegen, Schlüssel innerhalb der Bereichsabfrage besitzen. Offenbar ist v_{split} die Wurzel eines Teilbaumes von B und per Definition 4.7 gilt: $x_{\min} \leq \mathit{key}(v_{\text{split}}) \leq x_{\max}$. Wir sagen, ein Teilbaum B' liegt *innerhalb* einer Bereichsabfrage, wenn für $\mathit{root}(B')$ entweder gilt: $\mathit{root}(B') = \mathit{right}(v_i)$ mit $v_i \in p_{\min}, v_i \neq v_{\text{split}}$ und $\mathit{root}(B') \notin p_{\min}$. Oder wenn $\mathit{root}(B') = \mathit{left}(v_i)$ mit $v_i \in p_{\max}, v_i \neq v_{\text{split}}$ und $\mathit{root}(B') \notin p_{\max}$.

Beobachtung 4.10. *Alle Knoten $v \in B$, die innerhalb einer Bereichsabfrage liegen, gilt: $x_{\min} \leq \mathit{key}(v) \leq x_{\max}$.*

Beweis. Seien $p_{\min}$ und $p_{\max}$ Suchpfade einer Bereichsabfrage. Sei $\mathit{root}(B')$ ein Knoten innerhalb der Bereichsabfrage. Ohne Beschränkung der Gültigkeit sei $\mathit{root}(B') = \mathit{right}(v_i)$ mit $v_i \in p_{\min}$ und $v_i \neq v_{\text{split}}$. Denn dann gilt aufgrund der Ordnung in B für alle Knoten $v \in B'$: $x_{\min} \leq \mathit{key}(v)$. Da außerdem $v_i \in p_{\min}, v_i \neq v_{\text{split}}$ gilt, ist $\mathit{key}(v) \leq x_{\max}$. Der Fall $\mathit{root}(B') = \mathit{left}(v_i)$ mit $v_i \in p_{\max}$ und $v_i \neq v_{\text{split}}$ erfolgt analog. □

Wenn wir für ein Intervall $e = [l, r]$ die Aufliegerebene $\varphi(a(e))$ finden wollen, dann entspricht dies einer Bereichsabfrage mit $x_{\min} = l$ und $x_{\max} = r$. Wir können so eine Bereichsabfrage in $\mathcal{O}(\log n)$ beantworten und wegen Beobachtung 4.10 gilt dann für alle $v \in B_{\max(R)}$, die innerhalb der Bereichsabfrage liegen: $x_{\min} < r(e) < x_{\max}$. Dann gilt für jede Ebene, deren korrespondierender Knoten in der Bereichsabfrage liegt, dass dort ein zu e gerichtetes Intervall liegt. Wir können also effizient mit einer Bereichsabfrage an $B_{\max(R)}$ die Ebenen in L ermitteln, die für $\varphi(a(e))$ infrage kommen. Um nun die höchste dieser Ebenen zu finden, wollen wir in jedem Knoten zusätzliche Information speichern. Im folgenden Abschnitt zeigen wir, wie man innerhalb einer Bereichsabfrage effizient lokale Minima bzw. Maxima ermitteln kann.

4.1.2 Augmentierter Suchbaum

Wir suchen nach einer Datenstruktur, die uns gestattet, in einer Teillösung L für ein Intervall $e \notin L$ die Aufliegerebene zu finden (siehe Definition 3.4). Bisher haben wir gesehen, dass ein selbstbalancierender binärer Suchbaum uns das dynamische Ändern von Mengen und die Suche nach einem bestimmten Schlüssel in $\mathcal{O}(\log n)$ erlaubt. Mit einer Bereichsabfrage (Definition 4.7) können wir effizient alle Knoten in B entdecken, deren Schlüssel innerhalb eines Suchintervalls liegen. Man kann einen Baum so anpassen, dass mit einer Bereichsabfrage ein lokales Minimum oder Maximum ermittelt werden kann. Wir beschränken uns auf Maxima. Für Minima ist das Vorgehen analog. Eine *Augmentierung* eines Baumes B ist eine Zusatzinformation, die neben dem Schlüssel in jedem Knoten gespeichert wird. Die Ordnung des Baumes gemäß Definition 4.3 bleibt davon unberührt. Die Augmentierung kann uns jedoch dabei helfen, innerhalb eines Teilbaumes ein Maximum zu finden, ohne alle Knoten im Teilbaum zu betrachten.

Definition 4.11. *Sei B ein Baum. Dann ist B* augmentiert*, wenn jeder $v \in B$ ein* Augment *$\mu \in \mathbb{R}$ besitzt, wobei $\mu(\mathit{nil}) = -\infty$ sei. Zusätzlich sei in jedem Knoten ein Maximum über alle μ im darunter liegenden Teilbaum $\mu_{\max}(v)$ gespeichert, das wie folgt*

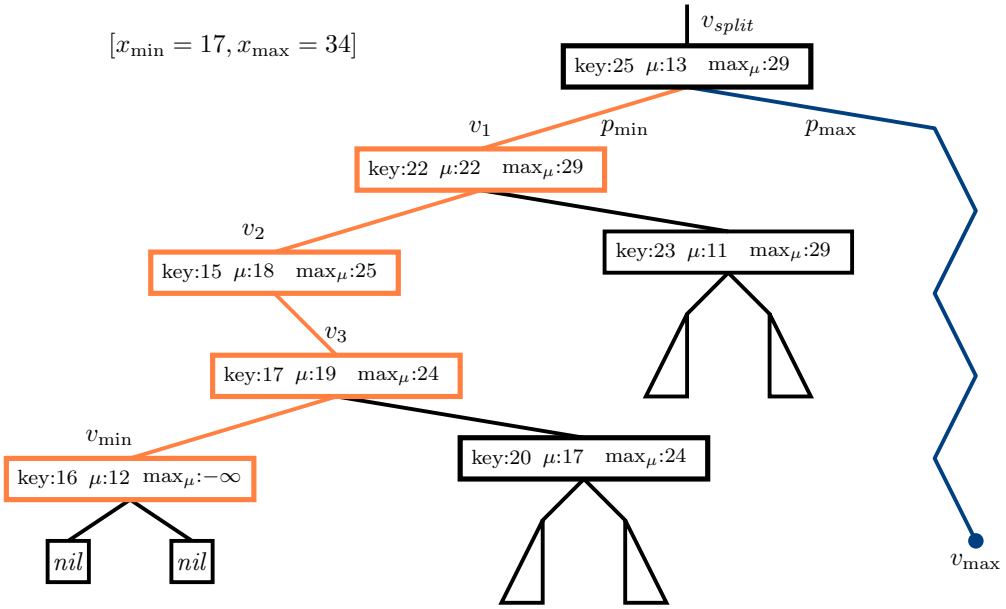


Abb. 4.3: Bereichsabfrage in einem augmentierten binären Suchbaum.

rekursiv definiert ist:

$$\mu_{\max}(v) = \begin{cases} -\infty, & \textit{falls } v = \textit{nil} \\ \max\{\mu(\textit{left}(v)), \mu_{\max}(\textit{left}(v)), \mu(\textit{right}(v)), \mu_{\max}(\textit{right}(v))\}, & \textit{sonst} \end{cases}$$

Beachte, dass dieses Augment auch *key* selbst sein kann. Dann ist $\mu_{\max}(v)$ das Maximum über alle Schlüssel im Teilbaum, dessen Wurzel v ist. Wir sind an dem Fall interessiert, dass $\mu(v) \neq \textit{key}(v)$ gilt. In $B_{\max(R)}$ wird in jedem Knoten die Ebene, die er repräsentiert, als $\mu(v)$ gespeichert. Entsprechend ist das Augment $\mu_{\max}(v)$ die höchste Ebene, die durch einen Knoten im darunter liegenden Teilbaum repräsentiert wird.

Lemma 4.12. *Sei B ein augmentierter Baum und $p_{\min}$, $p_{\max}$ Suchpfade einer Bereichsabfrage. Wir können mit höchstens $\mathcal{O}(h(B)) \in \mathcal{O}(\log n)$ Vergleichen das Maximum über alle $\mu(v), v \in B$ mit $x_{\min} \leq \textit{key}(v) \leq x_{\max}$ finden.*

Beweis. Sei $\max_{\text{temp}}$ eine Variable, die das Maximum von μ über alle bisher betrachteten Teilbäume von B speichert, und zu Beginn sei $\max_{\text{temp}} = 0$. Wir schreiten $p_{\min}$ ab. Für $p_{\max}$ wird symmetrisch verfahren. Gemäß Definition 4.7 kann für einen Knoten $v \in p_{\min}$ gelten: $\textit{key}(v) < x_{\min}$. Dann bleibt $\max_{\text{temp}}$ unverändert. Gelangen wir an einen $v \in p_{\min}$ mit $\textit{key}(v) \geq x_{\min}$, dann gilt wegen Beobachtung 4.10: Wenn $\textit{left}(v) \in p_{\min}$ gilt, dann sind alle Knoten im Teilbaum, dessen Wurzel $\textit{right}(v)$ ist, innerhalb der Bereichsabfrage. Dann ist $\max'_{\text{temp}} = \max\{\max_{\text{temp}}, \mu_{\max}(\textit{right}(v)), \mu(\textit{right}(v))\}$ das aktualisierte Maximum über alle bisher betrachteten Teilbäume innerhalb der Bereichsabfrage. Ist $\textit{right}(v)$ ein Blatt, so gilt wegen $\mu(\textit{nil}) = -\infty$: $\max'_{\text{temp}} = \max\{\max_{\text{temp}}, -\infty\} = \max_{\text{temp}}$. Somit ist es in $\mathcal{O}(1)$ Vergleichen möglich, für einen Teilbaum, der innerhalb der Bereichsabfrage

liegt, das Maximum über μ aufrecht zu erhalten. Wenn stattdessen $\mathit{left}(v) \notin p_{\min}$ gilt, so ist offenbar entweder v ein Blatt. Diesen Fall haben wir bereits abgehandelt. Oder es ist $\mathit{right}(v) \in p_{\min}$. In diesem Fall verfahren wir analog dazu, wie oben beschrieben. Symmetrisch gehen wir in $p_{\max}$ vor, was ebenso für jeden Knoten entlang des Suchpfades und jeden Teilbaum innerhalb der Bereichsabfrage in $\mathcal{O}(1)$ möglich ist. Da wir mit Lemma 4.9 wissen, dass die Pfade in ihrer Länge durch $\mathcal{O}(h(B))$ beschränkt sind, und wir auf einem balancierten Suchbaum arbeiten, liegt der Rechenaufwand für ein lokales Maximum über μ innerhalb einer Bereichsabfrage in $\mathcal{O}(\log n)$. □

Wenn wir $B_{\max(R)}$ augmentieren, in dem wir für jeden $v \in B_{\max(R)}$ die Ebene als $\mu(v)$ speichern, die der Knoten repräsentiert, dann können wir eine Bereichsabfrage innerhalb der Ebenen, auf denen sich zu einem Intervall $e \notin L$ gerichtete Intervalle befinden, durchführen. Mit dem Augment lässt sich dann effizient die höchste dieser Ebenen bestimmen.

Satz 4.13. *[CLRS09, Satz 14.1] Sei B ein augmentierter RS-Baum mit n Knoten und $v \in B$. Wenn $\mu_{\max}(v)$ aus Informationen, die in $\mathit{left}(v)$ und $\mathit{right}(v)$ gespeichert sind, neu berechnet werden kann, dann implementiert B die Operationen Suchen, Einfügen und Löschen in $\mathcal{O}(\log n)$.*

Mit einem entsprechend Definition 4.14 augmentierten RS-Baum können wir nun effizient $\varphi(a(e))$ berechnen. Die Abbildung 4.3 zeigt ein konkretes Beispiel einer Bereichsabfrage. Hier ist $p_{\max}$ (in Blau) nur angedeutet. Beachte auch, dass hier das Augment ungleich dem Schlüssel ist. Gemäß Beobachtung 4.8 ist v_{split} der höchste v in $p_{\min}$ bzw. $p_{\max}$ mit $\mathit{key}(v) \in [17, 34]$. Sei $\max_{\text{temp}} = 0$ das vorläufige Maximum über alle $\mu(v)$ innerhalb der Bereichsabfrage. Da $\mathit{key}(v_{\mathit{split}})$ innerhalb des Suchintervalls liegt, wird $\max_{\text{temp}}$ auf 13 gesetzt. Das Maximum, welches in v_{split} gespeichert ist, kann von Knoten stammen, die außerhalb der Bereichsabfrage stammen. Daher wird hier nur zwischen dem bisherigen Maximum und $\mu(v_{\text{split}})$ das Größere gewählt. In v_1 liegt der Fall vor, dass $\mathit{left}(v_1) \in p_{\min}$ gilt. Demnach sind alle Knoten in dem Teilbaum, dessen Wurzel $\mathit{right}(v_1)$ ist, innerhalb der Bereichsabfrage. Da auch $\mathit{key}(v_1)$ im Suchintervall liegt, wird nun zwischen dem bisher geltenden Maximum, $\mu(v_1)$, $\mu(\mathit{right}(v_1))$ und $\max_\mu(\mathit{right}(v_1))$, das Größere gewählt. Entsprechend wird $\max_{\text{temp}}$ auf 29 gesetzt. Weil $\mathit{key}(v_2) < x_{\min}$ gilt, bleibt das bisherige Maximum unverändert. In v_3 gilt wie zuvor: $\mathit{left}(v_3) \in p_{\min}$. Daher sind alle Knoten im Teilbaum, dessen Wurzel $\mathit{right}(v_3)$ ist, innerhalb der Bereichsabfrage. Zwischen $\max_{\text{temp}}, \mu(v_3), \mu(\mathit{right}(v_3))$ und $\max_\mu(\mathit{right}(v_3))$ wird das Größere gewählt und $\max_{\text{temp}}$ bleibt unverändert. Da $v_{\min}$ ein Blatt ist, ist das darin gespeicherte Maximum $-\infty$. Weil $\mathit{key}(v_{\min}) > x_{\min}$ ist, wird hier zwischen dem aktuellen Maximum und $\mu(v_{\min})$ das Größere gewählt und die Suche in $p_{\min}$ terminiert mit $\max_{\text{temp}} = 29$.

Man verfährt in $p_{\max}$ symmetrisch, indem man für alle Teilbäume, deren Wurzel linkes Kind eines $v \in p_{\max}$ sind, und die nicht selber in $p_{\max}$ liegen, das darin gespeicherte Augment, das Maximum über die darunter liegenden Augmente sowie das aktuelle Maximum über alle bereits betrachteten Teilbäume innerhalb der Suchanfrage und Knoten auf den Suchpfaden, aufrechterhält. Wenn wir $B_{\max(R)}$ (siehe Definition 4.6) anpassen, sodass jeder Knoten die höchste Ebene eines Knoten im darunter liegenden Teilbaum enthält, dann können wir wegen Lemmata 4.4 und 4.9 in $\mathcal{O}(\log n)$ die Aufliegerebene

eines Intervalls $e \notin L$ finden, wenn $B_{\max(R)}$ ein balancierter Baum ist. Denn diese ist, gemäß Definition 3.4 die höchste unter den Ebenen, auf denen $l(e) < \max_{r(k)} < r(e)$ gilt, mit $k \in B_{\max(R)}$.

Definition 4.14. *Sei $B_{\max(R)}$ ein Baum gemäß Definition 4.6. Dann ist $B_{\max(R)}$ augmentiert, wenn jeder $v \in B_{\max(R)}$ ein $\mu(v)$ und ein $\max_\mu(v)$ besitzt, sodass $\mu(v) = v$ gilt und $\max_\mu(v)$ ist rekursiv definiert gemäß Definition 4.11.*

Wenn wir ein Intervall e in einer Lösung L mit m Ebenen auf der tiefsten, zulässigen Ebene k platzieren, dann ändert sich $\max_{r(k)}$, wenn $k < m+1$ gilt. Denn dann wird e auf einer Ebene platziert, auf der bereits ein Intervall lag. Entsprechend ist dann $r(e)$ der rechteste Endpunkt auf Ebene k. In diesem Fall wird der Koten k zunächst aus $B_{\max(R)}$ gelöscht. Anschließend wird ein neuer Knoten k mit $\mu(k) = k$ und $key(k) = \max_{r(k)} = r(e)$ eingefügt. Wir müssen nun noch zeigen, dass wir das Augment bei Einfüge- und Löschoperationen in $\mathcal{O}(\log n)$ aufrecht erhalten können.

Lemma 4.15. *Sei B ein balancierter Suchbaum gemäß Definition 4.14. Wir können einen Knoten aus B löschen oder einfügen und in $\mathcal{O}(\log n)$ Rechenschritten das Augment aufrechterhalten.*

Beweis. Folgt aus Satz 4.13 und Definition 4.14. □

Lemma 4.16. *Sei $B_{\max(R)}$ ein augmentierter, balancierter, binärer Suchbaum gemäß Definition 4.14, L eine Teillösung und $e \notin L$ ein Intervall. Wir in können $\mathcal{O}(\log n)$ die Aufliegerebene $\varphi(a(e))$ berechnen.*

Beweis. Folgt aus Lemmata 4.12 und 4.15. □

Wir konnten zeigen, wie mit geeigneten Datenstrukturen die Aufliegerebene effizient gefunden werden kann. Nun wollen wir noch zeigen, dass unter den Ebenen, die oberhalb der Aufliegerebene liegen, die niedrigste freie Ebene ebenfalls effizient gefunden werden kann. Dazu verwenden wir erneut einen augmentierten RS-Baum und eine angepasste Bereichsabfrage.

4.2 Niedrigste zulässige Ebene

Wir wollen Layer-Assignment schneller lösen und dazu die Laufzeit von Algorithmus 1 verbessern. Wir haben gemäß Definition 4.1 den Rechenaufwand, um ein Intervall $e \notin L$ auf die niedrigste, zulässige Ebene in einer Teillösung L zu platzieren, in zwei Anfragen aufgeteilt: *Finde die Aufliegerebene $\varphi(a(e))$* und *Finde die niedrigste freie Ebene, die höher als $\varphi(a(e))$ ist.* Diese ist gemäß Beobachtung 3.5 die niedrigste zulässige Ebene für e. Mit einem augmentierten, binären Suchbaum kann die erste dieser Anfragen in $\mathcal{O}(\log n)$ beantwortet werden, wenn ein selbstbalancierender binärer Suchbaum verwendet wird (siehe Lemma 4.16). Nun wollen wir einen Baum definieren, der uns auch die zweite Anfrage effizient beantworten kann. Dieser sei ein augmentierter RS-Baum, der gemäß Lemma 4.5 alle grundlegenden Mengenoperationen effizient implementiert. Sei

L eine Teillösung mit m Ebenen. Dann existiert für jede Ebene in L ein Knoten im Baum, dessen Schlüssel die Ebene ist, die er repräsentiert. Weiter sei in jedem Knoten $\max_{r(k)}$ gespeichert sowie das Minimum über alle $\max_{r(k)}$ im darunter liegenden Baum. Wir nennen diesen Baum $B_{\min(\varphi)}$.

Definition 4.17. *Sei L eine Teillösung und $\{1, 2, \ldots, m\}$ die Ebenen in L. Dann ist $B_{\min(\varphi)}$ ein augmentierter, binärer Suchbaum mit den Knoten $\{1, 2, \ldots, m\}$ und für jeden $v \in \{1, 2, \ldots, m\}$ gilt $key(v) = v$ und $\mu(v) = \max_{r(v)}$. Dabei gelte $\mu(nil) = \infty$. Weiter sei in jedem Knoten v ein Minimum $\mu_{\min}(v)$ über alle $\mu(v')$ in dem Teilbaum, dessen Wurzel v ist, gespeichert. Dieses ist wie folgt rekursiv definiert:*

$$\mu_{\min}(v) = \begin{cases} \infty, & \text{falls } v = nil \\ \min\{\mu(left(v)), \mu_{\min}(left(v)), \mu(right(v)), \mu_{\min}(right(v))\}, & \text{sonst} \end{cases}$$

Sei $e \notin L$ ein Intervall, das in L platziert werden soll. Sei $v \in B_{\min(\varphi)}$, sodass $\varphi(a(e)) < key(v)$ und $\mu(v) < l(e)$ gilt. Dann ist v eine freie Ebene, da $\mu(v) = \max_{r(v)}$ ist, und eine zulässige Ebene, da auch noch $key(v) < \varphi(a(e))$ gilt. Wenn $\varphi(a(e))$ die Aufliegerebene ist, dann lautet eine Bereichsabfrage an $B_{\min(\varphi)}$, mit der man die niedrigste, freie Ebene oberhalb von $\varphi(a(e))$ finden kann: $[\varphi(a(e)), \infty]$. Dann gibt es nur einen Suchpfad $p_{\min}$, sodass für alle Knoten $v \in B_{\min(\varphi)}$ im rechten Teilbaum von v_{split} sowie alle Knoten innerhalb von $p_{\min}$ gilt: $\varphi(a(e)) < key(v)$. Anmerkung: Jeder Knoten in $B_{\min(\varphi)}$ repräsentiert exakt eine Ebene. Daher gilt auch die Ungleichheit anstelle von $\varphi(a(e)) \leq key(v)$.

Sei $v \in p_{\min}$ mit $right(v) \notin p_{\min}$ und $\mu_{\min}(right(v)) < l(e)$. Dann existiert im Teilbaum, dessen Wurzel $right(v)$ ist, eine Ebene, die frei und zulässig ist. Wir entdecken alle Teilbäume, für die diese Eigenschaft gilt in $\mathcal{O}(h(B_{\min(\varphi)}))$. Wegen der Ordnung von $B_{\min(\varphi)}$ gilt für zwei Teilbäume, die beide eine freie Ebene enthalten, und die beide innerhalb der Bereichsabfrage liegen: Alle Knoten in dem Teilbaum, der weiter links liegt, gehören zu niedrigeren Ebenen als alle Knoten im Teilbaum, der weiter rechts liegt. Wir suchen dann nur in dem Teilbaum nach der niedrigsten zulässigen Ebene, der am weitesten links liegt unter den Teilbäumen, die eine freie Ebene enthalten und innerhalb der Bereichsabfrage liegen. Innerhalb dieses Teilbaumes können wir anhand von $\mu_{\min}$ entscheiden, wo wir weiter suchen. Schreiten wir $p_{\min}$ ab, aktualisieren wir das Minimum an jedem $v \in p_{\min}$, wenn $\varphi(a(e)) < key(v)$ und $\mu(v) < l(e)$ gilt, und markieren alle Teilbäume innerhalb von $p_{\min}$, wenn ein solcher Teilbaum eine freie Ebene enthält. Wir schlagen daher folgenden Algorithmus vor, um in $B_{\min(\varphi)}$ die niedrigste zulässige Ebene zu finden:

Teilbaum auswählen Zunächst wählen wir den Teilbaum, der durchsucht wird. Dazu sei L eine Teillösung und $B_{\min(\varphi)}$ ein augmentierter RS-Baum gemäß 4.17. Weiter sei $e \notin L$ ein Intervall und $v, v' \in B_{\min(\varphi)}$ zwei Wurzeln von Teilbäumen, die innerhalb der Suchanfrage liegen, sodass $\mu_{\min}(v) < l(e)$ und $\mu_{\min}(v') < l(e)$ gilt. Und es sei v Knoten des linken Teilbaumes von $parent(v')$. Dann enthalten sowohl B' mit $root(B') = v$ als auch B'' mit $root(B'') = v'$ je eine freie Ebene, die oberhalb von $\varphi(a(e))$ liegt. Wegen

der Ordnung von $B_{\min(\varphi)}$ sind jedoch die Schlüssel für alle Knoten in B' höchstens so groß wie alle Schlüssel in B''. Folglich wählen wir, um eine minimale freie Ebene unter den Ebenen zu finden, die oberhalb von $\varphi(a(e))$ liegen, stets v aus. Da wir entlang von $p_{\min}$ maximal $(\log n)$ solche Teilbäume betrachten, können wir den Teilbaum, der die niedrigste zulässige Ebene enthält in $\mathcal{O}(\log n)$ Zeit finden.

Teilbaum durchsuchen Sei nun B' der Teilbaum, dessen Wurzel v ist und den wir im vorigen Schritt gewählt haben. Wenn nun $\mu_{\min}(\mathit{left}(v)) < l(e)$ gilt, so enthält der linke Teilbaum von v eine freie Ebene und wir suchen rekursiv im linken Teilbaum von v weiter. Andernfalls, und es gilt $\mu_{\min}(\mathit{right}(v)) < l(e)$, suchen wir rekursiv im rechten Teilbaum von v weiter. Gilt stattdessen $\mu(v) > l(e)$, so muss $\mu_{\min}(\mathit{right}(v)) < l(e)$ sein. Sonst würde B' keine freie Ebene enthalten. Dann ist v die niedrigste freie Ebene. Es ist leicht zu sehen, dass auch diese Suche durch $\mathcal{O}(h(B_{\min(\varphi)}))$ beschränkt ist.

Die Bäume $B_{\min(\varphi)}$ und $B_{\max(R)}$ enthalten je maximal n Knoten, wenn die Eingabe aus n Intervallen besteht. In jedem Knoten werden *key*, μ und $\mu_{\max}$ bzw. $\mu_{\min}$, sowie ein Zeiger auf *parent*, *left*, *right* gespeichert. Während der Anfragen an $B_{\min(\varphi)}$ wird nur ein vorläufiges Maximum über die Ebenen gespeichert. Für die Anfrage an $B_{\max(R)}$ wird neben dem vorläufigen Minimum über die Ebenen oberhalb von $\varphi(a(e))$ noch ein Zeiger auf den linkesten Teilbaum, der eine freie Ebene enthält, gespeichert. Zusammen liegt der Speicherbedarf daher in $\mathcal{O}(n)$. Da wir für jedes $e \in I \setminus L$ die niedrigste zulässige Ebene in $\mathcal{O}(\log n)$ Zeit finden können, ist die Laufzeit von Greedy-Layer-Assignment in $\mathcal{O}(n \log n)$, wenn n die Anzahl an Intervallen in der Eingabe ist.

Satz 4.18. *Greedy-Layer-Assignment kann in $\mathcal{O}(n \log n)$ Zeit und $\mathcal{O}(n)$ Platz gelöst werden, wenn $n = |I|$ die Größe der Eingabe ist.*

Nun wollen wir noch zeigen, dass unsere Laufzeitschranke optimal ist. Dazu bedienen wir uns einem Resultat, wonach das Sortieren von n natürlichen Zahlen durch einen vergleichsbasierten Algorithmus bis zu $\Omega(n \log n)$ viele Vergleiche benötigt.

Satz 4.19. *[CLRS09, Satz 8.1] In jedem vergleichsbasierten Sortieralgorithmus sind bis zu $\Omega(n \log n)$ viele Vergleiche nötig, um n natürliche Zahlen zu sortieren.*

Um zu zeigen, dass die Laufzeit von Greedy-Layer-Assignment optimal ist, nehmen wir an, dass ein Algorithmus existiert, der das Layer-Assignment-Problem in Laufzeit $\mathcal{o}(n \log n)$ löst. Dann zeigen wir, dass man mit so einem Algorithmus n natürliche Zahlen vergleichsbasiert sortieren kann. Dies steht im Widerspruch zu Satz 4.19. Folglich muss $\mathcal{O}(n \log n)$ eine scharfe untere Schranke zur Lösung von Layer-Assignment sein.

Satz 4.20. *Die Laufzeitschranke von $\mathcal{O}(n \log n)$ für Greedy-Layer-Assignment ist optimal.*

Beweis. Sei X ein Algorithmus, sodass $X(I)$ eine gültige Lösung für Layer-Assignment ist, wenn I eine Menge von Intervallen ist. Weiter sei die Laufzeit zur Berechnung von

$X(I)$ in $\mathcal{o}(n \log n)$, wenn $n = |I|$ gilt. Angenommen, wir wollten eine Menge A von n natürlichen Zahlen aufsteigend sortieren. Sei $\Delta_{\max} = \max\{a \in A\} - \min\{a \in A\}$. Wir können $\Delta_{\max}$ in $\mathcal{O}(n) \in \mathcal{o}(n \log n)$ berechnen. Nun erzeugen wir für jedes $a \in A$ ein Intervall $e = [a, a + \Delta_{\max}]$. Sei I die so erzeugte Instanz. Die resultierende Menge von Intervallen ist offensichtlich ein gerichteter Stapel gemäß Definition 2.11. Nun lassen wir $X(I)$ für die so erzeugte Instanz berechnen. Wir können nun die n Ebenen der Lösung iterativ durchlaufen, und für jedes $e, e' \in X(I)$ gilt: $l(e) < l(e') \Rightarrow \varphi(e) < \varphi(e')$. Da $l(e) = a$ gilt, ist die Lösung eine aufsteigende Sortierung von A, die in $\mathcal{o}(n \log n)$ berechnet wurde. Dies steht im Widerspruch zu Satz 4.19. □

Wir haben mit augmentierten RS-Bäumen die Laufzeit von Algorithmus 1 von $\mathcal{O}(n^2)$ auf $\mathcal{O}(n \log n)$ verbessert und konnten zeigen, dass diese Laufzeit optimal ist. Im kommenden Kapitel werden wir zeigen, dass Algorithmus 1 stets eine Lösung mit optimaler Höhe berechnet.

5 Layer-Assignment als Färbeproblem

Bisher haben wir Mengen von Intervallen betrachtet, von denen jedes das horizontale Geraden-Segment einer elektrischen Leitung darstellt. Diesen Intervallen wurden natürliche Zahlen, genannt Ebenen zugeordnet, sodass die Leitungen kompakt und ohne vermeidbare Überkreuzungen gezeichnet werden konnten. Bei einer Instanz dieses Problems, das wir Layer-Assignment-Problem nannten (siehe Kapitel 2), existiert eine Äquivalenz zum Färben eines gemischten Intervallgraphen. Betrachten wir noch einmal, was eine Färbung eines Graphen ist: Laut Definition 2.23 gilt in einer Färbung, dass zwei adjazenten Knoten u, v unterschiedliche Farben zugeordnet sind. Die Farben waren natürliche Zahlen, ebenso wie die Ebenen, welche den Intervallen zugewiesen wurden. Überlappen sich zwei Intervalle, so befindet sich im korrespondierenden Intervallgraphen eine Kante zwischen ihnen. Weiter galt, dass bei einer gerichteten Kante (u, v) für die zugewiesenen Farben gilt: $\text{col}(u) < \text{col}(v)$. Für zwei teilweise überlappende Intervalle galt analog, dass das weiter links befindliche auf einer niedrigeren Ebene liegen muss, wenn beide Intervalle rechtsgerichtet sind. Symmetrisch dazu muss das weiter rechts befindliche Intervall unten liegen, wenn beide Intervalle linksgerichtet sind. Wir führen zwei Graphen ein, die so konstruiert sind, dass sicherstellt wird, dass eine gültige Färbung jedes der beiden Graphen auch eine Lösung von Layer-Assignment ist. Diese Graphen heißen *Konfliktgraph* und *erweiterter Konfliktgraph* und gehen auf Brückner [Brü21] zurück.

5.1 Der Konfliktgraph

Bei einer Instanz von Layer-Assignment handelt es sich um eine Menge von Intervallen, zu denen auch ein Graph G existiert, der die Instanz abbildet. Wir wissen aus Definition 2.8, dass eine Lösung L den Intervallen Ebenen zuordnet, sodass zwei sich überlappende Intervalle auf unterschiedlichen Ebenen liegen. Darüber hinaus gibt bei Layer-Assignment jedoch eine weitere Bedingung, die wir abbilden müssen. Für zwei teilweise überlappenden Intervalle e, e' gilt, dass e auf einer höheren Ebene platziert sein muss als e', wenn e zu e' gerichtet ist. Diese Bedingung lässt sich darstellen, indem wir einen *Konfliktgraphen* K für I erstellen. Dieser ist ein gemischter Graph $K = (V, \vec{E}, E)$ mit $V = I$ und für jedes Paar $u, v \in I$, wenn u, v vollständig überlappen, existiert die ungerichtete Kante $\{u, v\} \in E$, und wenn u gerichtet zu v, existiert die gerichtete Kante $(u, v) \in \vec{E}$.

Definition 5.1. *Für eine Menge von Intervallen I ist $K = (V, \vec{E}, E)$ der* Konfliktgraph *mit der Knotenmenge $V = I$ und für jedes Paar an Intervallen $u, v \in I$, für das $u \cap v \neq \emptyset$*

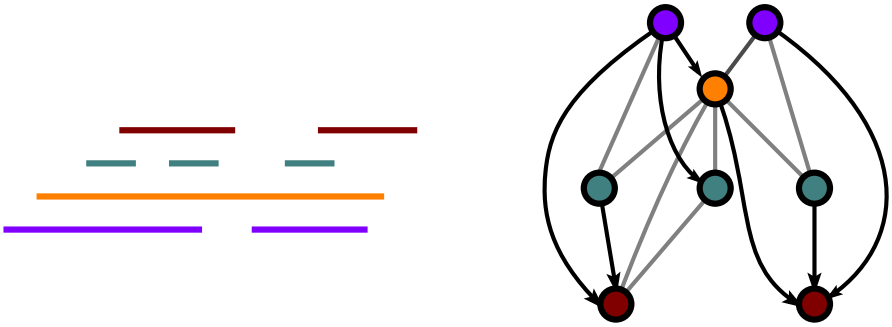


Abb. 5.1: Eine kleine Instanz mit gültiger Lösung auf vier Ebenen. Jede Ebene ist mit einer Farbe markiert. Rechts ist ein korrespondierender Graph zu sehen. Jedes Intervall entspricht einem Knoten im Graph. Die Intervalle auf derselben Ebene sind stets unabhängige Mengen im Graph. Ungerichtete Kanten sind in Grau, gerichtete in Schwarz dargestellt und mit Richtungspfeilen hervorgehoben.

gilt, ist die gerichtete Kante $(u, v) \in \vec{E}$*, wenn* $l(u) < l(v) \wedge r(u) < r(v)$ *gilt, und die ungerichtete Kante* $\{u, v\} \in E$*, wenn* $l(u) < l(v) \wedge r(u) > r(v)$ *gilt.*

Eine gültige Färbung von K weist nun die Farben so zu, dass für eine gerichtete Kante $(u, v) \in \vec{E}$ das Intervall u eine „größere“ Farbe als v erhält. Dies hat die Folge, dass für das Färben von K Satz 2.28 nicht mehr gilt. Es gibt Konfliktgraphen, in denen die benötigte Anzahl an Farben (und damit die Anzahl an Ebenen, die zum Zeichnen gebraucht werden) größer ist als die Cliquenzahl. Betrachten wir dazu noch einmal Abbildung 2.5(b): Die Intervalle e_1, e_3, e_5 sind disjunkt, ebenso e_2, e_4. Der zugehörige Intervallgraph hat Cliquenzahl $\omega(G) = 2$. Der Konfliktgraph besitzt jedoch einen gerichteten Pfad der Länge fünf, und dies ist hier auch die optimale Höhe. Zwei Intervalle können auch dann eine Reihenfolge in der Lösung haben, wenn sie nicht überlappen. Betrachten wir drei Intervalle a, b, c, wobei die Paare a, b und b, c teilweise überlappen, a und c jedoch disjunkt sind. Aus der teilweisen Überlappung wissen wir, dass b über a liegen muss und c über b. Daraus folgt aber, dass c über a liegen muss. Die Beziehung „a muss über b liegen“ ist also *transitiv*. Um dies auszudrücken, führen wir die *transitiven Kanten* ein und gelangen so zum *erweiterten Konfliktgraphen* K^+.

Definition 5.2. *Für eine Menge von Intervallen* I *ist* $K^+ = (V, \vec{E}, E)$ *der* erweiterter Konfliktgraph *mit der Knotenmenge* $V = I$ *und für alle* $u, v \in I$ *existiert die ungerichtete Kante* $\{u, v\} \in E$*, wenn* u *und* v *vollständig überlappen und die gerichtete Kante* $(u, v) \in \vec{E}$*, wenn* u *und* v *eine Reihenfolge laut Definition 2.10 haben.*

Mit dem erweiterten Konfliktgraphen haben wir jetzt die Möglichkeit, die Äquivalenz zwischen dem Färben des erweiterten Konfliktgraphen und einer Lösung L von Layer-Assignment zu zeigen.

Satz 5.3. *L ist eine Lösung für* Layer-Assignment $\iff$ *L ist eine gültige Färbung von* K^+*.*

Beweis. Wir zeigen zunächst, dass jede gültige Zuordnung der Intervalle zu Ebenen auch eine gültige Färbung des erweiterten Konfliktgraphen ist.
„⇒“
Eine Lösung von Layer-Assignment erfüllt die folgenden Eigenschaften:

1. Die „Ebenen“ sind natürliche Zahlen
2. Ist $u \cap v \neq \emptyset$, so gilt: $\varphi(u) \neq \varphi(v)$
3. Ist u gerichtet zu v, so gilt: $\varphi(u) < \varphi(v)$

Dies sind genau die Eigenschaften einer gültigen Färbung. Nun zeigen wir noch, dass jede gültige Färbung von K^+ auch eine Lösung für Layer-Assignment ist.
„⇐“
Umgekehrt erfüllt eine Färbung von K^+ folgende Eigenschaften:

1. Die „Farben“ sind natürliche Zahlen
2. Existiert die ungerichtet Kante $\{u, v\}$, so gilt: $\text{col}(u) \neq \text{col}(v)$
3. Existiert die gerichtete Kante (u, v), so gilt: $\text{col}(u) < \text{col}(v)$

Damit ist klar: Zwei Intervalle mit nicht-leerer Schnittmenge sind niemals in der selben Farbe eingefärbt. Liegt außerdem eine teilweise Überlappung vor, ist das rechte der beiden Intervalle „größer“ gefärbt. □

Hat man nun eine Färbung des erweiterten Konfliktgraphen, so ist diese äquivalent zu einer korrekten Zuordnung der Intervalle zu den Ebenen. Wir wollen im nächsten Abschnitt mithilfe des erweiterten Konfliktgraphen zeigen, dass der Algorithmus aus Kapitel 3 optimal ist.

5.2 Optimalität des Greedy-Verfahrens

Wir behaupten, dass der Algorithmus stets eine Lösung L mit der optimalen Höhe generiert. Das heißt, dass L unter allen gültigen Lösungen minimale Höhe besitzt. Sei I eine beliebige Eingabe und L eine Lösung mit Höhe m, die von Greedy-Layer-Assignment berechnet wurde. Wir konstruieren nun eine Teillösung genannt *Treppe* T von Intervallen aus L, sodass T aus exakt m Intervallen auf m Ebenen besteht. Wir zeigen, dass für T keine Lösung mit weniger als m Ebenen existiert. Um die Treppe zu konstruieren, nutzen wir, dass in einer Greedy-Layer-Assignment (Kapitel 3) berechneten Lösung für jedes Intervall e ein Stapel (siehe Definition 2.11) S existiert, den wir *Stufe* nennen und der bestimmte Eigenschaften erfüllt. Wir definieren dazu den Begriff *Auflieger*. Dieser ist das höchste zu e gerichtete Intervall, das in der Lösung unterhalb von e platziert wurde (vgl. Definition 3.4).

Definition 5.4. *Sei L eine Lösung, die von Greedy-Layer-Assignment berechnet wurde. Für ein $e \in L$ ist der* Auflieger *$a(e)$ das Intervall, welches zu e gerichtete ist, sodass kein anderes Intervall oberhalb von $a(e)$ zu e gerichtet ist.*

Lemma 5.5. *Für jedes $e \in L$ ist der Auflieger eindeutig.*

Beweis. Sei e ein beliebiges Intervall in L. Ein Intervall $a \in L$ ist laut Definition 5.1 zu e gerichtet, wenn a und e teilweise überlappen und e weiter rechts liegt als a. In diesem Fall enthält a den linken Endpunkt von e. Sei a nun der Auflieger von e und es gelte $\varphi(a) = k$. Dann kann es kein Intervall a' mit $\varphi(a') = k$ geben, das ebenfalls den linken Endpunkt von e enthält. Sonst gilt $a \cap a' \neq \emptyset$ und wir haben einen Widerspruch zur Definition einer gültigen Lösung. Daraus folgt, dass e auf jeder Ebene nur von einem einzigen Intervall eine gerichtete Kante haben kann und nur das höchste dieser Intervalle ist der Auflieger. □

Definition 5.6. *Für eine Lösung L, die von* Greedy-Layer-Assignment *berechnet wurde und ein Intervall $e \in L$ ist die* Stufe $S \subseteq L$ *von e ein Stapel, sodass e das oberste Intervall in S ist. Dann erfüllt S folgende Eigenschaften:*

(i) e gehört zu S. Unter e folgt eine Menge von Intervallen auf konsekutiven Ebenen, die zu e ungerichtete Kanten besitzen. Diese gehören zu S.

(ii) Alle zu e ungerichteten Intervallen in S überragen e zu beiden Seiten.

(iii) Das unterste von den zu e ungerichteten Intervallen liegt auf der niedrigsten Ebene oder es existiert $a(e)$. Dann gehört $a(e)$ zu S.

Wir schreiben $S(e)$ und meinen die Stufe in L, deren oberstes Intervall e ist. Alle Intervalle zwischen e und $a(e)$ heißen *Nicht-Auflieger-Intervalle* von e bzw. von $S(e)$. Wir zeigen nun, dass es stets zu einem Intervall, das unter e liegt, das eine ungerichtete Kante zu e hat und das vollständig in e enthalten ist, einen Auflieger $a(e)$ gibt.

Lemma 5.7. *Seien $e, s \in L$ Intervalle einer Lösung mit $\varphi(s) < \varphi(e)$ und e ist zu beiden Seiten breiter als s. Wenn für alle Intervalle v' mit $\varphi(s) < \varphi(v') < \varphi(e)$ und $e \cap e' \neq \emptyset$ gilt $l(e') < l(e)$ und $r(e') > r(e)$, dann existiert ein Intervall $a \in L$, das eine gerichtete Kante zu e hat mit $\varphi(s) = \varphi(a)$.*

Beweis. Sei e das Intervall, welches in Schritt i platziert wurde. Die Intervalle werden von Greedy-Layer-Assignment nach ihren linken Endpunkten aufsteigend sortiert verarbeitet. Daraus folgt, dass in Schritt i alle $i - 1$ bereits verarbeiteten Intervalle weiter links beginnen als e. Da s vollständig in e enthalten ist, muss der linke Endpunkt von s weiter rechts liegen als der von e. Daher muss s in einem späteren Schritt platziert worden sein als e. In Schritt i wurde e so tief wie möglich platziert. Daraus folgt, dass auf der Ebene, auf der s platziert wurde, bereits ein Intervall s lag, das *vor* e verarbeitet wurde und dessen linker Endpunkt entsprechend weiter links liegt als der von e. Da e nicht auf der Ebene von s platziert wurde, muss entweder ein Intervall a' existieren, das zwischen e und s liegt und eine gerichtete Kante zu e hat. Dies widerspricht aber der Annahme, dass alle Intervalle zwischen s und e vollständig mit e überlappen. Oder a schneidet e. Nun muss aber a enden, bevor s beginnt. Sonst hätte s dort nicht platziert werden dürfen. Dann müssen e und a teilweise überlappen und a ist gerichtet zu e. □

Lemma 5.8. *Zu* jedem *Intervall e in einer Lösung L, die von* Greedy-Layer-Assignment *berechnet wurde, existiert eine Stufe $S(e)$.*

Beweis. Sei e ein beliebiges Intervall in L. Liegt e auf der untersten Ebene von L, so sind die Eigenschaften (i) bis (iii) aus Definition 5.6 bereits erfüllt, wobei die Menge der Nicht-Auflieger leer und e das unterste Intervall von $S(e)$ ist.

Liegt e nicht auf der untersten Ebene, so muss ein Intervall e' mit $\varphi(e') = \varphi(e) - 1$ existieren und es gilt: $e \cap e' \neq \emptyset$. Andernfalls hätte e tiefer platziert werden dürfen. Ist e' gerichtet zu e, so ist e' der Auflieger und Eigenschaften (i) bis (iii) sind erfüllt, wobei auch hier die Menge der Nicht-Auflieger leer ist.

Ist e' ungerichtet zu e, so gilt wegen Lemma 5.7: Entweder ist e' zu beiden Seiten breiter als e. Dann ist e' ein Nicht-Auflieger und Eigenschaften (i) und (ii) sind erfüllt. Andernfalls existiert weiter links auf der Ebene von e' ein zu e gerichtetes Intervall a. Dann ist a der Auflieger von e und Eigenschaft (iii) ist erfüllt. Die übrigen Eigenschaften sind auch hier trivial erfüllt, da die Menge der Nicht-Auflieger leer ist.

Ist e' ein Nicht-Auflieger, dann muss auch unter e' ein Intervall existieren, das e schneidet oder e' liegt auf der untersten Ebene. Da e in diesem Fall zu e' eine ungerichtete Kante hat, gilt erneut: Ist die Ebene unter e' frei, so hätte e dort platziert werden müssen. Daraus folgt, dass bereits ein Intervall e'' unter e' liegt, das e schneidet. Dann ist e'' ein Nicht-Auflieger, vollständig in e enthalten oder der Auflieger von e.

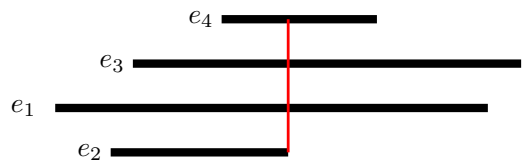
Ist e'' vollständig in e enthalten, so existiert wegen Lemma 5.7 ein Auflieger links von e''. Dann ist auch Eigenschaft (iii) erfüllt. Oder e'' ist breiter als e. Dann ist e'' ein Nicht-Auflieger und liegt auf der untersten Ebene. Andernfalls existiert erneut ein Intervall unter e'' und unser Kalkül setzt sich rekursiv fort. Daraus folgt, dass stets eine Menge von Intervallen unter e die Eigenschaften (i) und (ii) erfüllen, gefolgt vom Auflieger oder der untersten Ebene, wodurch auch Eigenschaft (iii) erfüllt ist. □

Beobachtung 5.9. *Sei $S(e)$ eine Stufe. Alle Intervalle in S enthalten den rechten Endpunkt von $a(e)$.*

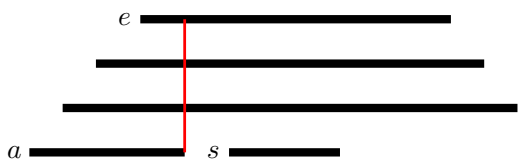
Beweis. Alle Nicht-Auflieger von e überragen e zu beiden Seiten. Da e eine teilweise Überlappung mit $a(e)$ hat, ist der rechte Endpunkt von $a(e)$ in e enthalten. Da e in allen Intervallen zwischen e und $a(e)$ enthalten ist, liegt auch der rechte Endpunkt von a in allen Nicht-Aufliegern. □

Definition 5.10. *Eine Treppe $T \subseteq L$ ist eine Teillösung von L aus k Stufen $S_1 \dots S_k$ gemäß Definition 5.6, sodass $T = \bigcup_{i=1}^{k} S_i$ gilt und für zwei Stufen S_i und S_j in T gilt $i = j + 1$ genau dann, wenn $S_i = S(e)$ und $S_j = S(a(e))$ ist. Die Menge der Auflieger in T heißt A und ein e' ist in A, genau dann, wenn eine Stufe $S(e')$ in T ist oder es existiert eine Stufe $S(e)$ in T, sodass $e' = a(e)$ gilt. Die Menge der Nicht-Auflieger ist $\overline{A} = T \setminus A$.*

In Abbildung 5.3 ist eine Treppe zu sehen, die Stufen sind Stapel nach Definition 5.6. Wir konstruieren T, beginnend mit einem Intervall e auf der obersten Ebene von L. Unter $S(e)$ folgt stets $S(a(e))$. Die Treppe endet, wenn das unterste Intervall einer Stufe



(a) Intervall e_4 liegt dem Stapel oben auf. Es müssen nicht zwingend alle Nicht-Auflieger untereinander ungerichtet sein. So sind e_1 und e_3 zueinander gerichtet. In e_2 haben wir ein zu e_4 gerichtetes Intervall, den Auflieger von e_4.



(b) Da s nach e verarbeitet wurde und unter e platziert wurde, muss ein Element a existieren, welches die Ebene, auf der s liegt, versperrt, vor s endet und vor e verarbeitet wurde. Dieses Intervall ist der Auflieger von e.

Abb. 5.2: Die abgebildeten Stapel sind Stufen laut Definition 5.6. In Rot hervorgehoben ist die x-Koordinate des rechten Endpunktes des Aufliegers, den alle Intervalle einer Stufe gemeinsam haben. Alle von der roten Vertikalen gekreuzten Intervalle gehören zu S.

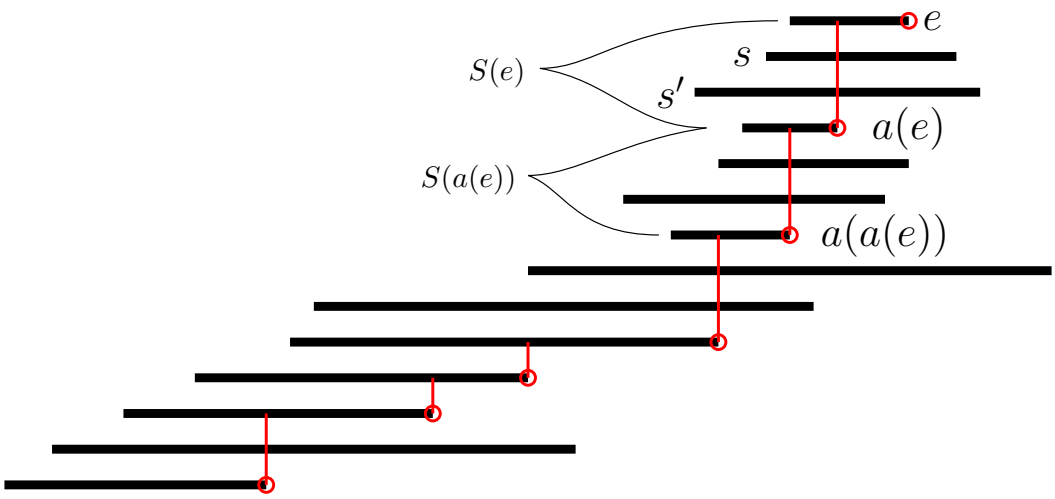


Abb. 5.3: Eine Treppe. Sie beginnt mit $S(e)$. Die Stufe darunter beginnt mit dem Auflieger $a(e)$. Hier sind s und s' die Nicht-Auflieger in $S(e)$. Die rechten Endpunkte der Auflieger in T sind rot markiert. Jede Stufe gehört zum Auflieger der vorherigen Stufe.

auf der untersten Ebene der Lösung liegt. Wenn L Höhe m hat, dann besteht T aus m Intervallen auf m konsekutiven Ebenen.

Beobachtung 5.11. *Wenn T mit einem Intervall auf Ebene m beginnt, so besteht T aus exakt m Intervallen auf konsekutiven Ebenen.*

Lemma 5.12. *Sei $A = (a_1, a_2, \ldots, a_k), A \subseteq T$ die Menge der Auflieger in T und $|A| = k$. Dann existiert ein gerichteter Pfad $(a_1, a_2, \ldots, a_k)$ im Konfliktgraphen.*

Beweis. Ohne Beschränkung der Gültigkeit sei a_k das oberste Intervall von T. Laut Definition 5.10 ist $S(a_k)$ die oberste Stufe von T. Die nächst-tiefere Stufe ist dann $S(a(a_k))) = S(a_{k-1})$. Daraus folgt, dass zwei Auflieger in aufeinanderfolgenden Stufen eine gerichtete Kante im Konfliktgraphen verbindet. Dann existiert auch ein gerichteter Pfad, der im untersten Auflieger a_1 beginnt und in a_k endet. □

Eine Konsequenz aus Lemma 5.12 ist, dass für jedes Paar an aufeinanderfolgenden Aufliegern a_i und a_{i+1} gilt, dass a_i früher beginnt und endet, als a_{i+1}. Dies folgt aus Definition 2.6. Diese Eigenschaft ist transitiv. Befindet sich ein Auflieger in T weiter oben als a_{i+1}, dann beginnt (und endet) er nicht nur weiter rechts als a_{i+1}, sondern auch als a_i und als jeder Auflieger unterhalb von a_i. Wir halten dies in einer Beobachtung fest:

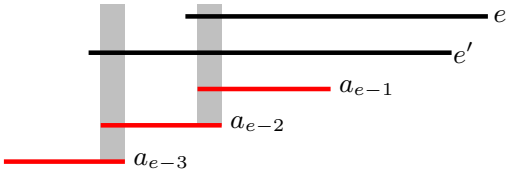
Beobachtung 5.13. *Sei $A = (a_1, a_2, \ldots, a_k), A \subseteq T$ die Menge der Auflieger in T. Dann gilt für jedes Paar $a_i, a_j, i < j$: $a_i(l) < a_{i+1}(l) \wedge a_i(r) < a_{i+1}(r)$*

Durch die Struktur von T können wir außer zu den Aufliegern auch Aussagen über die übrigen Intervalle in T treffen. So ist aus Eigenschaft (ii) klar, dass ein Nicht-Auflieger in einer Stufe der Treppe den nächst-höheren Auflieger zu beiden Seiten überragt. Dies bedeutet aber automatisch, dass ein Nicht-Auflieger alle Intervalle in der Stufe darüber sowie darunter schneidet.

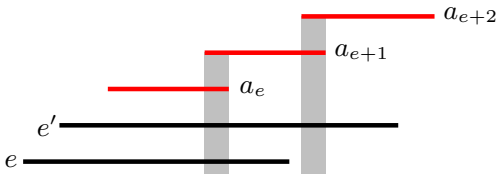
Lemma 5.14. *Sei $A \subseteq T$ die Menge der Auflieger in T. Für jeden Nicht-Auflieger $e \in T, e \notin A$ existiert ein Auflieger $a_i \in A$, der zu e gerichtet ist oder e beginnt weiter links als alle $a \in A$.*

Beweis. Wir betrachten Abb. 5.4. Sei $e \in T, e \notin A$ ein Nicht-Auflieger in T. Laut Eigenschaft (ii) ist e zu beiden Seiten breiter als der nächst-höhere Auflieger. Sei a_{i-1} der nächst-tiefere Auflieger in T. Da mit Beobachtung 5.13 die Auflieger einen gerichteten Pfad bilden gilt: Beginnt e weiter rechts als a_{i-1}, dann ist a_{i-1} in jedem Fall gerichtet zu e. Wenn nicht, gilt wieder wegen Beobachtung 5.13, dass a_{i-2} zu e gerichtet ist. Existiert kein Auflieger, der zu e gerichtet ist, folgt sofort, dass e weiter nach links beginnt als alle $a \in A$. □

Lemma 5.15. *Sei $A \subseteq T$ die Menge der Auflieger in T. Für jedes $e \in T$, $e \notin A$ existiert ein $a_i \in A$, zu dem e gerichtet ist, oder e endet weiter rechts als alle $a \in A$.*



(a) Nicht-Auflieger e und e' und der Ausschnitt aus T direkt darunter. Hier überlappt e den Auflieger a_{e-1} vollständig, sowie a_{e-2} teilweise und hat keine Überlappung mit a_{e-3}. Dagegen endet e' so weit links, dass es a_{e-1} und a_{e-2} vollständig, a_{e-3} nur teilweise überlappt.



(b) Nicht-Auflieger e und e' und der Ausschnitt aus T direkt daüber. Hier überlappt e den Auflieger a_e vollständig, a_{e+1} teilweise und hat keine Überlappung mit a_{e+2}. Dagegen ragt e' so weit nach rechts, dass es a_e sowie a_{e+1} vollständig, a_{e+2} jeodch nur teilweise überlappt.

Abb. 5.4: Ausschnitte einer Treppe direkt unter beziehungsweise über den Nicht-Aufliegern e und e'. Die Schnittmenge konsekutiver Auflieger ist grau hervorgehoben.

Beweis. Analog zu Lemma 5.14 und zu Abbildung 5.4 gilt für e, dass es a_e auch nach rechts überragt. Wegen Beobachtung 5.13 ist e entweder zu a_{i+1} gerichtet, oder es ragt so weit nach rechts, dass es a_{i+2} schneidet. Es gilt wieder: Entweder ist e zu a_{i+2} gerichtet, oder es schneidet auch den Auflieger darüber. Daraus folgt, dass e entweder gerichtet ist zu einem $a \in T$, oder es endet weiter rechts als alle Auflieger. □

Wir behaupten nun, dass T eine Clique in K^+ ist. Um die Behauptung zu beweisen, zeigen wir zuerst, dass die Menge der Auflieger A eine Clique bilden. Mithilfe der Lemmata 5.14 und 5.15 zeigen wir dann, dass Jedes Nicht-Auflieger-Intervall in T eine Clique mit A bildet. Anschließend zeigen wir, dass zwei Nicht-Auflieger eine Kante in K^+ verbindet. Daraus folgt dann die Behauptung.

Beobachtung 5.16. *Jeder gerichtete Pfad $(e_i, \ldots, e_j)$ in K^+ ist eine Clique.*

Beweis. Folgt direkt aus Definition 2.10 und Definition 5.2. □

Lemma 5.17. *Sei $A \subseteq T$ die Menge aller Auflieger in T. Dann bildet A eine Clique in K^+.*

Beweis. Folgt aus Beobachtung 5.16 und Lemma 5.12. □

Da A stets eine Clique in K^+ bildet, ist nun noch zu zeigen, dass jedes $e \in \overline{A}$ eine Clique mit A bildet, sowie das zwischen jedem Paar $e, e' \in \overline{A}$ eine Kante existiert. Wir zeigen zunächst, dass jeder Nicht-Auflieger eine Kante zu allen Aufliegern in T hat.

Lemma 5.18. *Sei A die Menge der Auflieger in T und $e \in T$ ein Nicht-Auflieger, dann ist $A \cup e$ eine Clique in K^+.*

Beweis. Aus Lemma 5.17 wissen wir, dass zwischen jedem $a \in A$ eine Kante existiert. Sei $S(a_i)$ die Stufe, in der e als Nicht-Auflieger vorkommt. Aus Definition 5.6 ist klar, dass e weiter links beginnt und weiter rechts endet als a_i. Falls e früher endet als a_{i+1}, dann liegt eine teilweise Überlappung zwischen e und a_{i+1} vor. In diesem Fall existiert ein gerichteter Pfad von e über a_{i+1} und zu allen Aufliegern a_j mit $j > i + 1$.

Falls e später als a_{i+1} endet, hat e auch eine ungerichtete Kante zu a_{i+1}. Dann liegt aber wieder eine Überlappung von e mit a_{i+2} vor. Diese kann wiederum vollständig oder teilweise sein. Entsprechend hat e eine gerichtete oder ungerichtete Kante zu a_{i+2}. Führen wir das Kalkül fort wird klar: Zwischen e und allen Aufliegern über a_i existiert eine gerichtete oder ungerichtete Kante in K^+.

Analog schneidet e auch a_{i-1}. Auch hier gilt wieder: Ist die Überlappung teilweise, so existiert ein gerichteter Pfad von a_1 zu e und e hat eine Kante mit allen Aufliegern unter a_i. Ist die Überlappung vollständig, so existiert eine ungerichtete Kante zwischen e und a_{i-1}. Dann liegt jedoch wieder eine Überlappung mit a_{i-2} vor. Auch hier folgt: Entweder hat e eine gerichtete oder ungerichtete Kante zu a_{i-2}. Daraus folgt: Es existiert eine Kante zwischen e allen Aufliegern a_j mit $j < i - 1$.

Da die ungerichtete Kante zu a_i bereits per Definition einer Stufe existiert, und eine Kante von e zu allen a_j mit $j < i$ und $j > i$ existiert, bildet e zusammen mit A eine Clique in K^+. □

Lemma 5.19. *Seien e und e' Nicht-Auflieger in T. Dann existiert entweder die ungerichtete Kante $\{e, e'\}$ oder die gerichtete Kante (e, e') in K^+.*

Beweis. Sei e ein Nicht-Auflieger in S_i und e' ein Nicht-Auflieger in S_j mit $i > j$. Wenn $i = j + 1$ gilt, dann ist S_j die Stufe des Aufliegers von S_i. In diesem Fall überragt e' den Auflieger von S_i zu beiden Seiten und es existiert wegen Beobachtung 5.9 eine Kante zwischen e' und allen Intervallen in S_i, also insbesondere auch zu e.

Sei also $i > j+1$. Dann existiert wegen Lemma 5.14 ein Auflieger in einer Stufe $S_{j'}$ mit $j' > j$, den e teilweise überlappt oder e überragt alle Auflieger in den Stufen oberhalb von S_j. Im letzteren Fall ist offensichtlich, dass e' auch eine Überlappung mit e hat. Im ersteren Fall gilt: Wenn $j' > i$ ist, dann hat e' eine vollständige Überlappung mit dem Auflieger von S_i und überlappt damit jedes Intervall S_i, also auch e.

Wenn dagegen $j' < i$ gilt, so folgt aus Lemma 5.15 analog, dass ein Auflieger in einer Stufe $S_{i'}$ existiert, der eine gerichtete Kante zu e hat, oder e beginnt weiter links als alle Auflieger unterhalb von S_i. Im ersten Fall folgt sowohl aus $i' < j$ als auch daraus, dass e alle Auflieger nach links überragt, dass e eine Überlappung mit e' hat.

Gilt hingegen $i' > j$, dann befinden sich die Auflieger von $S_{i'}$ und $S_{j'}$ oberhalb von S_j und unterhalb von S_i. Dann existiert ein gerichteter Pfad von e' über $S_{j'}$ und $S_{i'}$ zu e und es existiert eine gerichtete Kante zwischen e' und e in K^+. □

Wir haben in Lemma 5.17 gesehen, dass die Auflieger eine Clique im erweiterten Konfliktgraphen bilden. Lemma 5.18 sagt aus, dass jeder Nicht-Auflieger eine Clique mit den Aufliegern bildet. Und schließlich wissen wir aus Lemma 5.19, dass auch jedes Paar an Nicht-Aufliegern in K^+ durch eine Kante verbunden ist. Da die Menge der Nicht-Auflieger $\overline{A}$ definiert war als $T \setminus A$, existieren keine Intervalle in T, die weder Auflieger noch Nicht-Auflieger sind. Wir halten daher fest:

Lemma 5.20. *T ist eine Clique im erweiterten Konfliktgraph K^+.*

Beweis. Aus Definition 5.10 folgt, dass alle Intervalle in T entweder in A oder in $\overline{A}$ liegen. Wegen Lemma 5.17 gilt: A ist eine Clique in K^+. Wegen Lemma 5.18 folgt: Für

jedes $e \in \overline{A}$ gilt: $A \cup e$ ist eine Clique in K^+. Schließlich folgt aus Lemma 5.19: Für jedes $e, e' \in \overline{A}$ gilt: e und e' sind durch eine Kante in K^+ verbunden. Wenn jedes Paar an Aufliegern durch eine Kante in K^+ verbunden ist, ebenso jedes Paar an Nicht-Aufliegern und wenn jedes Paar aus einem Auflieger und einem Nicht-Auflieger durch eine Kante in K^+ verbunden ist, dann ist T eine Clique in K^+. □

In Abbildung 5.5 sehen wir eine Treppe laut Definition 5.10 und den dazugehörigen, erweiterten Konfliktgraphen K^+. Hier ist A in Rot markiert, e_i und e_j sind Intervalle aus $\overline{A}$. Die Auflieger $a_1, \ldots, a_5$ bilden einen gerichteten Pfad. Die einzigen ungerichteten Kanten in diesem Beispiel sind $\{e_j, a_3\}$ und $\{e_i, a_5\}$. Diese existieren wegen Definition 5.6. Da die gerichtete Kante (a_2, e_j) existiert, gibt es auch einen gerichteten Pfad von a_1 bis e_j. Ebenso existiert wegen der Kante (a_4, e_i) ein Pfad von a_1 bis e_i. Da außerdem die Kante (e_j, a_4) existiert, gibt es auch einen gerichteten Pfad von e_j zu e_i. Da jeder gerichtete Pfad eine Clique bildet (siehe Beobachtung 5.16), sind sowohl $A \cup e_j$ als auch $A \cup e_i$ eine Clique. Da in diesem Beispiel auch noch ein gerichteter Pfad zwischen e_j und e_i existiert, ist T eine Clique.

Wir nutzen nun die obigen Resultate um zu zeigen, dass der Algorithmus aus Kapitel 3 stets eine optimale Lösung berechnet. Da wir bereits mit Satz 5.3 gezeigt haben, dass das Färben von K^+ äquivalent zum Lösen von Layer-Assignment ist, können wir jetzt argumentieren, dass keine gültige Lösung mit weniger Ebenen auskommt als von Greedy-Layer-Assignment berechnet wurde.

Satz 5.21. *Greedy-Layer-Assignment ist optimal.*

Beweis. Sei I eine Menge von Intervallen und L eine Lösung für I mit Höhe m, die von Greedy-Layer-Assignment berechnet wurde. Wir wählen ein Intervall e aus mit $\varphi(e) = m$. Nun existiert wegen Lemma 5.8 und Beobachtung 5.11 auch eine Treppe T, deren oberstes Intervall e ist und die aus genau m Intervallen besteht. Aus Lemma 5.20 folgt: Im erweiterten Konfliktgraphen von I ist T eine Clique. Somit gilt $\omega(K^+) = m$. Aus Satz 5.3 folgt, dass eine gültige Lösung von Layer-Assignment stets auch eine Färbung von K^+ ist. Da zwei durch eine Kante verbundene Knoten nie in der selben Farbe gefärbt sein dürfen, und K^+ eine Clique der Größe m enthält wird klar: Es existiert keine Lösung, die weniger als m Ebenen verwendet. Da L unter allen gültigen Lösungen minimale Höhe besitzt, ist Greedy-Layer-Assignment optimal. □

Alle Aussagen und Resultat, die wir für rechtsgerichtete Intervalle gezeigt haben, sind auch für eine Menge von linksgerichteten Intervallen gültig. Die Beweise folgen symmetrisch zu denen für rechtsgerichtete Intervalle. Wendet man die naive Implementierung von Greedy-Layer-Assignment laut Algorithmus 1 auf eine Eingabe an, die absteigend nach den x-Koordinaten der rechten Endpunkte der Intervalle sortiert wurde, so erhält man eine gültige Lösung für den linksgerichteten Fall, in der sich ebenfalls eine Treppe konstruieren lässt. Da wir für den erweiterten Konfliktgraphen gezeigt haben, dass wir stets eine Färbung mit $\chi(K^+)$ Farben berechnen können, und laut Satz 5.3 auch jede Lösung von Layer-Assignment eine Färbung von K^+ ist, gilt offensichtlich stets $\omega(K^+) = \chi(K^+)$.

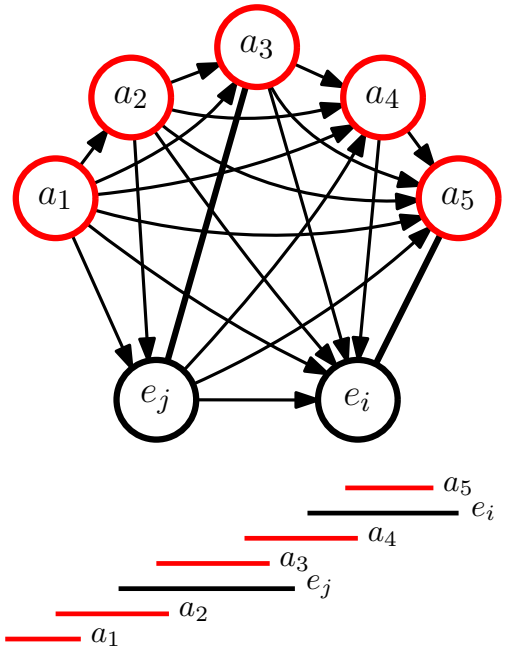


Abb. 5.5: Eine Treppe T und der dazugehörige, erweiterte Konfliktgraph K^+. Die Auflieger sind rot dargestellt. Die Nicht-Auflieger in diesem Beispiel sind e_i und e_j

Korollar 5.22. *Der erweiterte Konfliktgraph K^+ ist perfekt.*

Wir haben den erweiterten Konfliktgraphen K^+ kennengelernt und gezeigt, dass eine Färbung von K^+ stets auch eine Lösung von Layer-Assignment ist. Dies haben wir genutzt, um anhand einer Teillösung, die wir Treppe nannten, zu zeigen, dass Greedy-Layer-Assignment optimal ist. Nun wollen wir noch einen Ausblick auf offene Fragen und weiterführende Themen geben.

6 Ausblick

In einen Kabelplan, wie er in Kapitel 1 definiert wird, können sowohl links- wie auch rechtsgerichtete Leitungen vorkommen. Eine Erweiterung von Layer-Assignment ist, in einer Zeichnung beide Typen von Leitungen darzustellen. Dies führt zu einem Problem, das wir Beidseitiges Layer-Assignment nennen.

Definition 6.1. *Beim Beidseitigen Layer-Assignment Problem erhält man als Eingabe eine Menge I^r von rechtsgerichteten und eine Menge I^l von linksgerichteten Intervallen. Eine Lösung L ist dann eine Funktion $\varphi : I^r \cup I^l \to \mathbb{N}$, die allen $e \in I^r \cup I^l$ eine Ebene zuordnet, sodass für alle $e, e' \in I^r \cup I^l$ mit $e \cap e' \neq \emptyset$ folgende Bedingungen erfüllt sind:*

1. $\varphi(e) \neq \varphi(e')$
2. $e, e' \in I^r \wedge l(e) < l(e') \wedge r(e) < r(e') \Rightarrow \varphi(e) < \varphi(e')$
3. $e, e' \in I^l \wedge l(e) < l(e') \wedge r(e) < r(e') \Rightarrow \varphi(e) > \varphi(e')$

Abbildung 6.1 zeigt einen möglichen Kabelplan, indem beide Typen von Leitungen dargestellt sind. Hier wurde zunächst mit Greedy-Layer-Assignment eine Lösung für I^r mit Höhe m_r und eine Lösung für I^l mit Höhe m_l berechnet. Anschließend wurden die linksgerichteten Leitungen einfach oberhalb der rechtsgerichteten gezeichnet. Dazu wurde für jedes $e \in I^l$ die zugewiesene Ebene auf $\varphi(e) + m_r$ erhöht. Dies ist offenbar eine gültige Lösung, da alle $e \in I^r$ auf unterschiedlichen Ebenen als alle $e' \in I^l$ liegen. Die Gesamtlösung hat Höhe $m_r + m_l$. Da wegen Satz 5.21 die Lösungen von I^r und I^l optimal sind, ist eine Lösung für $I^r \cup I^l$ nach unten beschränkt durch $\max\{m_r, m_l\}$. Daraus folgt, dass $m_r + m_l$ höchstens doppelt so groß ist wie minimal möglich wäre.

Korollar 6.2. *Beidseitiges Layer-Assignment kann mit Faktor 2 approximiert werden.*

Aus den vorliegenden Resultaten ergeben sich folgende weiterführende Forschungsfragen:

- *Ist Beidseitiges Layer-Assignment $\mathcal{NP}$-schwer?*
- Wenn ja: Wie findet man in Polynomialzeit eine Lösung mit besserer Güte als 2?
- Wie lassen sich die Graphen, die zu Beidseitigem Layer-Assignment korrespondieren, charakterisieren?
- Wodurch ist die Färbezahl beschränkt?

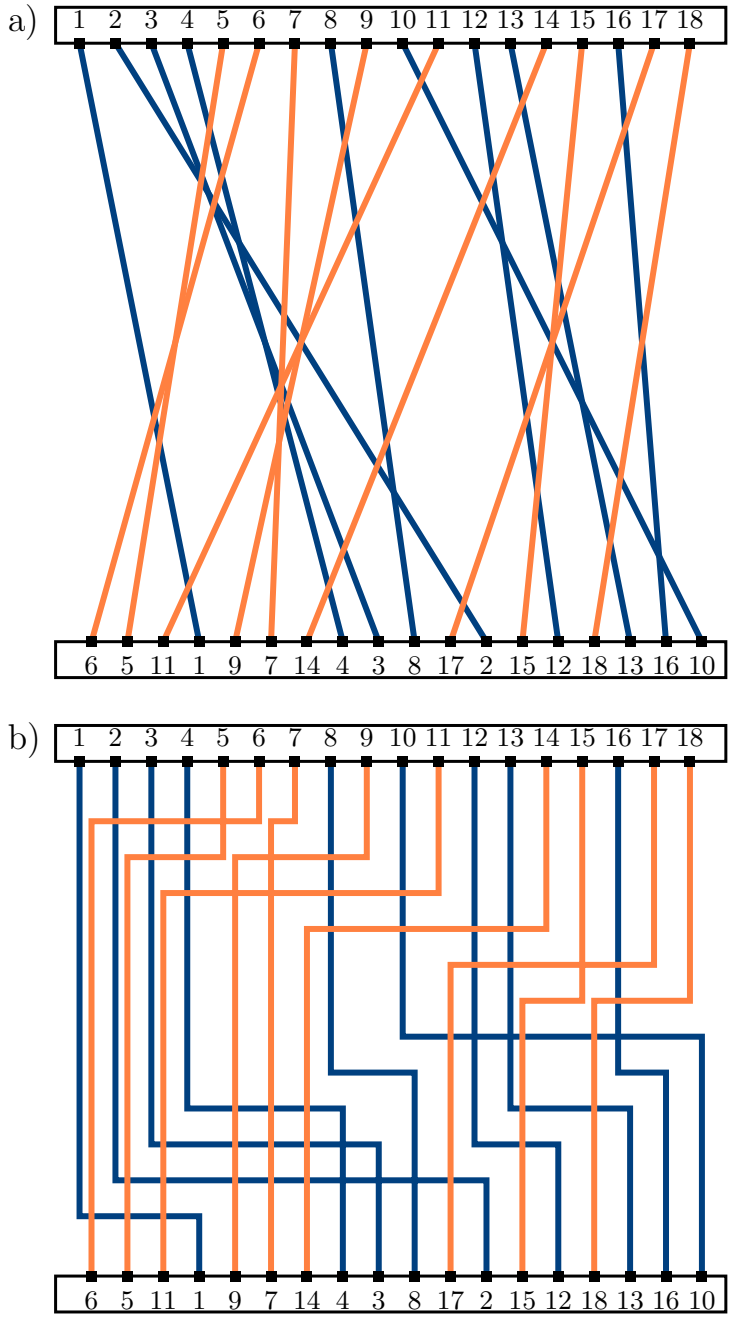


Abb. 6.1: Ein Kabelplan. Rechtsgerichtete Leitungen sind in Blau dargestellt, linksgerichtete in Orange. In (a) sind die Leitungen Geraden beliebiger Steigung zwischen oberem und unteren Endpunkt. Die Zeichnung hat vermeidbare Kreuzungen und ist unübersichtlicher. In (b) sind Leitungen dargestellt durch vertikale und horizontale Geraden-Segmente. Eine solche Zeichnung wird auch orthogonal genannt. Die links- und rechtsgerichteten horizontalen Segmente wurden jeweils optimal gelöst und danach übereinander gezeichnet.

Literaturverzeichnis

[AVL62] Georgy Adelson-Velskii und Evgenii Landis: An algorithm for the organization of information. *Proceedings of the USSR Academy of Sciences*, 146:2:263–266, 1962. `http://mi.mathnet.ru/dan26964`.

[Bay72] Rudolf Bayer: Symmetric Binary B-Trees: Data Structure and Maintenance Algorithms. *Acta Informatica*, 1(4):290–306, 1972, 10.1007/BF00289509.

[Ben59] Seymour Benzer: On the Topology of the Genetic Fine Structure. *Proceedings of the National Academy of Sciences*, 45(11):1607–1620, 1959, 10.1073/pnas.45.11.1607.

[Ber61] Claude Berge: Färbung von Graphen, deren sämtliche bzw. deren ungerade Kreise starr sind. *Wissenschaftliche Zeitschrift der Martin-Luther-Universität Halle-Wittenberg, Mathematisch-Naturwissenschaftliche Reihe*(10):114–115, 1961. `https://ci.nii.ac.jp/naid/10021313786/en/`.

[BNBYF+01] Amotz Bar-Noy, Reuven Bar-Yehuda, Ari Freund, Joseph Naor und Baruch Schieber: A Unified Approach to Approximating Resource Allocation and Scheduling. *J. ACM*, 48(5):1069–1090, 2001, 10.1145/502102.502107.

[Brü21] Lukas Brückner: Orthogonales Zeichnen als Färbeproblem in perfekten Graphen. Bachelorarbeit, Institut für Informatik, Universität Würzburg, 2021. `https://www1.pub.informatik.uni-wuerzburg.de/pub/theses/2021-brueckner-bachelor.pdf`.

[CLRS09] Thomas Cormen, Charles Leiserson, Ronald Rivest und Clifford Stein: *Introduction to algorithms.* The MIT Press, Cambridge, Massachusetts, 3. Auflage, 2009. `https://mitpress.mit.edu/books/introduction-algorithms-third-edition`.

[CRST06] Maria Chudnovsky, Neil Robertson, Paul Seymour und Robin Thomas: The strong perfect graph theorem. *Annals of Mathematics*, 164:51–229, 2006, 10.4007/annals.2006.164.51.

[GH64] Paul Gilmore und Alan Hoffman: A Characterization of Comparability Graphs and of Interval Graphs. *Canadian Journal of Mathematics*, 16:539–548, 1964, 10.4153/CJM-1964-055-5.

[GS78] Leonidas Guibas und Robert Sedgewick: A dichromatic framework for balanced trees. *Proceedings of the Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, 10(78):8–21, 1978, 10.1109/sfcs.1978.3.

[HKdW97] Pierre Hansen, Julio Kuplinsky und Dominique de Werra: Mixed graph colorings. *Mathematical Methods of Operations Research*, 45(1):145–160, 1997, https://doi.org/10.1007/BF01194253.

[LB62] Cornelis Lekkerkerker und Johan Boland: Representation of a finite graph by a set of intervals on the real line. *Fundamenta Mathematicae*, 51(1):45–64, 1962. `http://eudml.org/doc/213681`.

[NR73] Jürg Nievergelt und Edward Reingold: Binary Search Trees of Bounded Balance. *SIAM Journal on Computing*, 2(1):33–43, 1973, 10.1137/0202005.

[Sot20] Yuri Sotskov: Mixed Graph Colorings: A Historical Review. *Mathematics*, 8(3), 2020, 10.3390/math8030385.

[ST76] Yuri Sotskov und Vyacheslav Tanaev: A chromatic polynomial of a mixed graph. *Vestsi Akademii Navuk BSSR Seryya Fizika-Matematychnykh Navuk*, 6:20–23, 1976.

[STT81] Kozo Sugiyama, Shojiro Tagawa und Mitsuhiko Toda: Methods for Visual Understanding of Hierarchical System Structures. *IEEE Transactions on Systems, Man, and Cybernetics*, 11(2):109–125, 1981, 10.1109/TSMC.1981.4308636.

[Wes01] Douglas West: *Introduction to Graph Theory.* Prentice Hall, 2. Auflage, 2001. `https://ia801204.us.archive.org/35/items/igt_west/igt_west_text.pdf`.

[ZWBW22] Johannes Zink, Julian Walter, Joachim Baumeister und Alexander Wolff: Layered Drawing of Undirected Graphs with Generalized Port Constraints. *Comput. Geom. Theory Appl.*, 105–106(Artikel 101886):29 pages, 2022, 10.1016/j.comgeo.2022.101886. Erscheint in Kürze.

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 11. Mai 2022

. . . . . .
Florian Mittelstädt