Karlsruher Institut für Technologie

Erkennung von gerichteten Intervallgraphen

Masterarbeit von

Christoph Hoffmeyer

an der Fakultät für Informatik
Institut für Theoretische Informatik (ITI)

Erstgutachter:	Dr. rer. nat. Torsten Ueckerdt
Zweitgutachterin:	TT-Prof. Dr. Thomas Bläsius
Betreuer:	Dr. rer. nat. Torsten Ueckerdt

07.12.2023 – 07.09.2024

Karlsruher Institut für Technologie
Fakultät für Informatik
Postfach 6980
76128 Karlsruhe

Ich versichere wahrheitsgemäß, die Arbeit selbstständig verfasst, alle benutzten Quellen und Hilfsmittel vollständig und genau angegeben und alles kenntlich gemacht zu haben, was aus Arbeiten anderer unverändert oder mit Abänderungen entnommen wurde sowie die Satzung des KIT zur Sicherung guter wissenschaftlicher Praxis in der jeweils gültigen Fassung beachtet zu haben.

Karlsruhe, 07.09.2024

. . . . . .

(Christoph Hoffmeyer)

Zusammenfassung

Für eine Menge von Intervallen ist der korrespondierende *Intervallgraph* eine Repräsentation der Schnitte zwischen den Intervallen. Ein *gerichteter Intervallgraph* ist ein gemischter Graph, bei dem ungerichtete Kanten Intervalle repräsentieren, die sich vollständig überlappen, und gerichtete Kanten Intervalle repräsentieren, die sich schneiden. Wir untersuchen den Algorithmus zur Erkennung von gerichteten Intervallgraphen von G. Gutowski et al. aus [Gut+22]. Ziel dieser Arbeit ist es die Funktionsweise des Algorithmus und seine Bestandteile besser zu verstehen. Dabei wurden die verschiedenen Schritte des Algorithmus, einschließlich der Konstruktion und Rotation von PQ-Bäumen und der Konstruktion von partiell geordneten Mengen im Detail erläutert. Diese Arbeit stellt eine umfassende Aufarbeitung des Algorithmus dar und leistet einen Beitrag zu seinem besseren Verständnis und seiner möglichen Weiterentwicklung.

Inhaltsverzeichnis

1 Einleitung

Intervallgraphen tauchen ganz natürlich in vielen Anwendungen auf, die sich mit der Anordnung von überlappenden Strukturen beschäftigen. Ein ungerichteter Graph $G = (V, E)$ ist ein *Intervallgraph*, wenn die Knotenmenge V als Menge von Intervallen $\mathcal{I}$ auf der reellen Linie $\mathbb{R}$ repräsentiert werden können und Knoten genau dann adjazent sind, wenn sich ihre Intervalle schneiden.

Gerichtete Intervallgraphen erweitern das Konzept der klassischen Intervallgraphen. *Gemischte* Graphen $G = (V, E, A)$ haben sowohl eine Menge ungerichteter Kanten E als auch eine Menge gerichteter Kanten A. Ein gemischter Graph $G = (V, E, A)$ ist ein *gerichteter Intervallgraph*, wenn seine Knotenmenge V ebenfalls als Intervalle $\mathcal{I}$ repräsentiert werden können. Wenn eine ungerichtete Kante $\{u, v\} \in E$ existiert, dann muss eines der Intervalle das andere vollständig überlappen. Wenn es eine gerichtete Kante $(u, v) \in A$ gibt, dann schneidet das eine Intervall das andere nur.

1.1 Motivation

Intervallgraphen wurden ursprünglich unabhängig voneinander in den 1950ern von Mathematiker Hajós und einem Biologen Benzer eingeführt [Gol04]. Seitdem wurden sie eingehend untersucht. Sie finden Anwendung in verschiedenen Bereichen, von Schedulingproblemen bis hin zur Genomanalyse [Gol04].

Gemischte Graphen als Intervallgraphen bekamen erstmalig Aufmerksamkeit im Rahmen einer Anwendung des *Sugiyama-Frameworks* [ZWBW22]. Das Sugiyama-Framework wird zur Visualisierung von gerichteten, azyklischen Graphen verwendet, um diese für das bloße Auge in eine übersichtliche Darstellung zu bringen [STT81]. Das Framework unterteilt das Problem in mehrere NP-schwere Probleme, die durch Heuristiken möglichst effizient gelöst werden sollen. Eines dieser Teilprobleme kann als Färbungsproblem von *bigerichteten Intervallgraphen* betrachtet werden [ZWBW22]. Ein bigerichteter Intervallgraph $G = (V, E, A)$ ist gemischter Intervallgraph und eine Spezialisierung von gerichteten Intervallgraphen.

Im Zuge der Arbeit von Zink et al. [ZWBW22] wurden anschließend verschiedene Klassen von gemischten Intervallgraphen untersucht. Effiziente Algorithmen zur Erkennung und Färbung könnten zu einer Verbesserung des Sugiyama-Framework beitragen.

Ziel dieser Arbeit ist es, den Algorithmus zur Erkennung von gerichteten Intervallgraphen wie von G. Gutowski et al. [Gut+22] beschrieben, näher zu untersuchen. Zur Verständnis des Algorithmus werden erforderliche Grundlagen erläutert. Die einzelnen Bestandteile des Algorithmus werden erläutert und durch zusätzliche Grafiken unterstützt.

1.2 Verwandte Arbeiten

Der in dieser Arbeit vorgestellte Algorithmus zur Erkennung von gerichteten Intervallgraphen wurde von G. Gutowski et al. entwickelt [Gut+22]. Ihre Arbeit umfasst die Untersuchung von gemischten Intervallgraphen. Die Veröffentlichung stellt einen Beweis über die Optimalität

einer Färbung von gerichteten Intervallgraphen in $\mathcal{O}(n \log n)$ Schritten mit n als Anzahl der Intervalle vor. Es umfasst einen Beweis über das Entscheidungsproblem, ob ein gemischter Intervallgraph mit k Farben gefärbt werden kann, was als NP-vollständig bewiesen wurde. Außerdem enthält es den besagten Algorithmus zur Erkennung von gerichteten Intervallgraphen in $\mathcal{O}(|V|^2)$.

Die Veröffentlichung ist motiviert durch eine frühere Arbeit von Zink et al. [ZWBW22] über die geschichtete Zeichnung von ungerichteten Graphen mit generalisierten Port- Einschränkungen. In dieser Arbeit wird ein Algorithmus vorgestellt, der das Sugiyama-Framework verwendet, um effiziente Kabelpläne von komplexen Maschinen mit verschiedenen elektrischen Komponenten und verschiedenen Ports dieser Komponenten zu erstellen. Hier wurde erstmalig gezeigt, dass der Kantenrouting-Schritt im Sugiyama-Framework als Färbung von bigerichteten Intervallgraphen aufgefasst werden kann. Ein bigerichteter Intervallgraph ordnet den Intervallen zusätzlich zwei Richtungen (links gerichtet und rechts gerichtet) zu.

In einer nachfolgenden Veröffentlichung [Gut+23] haben G. Gutowski et al. eine weitere Klasse von gemischten Intervallgraphen untersucht. Die sogenannten *Intervall- Überdeckungsgraphen*. Ein gemischter Graph $G = (V, E, A)$ ist ein Intervall-Überdeckungsgraph, wenn seine Knotenmenge V ebenfalls als Intervalle $\mathcal{I}$ repräsentiert werden können. Die ungerichteten Kanten und gerichteten Kanten sind im Vergleich zu gerichteten Intervallgraphen vertauscht. Das heißt, wenn eine ungerichtete Kante $\{u, v\} \in E$ existiert, dann schneidet das eine Intervall das andere. Wenn eine gerichtete Kante $(u, v) \in A$ existiert, dann muss eines der Intervalle das andere überlappen. Für diese verwandte Klasse von gemischten Intervallgraphen konnte gezeigt werden, dass für einen beliebigen gemischten Graphen $G = (V, E, A)$ in einer Laufzeit von $\mathcal{O}(|V| \cdot |E|)$ entschieden werden kann, ob es ein Intervall-Überdeckungsgraph ist. Falls dies der Fall ist, wird eine Intervallrepräsentation bestimmt.

Für die Erkennung von gerichteten Intervallgraphen werden wir einen bestimmten Baum verwenden. Erstmalig wurde die Konstruktion eines Baumes eingesetzt von Booth und Lueker in [BL76], um für einen ungerichteten Graphen $G = (V, E)$ zu bestimmen, ob es sich um einen Intervallgraphen handelt. Sie führen die sogenannten PQ-Bäume ein. Dies ist eine Datenstruktur, die wir in der Einführung weiter erörtern werden. Durch die PQ-Bäume war es möglich, Intervallgraphen in $\mathcal{O}(|V| + |E|)$ zu bestimmen.

Nach Korte und Möhring gilt die Konstruktion dieser PQ-Bäume als kompliziert, da diese eine aufwendige und mehrschichtige Fallunterscheidung beinhaltet. Deshalb haben sie [KM89] eine modifizierte Variante eingeführt. Diese werden wir im hier vorgestellten Algorithmus verwenden. Die Konstruktion erfolgt ebenfalls in $\mathcal{O}(|V|+|E|)$. Mit modifizierten PQ-Bäumen ist es ebenfalls möglich zu entscheiden, ob ein ungerichteter Graph $G = (V, E)$ ein Intervallgraph ist. Allerdings haben die modifizierten MPQ-Bäume zusätzliche Eigenschaften, die wir uns zunutze machen werden, um gerichtete Intervallgraphen zu erkennen.

Eine weitere Variante von PQ-Bäumen sind *PC-Bäume*. Diese wurden von Hsu und McConnel entwickelt [HM03]. Im Gegensatz zu den gewurzelten PQ-Bäumen sind PC-Bäume ungewurzelt. Die Datenstruktur ist ansonsten analog aufgebaut. Es wurde auch gezeigt, dass die Konstruktion von PQ-Bäume auf die Konstruktion von PC-Bäumen mit linearem Overhead reduziert werden kann. In experimentellen Untersuchungen konnte gezeigt werden, dass die Implementierung der Konstruktion von PC-Bäumen die der PQ-Bäume um bis zu einen Faktor 4 schlägt [FPR23].

Wir werden partiell geordnete Mengen (Posets) verwenden, um nachzuweisen, dass ein Graph die Eigenschaften eines gerichteten Intervallgraphen hat. Ein Poset ist ein Tupel aus einer Menge und einer partiellen Ordnung, die auf dieser Menge definiert ist. Bei Gutowski

wird gezeigt, dass ein ungerichteter Graph $G = (V, E)$ ein gerichteter Intervallgraph ist, wenn das von ihnen konstruierte Poset zweidimensional ist [Gut+22]. In [MS99] wurde gezeigt, dass dies mit einer Laufzeit von $\mathcal{O}(|V| + |E|)$ überprüft werden kann.

1.3 Übersicht

In Kapitel 2 werden die Grundlagen und Konzepte erläutert, die im Algorithmus verwendet werden. Dazu gehören eine formelle Einführung in die Graphenklasse der Intervallgraphen, die Datenstruktur der PQ-Bäume sowie partiell geordnete Mengen.

Daraufhin erfolgt eine ausführliche Erläuterung des Algorithmus zur Erkennung von gerichteten Intervallgraphen. Ein zentraler Punkt dieses Kapitels ist es, dass wir die Datenstrukturen, die wir im vorherigen Kapitel eingeführt haben, konstruieren. Dies beinhaltet einen modifizierten PQ-Baum, eine partiell geordnete Menge und die Intervallrepräsentation. Außerdem analysieren wir die Laufzeit des Algorithmus.

Kapitel 4 fasst die wesentlichen Erkenntnisse zusammen und gibt einen Ausblick auf zukünftige Forschung.

2 Einführung

Im folgenden Kapitel führen wir grundlegende Konzepte und Notation ein, die in dieser Arbeit verwendet werden. Wir beginnen mit wesentlichen Eigenschaften und Notationen von Graphen, Ordnungen und Relationen. Dann führen wir die Klasse der Intervallgraphen, die PQ-Bäume und die partiell geordneten Mengen formell ein.

2.1 Graphen

Wir nehmen an, dass alle Graphen endlich und einfach sind (keine Schleifen oder Mehrfachkanten). Sei $G = (V, E)$ ein ungerichteter Graph. Dann bezeichnet $V(G)$ und V die Menge der Knoten von G und $E(G)$ und E die Menge der ungerichteten Kanten von G. Wenn es eine Kante $\{u, v\} \in E$ gibt, dann sagen wir, dass u und v *adjazent* sind. Die Menge aller adjazenten Knoten eines Knotens $v \in V$ ist $\text{Adj}(v) = \{u \mid \{u, v\} \in E\}$. Als *Nachbarschaft* bezeichnen wir die Vereinigung dieser Menge mit dem Knoten selbst $N(v) = \text{Adj}(v) \cup \{v\}$.

Zunächst definieren wir einige Klassen von Graphen.

Definition 2.1: *Ein vollständiger Graph $K_n = (V, E)$ hat $n = |V|$ Knoten, die paarweise adjazent sind.*

Ein Graph $G = (V, E)$, der nur aus einem Knoten ohne Kanten besteht, ist bereits ein vollständiger Graph. Ebenso ist ein Graph aus 2 Knoten mit einer Kante ein vollständiger Graph. Vollständige Graphen haben mit $|E| = \binom{n}{2} = \frac{n^2-n}{2}$ die maximale Anzahl an Kanten, die für einen Graphen möglich sind, solange keine Mehrfachkanten erlaubt sind.

Definition 2.2: *Sei $G = (V, E)$ ein ungerichteter Graph. Dann ist $\overline{G} = (V, \overline{E})$ der* Komplementgraph *von G mit*

$$\overline{E} = \{\{u, v\} \in V \times V \mid u \neq v \text{ und } \{u, v\} \notin E\} \tag{2.1}$$

Beim Komplementgraphen $\overline{G}$ bleibt die Knotenmenge gleich, aber alle Kanten, die in G enthalten sind, werden entfernt. Dafür enthält $\overline{G}$ alle Kanten, die mit den Knoten von G möglich sind, aber nicht in G enthalten sind. Die Kantenmenge von $\overline{G}$ ist somit das Inverse von G.

Definition 2.3: *Ein* Schnittgraph *$G = (V, E)$ ist ein ungerichteter Graph über eine Familie $\mathcal{F}$ von Mengen. Für jede Menge $F_i \in \mathcal{F}$ gibt es einen Knoten $v_i \in V$. Zwei Knoten v_i, v_j sind adjazent genau dann, wenn der Schnitt der korrespondierenden Mengen nicht leer ist. Es gilt:*

$$E(G) = \{\{v_i, v_j\} \mid i \neq j, F_i \cap F_j \neq \emptyset, v_i, v_j \in V\} \tag{2.2}$$

Schnittgraphen können auf verschiedenen geometrischen Strukturen konstruiert werden, die als Mengen ausgedrückt werden. Es gibt eine bijektive Abbildung zwischen den Mengen $\mathcal{F}$ und den Knoten V des Graphens. Entscheidend ist die Definition des Schnittes der Mengen, da dieser die Kantenbeziehung vorgibt. In Abbildung 2.1 sehen wir beispielsweise einen Schnittgraphen für Flächen in $\mathbb{R}^2$. Jede Fläche wird durch einen Knoten repräsentiert und wenn Flächen sich schneiden, gibt es eine Kante. Wir werden uns mit Schnittgraphen von Intervallen beschäftigen, die im 2.3 Abschnitt ausführlich vorgestellt werden.

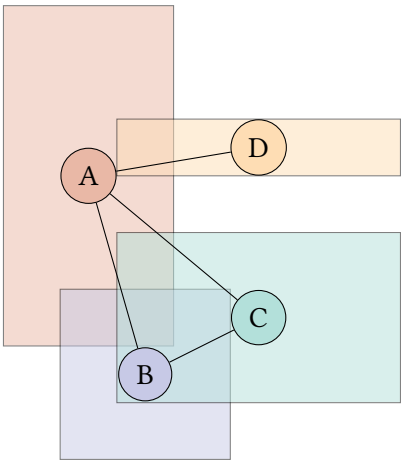

Abbildung 2.1: Schnittgraph von 4 Flächen in $\mathbb{R}^2$

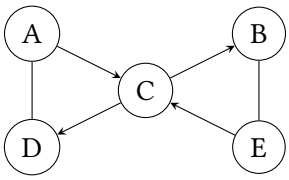

Abbildung 2.2: Gemischter Graph $G = (V, E, A)$

Definition 2.4: *Ein* gemischter Graph $G = (V, E, A)$ *hat ungerichtete Kanten* E *und gerichtete Kanten* A*. Jeder gemischte Graph hat einen* unterliegenden, ungerichteten Graphen $U(G) = (V, E')$ *mit* $E' = E \cup \{\{u, v\} \mid (u, v) \in A \vee (v, u) \in A\}$.

Die Kantenmenge des unterliegenden, ungerichteten Graphens ist die Vereinigung der beiden Kantenmengen. Hier betrachten wir nur gemischte Graphen, für die gilt, dass zwischen zwei Knoten höchstens eine gerichtete oder eine ungerichtete Kante existiert. Wir werden uns in dieser Arbeit besonders mit gemischten Graphen auseinandersetzen. Viele der Konzepte und Eigenschaften, die hier definiert werden, können auch für gemischte Graphen definiert werden. Der Einfachheit halber werden wir nicht jede Definition explizit für ungerichtete, gerichtete und gemischte Graphen angeben.

Definition 2.5: *Sei* $G = (V, E)$ *ein Graph. Ein Pfad* P *in* G *der Länge* l *ist eine Sequenz von Knoten* $(v_1, \ldots, v_l)$ *mit* $v_i \in V$ *für alle* $1 \leq i \leq l$*, wenn* $\{v_{j-1}, v_j\} \in E$ *für alle* j *mit* $1 < j \leq l$ *gilt.*

Pfade sind eine endliche Sequenz von Knoten, die durch eine endliche Sequenz von Kanten verbunden sind. Wir nehmen an, dass alle Pfade in dieser Arbeit einfach sind. Das heißt, jeder Knoten kann nur einmal auf einem Pfad vorkommen. In Abbildung 2.3 sehen wir beispielsweise in Rot die Knoten und die Kanten eines Pfades P markiert.

Definition 2.6: *Sei* $G = (V, A)$ *ein gerichteter Graph. Eine Kette* K *in* G *der Länge* l *ist eine Sequenz von Knoten* $(v_1, \ldots, v_l)$ *mit* $v_i \in V$ *für alle* $1 \leq i \leq l$*, wenn* $(v_i, v_j) \in A$ *für alle* $1 \leq i < j \leq l$ *gilt.*

Eine Kette ist sehr ähnlich aufgebaut wie ein Pfad, allerdings involvieren diese mehr Kanten. Jeder Knoten muss eine gerichtete Kante zu allen nachfolgenden Knoten der Kette haben.

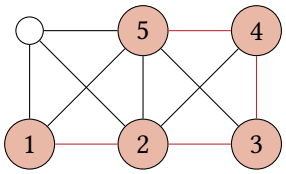

Abbildung 2.3: Pfad $P = (1, 2, 3, 4, 5)$ der Länge $l = 5$

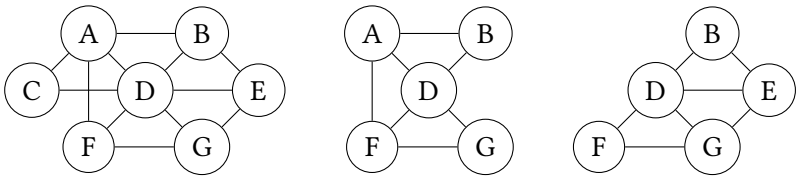

Abbildung 2.4: Graph G und seine induzierten Teilgraphen $G_{V'}$ und $G_{V''}$

Definition 2.7: *Ein* gerichteter Kreis $G = (V, A)$ *ist ein Graph mit* $n \geq 3$ *Knoten und den Kanten* $(v_i, v_{i+1}) \in A$ *für* $1 \leq i < n$ *und* $(v_n, v_1) \in A$*. Ein gerichteter Graph* $G = (V, A)$ *ist azyklisch, wenn er keinen gerichteten Kreis als Untergraphen enthält.*

Kreise sind so gesehen Pfade, die in allen Knoten beginnen und enden können.

Definition 2.8: *Sei* $T = (V, E)$ *ein ungerichteter Graph. Wir bezeichnen* T *als* Baum *genau dann, wenn für alle Knotenpaare* $u, v \in V$ *genau ein Pfad* $P = (u, \ldots, v)$ *existiert.*

Bäume haben die Eigenschaft, zusammenhängend zu sein. Das heißt, alle Knoten sind über einen Pfad verbunden. Bäume sind zusätzlich azyklisch.

Als nächstes definieren wir verschiedene Arten von Teilgraphen.

Definition 2.9: *Sei* $G = (V, E)$ *ein Graph. Ein Graph* $G' = (V', E')$ *ist ein Teilgraph von* G*, wenn* $V' \subseteq V$ *eine Teilmenge der Knotenmenge von* G *ist und* $E' \subseteq E$ *eine Teilmenge der Kantenmenge von* G *ist, wobei jede Kante in* E' *nur Knoten in* V' *verbindet.*

Bei einem einfachen Teilgraphen wählen wir eine Teilmenge von Knoten und Kanten aus, die wiederum einen Graphen bilden. Es können nur Kanten existieren, die auch Knoten haben. Jeder Graph G ist ein Teilgraph von sich selbst.

Definition 2.10: *Sei* $G = (V, E)$ *ein Graph. Für* $V' \subseteq V$ *ist ein Graph* $G_{V'} = (V', E')$ *ein* induzierter Teilgraph *von* G*, wenn* $G_{V'}$ *Teilgraph von* G *ist und alle Kanten aus* E *auch in* E' *vorhanden sind, wenn die Knoten in* V' *auftauchen. Das heißt,* $E' = E \cap (V' \times V')$*.*

Bei induzierten Teilgraphen achten wir darauf, dass wir die maximale Anzahl an Kanten des ursprünglichen Graphens übernehmen. Wenn wir eine Teilmenge an Knoten gewählt haben, übernehmen wir auch alle Kanten des ursprünglichen Graphens, die zu diesen Knoten gehören.

Definition 2.11: *Sei* $T = (V, E)$ *ein Baum und sei* $V' \subseteq V$ *eine Teilmenge von Knoten von* T*. Der induzierte Teilgraph* $T_{V'} = (V', E')$ *ist ein* induzierter Teilbaum*, wenn* $T_{V'}$ *ein Baum ist.*

Induzierte Teilbäume sind eine Spezialisierung von induzierten Teilgraphen. Wir fordern zusätzlich, dass der Teilgraph ein Baum sein muss. Das heißt, wenn wir einen induzierten Teilbaum $U(T)$ von einem Baum T bestimmen, dann muss dieser zusammenhängend sein, da es sich um einen Baum handelt.

Als nächstes definieren wir weitere Bestandteile und Eigenschaften von Graphen.

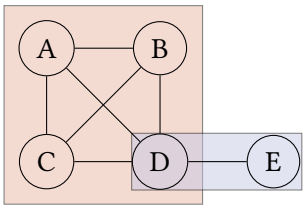

Abbildung 2.5: Clique C_1 in rot und C_2 in blau

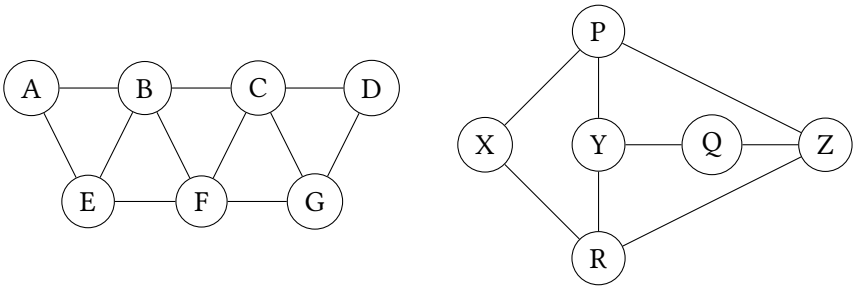

Abbildung 2.6: Zwei Beispielgraphen für simpliziale Knoten

Definition 2.12: *Sei $G = (V, E)$ ein ungerichteter Graph. Eine Teilmenge $U \subseteq V$ von Knoten ist eine* Clique*, wenn U einen vollständigen Teilgraphen induziert.*

Wir kombinieren die bereits definierten induzierten Teilgraphen und vollständigen Graphen, um Cliquen zu definieren.

Definition 2.13: *Ein Knoten $v \in V$ ist* simplizial *in G, wenn seine Nachbarschaft $N(v)$ eine Clique ist.*

In Abbildung 2.6 sind im linken Graphen die Knoten A und D simplizial. Im rechten Graphen gibt es keine simpliziale Knoten.

Definition 2.14: *Ein gerichteter Graph $G = (V, A)$ ist* transitiv *genau dann, wenn für alle Knoten $x, y, z \in V$ gilt: Wenn eine Kante $(x, y) \in A$ und eine Kante $(y, z) \in A$ existiert, dann gibt es auch eine Kante $(x, z) \in A$.*

Ein transitiver Graph hat also die Eigenschaft, dass jeder indirekte Weg zwischen zwei Knoten auch direkt durch eine einzelne Kante dargestellt wird. Zum Beispiel, wenn es von Knoten x über y einen Pfad zu z gibt, dann muss es auch direkt eine Kante $\{x, z\} \in E$ geben.

Definition 2.15: *Ein ungerichteter Graph $G = (V, E)$ ist* transitiv orientierbar*, wenn jeder Kante $\{a, b\} \in E$ eine Richtung zugeordnet werden kann, sodass der resultierende gerichtete Graph $G' = (V, A)$ die folgende Eigenschaft erfüllt:*

$$\textit{Für alle } a, b, c \in V\textit{:} \ (a, b) \in A \textit{ und } (b, c) \in A \Rightarrow (a, c) \in A$$

Dieser Graph wird auch als *Vergleichbarkeitsgraph* bezeichnet. Diese Eigenschaft kann auch als eine partielle Ordnung auf den Knoten und Kanten eines ungerichteten Graphens betrachtet werden.

Definition 2.16: *Sei $G = (V, A)$ ein gerichteter, azyklischer Graph. Eine* topologische Sortierung *$[v_1, v_2, \ldots, v_n]$ mit $v_i \in V$ für $1 \leq i|V|$ erfüllt die folgende Eigenschaft:*

$$\textit{Für alle i,j:} \ (v_i, v_j) \in A \rightarrow i < j$$

Eine topologische Sortierung wird auch als lineare Erweiterung *der transitiven Orientierung eines gerichteten Graphens bezeichnet.*

Wenn ein Graph G einen gerichteten Kreis hätte, könnten wir keine solche Sortierung finden. Wenn wir versuchen würden, einen gerichteten Kreis topologisch zu sortieren, dann können wir keinen Knoten finden, der einen höheren Index hat als alle anderen Knoten. Solange G azyklisch ist, können wir aber immer eine solche Sortierung finden. Dies ist beispielsweise durch eine modifizierte Tiefensuche in $\mathcal{O}(|V(G)| + |E(G)|)$ möglich.

Eine topologische Sortierung kann beispielsweise dadurch gefunden werden, indem wir immer wieder Knoten im Graphen finden, die keine eingehenden Kanten haben. Diese löschen wir dann mit allen ihren Kanten und suchen den nächsten Knoten. Solange der Graph azyklisch ist, können wir immer mindestens einen solchen Knoten finden.

2.2 Relationen

In diesem Abschnitt führen wir gängige Definitionen für Relationen ein.

Definition 2.17: *Eine* (totale) Ordnung *ist eine binäre Relation* $\leq \subseteq S \times S$ *auf einer Menge* S*, die die folgenden Eigenschaften erfüllt:*

- Reflexivität*: Für alle* $a \in S$ *gilt* $a \leq a$*.*
- Antisymmetrie*: Für alle* $a, b \in S$ *gilt, wenn* $a \leq b$ *und* $b \leq a$*, dann* $a = b$*.*
- Transitivität*: Für alle* $a, b, c \in S$ *gilt, wenn* $a \leq b$ *und* $b \leq c$*, dann* $a \leq c$*.*
- Totalität*: Für alle* $a, b \in S$ *gilt* $a \leq b$ *oder* $b \leq a$*.*

Bei einer totalen Ordnung sind alle Elemente miteinander vergleichbar. Das heißt, ein Element ist größer, kleiner oder gleich eines anderen Elementes. Die Menge der natürlichen Zahlen $\mathbb{N}$ mit der üblichen "kleiner-gleich"Relation $\leq$ ist beispielsweise eine Ordnung. Wir schreiben für eine totale Ordnung $\leq$ auch $\leq = [s_1, s_2, \ldots, s_n]$ oder $\leq = s_1 < s_2 < \cdots < s_n$.

Definition 2.18: *Eine* partielle Ordnung *ist eine binäre Relation* $\leq \subseteq P \times P$ *auf einer Menge* P*, die die folgenden Eigenschaften erfüllt:*

- Reflexivität*: Für alle* $a \in P$ *gilt* $a \leq a$*.*
- Antisymmetrie*: Für alle* $a, b \in P$ *gilt, wenn* $a \leq b$ *und* $b \leq a$*, dann* $a = b$*.*
- Transitivität*: Für alle* $a, b, c \in S$ *gilt, wenn* $a \leq b$ *und* $b \leq c$*, dann* $a \leq c$*.*

Der Unterschied zwischen einer totalen und einer partiellen Ordnung ist, dass Elemente in einer partiellen Ordnung nicht vergleichbar sein müssen. Die Menge der natürlichen Zahlen $\mathbb{N}$ mit der Relation der üblichen Teilbarkeit $/$ ist zum Beispiel eine partielle Ordnung. Die Zahl 1 steht beispielsweise in Relation zu jeder Zahl, weil sie alle natürlichen Zahlen teilt, aber 2 und 3 stehen nicht in Relation, weil keine die andere teilt.

Definition 2.19: *Eine* Äquivalenzrelation *ist eine binäre Relation* $\sim \subseteq A \times A$ *auf einer Menge* A*, die die folgenden Eigenschaften erfüllt:*

- Reflexivität*: Für alle* $a \in A$ *gilt* $a \sim a$*.*

Abbildung 2.7: Links ein offenes Intervall $< - >$ und ein geschlossenes Intervall $| - |$ und rechts der Schnitt der Intervalle $|- >$

- Symmetrie*: Für alle $a, b \in A$ gilt, wenn $a \sim b$, dann $b \sim a$.*
- Transitivität*: Für alle $a, b, c \in A$ gilt, wenn $a \sim b$ und $b \sim c$, dann $a \sim c$.*

Sei A eine Menge und $\sim$ eine Äquivalenzrelation auf A. Für einen Repräsentanten $a \in A$ ist die Äquivalenzklasse *von a bezüglich $\sim$ die Menge aller Elemente in A, die zu a äquivalent sind. Diese Äquivalenzklasse wird mit $[a] = \{b \in A \mid b \sim a\}$ bezeichnet.*

Eine Äquivalenzrelation partitioniert die Menge A in disjunkte Äquivalenzklassen, wobei alle Elemente einer Äquivalenzklasse zueinander in Relation stehen.

2.3 Intervallgraphen

In diesem Abschnitt beginnen wir damit, Intervallgraphen und die Spezialisierung der gerichteten Intervallgraphen einzuführen. Diese werden veranschaulicht und verschiedene Eigenschaften erörtert. Wir beginnen damit Intervalle zu definieren.

Definition 2.20: *Ein Intervall I ist eine zusammenhängende Teilmenge der reellen Zahlen für $a < b \in \mathbb{R}$. Ein offenes Intervall (a, b) enthält alle Zahlen x für die gilt $a < x < b$. Ein abgeschlossenes Intervall $[a, b]$ enthält alle reellen Zahlen x mit $a \leq x \leq b$. Der Schnitt zwischen zwei Intervallen I und J ist die Menge an Elementen, die in beiden Intervallen enthalten sind und wenn diese nicht leer ist, ist diese wiederum ein Intervall.*

Wir werden uns hier mit abgeschlossenen Intervallen auseinandersetzen. Der Schnitt zwischen zwei Intervallen ist die Überschneidung oder Überlappung in ihrem gemeinsamen Bereich. Wir unterscheiden zwischen Intervallen, die sich schneiden und Intervallen, die sich überlappen.

Definition 2.21: *Seien $I = [a, b]$ und $J = [a', b']$ zwei Intervalle und es gilt $a < a'$. Dann schneiden sich I und J, wenn $a' < b < b'$ gilt. Wenn $b' < b$ gilt, dann wird J von I überlappt.*

Wenn ein Intervall $I(v)$ ein anderes Intervall $I(w)$ überlappt, dann ist der Schnitt $I(v) \cap I(w) = I(w)$ wiederum das kleinere Intervall. Der Einfachheit halber werden aber alle Intervallen in Beispiel 2.8 mit unterschiedlichen Start- und Endpunkten dargestellt.

Wir definieren Intervallgraphen als Schnittgraphen von Intervallen.

Definition 2.22: *Ein* Intervallgraph *$G = (V, E)$ ist ein ungerichteter Graph, der isomorph zum Schnittgraphen einer Familie $\mathcal{I}$ von abgeschlossenen Intervallen auf der reellen Linie $\mathbb{R}$ ist. Wir nennen $\mathcal{I}$ die* Intervallrepräsentation *von G und $\mathcal{I}(v)$ ist jenes Intervall, dass einen Knoten v repräsentiert.*

Eine Intervallrepräsentation $\mathcal{I}$ korrespondiert zu einem Intervallgraphen G, aber ein Intervallgraph G kann mehrere Intervallrepräsentationen $\mathcal{I}$ haben. Wir betrachten in Abbildung 2.8 beispielsweise einen Intervallgraphen und zwei mögliche Intervallrepräsentationen. Wenn sich zwei Intervalle $I(v), I(w) \in \mathcal{I}$ in irgendeiner Reihenfolge überschneiden, sind die korrespon-

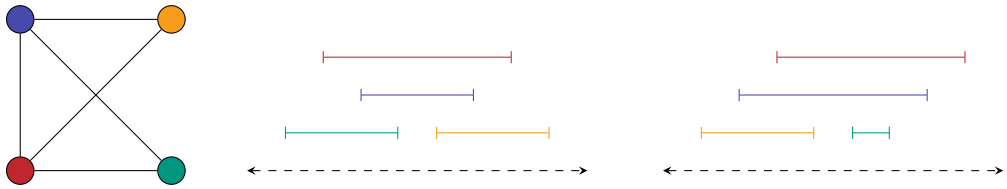

Abbildung 2.8: Intervallgraph G (links), Intervallrepräsentation $\mathcal{I}$ (Mitte) und Intervallrepräsentation $\mathcal{I}'$ (rechts)

dierenden Knoten $v, w \in V$ adjazent. Intervallrepräsentationen können sich beispielsweise darin unterscheiden, ob sich Intervalle überlappen oder schneiden. In beiden Fällen gibt es eine Kante im Graphen. Auch die Reihenfolge der Start- und Endpunkte von Intervallen kann variable sein.

Um bestimmen zu können, ob ein beliebiger ungerichteter Graph ein Intervallgraph ist, werden wir uns die folgende Eigenschaft zunutze machen.

Satz 2.23: *Ein Graph G ist ein Intervallgraph genau dann, wenn die maximalen Cliquen von G linear angeordnet werden können, sodass für jeden Knoten v gilt, dass die maximalen Cliquen mit v hintereinander auftreten. [FG65]*

Wir bezeichnen diese Anordnung als konsekutive Anordnung. Jeder Knoten muss als eine ununterbrochene Linie gezeichnet werden können, wie z. B. in Abbildung 2.8. Diese müssen sich mit allen Linien von adjazenten Knoten überschneiden. Daher muss es für jede maximale Clique $C = \{v_1, v_2, \ldots, v_k\}$ mindestens einen Punkt auf der reellen Linie geben an dem nur die korrespondierenden Intervalle $I(v_1), I(v_2), \ldots, I(v_k)$ von C auftreten. Diese Punkte, an denen nur die Intervalle einer Clique auftreten, definieren wir wie folgt:

Definition 2.24: *Sei C eine Menge von paarweise überschneidenden Intervallen auf der reellen Linie $\mathcal{R}$. Der am weitesten linke Punkt auf der reellen Linie, an dem sich alle Intervalle in C überschneiden, wird als* Cliquenpunkt *bezeichnet.*

Um die maximalen Cliquen eines Intervallgraphens zu erkennen, wollen wir simpliziale Knoten nutzen, um zu überprüfen, ob ein Graph ein Intervallgraph ist. Wir werden in Abschnitt 3 einen Algorithmus kennenlernen, der für einen Graphen $G = (V, E)$ eine Ordnung $\sigma = [v_1, v_2, \ldots, v_n]$ berechnet mit $n = |V|$. Wenn für alle Knoten $v_i \in \sigma$ mit $1 \leq i \leq n$ gilt, dass v_i ein simplizialer Knoten vom induzierten Teilgraphen $G_{\{v_i, \ldots, v_n\}}$ ist, dann ist G ein Intervallgraph. Die Idee dahinter ist wie folgt.

Wir stellen uns nun vor, dass wir den Knoten v_i als Intervall $I(v_i)$ zeichnen wollen. Wir haben bereits die Knoten v_{i+1} bis v_n gezeichnet und die Intervallrepräsentation $\mathcal{I} = \{I(v_{i+1}, \ldots, I(v_n)\}$ existiert. Alle Nachbarn von v_i bilden eine Clique. Deshalb muss es einen Cliquenpunkt in $\mathcal{I}$ geben. Wir können also $I(v_i)$ an diesen Punkt zeichnen und würden wieder eine Intervallrepräsentation erhalten.

Wir werden uns mit gerichteten Intervallgraphen auseinandersetzen. Diese sind eine Form von gemischten Intervallgraphen. Anschaulich gesprochen haben sie eine ungerichtete Kante, wenn ein Intervall das andere überlappt. Wenn Intervalle sich schneiden, dann hat ein gerichteter Intervallgraph eine gerichtete Kante vom Knoten des linken Intervalls zum Knoten des rechten Intervalls. Formell definieren wir einen gerichteten Intervallgraphen wie folgt:

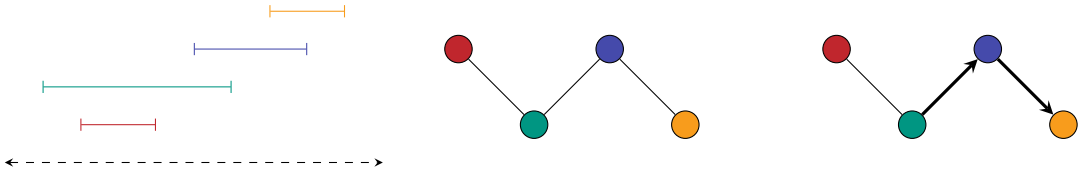

Abbildung 2.9: Intervallrepräsentation $\mathcal{I}$ (links), Intervallgraph (Mitte) und gerichteter Intervallgraph (rechts)

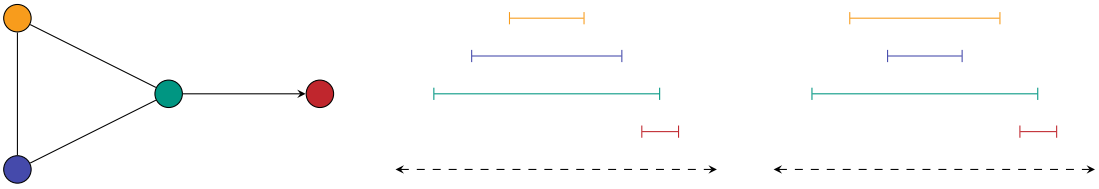

Abbildung 2.10: Gerichteter Intervallgraph G (links), gerichtete Intervallrepräsentation $\mathcal{I}$ (Mitte) und gerichtete Intervallrepräsentation $\mathcal{I}'$ (rechts)

Definition 2.25: *Für eine Familie $\mathcal{I}$ von abgeschlossenen Intervallen auf der reellen Linie $\mathbb{R}$ ist der dazugehörige* gerichtete Intervallgraph *$G = (V, E, A)$ ein gemischter Graph. Für jedes Knotenpaar $u, v \in V$ mit $I(u) = [l_u, r_u], I(v) = [l_v, r_v]$, wobei $l_u < l_v$ gilt, gilt genau eine der folgenden Bedienungen:*

1. *$I(u)$ und $I(v)$ sind unabhängig ($r_u < l_v$) genau dann, wenn u und v unabhängig in G sind.*
2. *$I(u)$ und $I(v)$ schneiden sich ($l_u < l_v < r_u < r_v$) genau dann, wenn es eine gerichtete Kante $(u, v) \in A$ gibt.*
3. *$I(v)$ überlappt $I(u)$ ($r_v < r_u$) genau dann, wenn es eine ungerichtete Kante $\{u, v\} \in E$ gibt.*

Ein gemischter Graph $G = (V, E, A)$ ist ein gerichteter Intervallgraph, wenn es eine Familie $\mathcal{I}$ von Intervallen gibt, sodass G der dazugehörige Graph ist.

Wir bezeichnen $\mathcal{I}$ als eine *gerichtete Intervallrepräsentation* von G. Ein gerichteter Intervallgraph G kann mehrere gerichtete Intervallrepräsentationen $\mathcal{I}$ besitzen. Wir betrachten beispielsweise die Abbildung 2.10. Intervalle, die sich überlappen, können ausgetauscht werden. Wir könnten auch leicht die Endpunkte von Intervallen verschieben und würden weitere Intervallrepräsentation erhalten.

2.4 PQ-Bäume

Ein PQ-Baum T ist eine Datenstruktur, die verwendet wird, um Permutationen über eine Menge von Elementen zu repräsentieren.

Definition 2.26: *Sei $S = \{a_1, a_2 \ldots, a_n\}$ eine Menge mit n Elementen. Eine* Permutation *ohne Wiederholung ist eine bijektive Abbildung $p : S \longrightarrow S$, bei der jedes Element a_i von S auf genau ein anderes Element a_j von S abgebildet wird. Wir bezeichnen $P(S)$ als die* Menge aller Permutationen *von S.*

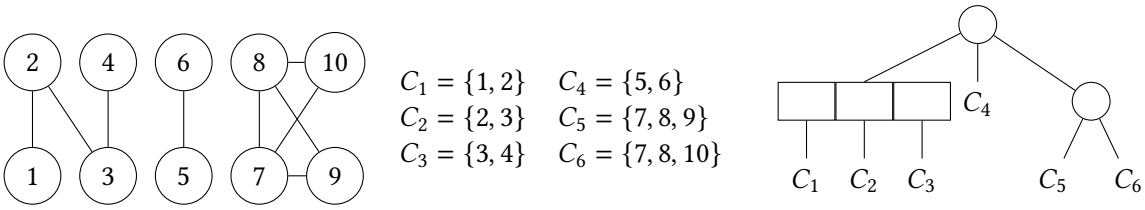

Abbildung 2.11: Intervallgraph G (links), maximale Cliquen von G (Mitte) und PQ-Baum T von G (rechts)

Eine Permutation ist also eine Anordnung oder Reihenfolge der Elemente einer Menge, bei der kein Element wiederholt wird und jedes Element genau einmal vorkommt. Durch die Bijektivität wird gewährleistet, dass jede Position in der Permutation eindeutig ist und keine Position leer bleibt. Beispielsweise hat die Menge $1, 2, 3$ sechs Permutationen:

$$(1, 2, 3), (1, 3, 2), (2, 1, 3), (2, 3, 1), (3, 1, 2), (3, 2, 1)$$

Die Menge $P(S)$ enthält alle möglichen Anordnungen der Elemente in S.

Diese Elemente einer Menge wollen wir nun als Blätter eines Baumes darstellen. Die Rotation des Baumes ist dann die Permutation dieser Elemente. Diese PQ-Bäume haben die folgende Struktur:

Definition 2.27: *Für eine endliche Menge Blätter B ist ein* PQ-Baum $T(B) = (V = P \cup Q \cup B, E, r)$ *definiert als ein geordneter Baum mit Wurzel $r \in V$. Die Knotenmenge $V = P \cup Q \cup B$ ist die Vereinigung der Blätter B und der inneren P- und Q-Knoten.* P-Knoten *haben mindestens zwei Kinder und werden als Kreis gezeichnet.* Q-Knoten *haben mindestens drei Kinder und werden als Rechteck gezeichnet. Jedes Element $b \in B$ tritt genau einmal als Blatt auf. Die* untere Grenze *$F(T)$ eines PQ-Baums ist die Ordnung der Blätter von links nach rechts.*

Wir haben zunächst einen ungerichteten Graphen gegeben und wollen mit einem PQ-Baum erkennen, ob es sich um einen Intervallgraphen handelt. Wir wählen als Menge B die Menge aller maximalen Cliquen eines Graphens G. Unser Ziel ist es, dass die untere Grenze eines PQ-Baumes eine konsekutive Anordnung von B ist. Damit können wir dann erkennen, ob es sich um einen Intervallgraphen handelt. In Abbildung 2.11 sehen wir beispielsweise einen PQ-Baum T, der über den maximalen Cliquen eines Intervallgraphen G aufgebaut wurde. Es kann leicht überprüft werden, dass es sich hierbei tatsächlich um eine konsekutive Anordnung handelt.

Wir wissen bereits, dass ein Intervallgraph G in der Allgemeinheit nicht nur eine Intervallrepräsentation $\mathcal{I}$ hat. Ebenso hat ein Intervallgraph nicht nur einen PQ-Baum. Was das genau bedeutet, wollen wir uns nun ansehen. Zunächst einmal definieren wir, wann PQ-Bäume äquivalent sind.

Definition 2.28: *Sei T und T' zwei PQ-Bäume. Sie sind äquivalent, wenn eine beliebige Anzahl von Transformationen angewendet werden kann, um einen Baum aus dem anderen zu erzeugen. Es gibt zwei Äquivalenztransformationen:*

1. *Die beliebige Anordnung der Kinder eines P-Knotens.*
2. *Die Umkehrung der Reihenfolge der Kinder eines Q-Knotens.*

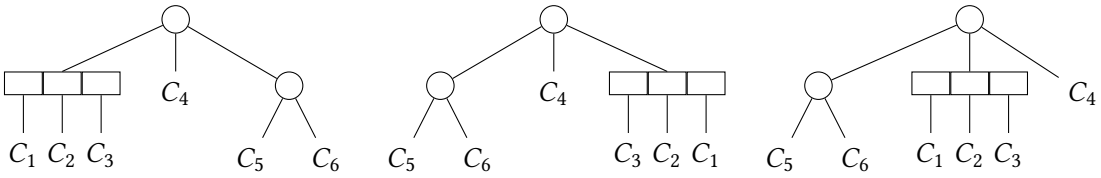

Abbildung 2.12: Äquivalente PQ-Bäume T, T', T''

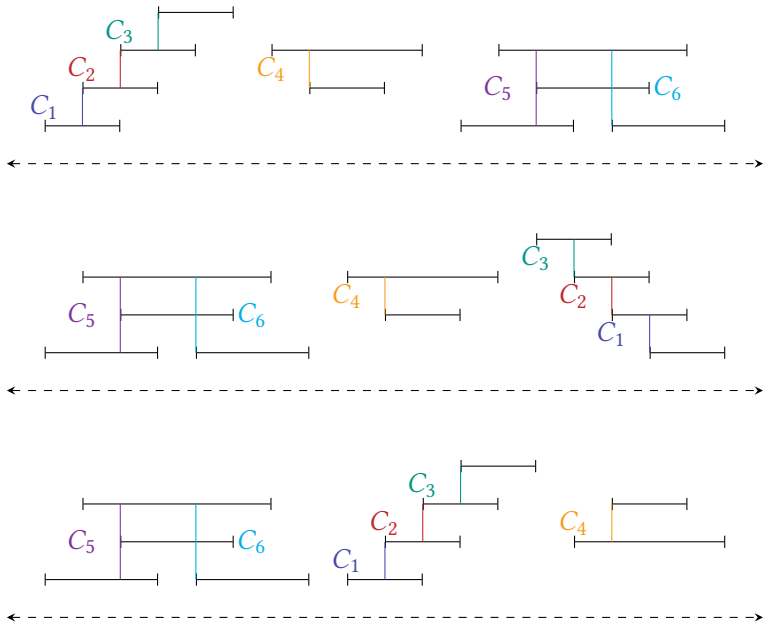

Abbildung 2.13: Intervallrepräsentationen $\mathcal{I}, \mathcal{I}', \mathcal{I}''$

Die Ordnung innerhalb der umgehängten Teilbäume bleibt unverändert.

Seien T und T' nun zwei äquivalente PQ-Bäume. Dann bezeichnen wir T auch als Rotation *von T' und umgekehrt.*

Die unteren Grenzen $F(T)$ und $F(T')$ von zwei äquivalenten PQ-Bäumen T und T' sind Permutationen von B. Ein PQ-Baum T bildet eine Äquivalenzklasse $[T]$. Wenn wir diese bestimmen, erhalten wir auch alle möglichen Permutationen $P(B)$, die zu T gehören.

In Abbildung 2.12 sehen wir beispielsweise äquivalente PQ-Bäume T, T' und T'', die zum Graphen G aus Abbildung 2.11 gehören. In Abbildung 2.13 sehen wir dann die korrespondierenden Intervallrepräsentationen $\mathcal{I}, \mathcal{I}'$ und $\mathcal{I}''$. Die Cliquen wurden als Punkte eingezeichnet, an denen sich alle Intervalle der Clique schneiden. Wir können leicht überprüfen, dass die unteren Grenzen $F(T), F(T')$ und $F(T'')$ der PQ-Bäume konsekutive Anordnungen sind und wir tatsächlich eine Intervallrepräsentation finden können, die ihr entspricht.

Als nächstes wollen wir definieren, was es heißt, dass ein PQ-Baum T zu einem Intervallgraph G gehört.

Definition 2.29: *Sei $G = (V, E)$ ein Intervallgraph und B die Menge aller maximalen Cliquen von G. Wir bezeichnen $T(B)$ als einen PQ-Baum von G genau dann, wenn für alle äquivalenten PQ-Bäume $T' \in [T(B)]$ eine Intervallrepräsentation $\mathcal{I}$ von G existiert und wenn für alle Intervallrepräsentationen $\mathcal{I}$ ein PQ-Baum $T' \in [T(B)]$ existiert. Die konsekutive Anordnung von $\mathcal{I}$ muss gleich der unteren Grenze $F(T')$ sein.*

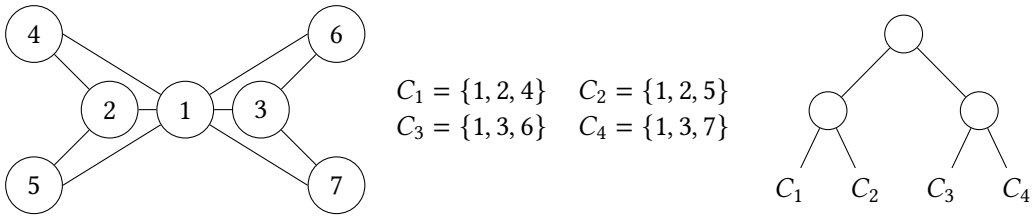

Abbildung 2.14: Intervallgraph G (links), maximale Cliquen von G (Mitte) und PQ-Baum T von G (rechts)

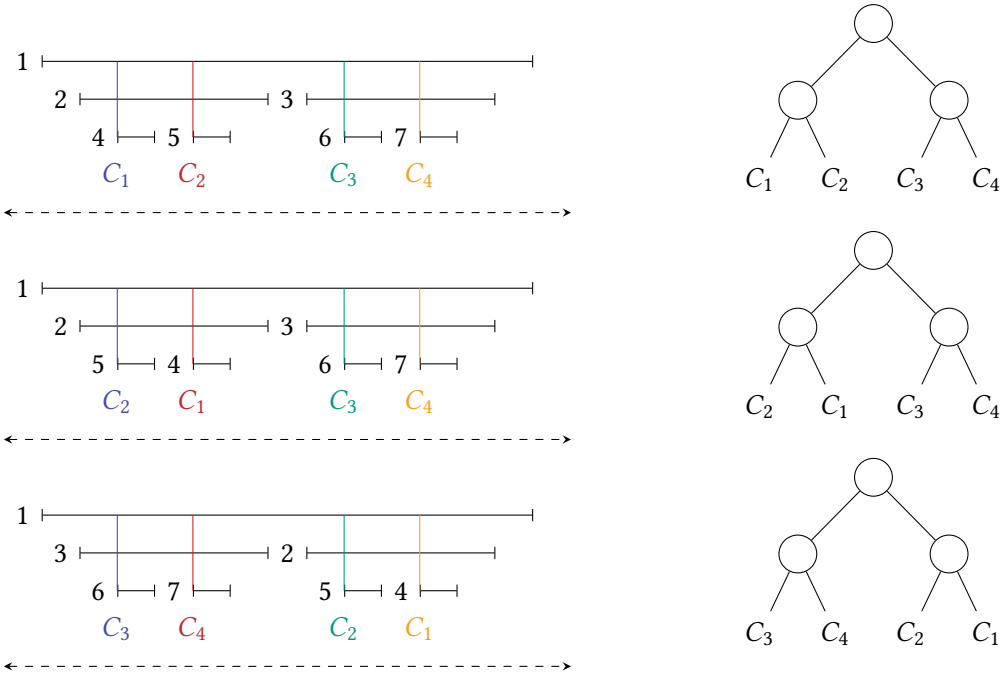

Abbildung 2.15: Intervallrepräsentationen I, I', I'' (links) und korrespondierende PQ-Bäume T, T', T'' (rechts)

Wir sagen auch, dass $T(B)$ dann mit einer Intervallrepräsentation $\mathcal{I}$ übereinstimmt.

Es genügt nicht nur, dass ein PQ-Baum T eine untere Grenze mit konsekutiver Anordnung hat. Wir könnten immer einen PQ-Baum aus nur einem P-Knoten konstruieren und alle maximalen Cliquen als Kinder in einer konsekutiven Anordnung anhängen. Stattdessen müssen wir in der Lage sein, alle Intervallrepräsentationen $\mathcal{I}$ aus T abzulesen, indem wir T rotieren.

Dabei ist es wichtig, dass die konsekutive Anordnung der Knoten gewährleistet bleibt. Wenn wir beispielsweise einen Teilbaum T' von einem PQ-Baum T betrachten, dann muss die untere Grenze dieses Teilbaums ebenfalls die konsekutive Anordnung der Knoten gewährleisten, da T' ebenfalls ein PQ-Baum ist. Betrachten wir beispielsweise den Graphen in Abbildung 2.14. Der Graph G besteht aus 4 Cliquen und der PQ-Baum T aus 3 P-Knoten. Die Cliquen C_1, C_2 umfassen beide die Knoten $C_1 \cap C_2 = \{1, 2\}$. Wenn wir den linken P-Knoten rotieren, dann bleibt die Reihenfolge dieser Knoten unverändert. Das sehen wir in Abbildung 2.15. Wenn wir dann den zentralen P-Knoten rotieren, bleibt nur das Intervall $I(1)$ unverändert. Wir können erkennen, dass die Intervalle $I(2), I(5), I(4)$ zum linken P-Knoten korrespondieren und die

Intervalle $I(3), I(6), I(7)$ zum rechten P-Knoten. Insgesamt können wir feststellen, dass es einen Zusammenhang zwischen den Cliquen eines (Teil)-Baumes und den Knoten geben muss, den dieser Baum repräsentiert. Diese Eigenschaft wollen wir nutzen, um PQ-Bäume zu modifizieren.

Jeder Intervallgraph G hat mindestens einen PQ-Baum T. Wir können einen solchen PQ-Baum in $\mathcal{O}(|V(G)| + |E(G)|)$ berechnen [BL76]. Die Konstruktion ist vergleichsweise aufwendig und stattdessen werden wir uns mit sogenannten modifizierten PQ-Bäumen beschäftigen. Sie werden so genannt, weil sie den Knoten eines PQ-Baums zusätzliche Informationen über die Knoten des korrespondierenden Intervallgraphen G zuordnen. Diese Bäume werden wir dann tatsächlich verwenden, um im späteren Algorithmus gerichtete Intervallgraphen zu erkennen.

Definition 2.30: *Sei $G = (V, E)$ ein Intervallgraph und sei $T = (V', E', r)$ ein PQ-Baum von G. Ein* modifizierter PQ-Baum *(MPQ-Baum) ist definiert als ein Tupel $T* = (V', E', r, \mathcal{V}, \mathcal{S})$ mit den folgenden Eigenschaften:*

- *Für jeden Knoten $v' \in V'$ von $T*$ gibt es eine zugeordnete Teilmenge von Knoten $V_{v'} \subseteq V$ des Graphens G. Die Familie dieser Teilmengen wird als $\mathcal{V} = \{V_{v'} \mid v' \in V'\}$ bezeichnet.*
- *Für jeden Q-Knoten $q \in V'$ von $T*$ und seinen Kindern $q_1, \ldots, q_k$ gibt es zu jedem Kind q_i mit $1 \leq i \leq k$ eine zugeordnete Menge S_{q_i}, die wir als Sektion von q bezeichnen. Die Menge aller solchen aller solchen Zuordnungen von Q-Knoten zu ihren Sektionen wird als $\mathcal{S} = \{(q, S_{q_1}, S_{q_2}, \ldots, S_{q_k}) \mid q \in V' \text{ und } q_1, \ldots, q_k \text{ Kinder von } q\}$ bezeichnet.*
- *Für alle Q-Knoten $q \in V'$ ist $V_q = \bigcup_{i=1}^{k} S_{q_i}$ die Vereinigung aller Sektionen.*

MPQ-Bäume übernehmen die Knotenstruktur von PQ-Bäumen und speichern zusätzlich zwei Mengen $\mathcal{V}$ und $\mathcal{S}$. Ziel ist es, Informationen über die maximalen Cliquen von G in $T*$ zu speichern. Jeder Knoten hat eine Menge $V_{v'}$, die Knoten des Intervallgraphens speichern. Q-Knoten sind zusätzlich in Sektionen unterteilt und haben für jedes ihrer Kinder eine Menge S_{q_i}. Ein Intervallgraph kann mehrere äquivalente PQ-Bäume haben und ebenso kann er mehrere äquivalente MPQ-Bäume haben. Wir betrachten nun genauer, was in $V_{v'}$ und S_{q_i} gespeichert wird.

Definition 2.31: *Sei $G = (V, E)$ ein Intervallgraph und $T = (V', E', r, \mathcal{V}, \mathcal{S})$ ein MPQ-Baum von G. Für alle P-Knoten p enthält die Menge V_p alle Knoten, die in allen maximalen Cliquen enthalten sind, die durch den von p induzierten Teilbaum $U(p)$ repräsentiert wird. Seien $C_1, \ldots, C_p$ alle Cliquen, die sich auf einem Pfad von einem Blatt b zu einem P-Knoten p befinden. Dann gilt $V_p = \bigcap_{i=1}^{p} C_i$.*

Sei $q_1, \ldots, q_k$ die Kinder eines Q-Knotens q und sei $U(q_i)$ der von q_i induzierte Teilbaum mit $1 \leq i \leq k$. Eine Sektion S_{q_i} enthält alle Knoten der maximalen Cliquen, die von $U(q_i)$ repräsentiert werden.

Für alle Blätter b enthält die Menge V_b nur Knoten, die sich in einer einzigen Clique befinden.

In Abbildung 2.16 sehen wir beispielsweise den gleichen Graphen wie in Abbildung 2.11 und diesmal einen korrespondierenden MPQ-Baum $T*$. Für alle P-Knoten p und für alle Blätter b geben wir die Mengen V_p und V_b explizit an, in dem wir sie in die Knoten schreiben. Für den obersten P-Knoten p_1 ist zum Beispiel $V_{p_1} = \emptyset$, da der Graph nicht zusammenhängend ist und daher die Cliquen keine Knoten gemeinsam haben. Für alle Q-Knoten q mit Kindern $q_1, \ldots, q_k$ geben wir nur die Sektionen $S_{q_1}, \ldots, S_{q_k}$ in Abbildungen an.

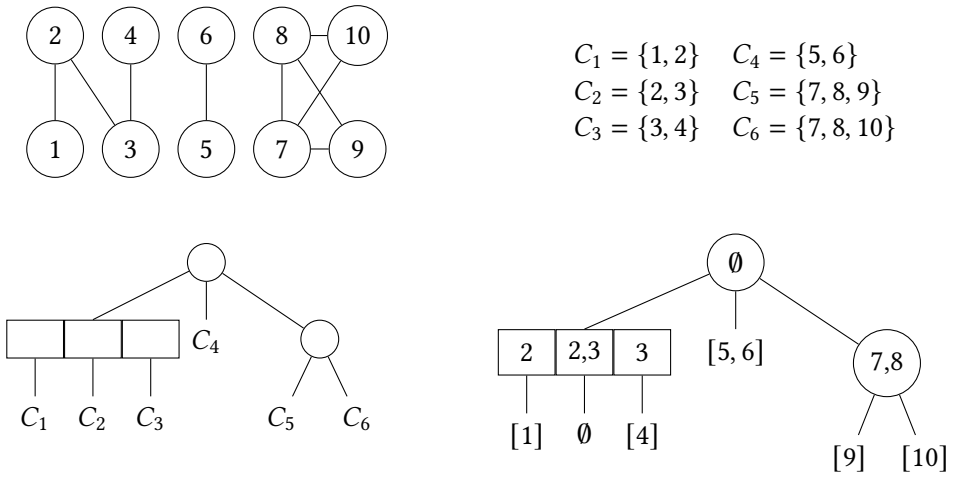

Abbildung 2.16: Graph G (links oben), maximale Cliquen von G (rechts oben), PQ-Baum T von G (links unten) und MPQ-Baum $T*$ (rechts unten)

Wenn wir einen Intervallgraphen G und einen zugehörigen PQ-Baum T haben, dann können wir daraus den MPQ-Baum $T*$ in $\mathcal{O}(|V(G)| + |E(G)|)$ berechnen [KM89].

Im Vergleich zu PQ-Bäumen besitzen MPQ-Bäume eine wichtige Eigenschaft zur Erkennung von Intervallgraphen:

Satz 2.32: *Sei $G = (V, E)$ ein Intervallgraph und $T = (V', E', r, \mathcal{V}, \mathcal{S})$ ein dazugehöriger MPQ-Baum. Außerdem sei $P = (r, \ldots, b)$ ein Pfad in T von der Wurzel r bis zu einem Blatt $b \in B$ und seien $V_1, \ldots, V_p$ die Mengen dieser Knoten. Jede maximale Clique C von G korrespondiert zu einem solchen Pfad mit $C = \bigcup_{i=1}^{p} V_i$ und jeder Knoten $v \in V$ wird in einer Menge X so dicht an der Wurzel wie möglich gespeichert. [KM89]*

Das bedeutet u.a., dass adjazente Knoten in G auf einem Pfad in T liegen. Diese Pfade werden wir uns später zunutze machen, um iterativ MPQ-Bäume zu konstruieren. Wenn wir die richtige Reihenfolge an Knoten V eines Graphen $G = (V, E)$ wählen, dann können wir nach und nach die Knoten $v \in V$ zu einem MPQ-Baum $T*$ hinzufügen.

Wie genau wir diese Bäume konstruieren, werden wir uns in Kapitel 3.2 ansehen. Hier wollen wir erstmals Mengen für diese Pfade definieren, die wir später benötigen werden.

Definition 2.33: *Sei $G = (V, E)$ ein Intervallgraph und $T = (V', E', r, \mathcal{V}, \mathcal{S})$ ein MPQ-Baum von G. Außerdem sei $v \in V$ ein Knoten von G und $v' \in V'$ ein Knoten von T. Wir sagen, dass v* über *v' ist, wenn v in einer Menge gespeichert wird, die sich auf einem Pfad $P = (v', \ldots, r)$ zur Wurzel befindet. Sei $v'' \in V'$ ein beliebiges Kind von v'. Analog zu über sagen wir, dass v* unter *v' ist, wenn es sich auf einem Pfad $P' = (v'', \ldots, b)$ zu einem Blatt b befindet.*

Dann definieren wir die Menge $A^T_{v'}$ als Menge aller Knoten aus G, die sich in oder über v' befinden. Analog ist $B^T_{v'}$ die Menge aller Knoten, die sich unterhalb von v' befinden.

Nach dieser Definition können wir die untere Grenze eines MPQ-Baums anders ausdrücken. Die untere Grenze eines MPQ-Baums ist die Ordnung der Mengen $A^T_{b_1}, A^T_{b_2}, \ldots, A^T_{b_k}$ für alle Blätter $b_1, b_2, \ldots, b_k \in B$ von T.

Was sich in den Mengen eines Q-Knotens befindet, wurde bisher noch recht allgemein gehalten. Es wurde zusätzlich noch in [KM89] einige Aussagen darüber gezeigt.

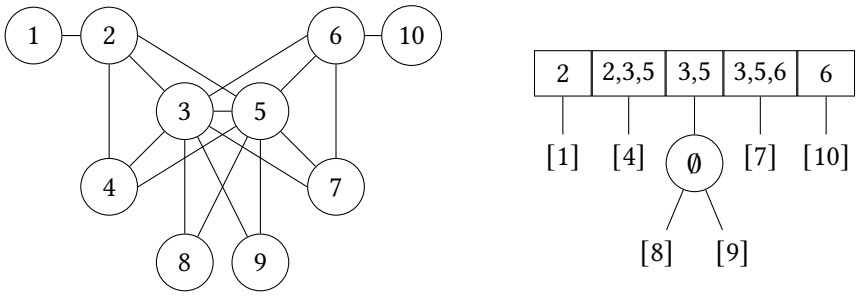

Abbildung 2.17: Intervallgraph G und ein MPQ-Baum $T*$ mit einem Q-Knoten als Wurzel

Satz 2.34: *Sei $G = (V, E)$ ein Intervallgraph und $T = (V', E', r, \mathcal{V}, \mathcal{S})$ ein MPQ-Baum von G. Außerdem sei q ein Q-Knoten von T, sei $S_1, \ldots, S_k$ die Sektionen von q und $B^T_{S_i} \subseteq V$ die Menge der Knoten, die sich unter S_i in T befinden mit $1 \leq i \leq k$. Dann gelten die folgenden Aussagen:*

1. $S_{i-1} \cap S_i \neq \emptyset$ *für* $i = 2, \ldots, k$
2. $S_1 \subseteq S_2$ *und* $S_k \subseteq S_{k-1}$
3. $B^T_{S_1} \neq \emptyset$ *und* $B^T_{S_k} \neq \emptyset$
4. $(S_i \cap S_{i+1}) \setminus S_1 \neq \emptyset$ *und* $(S_{i-1} \cap S_i) \setminus S_k \neq \emptyset$ *für* $i = 2, \ldots, k-1$

Anschaulich gesprochen bedeuten diese Aussagen das Folgende. Benachbarte Sektionen S_{i-1} und S_i müssen mindestens einen gemeinsamen Knoten speichern für $i = 2, \ldots, k$. Die erste Sektion S_1 ist immer eine Teilmenge der zweiten S_2 und analog ist die vorletzte Sektion eine Teilmenge der letzten. Die Mengen der Knoten $B^T_{S_1}$ und $B^T_{S_k}$, die sich unter der ersten Sektion S_1 und der letzten Sektion S_k befinden, können nicht leer sein. Die Mengen der Sektionen dazwischen können leer sein wie zum Beispiel in Abbildung 2.16. Der Schnitt zweier benachbarter Sektionen S_i und S_{i+1} oder S_{i-1} ohne die Elemente der ersten oder letzten Sektion kann nie leer sein. Das heißt, die Sektionen zwischen dem ersten und letzten müssen Knoten besitzen, die diese nicht enthalten.

2.5 Posets

Wir werden sogenannte partiell geordnete Mengen verwenden. Anschaulich gesprochen ist die Menge teilweise sortiert, weil nicht jedes Element miteinander verglichen werden kann. Sie sind eine Generalisierung von total geordneten Mengen, in welchen jedes Element mit jedem anderen Element verglichen werden kann und es immer eindeutig ist, wer Vorgänger und Nachfolger ist.

Definition 2.35: *Sei X eine Menge und $\leq$ eine partielle Ordnung auf X. Dann ist das Tupel $P = (X, \leq)$ eine* partiell geordnete Menge *oder auch* Poset *genannt (engl. partially ordered set). Ein Poset $P = (X, \leq)$ ist* total, *wenn $\leq$ eine totale Ordnung ist.*

Ein totales Poset können wir uns so vorstellen, dass alle Elemente vom Kleinsten bis zum Größten in einer Spalte sortiert werden können. Das heißt für je zwei Elemente $a, b \in X$ gilt entweder $a \leq b$ oder $b \leq a$. Wenn das Poset nicht total ist, dann können wir uns vorstellen, dass es mehrere Spalten gibt und die Elemente sich auf diese aufteilen. Wie wir diese visualisieren

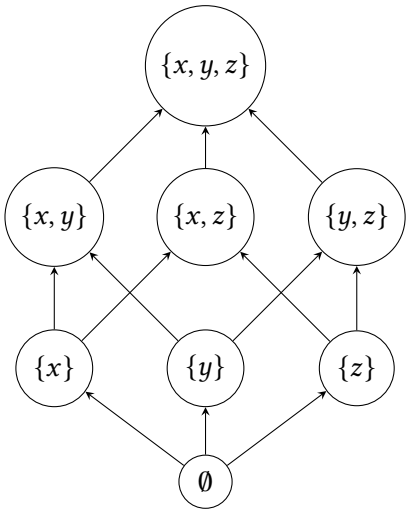

Abbildung 2.18: Hasse Diagramm der Potenzmenge von $\{x, y, z\}$

können, wollen wir uns jetzt anschauen. Es gibt verschiedene Arten und Weisen, aber die häufigste ist das sogenannte Hasse Diagramm. Dieses verwendet eine reduzierte Darstellung aller Relationen zwischen den Elementen. Wir definieren diese wie folgt:

Definition 2.36: *Ein* Hasse Diagramm $G = (V, A)$ *ist ein gerichteter Graph eines Posets* $P = (X, \leq)$. *Jeder Knoten* $v \in V$ *repräsentiert genau ein Element* $x \in X$ *des Posets. Es gibt eine gerichtete Kante* $(u, v) \in A$, *wenn* $u \leq v$ *gilt und kein* $w \in X \setminus \{u\}$ *existiert, sodass* $u \leq w \leq v$ *gilt.*

Das Hasse Diagramm wird von unten nach oben gezeichnet. Die Richtung impliziert die weggelassenen transitiven Kanten $(u, v) \in A$, für die $u \leq v$ gilt, aber ein $w \in X \setminus u$ existiert, sodass $u \leq w \leq v$. Diese reduzierte Darstellung wird auch *transitive Reduktion* genannt. In Abbildung 2.18 sehen wir beispielsweise ein Hasse Diagramm der Potenzmenge von $\{x, y, z\}$. Die partielle Ordnung dieses Posets ist die Teilmengenbeziehung zwischen den Elementen. Hier sehen wir auch gut, wie die partielle Ordnung die Elemente in mehrere Spalten aufteilt. Die Elemente $\{y\}$ und $\{x, z\}$ haben z. B. keine Beziehung, da keiner der beiden eine Teilmenge des anderen sind und daher können sie auch nicht in einer Spalte liegen.

Diese Art der Darstellung ist zur Visualisierung am besten geeignet, weil die Darstellung nicht durch zu viele Kanten überflutet wird. Für unseren Algorithmus sind die transitiven Relationen bzw. die transitiven Kanten notwendig. Wir werden Posets daher als transitive, gerichtete und azyklische Graphen betrachten. In Abbildung 2.19 sehen wir beispielsweise den Graphen mit allen transitiven Kanten. Wenn wir die gerichteten Kanten durch ungerichtete ersetzen, dann würden wir einen Vergleichbarkeitsgraphen erhalten.

Als Nächstes wollen wir uns eine Eigenschaft von Posets anschauen, die sogenannte Dimension. Dafür verwenden wir lineare Erweiterungen des Posets. Diese erhalten wir durch eine topologische Sortierung der Elemente, was wiederum eine totale Ordnung auf den Elementen darstellt. Obwohl eine partielle Ordnung im Allgemeinen nicht total ist, können wir die Elemente trotzdem in eine solche Ordnung bringen und die durch die partielle Ordnung vorgegebenen Relationen erhalten. Beispielsweise haben wir drei Elemente $X = \{a, b, c\}$ und es gilt $a < b$ und $a < c$. Die totale Ordnung $a < b < c$ erfüllt ebenfalls die Bedingungen der partiellen Ordnung. Wir definieren die Dimension dann wie folgt:

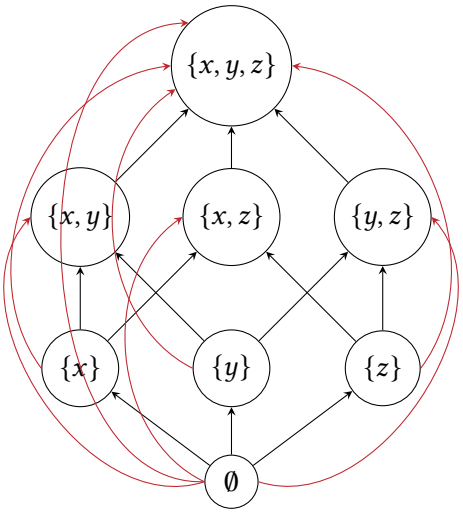

Abbildung 2.19: Hasse Diagramm der Potenzmenge von $\{x, y, z\}$ mit allen transitiven Kanten

Definition 2.37: *Sei $P = (X, \leq)$ ein Poset. Dann ist $\mathcal{L}(P)$ die Menge aller linearen Erweiterungen von P. Jede Teilmenge $\mathcal{L} \subseteq \mathcal{L}(P)$, die $\bigcap_{L \in \mathcal{L}} L = P$ erfüllt, heißt* Realizer *von P mit Größe $|\mathcal{L}|$. Wir definieren die Dimension $dim(P)$ von P als die kleinstmögliche Größe eines Realizers für P. Wir nennen diesen auch* minimaler Realizer *für P.*

Der Schnitt von zwei verschiedenen totalen Ordnungen ist eine partielle Ordnung. Wir betrachten ein Beispiel:

Beispiel 2.38: *Sei $P = (X, \leq)$ ein Poset mit $X = \{a, b, c, d\}$ und $\leq = \{(a, b), (a, c), (a, d), (b, d), (c, d)\}$. Dann gibt es zwei lineare Erweiterungen $L_1 = \{(a, b), (a, c), (a, d), (b, c), (b, d), (c, d)\}$ und $L_2 = \{(a, b), (a, c), (a, d), (b, d), (c, d)\}$ oder auch:*

$$L_1 = a < b < c < d \tag{2.3}$$

$$L_2 = a < c < b < d \tag{2.4}$$

Der Schnitt von L_1, L_2 ist der Schnitt der Mengen $L_1 \cap L_2 = P$.

Die Menge $\mathcal{L}(P)$ ist immer selbst ein Realizer von P. Wir können die Dimension auch wie folgt betrachten. Wenn ein Poset P die Dimension d hat, dann ist die partielle Ordnung $\leq$ der Schnitt aus d linearen Erweiterungen. In Abbildung 2.20 sehen wir beispielsweise ein Hasse Diagramm eines Posets P und zwei lineare Erweiterungen L_1, L_2. Wir können nun überprüfen, dass der Schnitt aus L_1, L_2 tatsächlich P ergibt. Das Hasse Diagramm und die Erweiterungen sind beide nach oben gerichtet. Die transitiven Kanten sind impliziert. Beispielsweise gilt in L_1 $a \leq x$, aber nicht in L_2. Daher stehen a und x in P nicht in Relation. Dafür gilt z. B. $a \leq b$ in L_1, L_2 und damit auch P.

Wir werden uns insbesondere mit zweidimensionalen Posets beschäftigen. Wir werden im Rahmen des Algorithmus ein Poset als Graphen konstruieren und dann überprüfen, ob dieses zweidimensional ist.

Satz 2.39: *Sei G_P der Vergleichbarkeitsgraph eines Posets P. Dann gilt $dim(P) \leq 2$ genau dann, wenn der Komplementgraph $\overline{G_P}$ ein Vergleichbarkeitsgraph ist [Gol04]*

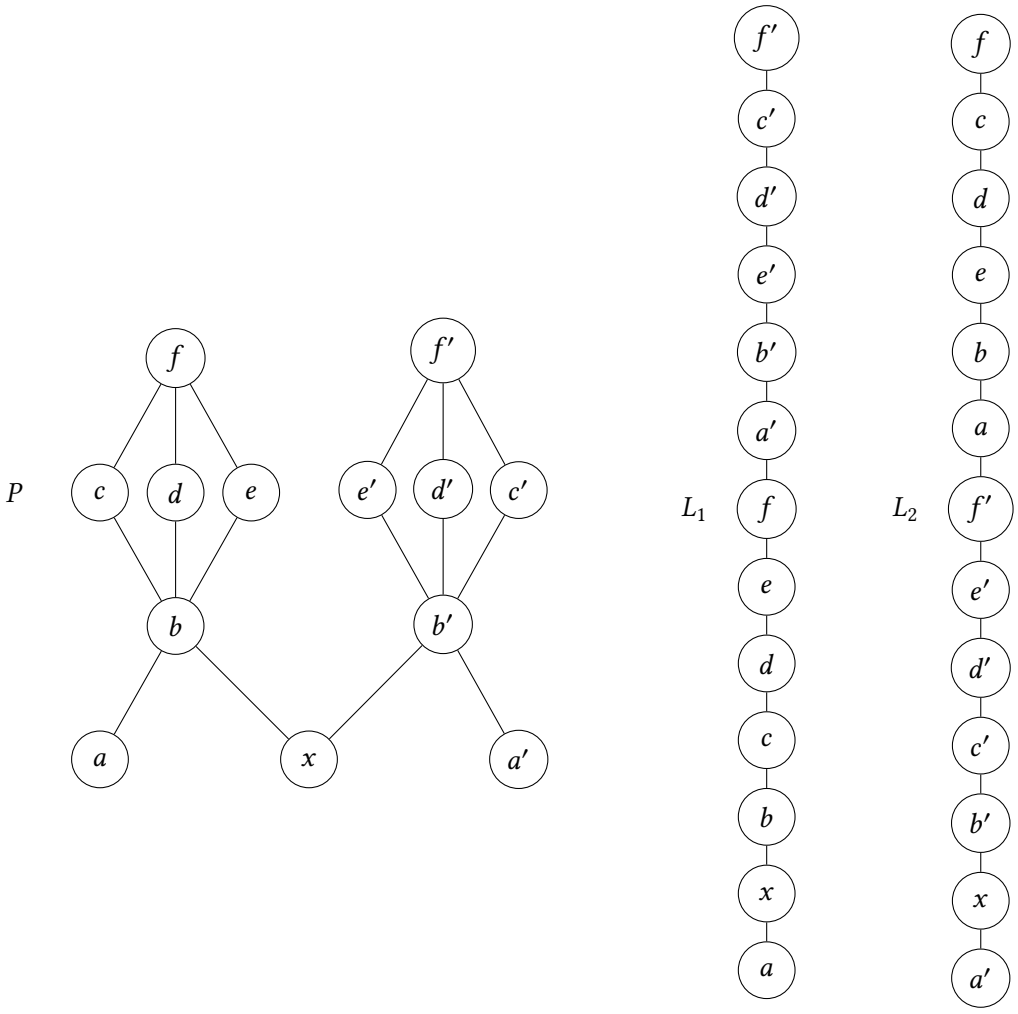

Abbildung 2.20: Ein zweidimensionales Poset P mit $P = L_1 \cap L_2$[Gol04]

Wenn wir also überprüfen können, ob P einen Vergleichbarkeitsgraph hat und dessen Komplement ebenfalls ein Vergleichbarkeitsgraph ist, dann können wir nachweisen, dass P höchstens zweidimensional ist.

Es gibt einen Algorithmus, der einen gerichteten Graphen $G = (V, A)$ als Eingabe erhält und entscheidet, ob dieser ein transitiver, azyklischer Graph eines zweidimensionalen Posets ist. Der Algorithmus verwendet eine modifizierte Tiefensuche, um lineare Erweiterungen für den Graphen und den Komplementgraphen zu konstruieren. Wenn dies möglich ist, handelt es sich um Vergleichbarkeitsgraphen. Dies ist in $\mathcal{O}(|V(G)|+|E(G)|)$ möglich [MS99]. Im Allgemeinen ist die Erkennung, ob ein Poset P die Dimension k hat, NP-vollständig für $k > 2$[Yan82].

3 Bestimmung von gerichteten Intervallgraphen

Wir haben einen beliebigen gemischten Graphen G gegeben und wollen überprüfen, ob es sich um einen gerichteten Intervallgraphen handelt und wenn dies der Fall ist, eine Intervallrepräsentation berechnen. G. Gutowski et al. haben in [Gut+22] gezeigt, dass gerichtete Intervallgraphen in $\mathcal{O}(|V(G)|^2)$ erkannt werden können. Der von ihnen vorgestellte Algorithmus wird hier näher erläutert und zusätzliche Informationen zu einzelnen Bestandteilen ergänzt.

Satz 3.1: *Es gibt einen Algorithmus, der für einen gegeben gemischten Graphen G in $\mathcal{O}(|V(G)|^2)$ entscheiden kann, ob es ein gerichteter Intervallgraph ist. Wenn dies der Fall ist, berechnet er eine gerichtete Intervallrepräsentation von G. [Gut+22]*

Der Algorithmus umfasst mehrere Schritte, die hier kurz vorgestellt werden. Unsere Eingabe ist ein beliebiger gemischter Graph $G = (V, E, A)$. Zunächst berechnen wir eine Ordnung σ des unterliegenden, ungerichteten Graphens $U(G)$. Mit dieser Knotenordnung berechnen wir einen MPQ-Baum T. Anschließend rotieren wir T, sodass er die Bedingungen einer gerichteten Intervallrepräsentation $\mathcal{I}$ erfüllt. Dann können wir ein Poset P konstruieren und überprüfen, ob dieses zweidimensional ist, indem wir zwei lineare Ordnungen berechnen. Wenn alle diese Schritte erfüllt sind, ist G ein gerichteter Intervallgraph. Dann können wir die Intervallrepräsentation konstruieren. Der Algorithmus 3.1 bietet zusätzlich eine Übersicht über die Schritte.

Algorithmus 3.1 : Erkennung von gerichteten Intervallgraphen

Input : Ein gemischter Graph $G = (V, E, A)$
Output : Gerichtete Intervallrepräsentation $\mathcal{I}$, falls möglich

1 3.2 Berechne der lexikographischen Ordnung σ von $U(G)$
2 3.3 Konstruiere einen MPQ-Baum T
3 3.4 Rotiere T, sodass er einer gerichteten Repräsentation entspricht
4 3.5 Konstruiere Poset P
5 Überprüfe P auf Zweidimensionalität und berechne Realizer R [MS99]
6 3.6 Konstruiere die Intervallrepräsentation $\mathcal{I}$

3.1 Lexikographische Breitensuche

Als Erstes wollen wir überprüfen, ob der unterliegende Graph $U(G)$ ein Intervallgraph ist. Wir wissen, dass dies der Fall ist, wenn wir einen MPQ-Baum T konstruieren können, dessen untere Grenze eine konsekutive Anordnung der maximalen Cliquen von $U(G)$ ist und alle äquivalenten MPQ-Bäume von T ebenfalls diese Eigenschaft haben. Dafür müssen wir zunächst

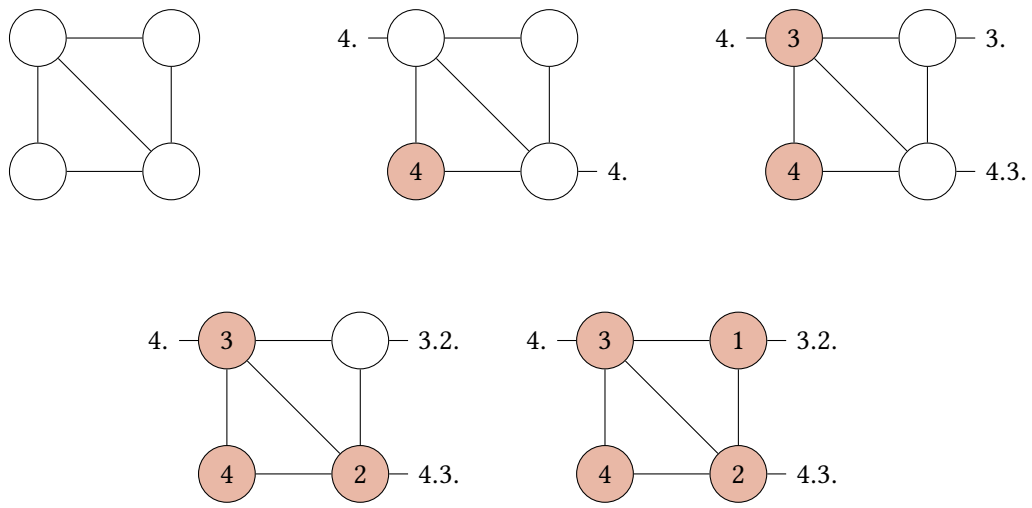

Abbildung 3.1: LEXBFS eines Graphen $G = (V, E)$. Die Label sind neben dem jeweiligen Knoten markiert

eine sogenannte lexikographische Breitensuche oder kurz LEXBFS durchführen und eine Ordnung σ von $U(G)$ berechnen. In dieser Reihenfolge können wir dann einen MPQ-Baum T iterativ aufbauen. Wenn dies gelingt, ist $U(G)$ ein Intervallgraph.

Wie der Name vermuten lässt, handelt es sich bei LEXBFS um eine modifizierte Breitensuche. Bei der normalen Breitensuche gibt es eine Mehrdeutigkeit, welcher Knoten als nächstes abgearbeitet wird. Wir werden hier den Knoten ein zusätzliches *Label* zuweisen, welche Knoten ihrer adjazenten Knoten bereits abgearbeitet wurden. Ein Label ist eine Menge von Zahlen, die in absteigender Reihenfolge aufgelistet werden.

Der Algorithmus geht wie folgt vor. Wir erhalten einen ungerichteten Graphen $G = (V, E)$ als Eingabe und wollen eine Knotenordnung $\sigma = [v_1, \ldots, v_n]$ berechnen mit $n = |V|$. Zu Beginn hat jeder Knoten ein leeres Label $\emptyset$. Wir beginnen bei einem beliebigen Knoten v, da alle Knoten gleich gewichtet sind und fügen diesen an das Ende unserer Ordnung ein. Wir bauen die Ordnung von hinten nach vorne auf. Dann weisen wir allen adjazenten Knoten das Label n zu. Anschließend wiederholen wir diesen Vorgang für einen Knoten mit dem höchsten Label, bis wir alle Knoten abgearbeitet haben. Wenn es mehrere Knoten mit dem gleichen Label gibt, wählen wir einen beliebigen. Der Algorithmus ist noch mal als Pseudocode in Algorithmus 3.2 aufgeführt. Seine Laufzeit benötigt $\mathcal{O}(|V(G)| + |E(G)|)$ Schritte [Gol04].

Algorithmus 3.2 : LEXBFS

Input : Ungerichteter Graph $G = (V, E)$
Output : Knotenordnung σ

```
1 Weise jedem Knoten das Label ∅ zu
2 for i ← n bis 1 do
3     wähle einen nicht nummerierten Knoten v mit größtem Label
4     σ[i] ← v
5     for jeden nicht nummerierten Knoten w ∈ Adj(v) do
6         füge i zu Label(w) hinzu
7 return σ
```

In Abbildung 3.1 sehen wir beispielsweise eine Durchführung von LEXBFS. Der Graph G wird von links unten nach rechts oben abgearbeitet. Die Knotenreihenfolge wird im Knoten notiert und das Label neben einem Knoten.

3.2 Konstruktion von MPQ-Bäumen

In diesem Abschnitt behandeln wir die Konstruktion eines MPQ-Baums T für den unterliegenden, ungerichteten Graphen $U(G)$ für unsere Eingabe $G = (V, E, A)$. Wir werden die LEXBFS-Ordnung σ verwenden, die wir gerade berechnet haben, um den Baum iterativ in dieser Reihenfolge aufzubauen. Wir können einen MPQ-Baum T_{i+1} berechnen, indem wir einen Knoten v zu einem MPQ-Baum T_i hinzufügen. Das heißt, wir fügen den Knoten v als Blatt an den Baum T_i an und passen anschließend die Knotenstruktur so an, dass es die Cliquen, die v enthält, repräsentiert. Wenn der Algorithmus erfolgreich ist, wissen wir, dass $U(G)$ ein Intervallgraph ist und können im nächsten Schritt T verwenden, um eine weitere Eigenschaft nachzuweisen.

Der Algorithmus zur Konstruktion von MPQ-Bäumen wurde erstmalig von Norbert Korte und Rolf H. Möhring in [KM89] vorgestellt.

Lemma 3.2: *Der Test, ob ein gegebener Graph $G = (V, E)$ ein Intervallgraph ist, und die Konstruktion seines MPQ-Baums sind in einer Laufzeit von $\mathcal{O}(|V(G)| + |E(G)|)$ möglich. [KM89]*

Wir gehen den Algorithmus nun im Detail durch. Als Eingabe erhalten wir einen ungerichteten Graphen $G = (V, E)$ und eine Ordnung $\sigma = [v_1, \ldots, v_n]$. Wir beginnen mit einem Baum T_1, der nur aus einem Blatt b mit $Z = \{v_n\}$ besteht. Wir durchlaufen σ von hinten nach vorne. Sei $T_i = (V', E', r, \mathcal{V}, \mathcal{S})$ unser aktueller MPQ-Baum und $u \in V$ unser aktueller Knoten, den wir in T_i einfügen wollen. Der Algorithmus hat eine Label- und eine Updatephase. In der Labelphase ordnen wir jedem Knoten $v' \in V'$ und jeder Sektion s von T_i ein *Label* zu, je nachdem, ob u zu allen ('∞'), manchen ('1') oder keinen Knoten ('0'), die v' oder s zugeordnet sind, adjazent ist. Die Label '∞' und '1' bezeichnen wir auch als *positiv*.

Als nächstes wollen die Label verwenden, um zu überprüfen, ob $G_{\{v_i,\ldots,v_n\}} + u$ ein Intervallgraph ist. In [KM89] wurde gezeigt, dass dies genau dann der Fall ist, wenn die folgenden zwei Bedingungen erfüllt sind:

1. Alle Knoten $v' \in V'$ mit positivem Label liegen auf einem Pfad P' in T_i
2. Für alle Q-Knoten q mit Sektionen $S_1, \ldots, S_k$ mit positivem Label ist der Schnitt der Menge der adjazenten Knoten $\mathrm{Adj}(u)$ und allen Knoten V_q in einer der äußeren Sektionen S_1 oder S_k enthalten. Es gilt $\mathrm{Adj}(u) \cap V_q \subseteq S_k$ oder $\mathrm{Adj}(u) \cap V_q \subseteq S_1$

Wenn diese Bedingungen erfüllt sind, dann ist u ein simplizialer Knoten in $G_{\{v_i,\ldots,v_n\}} + u$. Zum einen müssen seine adjazenten Knoten auf einem Pfad liegen, damit wir u an diesen anhängen können und dann weiterhin die Eigenschaft erfüllt ist, dass die Pfade in T Cliquen sind. Auf der anderen Seite muss ein simplizialer Knoten Teil einer äußeren Clique von $G_{\{v_1,\ldots,v_i\}} + u$ sein. Das heißt, es muss eine konsekutive Anordnung geben, sodass die maximale Clique C mit $u \in C$ am Anfang oder Ende der Anordnung liegt. Die erste Eigenschaft 1 gewährleistet auch, dass wir einen MPQ-Baum überhaupt iterativ aufbauen können, weil wir das Intervall $I(u)$ in einer Intervallrepräsentation an den Anfang oder das Ende hängen können, wenn wir uns an die LEXBFS Reihenfolge halten.

Algorithmus 3.3 : Konstruktion eines MPQ-Baum

Input : Ungerichteter Graph $G = (V, E)$ mit $\sigma[1, \ldots, n]$
Output : MPQ-Baum $T = (V', E', r, \mathcal{V}, \mathcal{S})$, falls möglich
`// Initialisiere` T
1 $V' \leftarrow b$
2 $r \leftarrow b$
3 $\mathcal{V} \leftarrow V_b = \{\sigma[n]\}$
4 **for** $i \leftarrow n - 1$ bis 1 **do**
5 $u \leftarrow \sigma[i]$
  `// Beginn der Labelphase`
6 **for** jeden Knoten $v \in V'$ **do**
7 Weise v ein Label zu je nachdem, ob alle ('∞'), manche ('1') oder keine Knoten ('0') von V_v adjazent sind zu u, die v zugeordnet sind
8 **if** v ist ein Q-Knoten **then**
9 Weise jeder Sektion $S_1, \ldots, S_k$ von v ebenfalls ein Label '∞', '1' oder '0' zu
10 **if** $Adj(u) \cap V_v \nsubseteq S_1 \vee \text{Adj}(u) \cap V_v \nsubseteq S_k$ **then**
11 **return** false
12 Bestimme einen Pfad $P = (r, \ldots, b)$ von der Wurzel r bis zu einem Blatt b, der alle Knoten und Sektionen mit positivem Label ('∞' oder '1') enthält
13 **if** P nicht existiert **then**
14 **return** false;
  `// Beginn der Updatephase`
15 **for** jeden Knoten $w \in P$ **do**
16 Überprüfe, ob w ein Blatt, P-Knoten oder Q-Knoten ist
17 Wende ein Template aus Abbildung 3.2 bis 3.5 an
18 **return** $T = (V', E', r, \mathcal{V}, \mathcal{S})$;

Die zweite Bedingung 2 können wir überprüfen, wenn wir die Label für die Q-Knoten setzen, indem wir die Mengen vergleichen. Für die erste Eigenschaft berechnen wir einen Pfad P von der Wurzel r zu einem Blatt b, der P' beinhalten. Wenn wir die Label für alle Knoten setzen, können wir uns bereits alle Knoten mit positivem Label in einer Warteschlange W speichern und anschließend überprüfen, ob diese auf einem konsekutiven Pfad liegen. Dazu arbeiten wir iterativ die Knoten $w \in W$ ab. Wir fügen w in einen neuen Baum B ein und fügen den Vater von w an das Ende von W ein. Wenn W leer ist, müssen wir nur einmal über B iterieren, um zu überprüfen, ob B ein Pfad ist. Sollte B nicht in einem Blatt von T enden, dann fügen wir solange Kinder des untersten Knotens zu B hinzu, bis dies der Fall ist.

Nachdem wir P berechnet haben und die Bedingungen geprüft haben, gehen wir in die Updatephase über. Sei v_* der niedrigste Knoten mit positivem Label in P, wobei die Wurzel r der höchste Knoten und das Blatt b der niedrigste Knoten ist. Sei v^* der höchste Knoten in P mit Label '0' oder '1'. Wenn kein solcher Knoten existiert oder es nur einen Knoten in P gibt, dann gilt $v_* = v^*$.

Der Algorithmus verwendet sogenannte *Templates*. Ein Template ist eine Kombination aus Mustererkennung und Modifikation. Wir unterscheiden zwischen Templates für Blätter, P-Knoten, Q-Knoten und zwei Hilfstemplates. Jedes Template hat bis zu 3 Unterkategorien, je nachdem, ob $u = v_*, v_* \neq v^*$, etc. gilt. Ein Template überführt einen Startzustand in einen Endzustand. Wir partitionieren die Menge von Knoten $V_{v'} = A \cup B$ eines Knotens $v' \in V'$ in eine Menge A von Knoten, die adjazent sind zu u und einer Menge B von Knoten, die nicht adjazent zu u sind. Diese Knoten werden aufgeteilt, damit verschiedene Cliquen auf unterschiedlichen Pfaden liegen. Die dargestellten Teilbäume T_i und T_i' mit i beliebig enthalten nur Knoten mit dem Label '0'. Angewendet werden die Templates von unten v_* bis nach oben v^* in P. Die Templates befinden sich in den Abbildungen 3.2 bis 3.5.

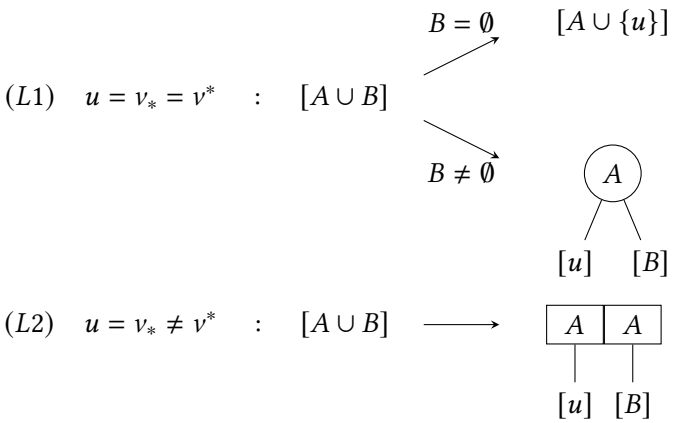

Abbildung 3.2: Templates für ein Blatt[KM89]

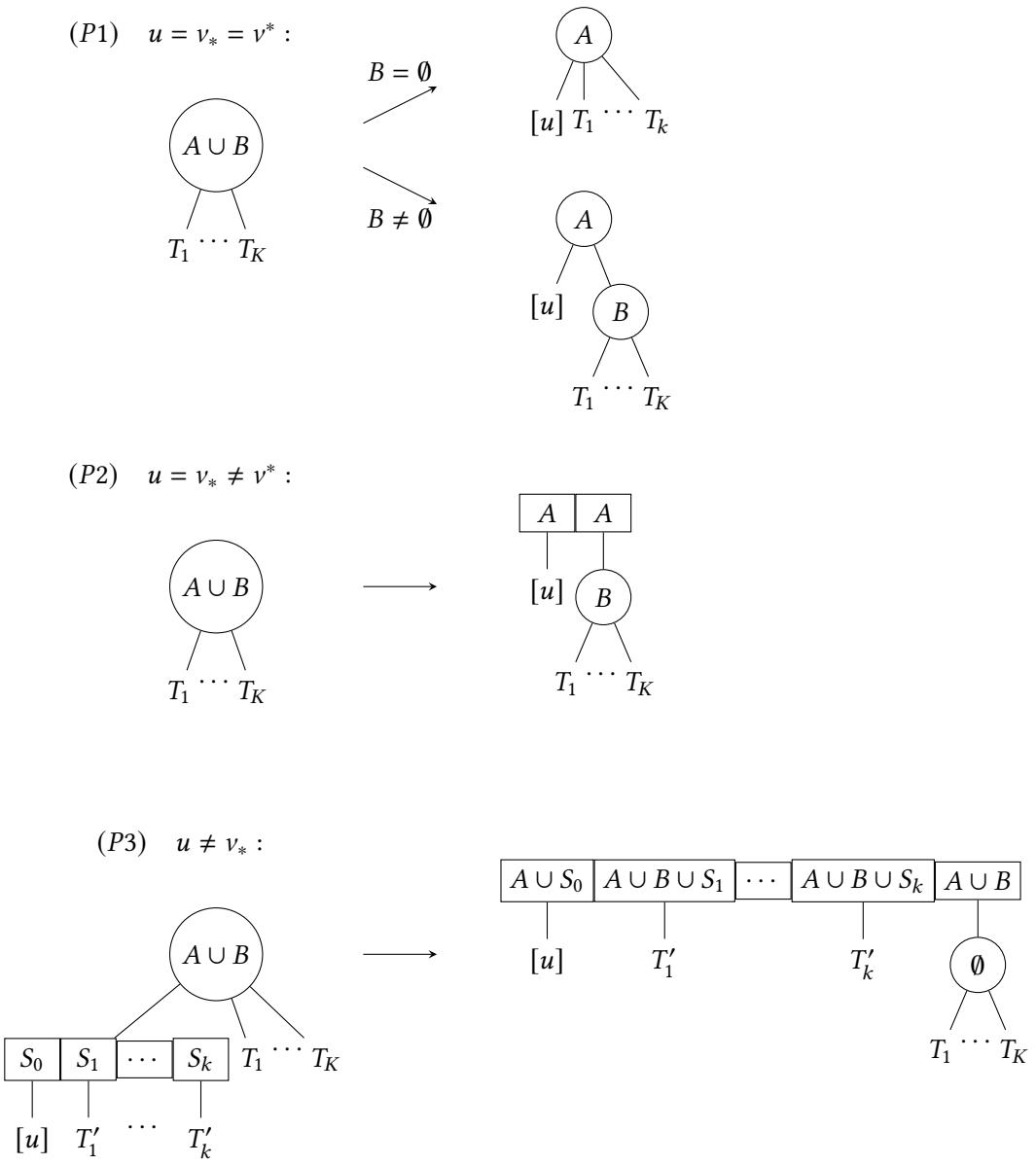

Abbildung 3.3: Templates für einen P-Knoten[KM89]

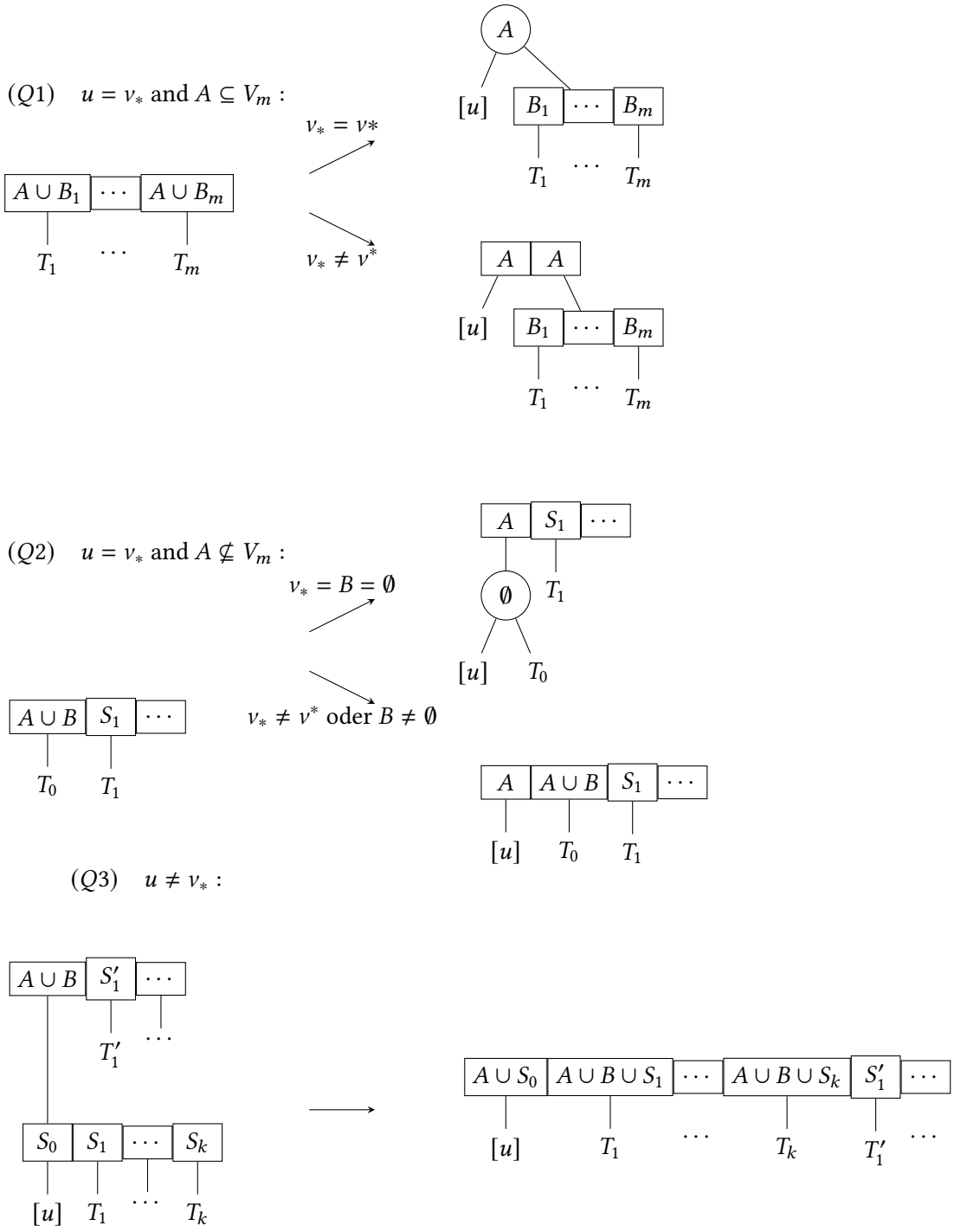

Abbildung 3.4: Templates für einen Q-Knoten[KM89]

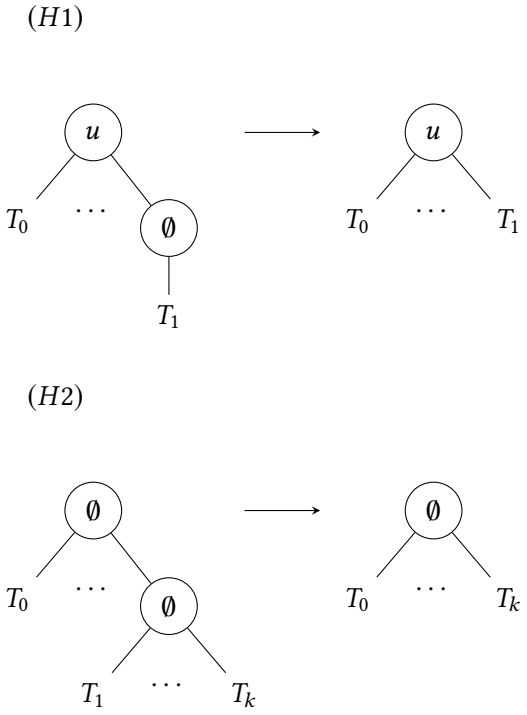

Abbildung 3.5: Hilfstemplates[KM89]

Beispiel 3.3: *Wir konstruieren beispielhaft einen MPQ-Baum für einen Graphen $G = (V, E)$, der nur aus einem Pfad P der Länge 4 besteht. Das heißt, die Knotenmenge ist $V = \{1, 2, 3, 4\}$ und die Kantenmenge ist $E = \{\{1, 2\}, \{2, 3\}, \{3, 4\}\}$. Die Konstruktion wird in Abbildung 3.6 veranschaulicht. Wir fügen die Knoten lexikographisch ein mit $\sigma = [1, 2, 3, 4]$.*

Wir beginnen mit einem leeren Baum und können 1 als Blatt einfügen. Als nächstes wollen wir 2 einfügen. In der Labelphase geben wir dem Blatt $b = [1]$ das Label ∞, da 1 und 2 adjazent sind. Wir haben nur einen Knoten und daher ist $b = v_* = v^*$ und wir befinden uns im Fall $(L1)$. Da alle Knoten in V_b adjazent sind zu 2 sind, ist $B = \emptyset$. Wir fügen 2 zu b hinzu.

Dann wollen wir 3 einfügen und geben unserem Blatt $b = [1, 2]$ das Label 1, da nur 2 adjazent zu 3 ist. In der Updatephase befinden wir uns wieder im Fall (L1), da es sich um ein Blatt handelt und wir mit einem Knoten weiterhin $b = v_* = v^*$ haben. Diesmal ist $A = \{1\}$ und $B = \{2\}$. Deshalb erhalten wir einen P-Knoten (2) mit zwei Kindern.

Wenn wir 4 einfügen, haben wir zum ersten Mal mehrere Fälle, die wir abarbeiten müssen. Der Pfad P vom Blatt $b' = [3]$ bis zur Wurzel $r = (2)$ besteht aus 2 Knoten und nur b' ist positiv gelabelt. Hier gilt $v_* = b'$ und $v^* = r$. Wir arbeiten uns von unten nach oben vor und beginnen mit b'. Diesmal sind wir im Fall $(L2)$ und fügen einen Q-Knoten ein. Anschließend aktualisieren wir den P-Knoten und befinden uns im Fall (P3). Damit der Leser leichter nachvollziehen kann, welche Bestandteile von T den Bestandteilen des Templates entsprechen, sind diese hier im Baum rot markiert. Es empfiehlt sich, für das bessere Verständnis einmal zu überprüfen, wie die einzelnen Bestandteile sich zum neuen Q-Knoten zusammensetzen. Dabei entsteht ein leerer P-Knoten, den wir mit dem Hilfstemplate $(H1)$ entfernen können.

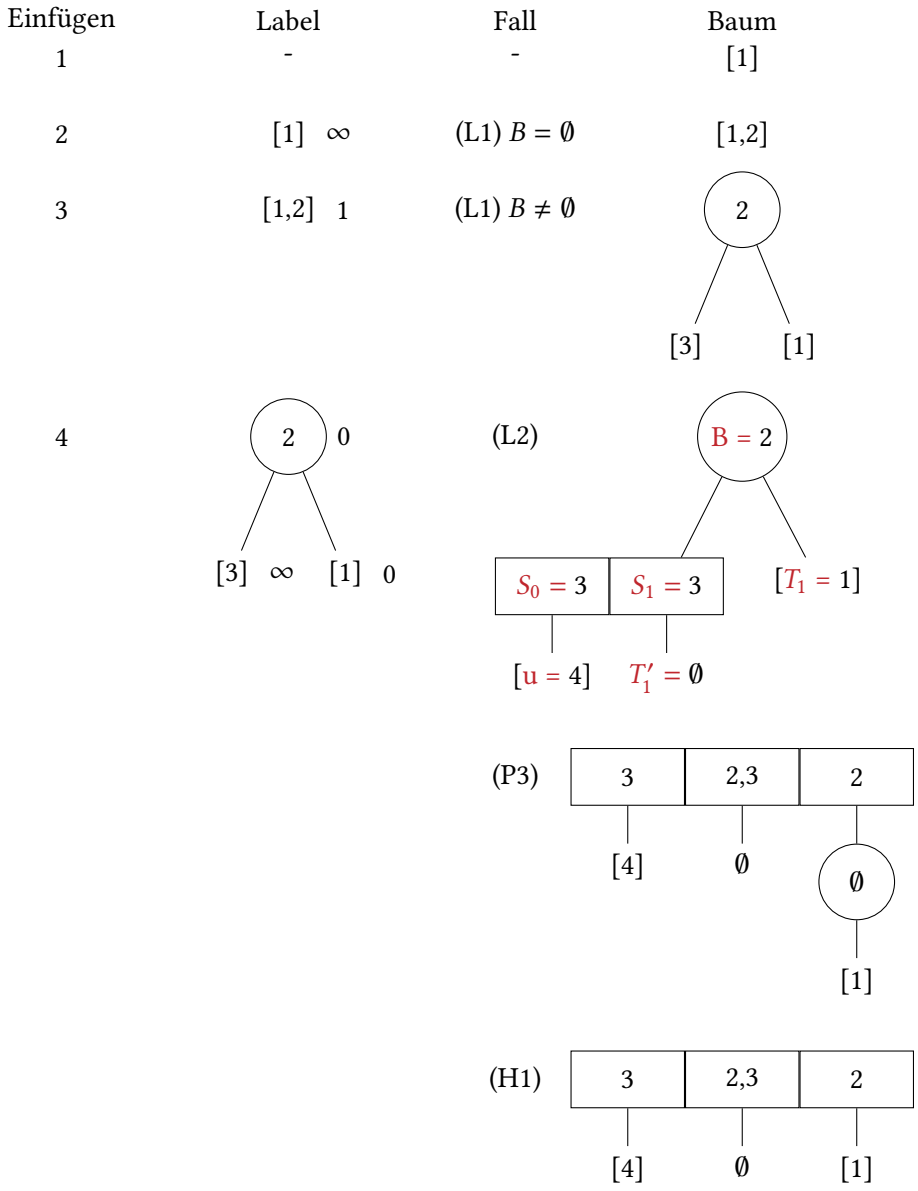

Abbildung 3.6: Konstruktion eines MPQ-Baums für P_4

3.3 Rotation von MPQ-Bäumen

Als nächstes wollen wir eine Rotation von unserem MPQ-Baum T finden, die mit einer gerichteten Repräsentation $\mathcal{I}$ von G übereinstimmt. Das heißt, wir suchen eine Menge von Intervallen $\mathcal{I}$, dessen Schnittgraph isomorph zum gemischten Graphen G ist. Bisher stimmt T mit einer ungerichteten Repräsentation überein, aber nicht jede Rotation von T erfüllt auch die Anforderungen eines gerichteten Intervallgraphens. Wenn G ein gerichteter Intervallgraph ist, dann muss es mindestens eine gerichtete Rotation von T geben.

Wenn es eine gerichtete Rotation von T gibt, dann können wir die untere Grenze $F(T)$ nutzen, um die Intervallrepräsentation $\mathcal{I}$ zu berechnen.

Lemma 3.4: *Es gibt einen Algorithmus, der für einen gegebenen gemischten Graphen G einen MPQ-Baum konstruieren kann, der mit einer gerichteten Repräsentation von G übereinstimmt, falls möglich. [Gut+22]*

Algorithmus 3.4 : MPQ-Baum rotieren

Input : MPQ-Baum T, gemischter Graph $G = (V, E, A)$
Output : Gerichteten MPQ-Baum T, falls möglich

1 **for** jeden Knoten $v' \in V'$ von T von den Blättern bis zur Wurzel **do**
2 $< y_1, \ldots, y_k >\leftarrow$ Kinder von v
3 **if** v' ist ein Q-Knoten **then**
4 **for** $i \leftarrow 2$ **bis** k **do**
5 **if** $S_1 \cap S_i \neq \emptyset$ **then**
6 $l \leftarrow i$
7 $u \leftarrow S_1 \cap S_l$
8 **if** $l < k$ **then**
9 $v \leftarrow (S_l \cap S_{l+1}) \setminus S_1$
10 Überprüfe die Richtung von $(u, v) \in A$. Rotiere v' entsprechend oder gib falsch zurück, wenn keine gerichtete Kante existiert
11 **if** v' ist ein P-Knoten **then**
12 **if** genau ein $a \in A^T_v$ und $b \in B^T_{v'}$ existiert mit $(a, b) \in A$ **then**
13 Rotiere den Teilbaum mit b an das Ende der Kinderliste von v'
14 **else**
15 **if** mehr als ein solches Knotenpaar existiert **then**
16 **return** false
17 **if** genau ein $\exists a \in A^T_v$ und $b \in B^T_{v'}$ existiert mit $(b, a) \in A$ **then**
18 Rotiere den Teilbaum mit b an den Anfang der Kinderliste von v'
19 **else**
20 **if** mehr als ein solches Knotenpaar existiert **then**
21 **return** false
22 **return** T

Jeder Knoten wird unabhängig voneinander rotiert. Wir beginnen bei den Blättern und rotieren von unten nach oben.

Wir wollen entscheiden, ob wir die Reihenfolge eines Q-Knotens invertieren oder nicht. Wenn ein Q-Knoten q in einen MPQ-Baum T auftritt, dann muss es mindestens zwei Knoten $u, v \in V_q$ geben, die eine gerichtete Kante $(u, v) \in A$ haben. Wir haben in Satz 2.34 gesehen,

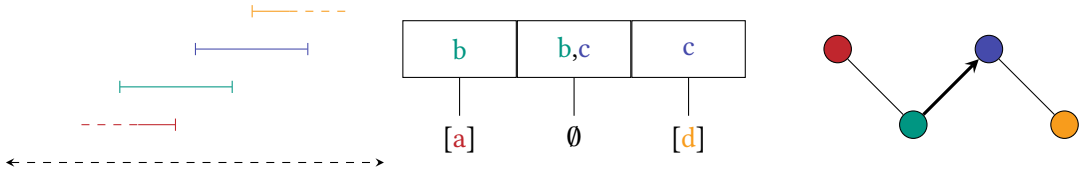

Abbildung 3.7: Intervallrepräsentation $\mathcal{I}$ (links), MPQ-Baum T (Mitte) and gerichteter Intervallgraph G (rechts)

dass der Schnitt $S_{i-1} \cap S_i \neq \emptyset$ der Mengen zweier benachbarter Sektionen S_{i-1}, S_i für $i = 2, \ldots, k$ nie leer sein kann. Außerdem ist dieser Schnitt ohne die Elemente von der ersten Sektion S_1 und der letzten Sektion S_k nicht leer: $(S_i \cap S_{i+1}) \setminus S_1 \neq \emptyset$ und $(S_{i-1} \cap S_i) \setminus S_k \neq \emptyset$ für $i = 2, \ldots, k-1$. Das heißt, zwei benachbarte Sektionen, die weder die erste noch die letzte Sektion sind, enthalten mindestens einen Knoten u, der nicht in S_1 vorkommt und einen Knoten v, der nicht in S_k vorkommt. Anschaulich gesprochen hängt es damit zusammen, dass Q-Knoten mehrere überlappende, maximale Cliquen in eine bestimmte Reihenfolge forcieren. Wir betrachten ein Beispiel, welches in Abbildung 3.7 dargestellt wird.

Beispiel 3.5: *Sei $T = (V', E', r, \mathcal{V}, \mathcal{S})$ ein MPQ-Baum von $U(G)$ mit $G = (V, E, A)$. Es gibt nur einen Q-Knoten q in T, der zu einem Pfad P mit 4 Knoten a, b, c, d korrespondiert. Es gibt 3 Cliquen $C_1 = \{a, b\}, C_2 = \{b, c\}$ und $C_3 = \{c, d\}$ in G. Die Intervalle $I(b)$ und $I(c)$ können sich nur schneiden, aber nicht überlappen, da b, d nicht adjazent sind und a, c nicht adjazent sind. Wenn das Intervall $I(b)$ das Intervall $I(c)$ überlappen würde, dann kann c nicht zu d adjazent sein, ohne das b zu d adjazent ist. Das heißt, es muss eine gerichtete Kante (b, c) in G geben.*

Sei q ein Q-Knoten in T und $q_1, \ldots, q_k$ seine Kinder. Wir wollen zwei Knoten $u, v \in \bigcup_{i=1}^{k} S_i$ bestimmen, deren Intervalle $I(v), I(u)$ sich in der Intervallrepräsentation $\mathcal{I}$ auf jeden Fall schneiden müssen. Wenn sie das tun, dann muss es eine gerichtete Kante $(u, v) \in A$ geben, die uns die Ausrichtung von q vorgibt.

Sei $l = \max\{i \mid S_1 \cap S_i \neq \emptyset\}$ der größte Index einer Sektion S_i von q, der mindestens einen Knoten mit der ersten Sektion S_1 teilt. Sei $u \in S_1 \cap S_l$ einer dieser Knoten. Es gilt $l < k$, weil es mindestens eine gerichtete Kante geben muss. Sei $v \in (S_l \cap S_{l+1}) \setminus S_1$. Wir haben mit l einen Punkt ausgemacht, an dem die Knoten u, v einer gemeinsamen Clique angehören, aber ausgeschlossen, dass sie in jeder Sektion gemeinsam auftreten. Daraus folgt, dass die Intervalle $I(v)$ und $I(u)$ sich in jeder Repräsentation $\mathcal{I}$ überschneiden, aber nicht überlappen. Es muss also eine gerichtete Kante $(u, v) \in A$ geben. Wenn wir die Richtung dieser Kante kennen, dann kennen wir auch die Ausrichtung des Q-Knotens. Wenn eine solche Kante nicht existiert, dann kann G kein gerichteter Intervallgraph sein.

P-Knoten werden wie folgt rotiert. Sei $p_1, \ldots, p_k$ die Kinder eines P-Knoten p in T. Diese Kinder bilden induzierte Teilbäume $T_{p_1}, \ldots, T_{p_k}$ mit $p_1, \ldots, p_k$ als Wurzel.

Wir überlegen uns nun, wo es gerichtete Kante zwischen den Knoten geben kann, die in den Teilbäumen gespeichert werden. Seien $u, v \in V$ zwei beliebige Knoten von G und $p', p'' \in V'$ zwei beliebige Knoten von T, die sich in verschiedenen Teilbäumen der Kinder von p befinden. Es gilt $u \in V_{p'}$ und $v \in V_{p''}$. Dann kann keine Kante $\{u, v\} \notin E$ existieren. Da die Pfade von den Blättern bis zur Wurzel von T die Cliquen von G sind, müssen Knoten in unterschiedlichen Teilbäumen in unterschiedlichen Cliquen sein. Daraus folgt, dass es nur gerichtete Kanten von einem Knoten oberhalb von p zu einem Knoten unterhalb von p geben kann oder umgekehrt.

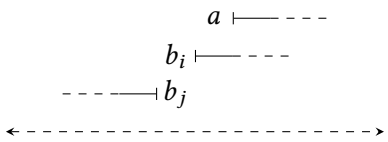

Abbildung 3.8: Gegenbeispiel

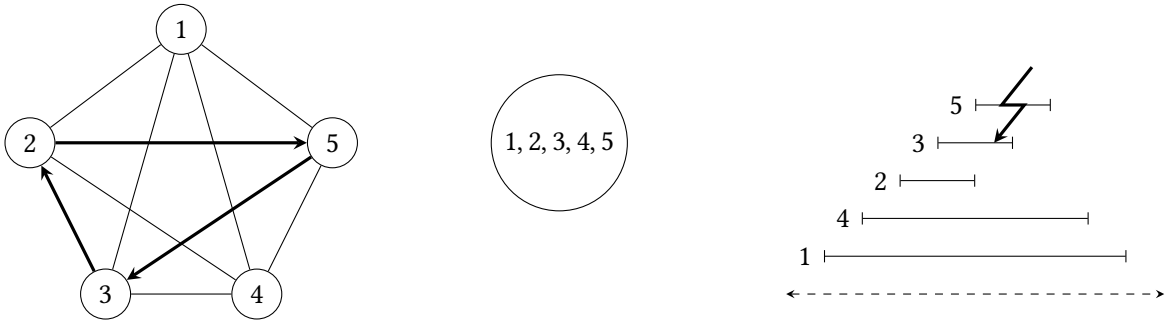

Abbildung 3.9: Graph $G = (V, E)$ (links) und MPQ-Baum T (Mitte) und gerichtete Intervallrepräsentation $\mathcal{I}$ (rechts)

Der Algorithmus rotiert Teilbäume an den Anfang der Kinderliste, wenn es eine eingehende Kante in den Teilbaum gibt und an das Ende der Liste, wenn es eine ausgehende Kante gibt. Es folgt ein Gegenbeispiel, was passieren würde, wenn wir einen solchen Teilbaum nicht an den Anfang der Liste rotieren würden.

Beispiel 3.6: *Sei $G = (V, E, A)$ ein gerichteter Intervallgraph und $T = (V', E', r, \mathcal{V}, \mathcal{S})$ ein MPQ-Baum von $U(G)$. Betrachte einen P-Knoten $p \in V'$. Nehmen wir an, dass $(b_i, a) \in A$ eine gerichtete Kante ist, wobei der Knoten $b_i \in V$ unter p liegt und $a \in V$ ein Knoten ist, der über p liegt. Sei T_{p_i} jener induzierte Teilbaum, in dem b_i in einer Menge gespeichert wird.*

Wir nehmen an, es gibt einen induzierten Teilbaum T_{p_j} mit $j \neq i$ und dieser ist das erste Kind von p. Sei b_j ein Knoten, der in einer Menge in diesem Teilbaum gespeichert wird. Der linke Endpunkt von a muss rechts von b_i liegen, da wir die gerichtete Kante (b_i, a) haben. Der linke Endpunkt von b_j muss links von b_i liegen, da der Teilbaum sich links von diesem befindet. Daraus folgt, dass b_j und a disjunkt sind (siehe Abbildung 3.8). Es kommt zum Widerspruch, da jeder Knoten oberhalb von p adjazent ist zu jedem Knoten unterhalb von p. Daraus folgt, dass der Teilbaum von p_i am Anfang der Kinderliste sein muss.

Wenn es mehrere induzierte Teilbäume geben sollte, die das erste oder letzte Kind von p sein müssen, dann können wir keine gerichtete Rotation von T finden und G ist kein gerichteter Intervallgraph. In [Gut+22] wurde darüber hinaus noch gezeigt, dass die Position der übrigen Kinder von p beliebig ist.

3.4 Konstruktion des Posets

Wir haben bisher für einen gemischten Graphen $G = (V, E, A)$ einen MPQ-Baum T konstruiert, dessen untere Grenze mit einer gerichteten Repräsentation $\mathcal{I}$ übereinstimmt. Dies reicht tatsächlich noch nicht aus, um nachzuweisen, dass G ein gerichteter Intervallgraph ist. Wir betrachten das folgende Beispiel:

Beispiel 3.7: *Sei $G = (V, E, A)$ ein vollständiger Graph mit fünf Knoten. Das heißt, G besteht aus einer maximalen Clique $C = V$. Ein MPQ-Baum T von G besteht daher nur aus einem P-Knoten. In G kann es einen gerichteten Kreis K geben. Wenn wir versuchen, für K eine gerichtete Intervallrepräsentation zu finden, kommt es zum Widerspruch. Dies ist in Abbildung 3.9 illustriert. Für den ungerichteten, unterliegenden Graphen $U(G)$ wäre es allerdings möglich, eine ungerichtete Intervallrepräsentation zu finden.*

Die konsekutive Anordnung der Cliquen reicht also nicht aus, wenn die Cliquen selbst nicht als gerichtete Intervallrepräsentation gezeichnet werden können. In [Gut+22] wurde gezeigt, dass G genau dann ein gerichteter Intervallgraph ist, wenn wir ein bestimmtes Poset $P = (X, <)$ konstruieren und überprüfen, ob dieses zweidimensional ist. Wir geben zunächst die Konstruktion von P an und werden dann eine Intuition für den Beweis geben.

Algorithmus 3.5 : Konstruiere Poset

Input : Gemischter Graph $G = (V, E, A)$ mit allen maximalen Cliquen $C_1, \dots, C_k$
Output : Graph $D = (V', A')$

$V' \leftarrow \bigcup_{i=1}^{k+1} \{a_i, b_i\}$
$A' \leftarrow \bigcup_{1 \le i < j \le k+1} \{(a_i, a_j), (b_i, b_j)\}$
$V' \leftarrow V' \cup V$
$A' \leftarrow A' \cup A$
for alle $v \in V$ **do**
    $lc(v) \leftarrow$ Index der ersten Clique C_i mit $v \in C_i$
    $rc(v) \leftarrow$ Index der letzten Clique C_i mit $v \in C_i$
    $A' \leftarrow A' \bigcup_{1 \le i \le lc(v)} \{(a_i, v)\}$
    $A' \leftarrow A' \bigcup_{1 \le i \le rc(v)} \{(b_i, v)\}$
    for alle $u \in V$ mit $u \neq v$ **do**
        if u, v sind unabhängig in G und $rc(u) < lc(v)$ **then**
            $A' \leftarrow A' \cup (u, v)$
return $D = (V', A')$

Wir beginnen nun unser Poset $P = (X, <)$ als transitiven, azyklischen Graphen $D = (V', A')$ zu konstruieren. In Abbildung 3.10 sehen wir ein Beispiel. Zunächst fügen wir zwei Ketten $K = (a_1, \dots, a_{k+1})$ und $K' = (b_1, \dots, b_{k+1})$ der Länge $k+1$ in D ein. Die Kette K hat gerichteten Kanten $(a_i, a_j) \in A'$ für $1 \le i < j \le k+1$ und die Kette K' hat gerichtete Kanten $(b_i, b_j) \in A'$ für $1 \le i < j \le k+1$. Diese Kanten sind gestrichelt in Abbildung 3.10 dargestellt. Dann fügen wir alle Knoten $v \in V$ in V' ein und alle gerichteten Kanten A aus G in D ein.

Sei $lc(v)$ bzw. $rc(v)$ der Index der ersten oder der letzten Clique, in der $v \in V$ vorkommt. Dann fügen wir gerichtete Kanten (a_i, v) für $1 \le i \le lc(v)$ und gerichtete Kanten (b_i, v) für $1 \le i \le rc(v)$ in D ein. Diese Kanten sind gepunktet in Abbildung 3.10 dargestellt. Außerdem fügen wir gerichtete Kanten (u, v) ein für jeden Knoten v und u, die in G unabhängig sind und für die gilt $rc(u) < lc(v)$.

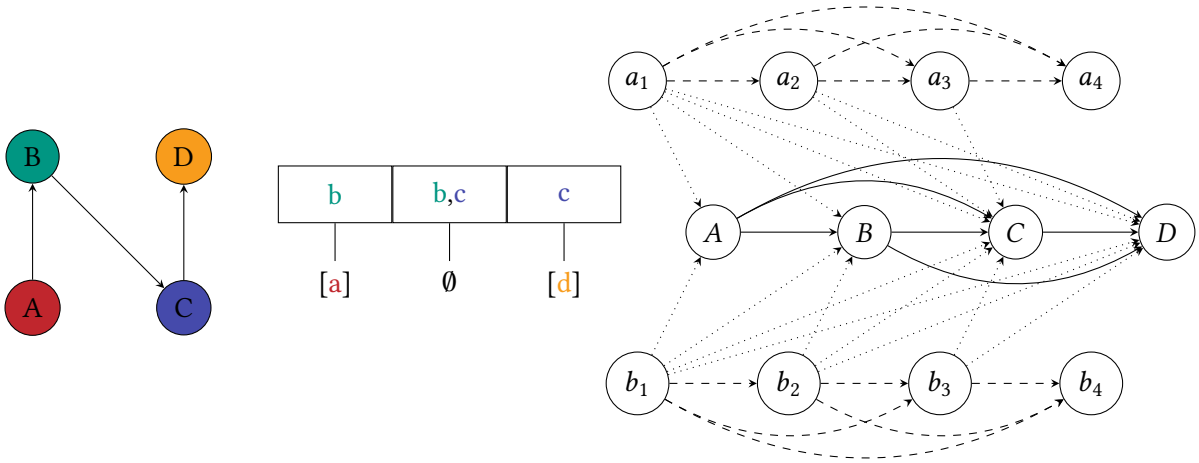

Abbildung 3.10: Gerichteter Intervallgraph G (links), MPQ-Baum T (Mitte) und Graph D des Posets P (rechts)

Der Beweis funktioniert über zwei Richtungen. Zum einen wurde gezeigt, dass mit einem gerichteten Intervallgraphen $G = (V, E, A)$ und einer gerichteten Intervallrepräsentation $\mathcal{I}$ ein zweidimensionaler Realizer für D konstruiert werden kann. Sei L_i die Folge aller Knoten v in G mit $lc(v) = i$ und analog R_i die Folge mit $rc(v) = i$. Die zwei linearen Ordnungen L und R haben die folgende Form:

$$L = b_1 < b_2 < \cdots < b_{k+1} < a_1 < L_1 < a_2 < L_2 < \cdots < a_k < L_k < a_{k+1} \tag{3.1}$$

$$R = a_1 < a_2 < \cdots < a_{k+1} < b_1 < R_1 < b_2 < R_2 < \cdots < b_k < R_k < b_{k+1} \tag{3.2}$$

Für die andere Richtung wird gezeigt, dass mit einem zweidimensionalen Realizer $Re = \{L, R\}$ von D die gerichtete Intervallrepräsentation $\mathcal{I}$ berechnet werden kann. Für uns bedeutet das, wenn wir Re von D berechnen können und Re zweidimensional ist, dann ist G ein gerichteter Intervallgraph. Wir verwenden den Algorithmus von McConnell und Spinrad, um Re zu berechnen [MS99].

3.5 Konstruktion der Intervallrepräsentation

Es verbleibt nur noch die Intervallrepräsentation $\mathcal{I}$ zu berechnen.

Lemma 3.8: *Es gibt einen Algorithmus, der für einen gegebenen MPQ-Baum T, der mit einer gerichteten Repräsentation von G übereinstimmt, eine gerichtete Repräsentation $\mathcal{I}$ von G konstruiert, sodass T mit $\mathcal{I}$ übereinstimmt. [Gut+22]*

Sei k die Anzahl aller maximalen Cliquen von G. Für $1 \leq i \leq k$ berechnen wir die Ordnungen L_i und R_i aus dem Realizer $Re = \{L, R\}$. Die Ordnung L_i ist die Reihenfolge der Knoten mit $lc(v) = i$, wie sie in der linearen Erweiterung L auftreten. Die Ordnung R_i ist analog.

Wir unterscheiden die Ordnungen L, R in Re wie folgt. Die Knoten b_{k+1} und a_1 sind unabhängig in D. Das heißt, in einer der beiden Ordnungen von Re muss $b_{k+1} < a_1$ gelten. Diese Ordnung ist L. Für alle Elemente $l_i \in L_i$ gilt: $a_i < l_i < a_{i+1}$ und analog für $r_i \in R_i$ gilt $b_i < r_i < b_{i+1}$. Dies haben wir auch bereits bei der Konstruktion des Posets in der Formell 3.1 und 3.2 angeben.

Algorithmus 3.6 : Konstruktion der Intervallrepräsentation

Input : Gemischter Graph $G = (V, E, A)$, Anzahl maximaler Cliquen k und Realizer $Re = \{L, R\}$
Output : Intervallrepräsentation $\mathcal{I}$

for $i = 1$ **bis** k **do**
  $L_i = [v \in V \mid a_i < v < a_{i+1} \text{ in } L]$
  $R_i = [v \in V \mid b_i < v < b_{i+1} \text{ in } R]$
for alle Knoten $v \in V$ **do**
  // $L^{-1}_{lc(v)}$ ist der Index von v in der Ordnung $L_{lc(v)}$
  $start \leftarrow lc(v) - \frac{1}{2} + lc(v)/L^{-1}_{lc(v)}(v)$
  $end \leftarrow rc(v) + \frac{1}{2} - rc(v)/L^{-1}_{rc(v)}(v)$
  $\mathcal{I} \leftarrow \mathcal{I} \cup \{(start, end)\}$
return $\mathcal{I}$

Für jede Clique $i = 1, \ldots, k$ wählen wir $|L_i|$ verschiedene reelle Punkte in $(i - \frac{1}{2}, i)$ und $|R_i|$ verschiedene reelle Punkte in $(i, i + \frac{1}{2})$. Für jeden Knoten v an der i-ten Stelle in $L_{lc(v)}$ wählen wir den i-ten Punkt in $(i - \frac{1}{2}, i)$ als Startpunkt und den j-ten Punkt in $(i, i + \frac{1}{2})$ als Endpunkt für die j-te Stelle in $R_{rc(v)}$.

In Abbildung 3.11 sehen wir ein Beispiel für unseren Graphen, der nur aus einem Pfad besteht. Es empfiehlt sich, einmal nachzuprüfen, wie wir von den angegebenen Funktionen bzw. Ordnungen auf die Intervallrepräsentation kommen. Für die erste Clique C_1 haben wir beispielsweise $|L_1| = 2$ Startpunkte zwischen $(\frac{1}{2}, 1)$ gewählt. Da A vor B in der Ordnung L_1 auftritt, erhält A den früheren Startpunkt.

Wenn wir die Intervalle auf diese Weise wählen, erhalten wir eine gerichtete Intervallrepräsentation $\mathcal{I}$ von G. Nun überprüfen wir, ob $\mathcal{I}$ korrekt ist. Zwei Intervalle $I(u)$ und $I(v)$ schneiden sich genau dann, wenn die Knoten u und v eine gemeinsame Clique haben. Da wir eine konsekutive Anordnung der maximalen Cliquen berechnet haben, tritt jede maximale Clique für jeden Knoten $v \in V$ hintereinander auf. Wenn u und v adjazent sind, muss es mindestens eine Clique C geben, in der beide Knoten enthalten sind, also $u, v \in C$. So wie die Start- und Endpunkte der Intervalle gewählt wurden, liegt der Start- und Endpunkt von u und v links bzw. rechts von der Position der Clique C. Daher müssen sie sich schneiden. Wenn u und v nicht adjazent sind, können sich ihre Intervalle in $\mathcal{I}$ nicht schneiden, da die Anordnung der Cliquen konsekutiv ist und es keine Clique C geben kann, die beide Knoten enthält.

Wenn es eine gerichtete Kante $(u, v) \in A$ in G gibt, dann gibt es diese Kante auch in D und $u < v$ muss ebenfalls in L und R gelten. Dadurch bekommt $I(u)$ einen früheren Start- und Endpunkt als $I(v)$.

Wenn es eine ungerichtete Kante $\{u, v\} \in E$ in G gibt, dann sind u, v unabhängig in D. Dadurch gilt $u < v$ in einer der Ordnungen L, R und $v < u$ in der anderen. Das heißt, einer der beiden Intervall muss vor dem anderen beginnen und nach dem anderen enden. Eines der Intervalle $I(u)$ oder $I(v)$ muss das andere überlappen.

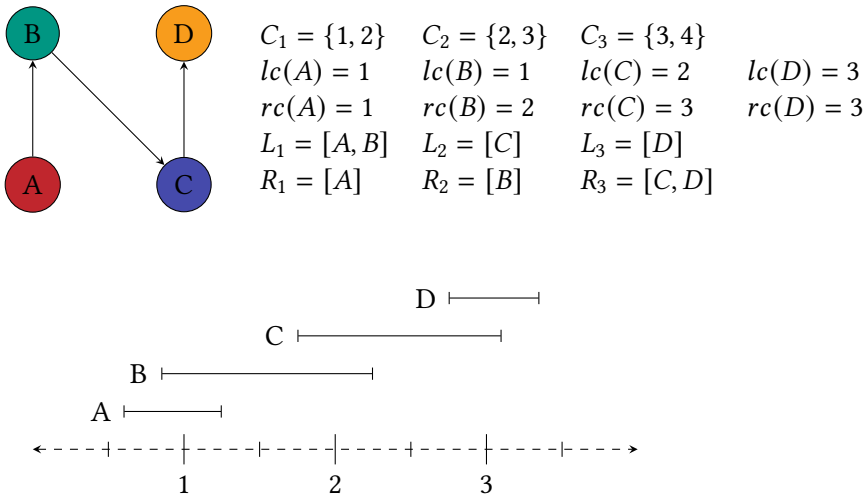

Abbildung 3.11: Gerichteter Intervallgraph G (oben) und gerichtete Intervallrepräsentation $\mathcal{I}$ (unten)

3.6 Laufzeit

Wir haben bereits ganz zu Anfang dieses Kapitels in Satz 3.1 gesehen, dass der Algorithmus zur Erkennung von gerichteten Intervallgraphen eine Laufzeit von $\mathcal{O}(|V(G)|^2)$ hat. Wir wollen uns in diesem Abschnitt ansehen, wie sich die Laufzeit der einzelnen Schritte zusammensetzt und wo sich die Schwachstelle befindet.

Sei $G = (V, E, A)$ ein gegebener gemischter Graph, für den wir bestimmen wollen, ob er ein gerichteter Intervallgraph ist. Wir berechnen eine Knotenordnung σ von dem unterliegenden, ungerichteten Graphen $U(G)$ mit LEXBFS. Die Laufzeit von LEXBFS ist in diesem Fall $\mathcal{O}(|V(G)| + |E(G)| + |A(G)|)$ [Gol04], da wir die gerichteten sowie ungerichteten Kantenmengen für $U(G)$ vereinigen. Einen MPQ-Baum T von $U(G)$ zu konstruieren hat ebenfalls eine Laufzeit von $\mathcal{O}(|V(G)| + |E(G)| + |A(G)|)$ [KM89].

Die Rotation von T hat naiv eine Laufzeit von $\mathcal{O}(|V|^2)$. Ein MPQ-Baum hat $\mathcal{O}(|V|)$ Knoten, die rotiert werden müssen. Q-Knoten und P-Knoten benötigen jeweils $\mathcal{O}(|V|)$ Schritte, um rotiert zu werden. In [Gut+22] wird behauptet, dass alle Knoten in linearer Zeit rotiert werden können, aber es ist nicht unmittelbar offensichtlich, wie dies möglich ist.

Der Graph D des Posets P hat $\mathcal{O}(|V(G)|)$ Knoten und $\mathcal{O}(|V(G)|^2)$ Kanten. Der zweidimensionale Realizer $Re = \{L, R\}$ von D kann in $\mathcal{O}(|V(D)| + |A(D)|)$ berechnet werden [MS99]. Insgesamt ist das eine Laufzeit von $\mathcal{O}(|V(G)|^2)$ für die Konstruktion von D und Re. Dies ist die erwähnte Schwachstelle des Algorithmus.

Mit L bzw. R können wir L_i bzw. R_i in $\mathcal{O}(|V|)$ Schritten berechnen, indem wir alle Knoten $v \in V$ in die Mengen hinzufügen. Die Start- und Endpunkte der Intervalle zu berechnen sind dann nur noch arithmetische Operationen. Sie für alle Knoten auszuführen ist ebenfalls in $\mathcal{O}(|V(G)|)$ Schritten möglich.

4 Fazit

Wir haben den Algorithmus zur Erkennung von gerichteten Intervallgraphen ausführlich behandelt. Für einen gemischten Graphen $G = (V, E, A)$ können wir in $\mathcal{O}(|V|^2)$ eine gerichtete Intervallrepräsentation $\mathcal{I}$ berechnen, falls diese existiert.

Begonnen haben wir mit der lexikographischen Breitensuche auf dem unterliegenden, ungerichteten Graphen $U(G)$, um eine Knotenordnung σ zu berechnen. Anhand dieser Ordnung haben wir erfolgreich iterativ einen MPQ-Baum T aufgebaut, wenn $U(G)$ ein Intervallgraph ist. Anschließend haben wir eine Rotation von T bestimmt, sodass T mit einer gerichteten Repräsentation $\mathcal{I}'$ von G übereinstimmt. Wenn dies ebenfalls erfolgreich war, konnten wir die untere Grenze $F(T)$ später verwenden, um die gerichtete Intervallrepräsentation $\mathcal{I}$ zu berechnen. Zuvor haben wir noch den Graphen $D = (V', A')$ eines Posets $P = (X, \leq)$ bestimmt. Um zu prüfen, ob G ein gerichteter Intervallgraph ist, haben wir einen zweidimensionalen Realizer Re von D berechnet. Anhand von $F(T)$ und Re konnten wir abschließend $\mathcal{I}$ berechnen.

4.1 Ausblick

Gemischte Intervallgraphen als Teilgebiet der Intervallgraphen sind ein nicht abgeschlossenes Thema in der Graphentheorie. Es bleibt offen, ob gerichtete Intervallgraphen in linearer Zeit erkannt werden können. Die Konstruktion des Graphen D des Posets P beinhaltet quadratisch viele Kanten, die in die Laufzeit der Konstruktion des Realizers einfließt. Die Frage ist, ob man die Zweidimensionalität auch erkennen könnte, wenn man die transitiven Kanten von D weglassen und nur implizit behandeln würde.

Eine weitere Möglichkeit wäre es, PC-Bäume anstelle von PQ-Bäumen zu verwenden, um festzustellen, ob der unterliegende Graph ein Intervallgraph ist. Eine analoge Form der modifizierten PC-Bäume oder auch MPC-Bäume könnte entwickelt werden. Es könnte untersucht werden, ob dies Vorteile gegenüber des bestehenden Algorithmus birgt.

Für das Kantenrouting im Sugiyama-Framework ist es von besonderem Interesse, ob bigerichtete Intervallgraphen in polynomieller Zeit gefärbt werden können. Leider konnte bereits gezeigt werden, dass die Färbung NP-schwer ist. Offen bleibt es, ob es einen α-Approximationsalgorithmus mit $\alpha < 2$ gibt. Ebenso ist es ungeklärt, ob bigerichtete Intervallgraphen in polynomieller Zeit erkannt werden können.

Literatur

[BL76] Kellogg S Booth und George S Lueker. „Testing for the consecutive ones property, interval graphs, and graph planarity using PQ-tree algorithms“. In: *Journal of computer and system sciences* Jg. 13 (1976), S. 335–379. DOI: *https://doi.org/10.1016/S0022-0000(76)80045-1*.

[FG65] Delbert Fulkerson und Oliver Gross. „Incidence matrices and interval graphs“. In: *Pacific journal of mathematics* Jg. 15 (1965), S. 835–855. DOI: *http://dx.doi.org/10.2140/pjm.1965.15.835*.

[FPR23] Simon D Fink, Matthias Pfretzschner und Ignaz Rutter. „Experimental comparison of pc-trees and pq-trees“. In: *ACM Journal of Experimental Algorithmics* Jg. 28 (2023), S. 1–24. DOI: *https://doi.org/10.1145/3611653*.

[Gol04] Martin Charles Golumbic. *Algorithmic graph theory and perfect graphs*. Elsevier, 2004. DOI: *https://doi.org/10.1016/C2013-0-10739-8*.

[Gut+22] Grzegorz Gutowski, Florian Mittelstädt, Ignaz Rutter, Joachim Spoerhase, Alexander Wolff und Johannes Zink. „Coloring mixed and directional interval graphs“. In: *International Symposium on Graph Drawing and Network Visualization*. Springer. 2022, S. 418–431. DOI: *https://doi.org/10.1007/978-3-031-22203-0_30*.

[Gut+23] Grzegorz Gutowski, Konstanty Junosza-Szaniawski, Felix Klesen, Paweł Rzążewski, Alexander Wolff und Johannes Zink. „Coloring and recognizing mixed interval graphs“. In: (2023). DOI: *https://doi.org/10.4230/LIPIcs.ISAAC.2023.36*.

[HM03] Wen-Lian Hsu und Ross M McConnell. „PC trees and circular-ones arrangements“. In: *Theoretical computer science* Jg. 296 (2003), S. 99–116. DOI: *https://doi.org/10.1016/S0304-3975(02)00435-8*.

[KM89] Norbert Korte und Rolf H Möhring. „An incremental linear-time algorithm for recognizing interval graphs“. In: *SIAM Journal on Computing* Jg. 18 (1989), S. 68–81. DOI: *http://dx.doi.org/10.1137/0218005*.

[MS99] Ross M McConnell und Jeremy P Spinrad. „Modular decomposition and transitive orientation“. In: *Discrete Mathematics* Jg. 201 (1999), S. 189–241. DOI: *https://doi.org/10.1016/S0012-365X(98)00319-7*.

[STT81] Kozo Sugiyama, Shojiro Tagawa und Mitsuhiko Toda. „Methods for visual understanding of hierarchical system structures“. In: *IEEE Transactions on Systems, Man, and Cybernetics* Jg. 11 (1981), S. 109–125. DOI: *https://doi.org/10.1109/TSMC.1981.4308636*.

[Yan82] Mihalis Yannakakis. „The complexity of the partial order dimension problem“. In: *SIAM Journal on Algebraic Discrete Methods* Jg. 3 (1982), S. 351–358. DOI: *https://doi.org/10.48550/arXiv.1501.01147*.

[ZWBW22] Johannes Zink, Julian Walter, Joachim Baumeister und Alexander Wolff. „Layered drawing of undirected graphs with generalized port constraints“. In: *Computational Geometry* Jg. 105 (2022), S. 101886. DOI: *https://doi.org/10.1016/j.comgeo.2022.101886*.