LMU LUDWIG-MAXIMILIANS-UNIVERSITÄT MÜNCHEN

FAKULTÄT FÜR MATHEMATIK, INFORMATIK UND STATISTIK
INSTITUT FÜR INFORMATIK

Skriptum

zur Vorlesung

Algorithmische Bioinformatik: Bäume und Graphen

gehalten im Sommersemester 2024

am Lehrstuhl für Praktische Informatik und Bioinformatik

Volker Heun

4. Juli 2024

Version 8.24

Vorwort

Dieses Skript entstand parallel zu der Vorlesung *Algorithmische Bioinformatik III* des Sommersemester 2003 und 2004, die als Fortsetzung der Vorlesungen *Algorithmische Bioinformatik I* und *Algorithmische Bioinformatik II* dient. Seit dem Sommersemester 2006 wurde die Vorlesung in *Algorithmische Bioinformatik: Bäume und Graphen* umbenannt, um den Inhalt besser zu charakterisieren.

Diese Vorlesungen wurde an der Ludwig-Maximilians-Universität speziell für Studenten der Bioinformatik, aber auch für Studenten der Informatik, im Rahmen des von der Ludwig-Maximilians-Universität München und der Technischen Universität München gemeinsam veranstalteten Studiengangs Bioinformatik gehalten.

Abschnitte, die im Sommersemester 2024 nicht Teil der Vorlesung waren, sind mit einem Stern (*) und Teile, die nur teilweise behandelt wurden, sind mit einem Plus-Zeichen (+) markiert.

Diese Fassung ist jetzt weitestgehend korrigiert, dennoch kann es immer noch den einen oder anderen Fehler enthalten. Daher bin ich für jeden Hinweis auf Fehler oder Ungenauigkeiten (an Volker.Heun@bio.ifi.lmu.de) dankbar.

An dieser Stelle möchte ich insbesondere meinen Mitarbeitern Johannes Fischer und Simon W. Ginzinger sowie den Übungsleitern Florian Erhard und Benjamin Albrecht für ihre Unterstützung bei der Veranstaltung danken, die somit das vorliegende Skript erst möglich gemacht haben. Auch möchte ich Sabine Spreer danken, die an der Erstellung der ersten Version dieses Skriptes in LaTeX2e maßgeblich beteiligt war. Weiterhin danke ich Frau Caroline Friedel, die zahlreiche Fehler im Skript gefunden hat.

München, im Sommersemester 2024 Volker Heun

Inhaltsverzeichnis

Physical Mapping 1

1.1 Biologischer Hintergrund und Modellierung

Bei der *genomischen Kartierung* (engl. *physical mapping*) geht es darum, einen ersten groben Eindruck des Genoms zu bekommen. Dazu soll für „charakteristische“ Sequenzen der genaue Ort auf dem Genom festgelegt werden. Im Gegensatz zu *genetischen Karten* (engl. *genetic map*), wo es nur auf die lineare und ungefähre Anordnung einiger bekannter oder wichtiger Gene auf dem Genom ankommt, will man bei *genomischen Karten* (engl. *physical map*) die Angaben nicht nur ungefähr, sondern möglichst genau bis auf die Position der Basenpaare ermitteln.

1.1.1 Genomische Karten

Wir wollen zunächst die Idee einer genomischen Karte anhand einer „Landkarte aus Photographien“ für Deutschland beschreiben. Wenn man einen ersten groben Überblick der Lage der Orte von Deutschland bekommen will, dann könnte ein erster Schritt sein, die Kirchtürme aus ganz Deutschland zu erfassen. Kirchtürme bieten zum einen den Vorteil, dass sich ein Kirchturm als solcher sehr einfach erkennen lässt, und zum anderen, dass Kirchtürme verschiedener Kirchen in der Regel doch deutlich unterschiedlich sind. Wenn man nun Luftbilder von Deutschland bekommt und die Kirchtürme den Orten zugeordnet hat, dann kann man für die meisten Photographien entscheiden, zu welchem Ort sie gehören, sofern denn ein Kirchturm darauf zu sehen ist. Ausgehend von Luftbildern, auf denen mehrere Kirchtürme zu sehen sind, kann man dann die relative Lage der Orte innerhalb Deutschlands festlegen. Die äquivalente Aufgabe bei der genomischen Kartierung ist die Zuordnung von auffälligen Sequenzen (Kirchtürme) auf Positionen im Genom (Koordinaten in Deutschland). Ein Genom ist dabei im Gegensatz zu Deutschland ein- und nicht zweidimensional.

Ziel der genomischen Kartierung ist es, ungefähr alle 10.000 Basenpaare eine charakteristische Sequenz auf dem Genom zu finden und zu lokalisieren. Dies ist wichtig für einen ersten Grob-Eindruck eines Genoms. Für das Human Genome Project war eine solche Kartierung wichtig, damit man das ganze Genom relativ einfach in viele kleine Stücke aufteilen konnte, so dass die einzelnen Teile von unterschiedlichen Forscher-Gruppen sequenziert werden konnten. Die einzelnen Teile konnten dann unabhängig und somit hochgradig parallel sequenziert werden. Damit zum Schluss

die einzelnen sequenzierten Stücke wieder den Orten im Genom zugeordnet werden konnten, wurde dann eine genomische Karte benötigt.

Obwohl Celera Genomics mit dem Whole Genome Shotgun Sequencing gezeigt hat, dass für die Sequenzierung großer Genome eine genomische Karte prinzipiell nicht unbedingt benötigt wird, so mussten diese Daten letztendlich für die Sequenzierung des menschlichen Genoms mitverwendet werden. Weiterhin ist diese zum einen immer noch hilfreich zur vollständigen Sequenzierung eines Genoms und zum anderen auch beim Vergleich von ähnlichen Genomen. Weiterhin finden die verwendeten Methoden mittlerweile unter anderem auch im Gebiet der komparativen Genomik Anwendung.

1.1.2 Konstruktion genomischer Karten

Wie erstellt man nun solche genomischen Karten. Das ganze Genom wird in viele kleinere Stücke, so genannte *Fragmente* zerlegt. Dies kann mechanisch durch feine Sprühdüsen oder biologisch durch Restriktionsenzyme geschehen. Diese einzelnen kurzen Fragmente werden dann auf spezielle Landmarks hin untersucht.

Als Landmarks können zum Beispiel so genannte *STS*, d.h. *Sequence Tagged Sites*, verwendet werden. Dies sind kurze Sequenzabschnitte, die im gesamten Genom eindeutig sind. In der Regel sind diese 100 bis 500 Basenpaare lang, wobei jedoch nur die Endstücke von jeweils 20 bis 40 Basenpaaren als Sequenzfolgen bekannt sind. Vorteil dieser STS ist, dass sie sich mit Hilfe der Polymerasekettenreaktion sehr leicht nachweisen lassen, da gerade die für die PCR benötigten kurzen Endstücke als Primer bekannt sind. Somit lassen sich die einzelnen Fragmente daraufhin untersuchen, ob sie eine STS enthalten oder nicht. Alternativ kann auch mit Hybridisierungsexperimenten festgellt werden, ob eine STS in einem Fragment enthalten ist oder nicht.

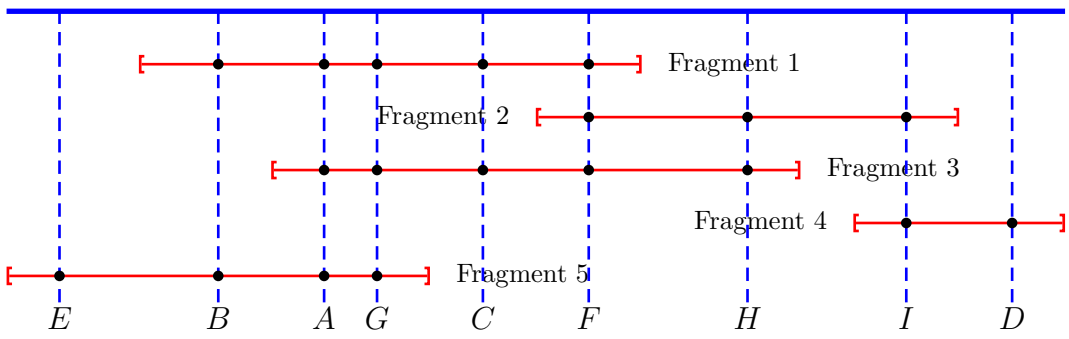


Abbildung 1.1: Skizze: Genomische Kartierung

In Abbildung 1.1 ist eine Aufteilung in Fragmente und die zugehörige Verteilung der STS illustriert. Dabei ist natürlich weder die Reihenfolge der STS im Genom, noch die Reihenfolge der Fragmente im Genom (aufsteigend nach Anfangspositionen) bekannt. Die Experimente liefern nur, auf welchem Fragment sich welche STS befindet. Die Aufgabe der genomischen Kartierung ist es nun, die Reihenfolge des STS im Genom (und damit auch die Reihenfolge des Auftretens der Fragmente im Genom) zu bestimmen. Im Beispiel, das in der Abbildung 1.1 angegeben ist, erhält man als Ergebnis des Experiments nur die folgende Information (neben der ungefähren Länge der Fragmente):

$$\begin{aligned} F_1 &= \{A, B, C, F, G\}, \\ F_2 &= \{F, H, I\}, \\ F_3 &= \{A, C, F, G, H\}, \\ F_4 &= \{D, I\}, \\ F_5 &= \{A, B, E, G\}. \end{aligned}$$

Hierbei gibt die Menge F_i an, welche STS das Fragment i enthält. In der Regel sind natürlich die Fragmente nicht in der Reihenfolge ihres Auftretens durchnummeriert, sonst wäre die Aufgabe ja auch trivial.

Aus diesem Beispiel sieht man schon, das sich die Reihenfolge aus diesen Informationen nicht immer eindeutig rekonstruieren lässt. Obwohl im Genom A vor G auftritt, ist dies aus den experimentellen Ergebnissen nicht ablesbar.

1.1.3 Modellierung mit Permutationen und Matrizen

In diesem Abschnitt wollen wir zwei recht ähnliche Methoden vorstellen, wie man die Aufgabenstellung mit Mitteln der Informatik modellieren kann. Eine Modellierung haben wir bereits kennen gelernt: Die Ergebnisse werden als Mengen angegeben. Was wir suchen ist eine Permutation der STS, so dass für jede Menge gilt, dass die darin enthaltenen Elemente in der Permutation zusammenhängend vorkommen, also durch keine andere STS separiert werden. Für unser Beispiel wären also $EBAGCFHID$ und $EBGACFHID$ sowie $DIHFCGABE$ und $DIHFCAGBE$ zulässige Permutationen, da hierfür gilt, dass die Elemente aus F_i hintereinander in der jeweiligen Permutation auftreten.

Wir merken hier bereits an, dass wir im Prinzip immer mindestens zwei Lösungen erhalten, sofern es überhaupt eine Lösung gibt. Aus dem Ergebnis können wir nämlich die Richtung nicht feststellen. Mit jedem Ergebnis ist auch die rückwärts aufgelistete Reihenfolge eine Lösung. Dies lässt sich in der Praxis mit zusätzlichen Experimenten jedoch leicht lösen.

	A	B	C	D	E	F	G	H	I
1	1	1	1	0	0	1	1	0	0
2	0	0	0	0	0	1	0	1	1
3	1	0	1	0	0	1	1	1	0
4	0	0	0	1	0	0	0	0	1
5	1	1	0	0	1	0	1	0	0

	E	B	A	G	C	F	H	I	D
1	0	1	1	1	1	1	0	0	0
2	0	0	0	0	0	1	1	1	0
3	0	0	1	1	1	1	1	0	0
4	0	0	0	0	0	0	0	1	1
5	1	1	1	1	0	0	0	0	0

Abbildung 1.2: Beispiel: Matrizen-Darstellung

Eine andere Möglichkeit wäre die Darstellung als eine $n \times m$-Matrix, wobei wir annehmen, dass wir n verschiedene Fragmente und m verschiedene STS untersuchen. Der Eintrag an der Position (i, j) ist genau dann 1, wenn die STS j im Fragment i enthalten ist, und 0 sonst. Diese Matrix für unser Beispiel ist in Abbildung 1.2 angegeben. Hier ist es nun unser Ziel, die Spalten so zu permutieren, dass die Einsen in jeder Zeile aufeinander folgend (konsekutiv) auftreten. Wenn es eine solche Permutation gibt, ist es im Wesentlichen dieselbe wie die, die wir für unsere andere Modellierung erhalten. In der Abbildung 1.2 ist rechts eine solche Spaltenpermutation angegeben. Daher sagt man auch zu einer 0-1 Matrix, die eine solche Permutation erlaubt, dass sie die *Consecutive Ones Property*, kurz *C1P* oder *COP*, erfüllt.

1.1.4 Fehlerquellen

Im vorigen Abschnitt haben wir gesehen, wie wir unser Problem der genomischen Kartierung geeignet modellieren können. Wir wollen jetzt noch auf einige biologische Fehlerquellen eingehen, um diese bei späteren anderen Modellierungen berücksichtigen zu können. Diese prinzipiellen Fehlerquellen treten zumindest teilweise auch bei anderen Anwendungen auf.

False Positives: Leider kann es bei den Experimenten auch passieren, dass eine STS in einem Fragment i identifiziert wird, obwohl sie gar nicht enthalten ist. Dies kann zum Beispiel dadurch geschehen, dass in der Sequenz sehr viele Teilsequenzen auftreten, die den Primern der STS zu ähnlich sind, oder aber die Primer tauchen ebenfalls sehr weit voneinander entfernt auf, so dass sie gar keine STS bilden, jedoch dennoch vervielfältigt werden. Solche falschen Treffer werden als *False Positives* bezeichnet.

False Negatives: Analog kann es passieren, dass, obwohl eine STS in einem Fragment enthalten ist, diese durch die PCR nicht multipliziert wird. Solche fehlenden Treffer werden als *False Negatives* bezeichnet.

Chimeric Clones: Außerdem kann es nach dem Aufteilen in Fragmente passieren, dass sich die einzelnen Fragmente zu längeren Teilen rekombinieren. Dabei

könnten sich insbesondere Fragmente aus ganz weit entfernten Bereichen des untersuchten Genoms zu einem neuen Fragment kombinieren und fälschlicherweise Nachbarschaften liefern, die gar nicht existent sind. Solche Rekombinationen werden als *Chimeric Clones* bezeichnet.

Non-Unique Probes: Ein weiteres Problem stellen so genannte Non-Unique Probes dar, also STS, die mehrfach im Genom vorkommen und fälschlicherweise als einzigartig angenommen wurden.

1.2 PQ-Bäume

In diesem Abschnitt wollen wir einen effizienten Algorithmus zur Entscheidung der Consecutive Ones Property vorstellen. Obwohl dieser Algorithmus mit keinem der im vorigen Abschnitt erwähnten Fehler umgehen kann, ist er dennoch von grundlegendem Interesse, da andere Algorithmen auf diesen Methoden aufbauen.

1.2.1 Definition von PQ-Bäumen

Zur Lösung der C1P benötigen wir das Konzept eines PQ-Baumes. Im Prinzip handelt es sich hier um einen gewurzelten Baum mit besonders gekennzeichneten inneren Knoten und markierten Blättern.

Definition 1.1 *Sei Σ ein endliches Alphabet. Dann ist ein* PQ-Baum *über Σ induktiv wie folgt definiert:*

- *Jeder einelementige Baum (also ein Blatt), das mit einem Zeichen aus Σ markiert ist, ist ein PQ-Baum über Σ.*
- *Sind $T_1, \ldots, T_k$ PQ-Bäume über Σ, dann ist der Baum, der aus einem so genannten* P-Knoten *als Wurzel entsteht und dessen Kinder die Wurzeln der Bäume $T_1, \ldots, T_k$ sind, ebenfalls ein PQ-Baum über Σ.*
- *Sind $T_1, \ldots, T_k$ PQ-Bäume über Σ, dann ist der Baum, der aus einem so genannten* Q-Knoten *als Wurzel entsteht und dessen Kinder die Wurzeln der Bäume $T_1, \ldots, T_k$ sind, ebenfalls ein PQ-Baum über Σ.*

In der Abbildung 1.3 ist skizziert, wie wir in Zukunft P- bzw. Q-Knoten graphisch darstellen wollen. P-Knoten werden durch Kreise, Q-Knoten durch lange Rechtecke dargestellt. Für die Blätter führen wir keine besondere Konvention ein. In der Abbildung 1.4 ist ein Beispiel eines PQ-Baumes angegeben.

Abbildung 1.3: Skizze: Darstellung von P- und Q-Knoten

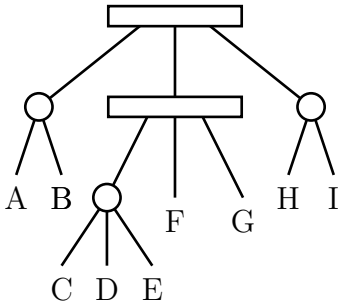


Abbildung 1.4: Beispiel: Ein PQ-Baum

Im Folgenden benötigen wir spezielle PQ-Bäume, die wir jetzt definieren wollen.

Definition 1.2 *Sei Σ ein endliches Alphabet. Ein PQ-Baum über Σ heißt* echt*, wenn die folgenden Bedingungen erfüllt sind:*

- *Jedes Element $a \in \Sigma$ kommt genau einmal als Blattmarkierung vor;*
- *Jeder P-Knoten hat mindestens zwei Kinder;*
- *Jeder Q-Knoten hat mindestens drei Kinder.*

Der in Abbildung 1.4 angegebene PQ-Baum ist also ein echter PQ-Baum.

An dieser Stelle wollen wir noch ein elementares, aber fundamentales Ergebnis über gewurzelte Bäume wiederholen, das für PQ-Bäume im Folgenden sehr wichtig sein wird.

Lemma 1.3 *Sei T ein gewurzelter Baum, wobei jeder innere Knoten mindestens zwei Kinder besitzt, dann ist die Anzahl der inneren Knoten echt kleiner als die Anzahl der Blätter von T.*

Da ein echter PQ-Baum diese Eigenschaft erfüllt (ein normaler in der Regel nicht), wissen wir, dass die Anzahl der P- und Q-Knoten kleiner als die Kardinalität des betrachteten Alphabets Σ ist.

Die P- und Q-Knoten besitzen natürlich eine besondere Bedeutung, die wir jetzt erläutern wollen. Wir wollen PQ-Bäume im Folgenden dazu verwenden, Permutation

zu beschreiben. Daher wird die Anordnung der Kinder an P-Knoten willkürlich sein (d.h. alle Permutationen der Teilbäume sind erlaubt). An Q-Knoten hingegen ist die Reihenfolge bis auf das Umdrehen der Reihenfolge fest. Um dies genauer beschreiben zu können benötigen wir noch einige Definitionen.

Definition 1.4 *Sei T ein echter PQ-Baum über Σ. Die* Frontier *von T, kurz $f(T)$ ist die Permutation über Σ, die durch das Ablesen der Blattmarkierungen von links nach rechts geschieht (also die Reihefolge der Blattmarkierungen in einer Tiefensuche unter Berücksichtigung der Ordnung auf den Kindern jedes Knotens).*

Die Frontier des Baumes aus Abbildung 1.4 ist dann ABCDEFGHI.

Definition 1.5 *Zwei PQ-Bäume T und T' heißen* äquivalent*, kurz $T \cong T'$, wenn sie durch endliche Anwendung folgender Regeln ineinander überführt werden können:*

- *Beliebiges Umordnen der Kinder eines P-Knotens;*
- *Umkehren der Reihenfolge der Kinder eines Q-Knotens.*

Damit kommen wir zur Definition konsistenter Frontiers eines PQ-Baumes.

Definition 1.6 *Sei T ein echter PQ-Baum, dann ist consistent(T) bzw. cons(T) die Menge der konsistenten Frontiers von T, d.h.:*

$$\mathit{cons}(T) := \mathit{consistent}(T) := \{f(T') \ : \ T \cong T'\}\,.$$

Beispielsweise befinden sich dann in der Menge cons(T) für den Baum aus der Abbildung 1.4: BADCEFGIH, ABGFCDEHI oder HIDCEFGBA, insgesamt gibt es 96 verschiedene Frontiers für diesen echten PQ-Baum.

Definition 1.7 *Sei Σ ein endliches Alphabet und $\mathcal{F} = \{F_1, \ldots, F_k\} \subseteq 2^\Sigma$ eine so genannte Menge von* Restriktionen*, d.h. von Teilmengen von Σ. Dann bezeichnet $\Pi(\Sigma, \mathcal{F})$ die Menge der Permutationen über Σ, in der die Elemente aus F_i für jedes $i \in [1:k]$ konsekutiv vorkommen.*

Mit Hilfe dieser Definitionen können wir nun das Ziel dieses Abschnittes formalisieren. Zu einer gegebenen Menge $\mathcal{F} \subset 2^\Sigma$ von Restriktionen (nämlich den Ergebnissen unserer biologischen Experimente zur Erstellung einer genomischen Karte) wollen wir einen echten PQ-Baum T mit

$$\mathrm{cons}(T) = \Pi(\Sigma, \mathcal{F})$$

konstruieren, sofern dies möglich ist.

1.2.2 Konstruktion von PQ-Bäumen

Wir werden versuchen, den gewünschten PQ-Baum für die gegebene Menge von Restriktionen iterativ zu konstruieren, d.h. wir erzeugen eine Folge $T_0, T_1, \ldots, T_k$ von PQ-Bäumen, so dass

$$\text{cons}(T_i) = \Pi(\Sigma, \{F_1, \ldots, F_i\})$$

gilt. Dabei ist $T_0 = T(\Sigma)$ der PQ-Baum, dessen Wurzel aus einem P-Knoten besteht und an dem n Blätter hängen, die eineindeutig mit den Zeichen aus $\Sigma = \{a_1, \ldots, a_n\}$ markiert sind. Wir müssen daher nur noch eine Prozedur reduce entwickeln, für die $T_i = \text{reduce}(T_{i-1}, F_i)$ gilt.

16.04.24

Prinzipiell werden wir zur Realisierung dieser Prozedur den Baum T_{i-1} von den Blättern zur Wurzel hin durchlaufen, um die Restriktion F_i einzuarbeiten. Dazu werden alle Blätter, deren Marken in F_i auftauchen markiert und wir werden nur den Teilbaum mit den markierten Blättern bearbeiten. Dazu bestimmen wir zuerst den niedrigsten Knoten $r(T_{i-1}, F_i)$ in T_i, so dass alle Blätter aus F_i in dem an diesem Knoten gewurzelten Teilbaum enthalten sind. Diesen Teilbaum selbst bezeichnen wir mit $T_r(T_{i-1}, F_i)$ als den *reduzierten Teilbaum*.

Weiterhin vereinbaren wir noch den folgenden Sprachgebrauch:

- Ein Blatt heißt *voll*, wenn es in F_i vorkommt und ansonsten *leer*.
- Ein innerer Knoten heißt *voll*, wenn alle seine Kinder voll sind.
- Analog heißt ein innerer Knoten *leer*, wenn alle seine Kinder leer sind.
- Andernfalls nennen wir den Knoten *partiell*.

Im Folgenden werden wir auch Teilbäume als *voll* bzw. *leer* bezeichnen, wenn alle darin enthaltenen Knoten voll bzw. leer sind (was äquivalent dazu ist, dass dessen Wurzel voll bzw. leer ist). Andernfalls nennen wir einen solchen Teilbaum *partiell*.

Da es bei P-Konten nicht auf die Reihenfolge ankommt, wollen wir im Folgenden immer vereinbaren, dass die leeren Kinder und die vollen Kinder eines P-Knotens immer konsekutiv angeordnet sind (siehe Abbildung 1.5).

Abbildung 1.5: Skizze: Anordnung leerer und voller Kinder eines P-Knotens

Im Folgenden werden wir volle und partielle Knoten bzw. Teilbäume immer rot kennzeichnen, während leere Knoten bzw. Teilbäume weiß bleiben. Man beachte, dass ein PQ-Baum nie mehr als zwei partielle Knoten besitzen kann, von denen nicht einer ein Nachfahre eines anderen ist. Andernfalls könnten die gewünschten Permutationen aufgrund der gegebenen Restriktionen nicht konstruiert werden. Die Abbildung 1.6 mag dabei helfen, sich dies klar zu machen, wobei die gestreiften Teilbäume partielle Teilbäume darstellen.

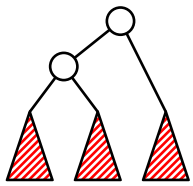


Abbildung 1.6: Skizze: Drei partielle Teilbäume

In den folgenden Teilabschnitten werden wir verschiedene Schablonen beschreiben, die bei unserer bottom-up-Arbeitsweise im reduzierten Teilbaum angewendet werden, um die aktuelle Restriktion einzuarbeiten. Wir werden also immer annehmen, dass die Teilbäume des aktuell betrachteten Knoten (oft auch als Wurzel des Teilbaums bezeichnet) bereits abgearbeitet sind.

Wir werden dabei darauf achten, folgende Einschränkung aufrecht zu erhalten. Wenn ein Knoten partiell ist, wird es ein Q-Knoten sein. Wir werden also nie einen partiellen P-Knoten konstruieren, außer es handelt sich um die Wurzel des reduzierten Teilbaums. Dann wird die Prozedur reduce allerdings auch abbrechen. Weiterhin wird ein konstruierter partieller Q-Knoten nur leere und volle Kinder besitzen, wobei die leeren ohne Beschränkung der Allgemeinheit links und die vollen rechts vorkommen und jeweils konsekutiv sind, außer es handelt sich um die Wurzel des reduzierten Teilbaums, dann wird die Prozedur aber auch wieder abbrechen.

1.2.2.1 Schablone P_0

Die Schablone P_0 in Abbildung 1.7 ist sehr einfach. Wir betrachten einen P-Knoten, an dem nur leere Teilbäume hängen. Somit ist nichts zu tun und wir steigen im Baum einfach weiter auf.

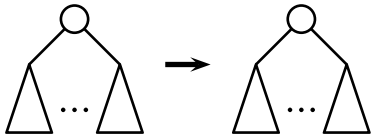


Abbildung 1.7: Skizze: Schablone P_0

1.2.2.2 Schablone P_1

Die Schablone P_1 in Abbildung 1.8 ist auch nicht viel schwerer. Wir betrachten einen P-Knoten, an dem nur volle Unterbäume hängen. Wir markieren daher die Wurzel als voll und gehen weiter bottom-up vor.

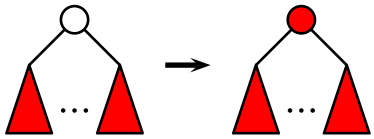


Abbildung 1.8: Skizze: Schablone P_1

1.2.2.3 Schablone P_2

Jetzt betrachten wir einen P-Knoten p, an dem nur volle und leere (also keine partiellen) Teilbäume hängen (siehe Abbildung 1.9). Weiter nehmen wir an, dass der Knoten p die Wurzel des reduzierten Teilbaums T_r ist. In diesem Fall fügen wir einen neuen P-Knoten als Kind der Wurzel ein und hängen alle vollen Teilbäume der ursprünglichen Wurzel an diesen Knoten. Da wir die Wurzel des reduzierten Teilbaumes erreicht haben, können wir mit der Umordnung des PQ-Baumes aufhören, da nun alle markierten Knoten aus F in den durch den PQ-Baum dargestellten Permutationen konsekutiv sind.

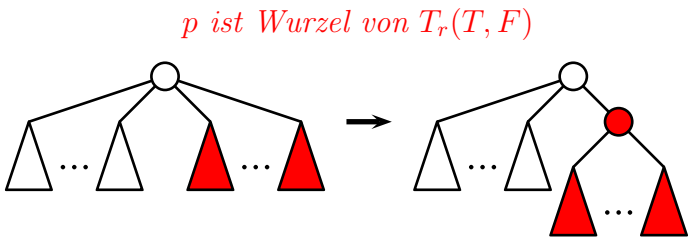


Abbildung 1.9: Skizze: Schablone P_2

Hierbei ist nur zu beachten, dass wir eigentlich nur echte PQ-Bäume konstruieren wollen. Hing also ursprünglich nur ein voller Teilbaum an der Wurzel, dann wäre nicht p die Wurzel des reduzierten Teilbaums, sondern dessen einziges volles Kind.

In jedem Falle überzeugt man sich leicht, dass alle Frontiers, die nach der Transformation eines äquivalenten PQ-Baumes abgelesen werden können, auch schon vorher abgelesen werden konnten. Des Weiteren haben wir durch die Transformation erreicht, dass alle Zeichen der aktuell betrachteten Restriktion nach der Transformation konsekutiv auftreten müssen.

1.2.2.4 Schablone P_3

Nun betrachten wir einen P-Knoten, an dem nur volle oder leere Teilbäume hängen, der aber noch nicht die Wurzel der reduzierten Teilbaumes ist (siehe Abbildung 1.10). Wir führen als neue Wurzel einen Q-Knoten ein. Alle leeren Kinder der ursprünglichen Wurzel belassen wir diesem P-Knoten und machen diesen P-Knoten zu einem Kind der neuen Wurzel. Weiter führen wir einen neuen P-Knoten ein, der ebenfalls ein Kind der neuen Wurzel wird und schenken ihm als Kinder alle vollen Teilbäume der ehemaligen Wurzel.

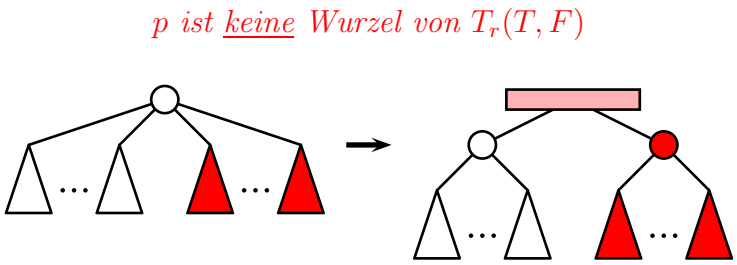


Abbildung 1.10: Skizze: Schablone P_3

Auch hier müssen wir wieder beachten, dass wir einen korrekten PQ-Baum generieren. Gab es vorher nur einen leeren oder nur einen vollen Unterbaum, so wird das entsprechende Kind der neuen Wurzel nicht wiederverwendet bzw. eingefügt, sondern der leere bzw. volle Unterbaum wird direkt an die neue Wurzel gehängt. Des Weiteren haben wir einen Q-Knoten konstruiert, der nur zwei Kinder besitzt. Dies würde der Definition eines echten PQ-Baumes widersprechen. Da wir jedoch weiter bottom-up den reduzierten Teilbaum abarbeiten müssen, werden wir später noch sehen, dass dieser Q-Knoten mit einem anderen Q-Knoten verschmolzen wird, so dass auch das kein Problem sein wird.

1.2.2.5 Schablone P_4

Betrachten wir nun den Fall, dass die Wurzel p ein P-Knoten ist, der neben leeren und vollen Kindern noch ein partielles Kind hat, das dann ein Q-Knoten sein muss. Dies ist in Abbildung 1.11 illustriert, wobei wir noch annehmen, dass der betrachtete Knoten die Wurzel des reduzierten Teilbaumes ist

Wir werden alle vollen Kinder, die direkt an der Wurzel hängen, unterhalb des partiellen Knotens einreihen. Da der partielle Knoten ein Q-Knoten ist, müssen die vollen Kinder an dem Ende hinzugefügt werden, an dem bereits volle Kinder hängen. Da die Reihenfolge der Kinder, die an der ursprünglichen Wurzel (einem P-Knoten) hingen, egal ist, werden wir die Kinder nicht direkt an den Q-Knoten

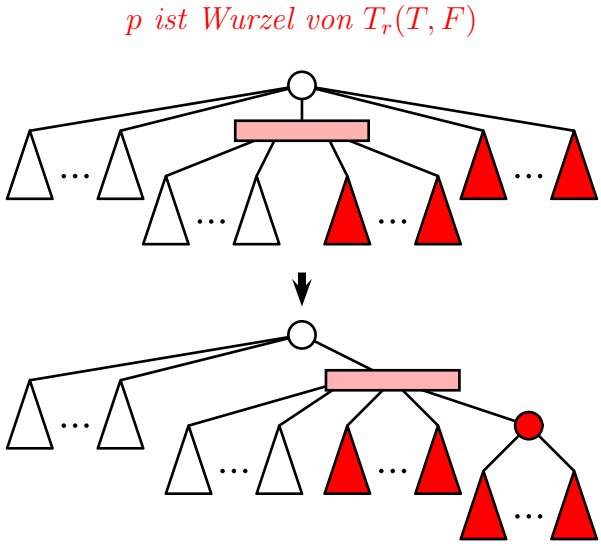


Abbildung 1.11: Skizze: Schablone P_4

hängen, sondern erst einen neuen P-Knoten zum äußersten Kind dieses Q-Knotens machen und daran die vollen Teilbäume anhängen. Dies ist natürlich nicht nötig, wenn an der ursprünglichen Wurzel nur ein voller Teilbaum gehangen hat. Falls der betrachtete P-Knoten keine leeren Kinder besitzt, wird dieser P-Knoten entfernt.

Auch hier machen wir uns wieder leicht klar, dass die Einschränkungen der Transformation lediglich die aktuell betrachtete Restriktion widerspiegelt und wir den Baum bzw. seine dargestellten Permutationen nicht mehr einschränken als nötig.

Wir müssen uns jetzt nur noch Gedanken machen, wenn der Q-Knoten im vorigen Schritt aus der Schablone P_3 entstanden ist. Dann hätte dieser Q-Knoten nur zwei Kinder gehabt. Besaß die ehemalige Wurzel p vorher noch mindestens einen vollen Teilbaum, so hat sich dieses Problem erledigt, das der Q-Knoten nun noch ein drittes Kind erhält. Hätte p vorher kein volles Kind gehabt (also nur einen partiellen Q-Knoten und lauter leere Bäume als Kinder), dann könnte p nicht die Wurzel des reduzierten Teilbaumes sein (dann hätte der partielle Q-Knoten die Wurzel des reduzierten Teilbaumes sein müssen). Dieser Fall kann also nicht auftreten.

1.2.2.6 Schablone P_5

Nun betrachten wir den analogen Fall, dass an der Wurzel ein partielles Kind hängt, aber der betrachtete Knoten nicht die Wurzel des reduzierten Teilbaumes ist. Dies ist in Abbildung 1.12 illustriert.

Wir machen also den Q-Knoten zur neuen Wurzel des betrachteten Teilbaumes und hängen die ehemalige Wurzel des betrachteten Teilbaumes mitsamt seiner leeren

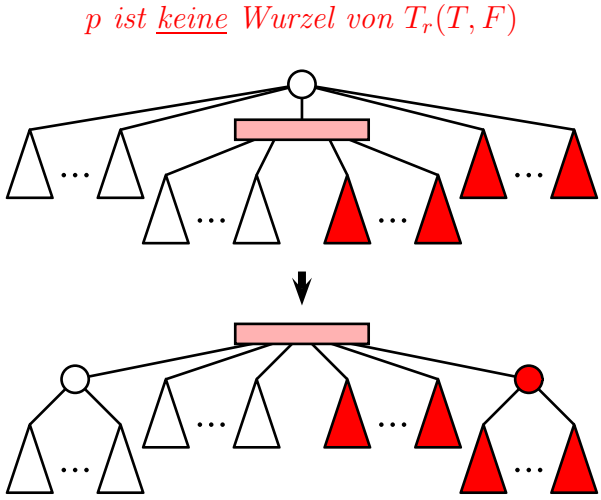


Abbildung 1.12: Skizze: Schablone P_5

Kinder ganz außen am leeren Ende an den Q-Knoten an. Die vollen Kinder der ehemaligen Wurzel des betrachteten Teilbaumes hängen wir am vollen Ende des Q-Knotens über einen neuen P-Knoten an. Man beachte wieder, dass die P-Knoten nicht benötigt werden, wenn es nur einen leeren bzw. vollen Teilbaum gibt, der an der Wurzel des betrachteten Teilbaumes hing.

Auch hier machen wir uns wieder leicht klar, dass die Einschränkungen der Transformation lediglich die aktuell betrachtete Restriktion widerspiegelt und wir den Baum bzw. seine dargestellten Permutationen nicht mehr einschränken als nötig.

Falls der Q-Knoten vorher aus der Schablone P_3 neu entstanden war, so erhält er nun mindestens eine weitere Kind, um der Definition eines echten PQ-Baumes zu genügen. Man beachte hierzu nur, dass die Wurzel p vorher mindestens einen leeren oder einen vollen Teilbaum besessen haben muss. Andernfalls hätte der P-Knoten p als Wurzel nur ein Kind besessen, was der Definition eines echten PQ-Baumes widerspricht.

1.2.2.7 Schablone P_6

Es bleibt noch der letzte Fall zu betrachten, dass die Wurzel des betrachteten Teilbaumes ein P-Knoten ist, an der neben vollen und leeren Teilbäume genau zwei partielle Kinder hängen (die dann wieder Q-Knoten sein müssen). Dies ist in Abbildung 1.13 illustriert.

Man überlegt sich leicht, dass die Wurzel p des betrachteten Teilbaumes dann auch die Wurzel des reduzierten Teilbaumes sein muss, da andernfalls die aktuell betrach-

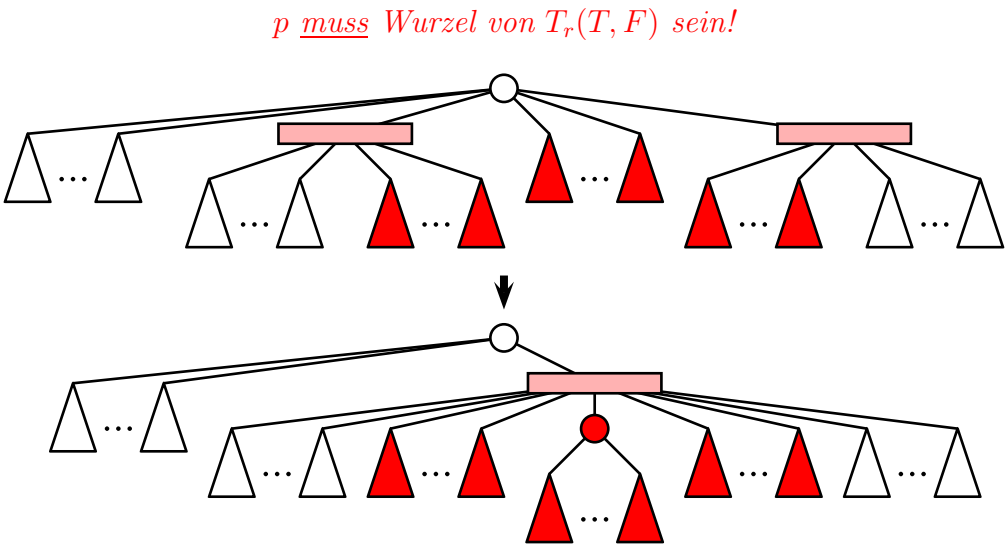


Abbildung 1.13: Skizze: Schablone P_6

tete Restriktion sich nicht mit den Permutationen des bereits konstruierten PQ-Baumes unter ein Dach bringen lässt.

Wir vereinen einfach die beiden Q-Knoten zu einem neuen und hängen die vollen Kinder der Wurzel des betrachteten Teilbaumes über einen neu einzuführenden P-Knoten in der Mitte des verschmolzenen Q-Knoten ein.

Falls hier einer oder beide der betrachteten Q-Knoten aus der Schablone P_3 entstanden ist, so erhält er auch hier wieder genügend zusätzliche Kinder, so dass die Eigenschaft eines echten PQ-Baumes wiederhergestellt wird.

1.2.2.8 Schablone Q_0

Nun haben wir alle Schablonen für P-Knoten als Wurzeln angegeben. Es folgen die Schablonen, in denen die Wurzel des betrachteten Teilbaumes ein Q-Knoten ist. Die Schablone Q_0 ist analog zur Schablone P_0 wieder völlig simpel. Alle Kinder sind leer und es ist also nichts zu tun (siehe Abbildung 1.14) und wir steigen im Baum weiter auf.

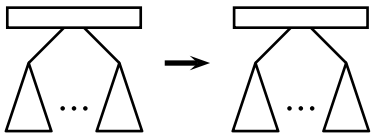


Abbildung 1.14: Skizze: Schablone Q_0

1.2.2.9 Schablone Q_1

Auch die Schablone Q_1 ist völlig analog zur Schablone P_1. Alle Kinder sind voll und daher markieren wir den Q-Knoten als voll und arbeiten uns weiter bottom-up durch den reduzierten Teilbaum (siehe auch Abbildung 1.15).

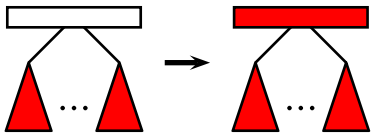


Abbildung 1.15: Skizze: Schablone Q_1

1.2.2.10 Schablone Q_2

Betrachten wir nun den Fall, dass sowohl volle wie leere Teilbäume an einem Q-Knoten hängen (siehe Abbildung 1.16). In diesem Fall überprüfen wir nur, ob die vollen Kinder konsekutiv vorkommen, denn dann ist die Wurzel ein partieller Q-Knoten, und wir steigen einfach im Baum weiter auf. Falls die vollen Kinder nicht konsekutiv vorkommen, kann die Restriktion F nicht in den PQ-Baum eingearbeitet werden und der Algorithmus bricht ab und meldet, dass es keine Lösung für $\mathcal{F}$ geben kann.

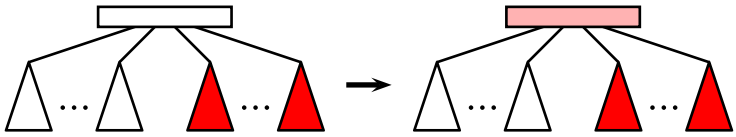


Abbildung 1.16: Skizze: Schablone Q_2

1.2.2.11 Schablone Q_3

Kommen wir also gleich zu dem Fall, in dem an der Wurzel p des aktuell betrachteten Teilbaumes volle und leere sowie genau ein partieller Q-Knoten hängt. Zuerst prüfen wir, ob die vollen Kinder konsekutiv sind und alle entweder rechts oder links vom partiellen Kind vorkommen. Falls nicht, kann die Restriktion F nicht in den PQ-Baum eingearbeitet werden und der Algorithmus bricht ab und meldet, dass es keine Lösung für $\mathcal{F}$ geben kann. Andernfalls verschmelzen wir nun einfach den partiellen Q-Knoten mit der Wurzel (die ebenfalls ein Q-Knoten ist), wie in Abbildung 1.17 illustriert. Falls der partielle Q-Knoten aus der Schablone P_3 entstanden ist, erhält er auch hier wieder ausreichend viele zusätzliche Kinder.

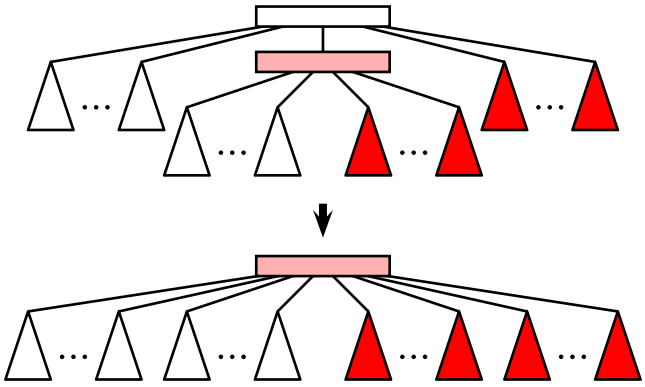


Abbildung 1.17: Skizze: Schablone Q_3

1.2.2.12 Schablone Q_4

Als letzter Fall bleibt der Fall, dass an der Wurzel des aktuell betrachteten Teilbaumes zwei partielle Q-Knoten hängen (sowie volle und leere Teilbäume). Zuerst prüfen wir, ob alle vollen Kinder konsekutive zwischen den beiden partiellen Kindern vorkommen. Falls nicht, kann die Restriktion F nicht in den PQ-Baum eingearbeitet werden und der Algorithmus bricht ab und meldet, dass es keine Lösung für $\mathcal{F}$ geben kann. Andernfalls vereinen wir auch hier die drei Q-Knoten zu einem neuen wie in Abbildung 1.18 angegeben. In diesem Fall muss der betrachtete Q-Knoten bereits die Wurzel des reduzierten Teilbaumes sein und die Prozedur bricht ab. Falls einer der beiden partiellen Q-Knoten aus der Schablone P_3 entstanden ist, erhält er auch hier wieder ausreichend viele zusätzliche Kinder.

23.04.24

p muss Wurzel sein!

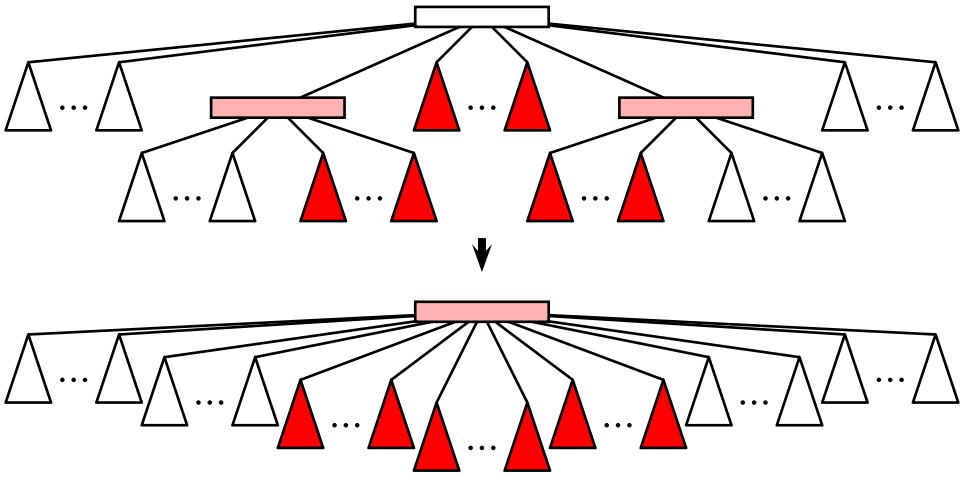


Abbildung 1.18: Skizze: Schablone Q_4

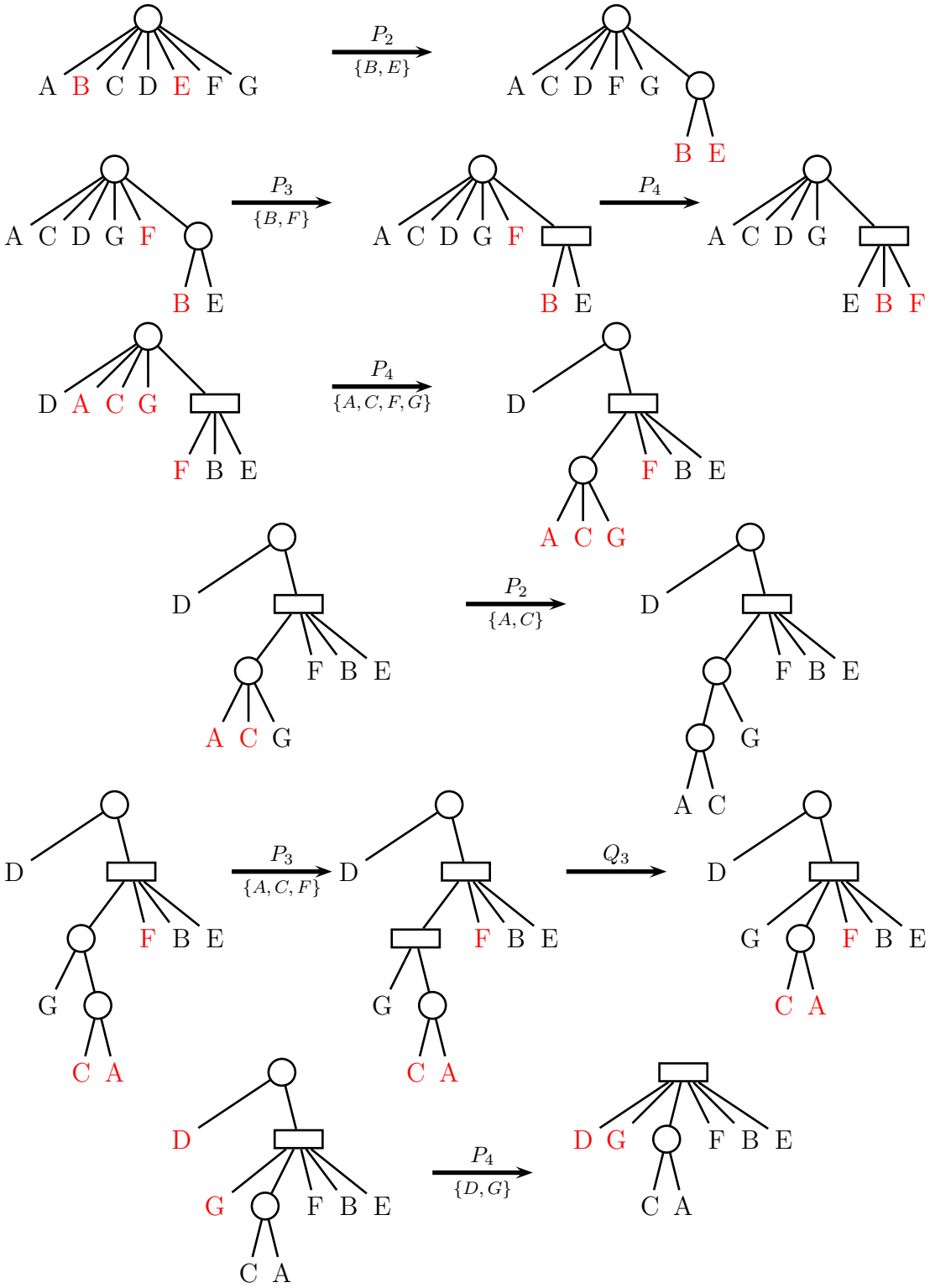


Abbildung 1.19: Beispiel: Konstruktion eines PQ-Baumes

In der Abbildung 1.19 auf Seite 17 ist ein Beispiel zur Konstruktion eines PQ-Baumes für die folgende Restriktionsmenge angegeben:

$$\mathcal{F} = \Big\{\{B,E\},\{B,F\},\{A,C,F,G\},\{A,C\},\{A,C,F\},\{D,G\}\Big\}.$$

1.2.3 Korrektheit

In diesem Abschnitt wollen wir kurz die Korrektheit beweisen, d.h. dass der konstruierte PQ-Baum tatsächlich die gewünschte Menge von Permutationen bezüglich der vorgegebenen Restriktionen darstellt. Dazu definieren wir den *universellen PQ-Baum* $T(\Sigma, F)$ für ein Alphabet Σ und eine Restriktion $F = \{a_{i_1}, \ldots, a_{i_r}\}$. Die Wurzel des universellen PQ-Baumes ist ein P-Knoten an dem sich lauter Blätter, je eines für jedes Zeichen aus $\Sigma \setminus F$, und ein weiterer P-Knoten hängen, an dem sich seinerseits lauter Blätter befinden, je eines für jedes Element aus F.

Theorem 1.8 *Sei T ein beliebiger echter PQ-Baum und $F \subseteq \Sigma$. Dann gilt:*

$$\text{cons}(\text{reduce}(T,F)) = \text{cons}(T) \cap \text{cons}(T(\Sigma,F)).$$

Beweis: Zuerst führen wir zwei Abkürzungen ein:

$$\begin{aligned} A &:= \text{cons}(\text{reduce}(\text{T},\text{F})) \\ B &:= \text{cons}(T) \cap \text{cons}(T(\Sigma,F)) \end{aligned}$$

Wir zeigen jeweils die entsprechende Mengeninklusion.

$\boldsymbol{A \subseteq B}$ **:** Ist $A = \emptyset$, so ist nichts zu zeigen. Ansonsten existieren

$$\pi \in \text{cons}(\text{reduce}(T,F)) \quad \text{und} \quad T' \cong \text{reduce}(T,F) \quad \text{mit} \quad f(T') = \pi.$$

Nach Konstruktion gilt $\pi \in \text{cons}(T)$. Andererseits gilt nach Konstruktion (vgl. die einzelnen Schablonen) für jeden erfolgreich abgearbeiteten Knoten x im reduzierten Teilbaum $T_r(T,F)$ genau eine der folgenden Aussagen:

- x ist ein Blatt (leer oder voll),
- x ist ein voller oder leerer P-Knoten,
- x ist ein Q-Knoten, der nur leere oder volle Unterbäume besitzt und dessen volle markierte Unterbäume alle konsekutiv vorkommen.

Somit kommen alle markierten Blätter aus F im jeweiligen betrachteten Teilbaum konsekutiv vor und am Ende ist nur die Wurzel ein partieller Knoten, der dann ein Q-Knoten sein muss. Daraus folgt unmittelbar, dass $\pi \in \text{cons}(T(\Sigma,F))$.

***B* ⊆ *A* :** Ist $B = \emptyset$, so ist nichts zu zeigen. Sei also $\pi \in B$. Sei T' so gewählt, dass $T' \cong T$ und $f(T') = \pi$. Nach Voraussetzung kommen die Zeichen aus F in π hintereinander vor. Somit hat im reduzierten Teilbaum $T_r(T', F)$ jeder Knoten außer der Wurzel maximal ein partielles Kind und die Wurzel maximal zwei partielle Kinder. Jeder partielle Knoten wird nach Konstruktion durch einen Q-Knoten ersetzt, dessen Kinder entweder alle voll oder leer sind und deren volle Unterbäume konsekutiv vorkommen. Damit ist bei der bottom-up-Vorgehensweise immer eine Schablone anwendbar und es gilt $\pi \in \text{cons}(\text{reduce}(T', F))$. Damit ist auch $\pi \in \text{cons}(\text{reduce}(T, F))$, da die Anwendbarkeit der Regeln nicht von der Reihenfolger der Kinder eines P-Knotens oder der Umkehrbarkeit der Reihenfolge der Kinder eines Q-Knotens abhängt. ■

1.2.4 Implementierung

An dieser Stelle müssen wir noch ein paar Hinweise zur effizienten Implementierung geben, da mit ein paar Tricks die Laufzeit zur Generierung von PQ-Bäumen drastisch gesenkt werden kann. Überlegen wir uns zuerst die Eingabegröße. Die Eingabe selbst ist $(\Sigma, \mathcal{F})$ und somit ist die Eingabegröße $\Theta(|\Sigma| + \sum_{F \in \mathcal{F}} |F|)$. In der Regel gilt dabei $|\Sigma| = O(\sum_{F \in \mathcal{F}} |F|)$.

Betrachten wir den Baum T, auf den wir die Operation $\text{reduce}(T, F)$ loslassen. Mit $T_r(T, F)$ bezeichnen wir den reduzierten Teilbaum von T bezüglich F. Dieser ist über die niedrigste Wurzel beschrieben, so dass alle aus F markierten Blätter Nachfahren dieser Wurzel sind. Der Baum $T_r(T, F)$ selbst besteht aus allen Nachfahren dieser Wurzel. Offensichtlich läuft die Hauptarbeit innerhalb dieses Teilbaumes ab. Dieser Teilbaum von T sind in Abbildung 1.20 schematisch rot schraffiert dargestellt.

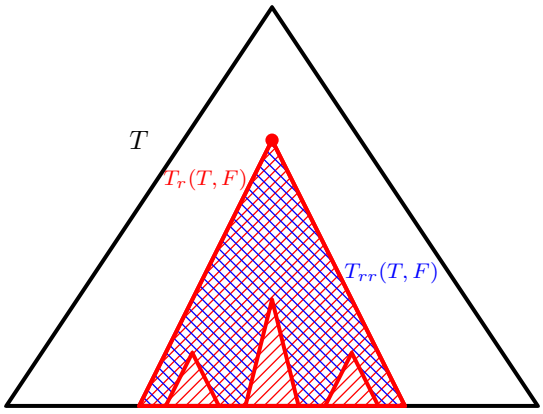


Abbildung 1.20: Skizze: Bearbeitete Teilbäume bei $\text{reduce}(T, F)$

Aber selbst bei nur zwei markierten Blättern, kann dieser Teilbaum sehr groß werden, bspw. zwei Blätter, deren niedrigster gemeinsamer Vorfahre die Wurzel des Baumes T ist. Also betrachten wir den so genannten *relevanten reduzierten Teilbaum* $T_{rr}(T, F)$ Dieser besteht aus dem kleinsten zusammenhängenden Teilgraphen von T, der alle markierten Blätter aus F enthält. Offensichtlich ist $T_{rr}(T, F)$ ein Teilbaum von $T_r(T, F)$, wobei die Wurzeln der beiden Teilbäume von T dieselben sind. Man kann auch sagen, dass der relevante reduzierte Teilbaum aus dem reduzierten Teilbaum entsteht, indem man leere Teilbäume herausschneidet. Dieser relevante reduzierte Teilbaum von T sind in Abbildung 1.20 schematisch blau schraffiert dargestellt.

Wir werden zeigen, dass die gesamte Arbeit im Wesentlichen im Teilbaum $T_{rr}(T, F)$ erledigt wird und diese somit für eine reduce-Operation proportional zu $|T_{rr}(T, F)|$ sein wird. Somit ergibt sich für die Konstruktion eines PQ-Baumes für eine gegebene Menge $\mathcal{F} = \{F_1, \ldots, F_n\}$ von Restriktionen die folgende Laufzeit von

$$\sum_{i=1}^{k} O(|T_{rr}(T_{i-1}, F_i)|),$$

wobei $T_0 = T(\Sigma)$ ist und $T_i = \text{reduce}(T_{i-1}, F_i)$. Wir müssen uns jetzt noch um zwei Dinge Gedanken machen: Wie kann man die obige Laufzeit besser, anschaulicher abschätzen und wie kann man den relevanten reduzierten Teilbaum $T_{rr}(T, F)$ in Zeit $O(|T_{rr}(T, F)|)$ ermitteln und darin die Schablonen mit derselben Zeitkomplexität anwenden.

Zuerst kümmern wir uns um die Bestimmung des relevanten reduzierten Teilbaumes. Dazu müssen wir uns aber erst noch ein paar genauere Gedanken zur Implementierung des PQ-Baumes selbst machen. Die Kinder eines Knotens werden als doppelt verkettete Liste abgespeichert, da ja für die Anzahl der Kinder a priori keine obere Schranke bekannt ist. Bei den Kindern eines P-Knoten ist die Reihenfolge, in der sie in der doppelt verketteten Liste abgespeichert werden, beliebig. Bei den Kindern eines Q-Knoten respektiert die Reihenfolge innerhalb der doppelt verketteten Liste gerade die Ordnung, in der sie unter dem Q-Knoten hängen.

Zusätzlich werden wir zum bottom-up Aufsteigen auch noch von jedem Knoten den zugehörigen Elter wissen wollen. Leider wird sich herausstellen, dass es zu aufwendig ist, für jeden Knoten einen Verweis zu seinem Elter aktuell zu halten. Daher werden wie folgt vorgehen. Jedes Kind eines P-Knotens erhält jeweils eine Verweis auf seinen Elter. Bei Q-Knoten werden nur die beiden äußersten Kinder (also das älteste und das jüngste Kind) einen Verweis auf ihren Elter erhalten. Wir werden im Folgenden sehen, dass dies völlig ausreichend sein wird.

In Abbildung 1.21 ist der Algorithmus zum Ermitteln des relevanten reduzierten Teilbaumes im Pseudo-Code angegeben. Prinzipiell versuchen wir ausgehend von

Find_Tree (tree T, set F)

```
begin
    int sectors := 0;
    queue free := ∅;
    set blocked := ∅;
    tree T_rr := (∅, ∅);
    forall (f ∈ F) do
        free.add(f);
        V(T_rr) := V(T_rr) ∪ {f};
    while (free.size() + sectors > 1) do
        if (free.size = 0) then
            return (∅, ∅);                    /* F leads to a contradiction */
        else
            v := free.remove_FIFO();
            if (parent(v) ≠ nil) then
                if (parent(v) ∉ V(T_rr)) then
                    V(T_rr) := V(T_rr) ∪ {parent(v)};
                    free.add(parent(v));
                E(T_rr) := E(T_rr) ∪ {{v, parent(v)}};
            else
                blocked.add(v);
                if (∃x ∈ N(v) s.t. parent(x) ≠ nil) then
                    if (parent(x) ∉ V(T_rr)) then
                        V(T_rr) := V(T_rr) ∪ {parent(x)};
                        free.add(parent(x));
                    let y s.t. x ⇌ v ⇌ y;
                    let S be the sector containing v;
                    if (y ∈ blocked) then
                        sectors−−;
                    forall (s ∈ S) do
                        blocked.remove(s);
                        E(T_rr) := E(T_rr) ∪ {{s, parent(x)}};
                else if (both neighbors of v are blocked) then
                    sectors−−;
                else if (both neighbors of v are not blocked) then
                    sectors++;
    return T_rr;
end
```

Abbildung 1.21: Algorithmus: Ermittlung von $T_{rr}(T, F)$

der Menge der markierten Blätter aus F einen zusammenhängen Teilgraphen von T zu konstruieren, indem wir mit Hilfe der Verweise auf die Eltern im Baum T von den Blätter aus F nach oben laufen. Falls wir bereits beim Auffinden des relevanten reduzierten Baumes feststellen, dass sich die gegeben Restriktion nicht widerspruchsfrei einarbeiten lässt, gibt der Algorithmus bereits dann einen leeren Baum zurück.

25.04.24

Um diesen Algorithmus genauer verstehen zu können, müssen wir erst noch ein paar Notationen vereinbaren. Ein Knoten heißt *aktiv*, wenn wir im Algorithmus festgestellt haben, dass er ein Vorfahr eines markierten Blattes aus F ist. Ein aktiver Knoten heißt *frei*, wenn die Kante zu seinem Elter im Algorithmus noch nicht betrachtet wurde (d.h. abgefragt wurde). Ein aktiver Knoten heißt *blockiert*, wenn wir festgestellt haben, dass wir seinen Elter nicht kennen. Es kann also durchaus freie Knoten geben, die keinen Verweis auf ihren Elter haben, aber deren Eltern selbst schon aktiv sind (wir haben dies nur noch nicht bemerkt). Dies ist auch in Abbildung 1.22 illustriert.

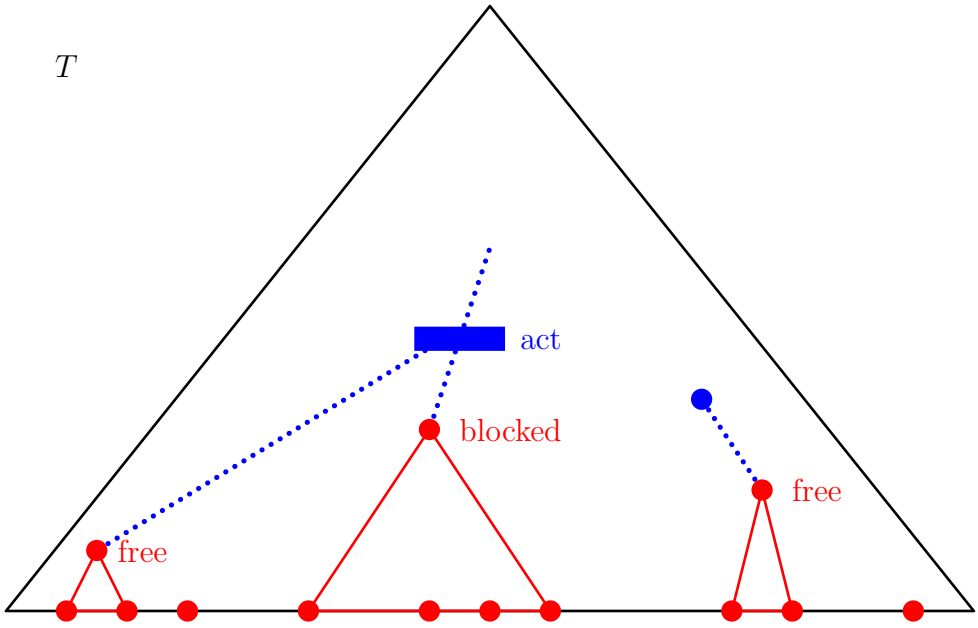


Abbildung 1.22: Skizze: Freie und blockierte Knoten im Teilbaum bei reduce(T, F)

Hierzu halten wir eine FIFO-Queue für die Menge free der so genannten *freien Knoten*. Weiterhin speichern wir eine Menge blocked der so genannten *blockierten Knoten*. Diese kann sehr einfach mit Hilfe eines Booleschen Feldes bzw. besser mit Flags zu den einzelnen Knoten implementiert werden.

Wenn wir jetzt versuchen den kleinsten zusammenhängenden Teilbaum zu konstruieren, der alle markierten Blätter enthält, gehen wir bottom-up durch den Baum und konstruieren dabei viele kleine Teilbäume, die durch Verschmelzen letztendlich

im Wesentlichen den relevanten reduzierten Teilbaum ergeben. Zu Beginn besteht diese Menge der Teilbäume aus allen markierten Blättern.

Eine maximale Folge von blockierten Knoten, die aufeinander folgende Kinder desselben Knotens sind (der dann ein Q-Knoten sein muss), nennen wir einen *Sektor*. Beachte, dass ein Sektor nie eines der äußersten Kinder eines Q-Knoten enthalten kann, da diese nach Definition ihren Elter kennen.

Zuerst überlegen wir uns, wann wir die Prozedur abbrechen. Wenn es nur noch einen freien Knoten und keine blockierten Knoten (und damit auch keine Sektoren) mehr gibt, brechen wir ab. Dann haben wir entweder die Wurzel des relevanten reduzierten Teilbaumes gefunden oder wir befinden uns mit der freien Wurzel bereits auf dem Weg von der gesuchten Wurzel zur Wurzel des Gesamtbaumes T. Wir wissen ja leider nicht in welcher Reihenfolge wir die Knoten des relevanten reduzierten Teilbaumes aufsuchen. Es kann durchaus passieren, dass wir die Wurzel recht schnell finden und den restlichen Teil des Baumes noch gar nicht richtig untersucht haben. Dies passiert insbesondere dann, wenn an der Wurzel bereits ein markiertes Blatt hängt. Dies ist im linken Teil in Abbildung 1.23 illustriert.

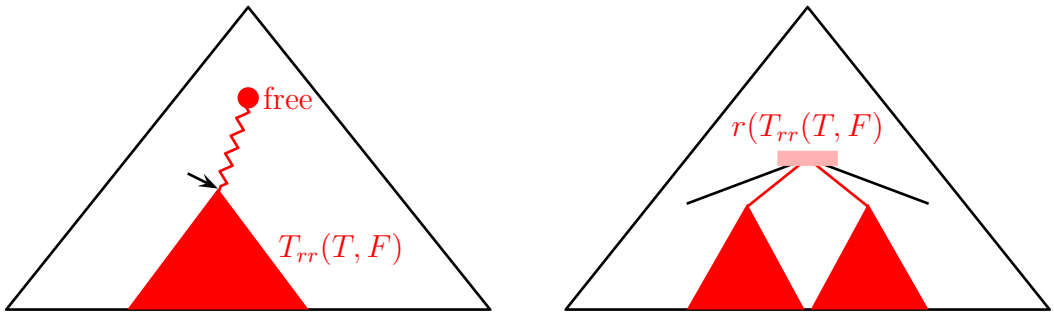


Abbildung 1.23: Skizze: Abbruch bei reduce(T, F)

Andererseits brechen wir ab, wenn wir nur noch einen Sektor bearbeiten. Der Elter der Knoten dieses Sektors muss dann die gesuchte Wurzel des relevanten reduzierten Teilbaumes sein. Dies ist im rechten Teil in Abbildung 1.23 illustriert.

Wann immer wir mindestens zwei Sektoren und keinen freien Knoten mehr besitzen, ist klar, dass wir im Fehlerfall sind, d.h für die gegebene Menge $\mathcal{F}$ von Restriktionen kann es keinen korrespondierenden PQ-Baum geben. Andernfalls müssten wir die Möglichkeit haben, diese beide Sektoren mithilfe von freien Knoten irgendwie zu verschmelzen, die es aber nicht gibt.

Wie geht unser Algorithmus also weiter vor, wenn es noch freie Wurzeln gibt? Er nimmt eine solche freie Wurzel v her und testet, ob der Elter von v bekannt ist. Falls ja, fügt er die Kante zum Elter in den relevanten reduzierten Teilbaum ein. Ist der

Elter selbst noch nicht im relevanten reduzierten Teilbaum enthalten, so wird auch dieser darin aufgenommen und der Elter selbst als frei markiert.

Andernfalls wird der betrachtete Knoten v als blockiert erkannt. Jetzt müssen wir nur die Anzahl der Sektoren aktualisieren. Dazu stellen wir zunächst fest, ob v ein direktes Geschwister x (Nachbar in der doppelt verketteten Liste) besitzt, der seinen Elter schon kennt. Wenn ja, dann sei y das andere direkte Geschwister von v (man überlege sich, dass dieses existieren muss). Die Folge (x, v, y) kommt also so oder in umgekehrter Reihenfolge in der doppelt verketteten Liste der Geschwister vor. Mit S bezeichnen wir jetzt den Sektor, der v enthält (wie wir diesen bestimmen, ist im Algorithmus nicht explizit angegeben und die technischen Details seien dem Leser überlassen).

Da S nun mit v einen blockierten Knoten enthält, der ein Geschwister hat, der seinen Elter kennt, können wir jetzt auch allen Knoten dieses Sektors S seinen Elter zuweisen und die die entsprechenden Kanten in den relevanten reduzierten Teilbaum aufnehmen. War y vorher blockiert, so reduziert sich die Anzahl der Sektoren um eins, da alle Knoten im Sektor von y jetzt ihren Elter kennen.

Es bleibt der Fall übrig, in dem kein direktes Geschwister von v seinen Elter kennt. In diesem Fall muss jetzt nur noch die Anzahl der Sektoren aktualisiert werden. Ist v ein isolierter blockierter Knoten (besitzt also kein blockiertes Geschwister), so muss die Anzahl der Sektoren um eins erhöht werden. Waren beide Geschwister blockiert, so werden diese Sektoren mithilfe von v zu einem Sektor verschmolzen und die Anzahl der Sektoren sinkt um eins. War genau ein direktes Geschwister blockiert, so erweitert v diesen Sektor und die Anzahl der Sektoren bleibt unverändert.

Damit haben wir die Korrektheit des Algorithmus zur Ermittlung des relevanten reduzierten Teilbaumes bewiesen. Bleibt am Ende des Algorithmus eine freie Wurzel oder ein Sektor übrig, so haben wir den relevanten reduzierten Teilbaum im Wesentlichen gefunden. Im ersten Fall befinden wir uns mit der freien Wurzel auf dem Pfad von der eigentlichen Wurzel zur Wurzel der Gesamtbaumes. Durch Absteigen können wir die gesuchte Wurzel als den Knoten identifizieren, an dem eine Verzweigung auftritt (der Leser möge sich überlegen, wie). Im zweiten Fall ist, wie gesagt, der Elter der blockierten Knoten im gefunden Sektor die gesuchte Wurzel.

Eigentlich haben wir im zweiten Fall die Wurzel des reduzierten Teilbaumes nicht wirklich gefunden, sondern nur einige (konsekutive) Kinder der Wurzel, die einen Sektor bilden. An die Wurzel selbst kommen wir ohne größeren Zeitaufwand eigentlich auch gar nicht heran. Algorithmisch ist es jedoch völlig ausreichend, dass wir den Sektor ermittelt haben (mit seinen beiden Knoten am Rand). In den zutreffenden Schablonen (Q_2, Q_3 und Q_4) werden die Kinder des bzw. der partiellen Q-Knoten ja in die Geschwisterliste der Wurzel eingehängt. Dazu muss man die Wurzel des reduzierten Teilbaumes überhaupt nicht kennen, sondern nur den Zugriff an der

richtigen Stelle auf dessen Kinderliste haben. Diese ist durch die doppelt verkettete Geschwisterliste jedoch gegeben, da der Rand des Sektors genau auf diese Stellen verweist, wo die Kinderliste(n) eingefügt werden müssen.

Implementierungstechnisch müssen wir noch darauf hinweisen, dass wir beim Verschmelzen von Sektoren, wovon einer der Sektoren seinen Elter kennt, nicht permanent die Elter-Informationen aktualisieren. Nach Einbau einer Restriktion müssen wir die Elter-Informationen von Kindern von Q-Knoten, die nicht das älteste oder jüngste Kind sind, wieder löschen. Sonst könnten beim Einbauen anderer Restriktionen alte, nicht mehr aktuelle Verweise auf Eltern überleben.

Dazu müssen wir uns bei der Bestimmung des reduzierten Teilbaumes merken, welche Kinder von Q-Knoten, die dann nicht älteste bzw. jüngste Kinder sind, ihren Verweis auf ihr Elter vorübergehend gesetzt haben und diese Verweise am Ende wieder löschen. Dies verursacht keinen wesentlichen zeitlichen Zusatzaufwand. Ein allgemeines Löschen aller Elterinformationen von Kindern von Q-Knoten wäre hingegen viel zu teuer.

1.2.5 Laufzeitanalyse

Wir haben bereits behauptet, dass die Lauzeit mit

$$\sum_{i=1}^{k} O(|T_{rr}(T_{i-1}, F_i)|)$$

abgeschätzt werden kann, wobei $T_0 = T(\Sigma) = T(\Sigma, \emptyset)$ ist und $T_i = \text{reduce}(T_{i-1}, F_i)$. Zuerst wollen wir uns noch wirklich überlegen, dass diese Behauptung stimmt. Das erste, was wir uns überlegen müssen, ist, dass die Prozedur `Find_Tree` den relevanten reduzierten Teilbaum für eine Menge von Restriktionen F in Zeit $|T_{rr}(T, F)|$ erkennen kann. Jeder Knoten des relevanten reduzierte Teilbaum wird nach dem er zu ersten Mal aktiv wird, in die FIFO-Queue aufgenommen, was in konstanter Zeit möglich ist. Auch die Feststellung, ob ein Knoten blockiert ist oder nicht, ist in konstanter Zeit möglich. Die Befreiung blockierter Knoten ist proportional zur Anzahl der Knoten im Sektor. Somit ist der Zeitaufwand pro blockierten Knoten in einem Sektor auch wieder konstant. Da kein Knoten, dessen Blockierung aufgehoben wurde, wieder blockiert werden kann, gilt die Behauptung.

Dem Leser sei als Übungsaufgabe überlassen, dass man anschließend in linearer Zeit alle Knoten des relevanten reduzierten Teilbaums als volle oder partielle Knoten erkennen kann. Darüber hinaus ist es in der gleichen Zeit möglich, eine Aufzählung der Knoten zu konstruieren, so dass alle Kinder eines Knotens $v \in V(T_{rr})$ in dieser Aufzählung vor dem Knoten v vorkommen.

Das einzige weitere Problem ist, dass ja aus dem relevanten reduzierte Teilbaum Kanten herausführen, an denen andere Knoten des reduzierten Teilbaumes hängen, die jedoch nicht zum relevanten reduzierten Teilbaum gehören (in Abbildung 1.20 sind dies Kanten aus dem blauen in den roten Bereich). Wenn wir für jede solche Kante nachher bei der Anwendung der Schablonen den Elterverweis in den relevanten reduzierten Teilbaum aktualisieren müssten, hätten wir ein Problem. Dies ist jedoch, wie wir gleich sehen werden, glücklicherweise nicht der Fall.

Im Folgenden werden wir zeigen, dass wir die Kosten (d.h. den Zeitbedarf), die bei Anwendung einer Schablone entstehen, so auf Knoten des relevanten reduzierten Teilbaumes $T_{rr}(T_{i-1})$ verteilen können, dass jedem Knoten des relevanten reduzierten Teilbaumes $T_{rr}(T_{i-1})$ nur konstante Kosten zugeordnet werden. Dazu bemerken wir vorab, dass alle Knoten voller Teilbäume, die beim Abarbeiten von Schablonen betrachtet werden und denen Kosten zugeordnet werden, auch bereits Knoten im relevanten reduzierten Teilbaum $T_{rr}(T_{i-1})$ waren. Die konstanten Kosten pro Knoten werden dabei nur auf volle Kinder des gerade betrachteten Knotens oder auf Knoten, die in diesem Schritt eliminiert werden (bzw. sich nach Anwendung der Schablone außerhalb des relevanten reduzierten teilbaumes befindet), oder auf die (partielle) Wurzel des betrachteten relevanten reduzierten Teilbaums verteilt.

Wir werden weiterhin ausnutzen, dass wir bei der Erkennung des relevanten reduzierten Teilbaumes bereits für jeden Knoten festgestellt haben, ob er voll oder partiell ist (der relevante reduzierte Teilbaum selbst enthält ja keine leeren Knoten) und auch welches seine partiellen und vollen Kinder jeweils sind.

Für P-Knoten sammeln wir die vollen Kinder beispielsweise in einer eigenen zusätzlichen Geschwisterliste oder die vollen Kinder werden an einem Ende der Geschwisterliste gehalten. Die partiellen Kinder werden in maximal zwei Verweisen des Elters gehalten (wenn ein P-Knoten mehr als zwei partielle Kinder hat, kann die aktuelle Restriktion nicht eingearbeitet werden), ebenso der Verweis auf den Beginn der Liste der vollen Kinder.

Für Q-Knoten sind die vollen Kinder immer an einem Ende der Geschwisterliste zu finden, außer wir haben nur noch genau einen Sektor, der dann die konsekutiven vollen und partiellen Kinder beinhaltet und quasi die Wurzel des relevanten reduzierten Teilbaumes ist (bei mehreren Sektoren, kann die Restriktion ja nicht eingearbeitet werden). Dazu merken wir uns für jeden Q-Knoten auch die Verweise auf das älteste und jüngste Kind in der Geschwisterliste seiner Kinder.

Damit kann in Zeit proportional zu den nichtleeren (d.h. den partiellen oder vollen) Kindern erkannt werden, welche Schablone anzuwenden ist (bzw. dass die aktuelle Restriktion sich nicht einarbeiten lässt). Damit ist auch die Entfernung aller nichtleerer (bzw. aller voller) Kinder aus einer Geschwisterliste in Zeit proportional zur Anzahl aller nichtleerer (bzw. aller voller) Kinder möglich. Des Weiteren merken wir

uns bei Abarbeitung partieller Q-Knoten, an welchem Ende die vollen bzw. leeren Kinder hängen.

1.2.5.1 Die Schablonen P_0 und Q_0

Zuerst bemerken wir, dass die Schablonen P_0 und Q_0 nie angewendet werden, da diese nur außerhalb des relevanten reduzierten Teilbaums anwendbar sind.

1.2.5.2 Die Schablonen P_1 und Q_1

Bei den Schablonen P_1 und Q_1 sind keine Veränderungen des eigentlichen PQ-Baumes durchzuführen. Es muss nur die Situation festgestellt werden, was, wie oben bemerkt, in Zeit proportional zur Anzahl der vollen Teilbäume möglich ist. Somit können wir die Kosten auf jedes volle Kind des betrachteten P-Knotens so verteilen, dass jedes volle Kind nur konstante Kosten erhält.

1.2.5.3 Die Schablone P_2

Bei der Schablone P_2 (siehe Abbildung 1.9 auf Seite 10) bleiben die Knoten außerhalb des relevanten reduzierten Teilbaumes unverändert und auch die Wurzel ändert sich nicht. Wir müssen nur die Wurzeln der vollen Teilbäume und den neu eingefügten Knoten aktualisieren. Beides geht in Zeit proportional zu Anzahl der vollen Kinder des betrachteten P-Knotens. Somit können wir jedem vollen Kind des betrachteten P-Knotens konstante Kosten zuordnen. Da danach die Prozedur abbricht, können dem neu generierten P-Knoten keine Kosten zugeordnet werden.

1.2.5.4 Die Schablone P_3

Bei der Schablone P_3 (siehe Abbildung 1.10 auf Seite 11) verwenden wir den Trick, dass wir den aktuellen Knoten als Elter der leeren Teilbäume belassen. Somit muss ebenfalls nur an den Wurzeln der vollen Teilbäumen und am neu eingeführten Q-Knoten etwas verändert werden. Dass wir dabei auch den Elter-Zeiger der alten Wurzel des betrachteten Teilbaumes aktualisieren müssen, ist nicht weiter tragisch, da dies nur konstante Kosten pro Schablone (und somit pro Knoten des betrachteten relevanten reduzierten Teilbaumes) verursacht. Diese Kosten können wir nun den Wurzeln der vollen Teilbäume zuordnen, die ja dann zu Enkeln oder ggf. Kindern werden. Falls vorher nur ein volles Kind existierte, bekommt dieses die Kosten. Da partiellen Knoten als Kind keine Kosten zugewiesen werden, bekommen auch hier die die neu eignefügten Knoten später keine Kosten zugewiesen.

1.2.5.5 Die Schablone P_4

Bei der Schablone P_4 (siehe Abbildung 1.11 auf Seite 12) ist dies wieder offensichtlich, da wir nur ein paar volle Teilbäume umhängen und einen neuen P-Knoten einführen. Da am betrachteten P-Knoten mindestens ein voller Teilbaum hängen muss (sonst könnte er nicht die Wurzel des relevanten reduzierten Teilbaumes sein), können die Kosten auf die vollen Kinder des P-Knotens so verteilt werden, dass jedes volle Kind nur konstante Kosten zahlen muss. Wenn es nur ein vollen Kind am P-Knoten gab, muss der neuen P-Knoten als Kind des Q-Konten nicht eingefügt werden. Wenn es keine leeren Kinder am P-Knoten gab, muss der alte P-Knoten eliminiert werden.

1.2.5.6 Die Schablone P_5

Bei der Schablone P_5 (siehe Abbildung 1.12 auf Seite 13) verwenden wir denselben Trick wie bei Schablone P_3. Der aktuell betrachtete Knoten mitsamt seiner Kinder wird umgehängt, so dass die eigentliche Arbeit an der vollen und am neuen Knoten stattfindet.

Hängt am betrachteten P-Knoten mindestens ein voller Teilbaum, können die Kosten auf die vollen Kinder des P-Knotens so verteilt werden, dass jedes volle Kind nur konstante Kosten erhält. Falls am betrachteten P-Knoten kein voller Teilbaum hängt, so sind sind die Kosten konstant und können dem betrachteten P-Knoten zugeordnet werden (der dann jaus dem relevanten reduzierten Baum bzw. ganz verschwindet, also keine weiteren Kosten in späteren Schritten mehr erhalten kann).

1.2.5.7 Die Schablone P_6

Bei der Schablone P_6 (siehe Abbildung 1.13 auf Seite 14) gilt dasselbe. Hier werden auch zwei Q-Knoten verschmolzen und ein P-Knoten in deren Kinderliste mitaufgenommen. Da wir die Menge der Kinder als doppelt verkettete Liste implementiert haben, ist der Aufwand wieder proportional zu 1 plus der Anzahl voller Kinder des betrachteten P-Knotens (die richtigen Enden der jeweiligen Geschwisterliste der Kinder des Q-Knotens finden wir durch die Verweise auf das jeweils älteste und jüngste Kind, die nach dem Verschmelzen auch einfach wieder aktualisiert werden können).

Hängt also am betrachteten P-Knoten mindestens ein volles Kind, so können die Kosten wieder so verteilt werden, dass jedes volle Kind des P-Knotens konstante Kosten erhält (den Kindern des Q-Knoten werden keine Kosten zugewiesen). Hängt am betrachteten P-Knoten kein volles Kind, so sind die Kosten insgesamt konstant und können dem betrachteten P-Knoten zugeordnet werden. Da die Prozedur abbricht, können diesem P-Knoten keine weiteren Kosten mehr zugewiesen werden.

1.2.5.8 Die Schablone Q_2

In diesem Fall ist nur festzustellen, dass der entsprechende Q-Knoten ein partieller Q-Knoten ist. Die Kosten hierfür sind proportional zur Anzahl nichtleerer (also voller) Kinder des betrachteten Q-Knotens. Somit bekommt jedes volle Kind konstante Kosten zugeordnet.

1.2.5.9 Die Schablone Q_3

Bei der Schablone Q_3 (siehe Abbildung 1.17 auf Seite 16) wird nur ein Q-Knoten in einen anderen Knoten hineingeschoben. Da die Kinder eines Knoten als doppelt verkettete Liste implementiert ist, kann dies in konstanter Zeit geschehen, insbesondere da die richtegen Enden der Geschwisterlisten durch die Verweise auf das älteste und jüngste Kind der Q-Knoten gefunden werden können. Beim Erkennen dieser Situation sind noch Kosten fällig, die proportional zu 1 plus der Anzahl voller Kinder sind. Wenn der Q-Knoten die Wurzel des relevanten reduzierten Teilbaumes ist, muss es mindestens ein volles Kind geben und jedes volle Kind erhält konstante Kosten. Andernfalls werden die insgesamt konstanten Kosten dem betrachteten Q-Knoten zugeordnet (der aus dem Baum verschwindet und keine Kosten mehr erhalten kann).

Einziges Problem ist die Aktualisierung der Kinder des Q-Knotens. Würde jedes Kind einen Verweis auf seinen Elter besitzen, so könnte dies teuer werden. Da wir dies aber nur für die äußersten Kinder verlangen, müssen nur von den äußersten Kindern des Kinder-Q-Knotens die Elter-Information eliminiert werden, was sich in konstanter Zeit realisieren lässt. Alle inneren Kinder eines Q-Knotens sollen ja keine Informationen über ihren Elter besitzen. Ansonsten könnte nach ein paar Umorganisationen des PQ-Baumes diese Information falsch sein. Da ist dann keine Information besser als eine falsche.

1.2.5.10 Die Schablone Q_4

Bei der Schablone Q_4 (siehe Abbildung 1.18 auf Seite 16) sind die Kosten proportional zu 1 plus der Anzahl der vollen Kinder des betrachteten Q-Knotens. Sofern mindestens ein volles Kind des betrachteten Q-Knotens existiert, können diese auf die vollen Kinder so verteilt werden, dass jedes konstante Kosten erhält. Andernfalls sind die Kosten konstant und wir weisen sie der Wurzel des relevanten reduzierten Teilbaumes zu, da danach die Prozedur abbricht. Man beachte, dass dieser Fall genau dann eintritt, wenn beim Auffinden des relevanten reduzierten Teilbaums die Prozedur mit einem Sektor abbricht und somit die Listen der partiellen und vollen und partiellen Kinder des Q-Knotens bekannt sind (aber nicht der Q-Konten als Wurzel selbst).

1.2.5.11 Zusammenfassung Schablonen

Somit haben wir bei jeder Schablone einigen Knoten des relevanten reduzierten Teilbaumes konstante Kosten zugeordnet, nämlich einigen vollen Kindern, verschwindenden Knoten oder der Wurzel. Jedem Knoten werden also nur einmal Kosten zugeordnet, da er bei der Bearbeitung des Elters dann bereits der Enkel des betrachteten Knotens ist oder aus dem Baum entfernt wurde oder die Anwendung der Schablone beendet ist. Also sind die Kosten insgesamt proportional zu $|T_{rr}(T_{i-1}, F)|$.

30.04.24

1.2.5.12 Der Pfad zur Wurzel

Zum Schluss müssen wir uns nur noch überlegen, dass wir eventuell Zeit verbraten, wenn wir auf dem Weg von der Wurzel des relevanten reduzierten Teilbaumes zur eigentlichen Wurzel des Baumes weit nach oben laufen. Dieser Pfad könnte wesentlich größer sein als die Größe des relevanten reduzierten Teilbaumes.

Hierbei hilft uns jedoch, dass wir die Knoten aus der Menge free in FIFO-Manier (first-in-first-out) entfernen. Das bedeutet, bevor wir auf diesem Wurzelweg einen Knoten nach oben steigen, werden zunächst alle anderen freien Knoten betrachtet. Dies ist immer mindestens ein anderer. Andernfalls gäbe es nur einen freien Knoten und (mindestens) einen Sektor. Aber da der freie Knoten auf dem Weg von der Wurzel des relevanten reduzierten Teilbaumes zur Wurzel des Baumes könnte den blockierten Sektor nie befreien. In diesem Fall könnten wir zwar den ganzen Weg bis zur Wurzel hinauflaufen, aber dann gäbe es keine Lösung und ein einmaliges Durchlaufen des Gesamt-Baumes können wir uns leisten.

Somit sind also immer mindestens zwei freie Knoten in der freien Menge. Also wird beim Hinauflaufen jeweils der relevante reduzierte Teilbaum um eins vergrößert. Damit können wir auf dem Weg von der relevanten reduzierten Wurzel zur Wurzel des Baumes nur so viele Knoten nach oben ablaufen, wie es insgesamt Knoten im relevanten reduzierten Teilbaum geben kann. Diese zusätzlichen Faktor können wir jedoch in unserer Groß-O-Notation verstecken.

1.2.6 Anzahlbestimmung angewendeter Schablonen

Da die Anzahl der inneren Knoten im relevanten reduzierten Teilbaum gleich der Anzahl der angewendeten Schablonen ist, werden wir für die Laufzeitabschätzung die Anzahl der angewendeten Schablonen abzählen bzw. abschätzen. Mit $\sharp P_i$ bzw. $\sharp Q_i$ bezeichnen wir die Anzahl der angewendeten Schablonen P_i bzw. Q_i zur Konstruktion des PQ-Baumes für $\Pi(\Sigma, \mathcal{F})$. Hinzu kommen dann noch die Anzahl aller jemals markierten Blätter, die durch $\sum_{i=1}^{k} |F_i|$ gegeben ist.

1.2.6.1 Bestimmung von $\sharp P_0$ und $\sharp Q_0$

Diese Schablonen werden, wie bereits erwähnt, im relevanten reduzierten Teilbaum nie explizit angewendet.

1.2.6.2 Bestimmung von $\sharp P_1$ und $\sharp Q_1$

Man überlegt sich leicht, dass solche Schablonen nur in Teilbäumen angewendet werden können, in denen alle Blätter markiert sind, also an inneren Knoten von vollen gewurzelten Teilbäumen des reduzierten relevanten Teilbaumes. Da nach Lemma 1.3 die Anzahl der inneren Knoten durch die Anzahl der markierten Blätter beschränkt sind, gilt:

$$\sharp P_1 + \sharp Q_1 = O\left(\sum_{F \in \mathcal{F}} |F|\right).$$

1.2.6.3 Bestimmung von $\sharp P_2$, $\sharp P_4$, $\sharp P_6$ und $\sharp Q_4$

Da nach diesen Schablonen die Prozedur reduce(T, F) abgeschlossen ist, können diese nur einmal für jede Restriktion angewendet werden uns daher gilt:

$$\sharp P_2 + \sharp P_4 + \sharp P_6 + \sharp Q_4 \leq |\mathcal{F}| = O(|\mathcal{F}|).$$

1.2.6.4 Bestimmung von $\sharp P_3$ und $\sharp Q_2$

Die Schablone P_3 generiert einen neuen partiellen Q-Knoten, der vorher noch nicht da war (siehe auch Abbildung 1.10), und die Schablone Q_2 ermittelt einen neuen partiellen Q-Knoten. Da in einem PQ-Baum nicht mehr als zwei partielle Q-Knoten (die nicht Vorfahr eines anderen sind) auftreten können und partielle Q-Knoten nicht wieder verschwinden können, kann für jede Anwendung reduce(T, F) nur zweimal die Schablone P_3 bzw. Q_2 angewendet werden (andernfalls wird festgestellt, dass sich F_i nicht einarbeiten lässt und es wird der leere PQ-Baum zurückgegeben). Daher gilt nun

$$\sharp P_3 + \sharp Q_2 \leq 2|\mathcal{F}| = O(|\mathcal{F}|).$$

1.2.6.5 Bestimmung von $\sharp P_5 + \sharp Q_3$

Wir definieren zuerst einmal recht willkürlich die Norm eines PQ-Baumes wie folgt.

Definition 1.9 *Die* Norm eines PQ-Baumes T *(kurz* $\|T\|$*) ist die Summe aus der Anzahl der Q-Knoten in* T *plus der Anzahl der inneren Knoten von* T*, die Kinder eines P-Knotens sind.*

Man beachte, dass Q-Knoten in der Norm zweimal gezählt werden können, nämlich genau dann, wenn sie ein Kind eines P-Knotens sind.

Zuerst halten wir ein paar elementare Eigenschaften dieser Norm fest:

1. Es gilt $\|T\| \geq 0$ für alle PQ-Bäume T;
2. $\|T(\Sigma)\| = 0$;
3. Die Anwendung einer beliebige Schablone erhöht die Norm um maximal eins, d.h $\|S(T)\| \leq \|T\| + 1$ für alle PQ-Bäume T, wobei $S(T)$ der PQ-Baum ist, der nach Ausführung einer Schablone S entsteht.
4. Die Schablonen P_5 und Q_3 erniedrigen die Norm um mindestens eins, d.h. $\|S(T)\| \leq \|T\| - 1$ für alle PQ-Bäume T, wobei $S(T)$ der PQ-Baum ist, der nach Ausführung einer Schablone $S \in \{P_5, Q_3\}$ entsteht.

Die ersten beiden Eigenschaften folgen unmittelbar aus der Definition der Norm. Die letzten beiden Eigenschaften werden durch eine genaue Inspektion der Schablonen klar (dem Leser sei explizit empfohlen, dies zu verifizieren).

Da wir mit den Schablonen P_5 und Q_3 die Norm ganzzahlig erniedrigen und mit jeder anderen Schablone die Norm ganzzahlig um maximal 1 erhöhen, können die Schablonen P_5 und Q_3 nur so oft angewendet werden, wie die anderen. Grob gesagt, es kann nur das weggenommen werden, was schon einmal hingelegt wurde. Es gilt also:

$$\begin{aligned} \sharp P_5 + \sharp Q_3 &\leq \sharp P_1 + \sharp P_2 + \sharp P_3 + \sharp P_4 + \sharp P_6 + \sharp Q_1 + \sharp Q_2 + \sharp Q_4 \\ &= O\left(|\mathcal{F}| + \sum_{F \in \mathcal{F}} |F|\right) \\ &= O\left(\sum_{F \in \mathcal{F}} |F|\right). \end{aligned}$$

Hinzu kommt noch die Laufzeit für der Erstellung des ersten PQ-Baumes $T_0 = T(\Sigma)$. Da dies einfach der PQ-Baum mit einem P-Knoten als Wurzel und genau einem Kind für jedes Zeichen aus Σ ist, ist dies in Zeit $O(|\Sigma|)$ machbar. Damit haben wir die Laufzeit für einen erfolgreichen Fall berechnet.

Wir müssen uns nur noch überlegen, was im erfolglosen Fall passiert, wenn also der leere PQ-Baum die Lösung darstellt. In diesem Fall berechnen wir zuerst für eine

Teilmenge $\mathcal{F}' \subsetneq \mathcal{F}$ einen konsistenten PQ-Baum. Bei Hinzunahme der Restriktion F stellen wir fest, dass $\mathcal{F}'' := \mathcal{F}' \cup \{F\}$ keine Darstellung durch einen PQ-Baum besitzt. Für die Berechnung des PQ-Baumes von $\mathcal{F}'$ benötigen wir, wie wir gerade eben gezeigt haben:

$$O\left(|\Sigma| + \sum_{F\in\mathcal{F}'} |F|\right) = O\left(|\Sigma| + \sum_{F\in\mathcal{F}} |F|\right).$$

Um festzustellen, dass $\mathcal{F}''$ keine Darstellung durch einen PQ-Baum besitzt, müssen wir im schlimmsten Fall den PQ-Baum T' für $\mathcal{F}'$ durchlaufen. Da dieser ein PQ-Baum ist und nach Lemma 1.3 maximal $|\Sigma|$ innere Knoten besitzt, da er genau $|\Sigma|$ Blätter besitzt, folgt, dass der Aufwand höchstens $O(|\Sigma|)$ ist. Fassen wir das Ergebnis noch einmal zusammen.

Theorem 1.10 *Die Menge $\Pi(\Sigma, \mathcal{F})$ kann durch einen PQ-Baum T mit*

$$\text{cons}(T) = \Pi(\Sigma, \mathcal{F})$$

dargestellt und in Zeit $O\left(|\Sigma| + \sum_{F\in\mathcal{F}} |F|\right)$ berechnet werden.

Somit haben wir einen effizienten Algorithmus zur genomischen Kartierung gefunden, wenn wir voraussetzen, dass die Experimente fehlerfrei sind. In der Regel wird dies jedoch nicht der Fall sein, wie wir das schon am Ende des ersten Abschnitt dieses Kapitels angemerkt haben. Wollten wir False Negatives berücksichtigen, dann müssten wir erlauben, dass die Zeichen einer Restriktion nicht konsekutiv in einer Permutation auftauchen müssten, sondern durchaus wenige (ein oder zwei) sehr kurze Lücken (von ein oder zwei Zeichen) auftreten dürften. Für False Positives müssten wir zudem wenige einzelne isolierte Zeichen einer Restriktion erlauben. Und für Chimeric Clones müsste auch eine oder zwei zusätzliche größere Lücken erlaubt sein. Leider hat sich gezeigt, dass solche modifizierten Problemstellung bereits $\mathcal{NP}$-hart sind und somit nicht mehr effizient lösbar sind.

PQ-Bäume werden auch zur Erkennung von planaren Graphen in Linearzeit eingesetzt. Darüber hinaus lassen sich auch Intervallgraphen (auf die wir in Abschnitt1.6 eingehen werden) mit Hilfe von PQ-Bäumen in linearer Zeit erkennen. Eine weitere, biologisch interessante Anwendung werden wir folgenden Abschnitt skizzieren.

1.2.7 Anwendung: Gene-Clustering (+)

In diesem Abschnitt wollen wir kurz eine weitere Anwendung von PQ-Bäumen skizzieren. Im Folgenden bezeichne $S(n)$ die symmetrische Gruppe der Ordnung n, d.h. die Gruppe aller Permutationen auf n Elementen (in der Regel auf $[1:n]$).

Definition 1.11 *Seien $\pi_1, \ldots, \pi_k \in S(n)$. Ein* common interval *ist eine Folge von k Paaren $(\ell_1, u_1), \ldots (\ell_k, r_k)$, für die es ein $C \subseteq [1:n]$ mit $C = \{\pi_i(j) : j \in [\ell_i : u_i]\}$ für alle $i \in [1:k]$ gibt.*

Biologisch werden hierbei die Permutationen als die Reihenfolge von n Genen interpretiert, die alle auf den gegebenen k Genomen auftreten. Ein solches common interval beschreibt damit eine Gruppe von Genen, die auf allen Genomen in der gleichen Gruppierung, aber eventuell unterschiedlicher Reihenfolge auftreten. Solche Gen-Cluster haben meist eine funktionelle Abhängigkeit und sind daher biologisch sehr interessant.

In den drei Sequenzen (a, b, c, d, e, f), (d, f, e, c, b, a) und (c, e, f, d, a, b) sind beispielsweise (a, b) und (d, e, f) solche common intervals. Dies gilt immer auch für die trivialen einelementigen Teilmengen und die gesamte Menge selbst.

Wenn man Gene auf Genomen durch Permutationen beschreibt, betrachtet man nur orthologe Gene. Will man auch paralog Gene betrachten, so muss man auch Vielfachheiten der einzelnen Symbole erlauben, d.h. von Permutation zu Zeichenreihen übergehen.

Definition 1.12 *Sei $p = p_1 \cdots p_m \in \Sigma^m$, $s = s_1 \cdots s_n \in \Sigma^n$. Das Muster p erscheint in s an Position i, wenn $\{p_j : j \in [1:m]\} = \{s_j : j \in [i : i+m-1]\}$.*

Soll in der letzten Definition ein mehrfaches Vorkommen von Zeichen in den Sequenzen erlaubt sein, so ist die Mengengleichheit im Sinne von Multimengen zu verstehen, d.h. die Vielfachheiten müssen auch übereinstimmen.

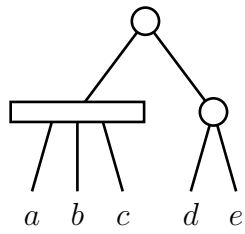
Definition 1.13 *Die* Linearisierung *eines PQ-Baumes T ist induktiv wie folgt definiert:*

a) Die Linearisierung eines mit a markiertes Blatt ist das Symbol selbst.

b) Wenn die Wurzel des PQ-Baumes ein P-Knoten ist und die Linearisierung der Kinder durch $\ell_1, \ldots, \ell_k$ gegeben ist, dann ist die Linearisierung des Baums durch $(\ell_1, \ldots, \ell_k)$ gegeben.

c) Wenn die Wurzel des PQ-Baumes ein Q-Knoten ist und die Linearisierung der Kinder durch $\ell_1, \ldots, \ell_k$ gegeben ist, dann ist die Linearisierung des Baums durch $(\ell_1 - \cdots - \ell_k)$ gegeben.

In Abbildung 1.24 ist ein Beispiel für eine Linearisierung eines PQ-Baumes angegeben. Damit ergibt sich das erste Ziel, für eine gegebene Menge von Permutationen $\pi = (\pi_1, \ldots, \pi_k)$ einen PQ-Baum T zu erstellen, so dass die Linearisierung von T der maximalen Notation von π entspricht (die maximale Notation ist



$((a-b-c),(d,e))$

Abbildung 1.24: Beispiel: Linearisierung eines PQ-Baum

eine der Linearisierung ähnliche Darstellung für mögliche Permutationen). Dies ist leider nicht immer zu erreichen, wie das folgende Beispiel zeigt. Für die Menge $\pi = \{abcde, abced, cbade, edabc\}$ ist der PQ-Baum in Abbildung 1.24 der kleinste PQ-Baum, der alle diese Permutationen in $\text{cons}(T)$ enthält, aber leider auch andere, wie z.B. *adcba* und *cbaed*.

Die Art der Darstellung von Permutation, wie sie bei Linearisierung von PQ-Bäumen erzeugt wird, ist für die Analyse von Gen-Clustern weit verbreitet (bekannt als so genannte maximale Notation). Man kann also mit PQ-Bäumen eine sehr kompakte Darstellung der Cluster erzielen. Wir werden uns jetzt darum kümmern, wie man diese PQ-Bäume konstruieren kann.

Definition 1.14 *Sei $\pi = (\pi_1, \ldots, \pi_k) \subseteq S(n)$. Ein* Konsensus-PQ-Baum *für π ist ein PQ-Baum T mit $\pi \subseteq \text{cons}(T)$.*

Ein Konsensus-PQ-Baum T für π heißt minimal*, wenn kein Konsensus-PQ-Baum T' für π mit $\pi \subseteq \text{cons}(T')$ und $|\text{cons}(T')| < |\text{cons}(T)|$ existiert.*

Beachte, dass nach der Definition ein Konsensus-PQ-Baum nicht notwendigerweise eindeutig sein muss. Im Folgenden zeigen wir kurz ohne Beweis, dass diese minimalen Konsensus-PQ-Bäume im Wesentlichen eindeutig sind. Wir betrachten dazu die folgende Variante der Definition 1.11, wobei π_1 als Identität festgelegt ist.

Definition 1.15 *Sei $\pi = (\pi_1, \ldots, \pi_k) \subseteq S(n)$ mit $\pi_1 = \text{id} = (1, \ldots, n)$. Ein Intervall $[r : s] \subseteq [1 : n]$ heißt* common interval *von π, wenn die Elemente der Menge $[r : s]$ konsekutiv in π_i für alle $i \in [1 : k]$ auftreten.*

Die Menge aller common intervals von π werden mit $C(\pi)$ bezeichnet.

Das folgende Lemma besagt, dass es für jeden echten PQ-Baum mindestens eine Menge von Restriktion gibt, die diesen erzeugt.

Lemma 1.16 *Sei T ein echter PQ-Baum über Σ. Dann existiert $\mathcal{F} \subseteq 2^\Sigma$ mit $\Pi(\Sigma, \mathcal{F}) = cons(T)$ gibt.*

Beweis: Übungsaufgabe. ■

Theorem 1.17 *Für $\pi \subseteq S(n)$ ist $T = \text{reduce}(T(\Sigma), C(\pi))$ ein minimaler Konsensus-PQ-Baum für π.*

Beweis: Zuerst stellen wir fest, dass $T = \text{reduce}(T(\Sigma), C(\pi))$ ein Konsensus-PQ-Baum für π ist. Für einen Widerspruchsbeweis nehmen wir an, dass $\hat{T} \neq T$ ein minimaler Konsensus-PQ-Baum für π ist. Nach Lemma 1.16 existiert ein $\mathcal{F} \subseteq 2^{\Sigma}$ mit $\text{cons}(T) = \Pi(T(\Sigma), \mathcal{F})$.

Für jedes $F \in \mathcal{F}$ kommen also die Symbole aus F in jedem $f(T')$ mit $T' \cong T$ konsekutiv vor. Da $\hat{T}$ ein minimaler Konsensus-PQ-Baum ist, muss $F \in C(\pi)$ gelten. Also gilt $\text{cons}(T) \subseteq \text{cons}(\hat{T})$, was ein Widerspruch zur Minimalität von $\hat{T}$ ist. ■

Beachte, dass auch nach dem obigen Beweis ein minimaler Konsensus-PQ-Baum nicht notwendigerweise eindeutig sein muss.

Korollar 1.18 *Sei $\pi \subseteq S(n)$ und seien T_1 und T_2 zwei minimale Konsens-PQ-Bäume für π, dann gilt $\text{cons}(T_1) = \text{cons}(T_2)$.*

Für den Beweis dieses Korollars sind im Wesentlichen nur die Beweisideen des vorigen Satzes nötig.

Theorem 1.19 *Seien T_1 und T_2 zwei PQ-Bäume mit $\text{cons}(T_1) = \text{cons}(T_2)$, dann ist $T_1 \cong T_2$.*

Diesen Satz wollen wir hier nicht beweisen und verweisen auf die Originalliteratur. Wir erhalten nur das gewünschte Ergebnis im folgenden Korollar fest.

Korollar 1.20 *Sei $\pi \subseteq S(n)$, dann ist der minimale Konsensus-PQ-Baum für π bis auf Äquivalenz eindeutig.*

Da minimale Konsensus-PQ-Bäume mehr Permutationen anzeigen als gewünscht (siehe auch das Beispiel in Abbildung 1.24), werden die inneren P- und Q-Knoten noch annotiert. Q-Knoten erhalten noch die Annotation $\leftrightarrow$, wenn die beide Richtungen erlaubt sind (sonst nur die Abfolge der Kinder von Links nach rechts). P-Knoten erhalten die Annotation, welche Permutationen der Kinder wirklich zulässig sind (statt allen, es sind hier aber nur die Permutationen zusätzlich zur kanonischen

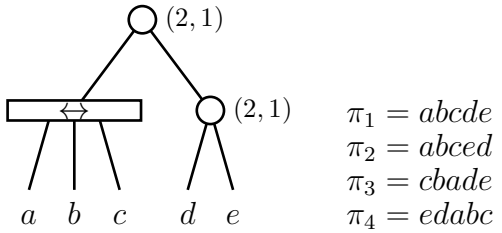


Abbildung 1.25: Beispiel: Spezialisierung eines PQ-Baum

angegeben). Wie man im selben Beispiel in der Abbildung 1.25 sehen kann, hilft diese Annotation auch nicht immer. Sie kann auch nur einige Fälle ausschließen, im angegeben Beispiel sogar keines.

Ein minimaler Konsensus-PQ-Baum für π kann sehr einfach konstruiert werden. Zuerst wird die Menge $C(\pi)$ der common intervals von π konstruiert und mit diesem dann der PQ-Baum $T = \text{reduce}(T(\Sigma, C(\pi))$ konstruiert. Da bereits $|C(\pi)| \leq \binom{|\Sigma|}{2}$ ist, ist der Algorithmus aber in der Regel nicht mehr linear in der Eingabegröße kn.

Für einen Linearzeit-Algorithmus wird zuerst die Menge der irreducible common intervals $I(\pi) \subseteq C(\pi)$ konstruiert. Dies ist quasi die kleinste Teilmenge von $C(\pi)$, für die die folgende Beziehung gilt: $\text{reduce}(T(\Sigma, I(\pi))) = \text{reduce}(T(\Sigma, C(\pi)))$. Die Menge der irreducible common intervals kann in Zeit $O(kn)$ aus π erzeugt werden. Obwohl die Anzahl der irreducible common intervals linear in n ist, ist die Menge der einzubauenden Restriktionen in den PQ-Baum nur durch $O(n^2)$ beschränkt, so dass die Konstruktion einen Zeitbedarf von $O(n^2 + kn)$ hat. Man kann aber zeigen, dass man den PQ-Baum für die Menge der irreducible common intervals auch in Linearzeit aufbauen kann.

Theorem 1.21 *Für $\pi \subseteq S(n)$ kann ein minimaler Konsensus-PQ-Baum in Zeit $O(kn)$ konstruiert werden.*

Für eine genauere Darstellung verweisen wir auf die Originalliteratur von G. Landau, L. Parida und O. Weimann. Für ein ähnliches Problem, namentlich die Erkennung zusammenhängender genomischer Bereiche von Vorfahren (Contiguous Ancestral Regions), werden ebenfalls PQ- und die im folgenden Abschnitte eingeführten PQR-Bäume verwendet. Auch hier verweisen wir auf die Originalliteratur von C. Chauve und E. Tannier. Für die Bestimmung des common intervals verweisen wir auf die Originalliteratur von Bergeron et al.

Eine weitere Anwendung ist die Berechnung evolutionärer Distanzen, wenn für die Taxa (Blätter) die Reihenfolge der Gene (Permutationen) jeweils bekannt sind und für die Vorfahren (innere Knoten) jeweils eingeschränkte Reihenfolgen (PQ-Bäume) vorgegeben sind. Hierzu verweisen wir auf die Originalliteratur von Jiang et al.

Auch eine approximative Suche nach bekannten Gen-Cluster aus bekannten Genomen (dargestellt durch PQ-Bäume) in einem neu sequenzierten Genom ist möglich, siehe die Originalliteratur von Ziemerman et al.

02.05.24

1.3 PQR-Bäume

In diesem Abschnitt wollen wir eine Verallgemeinerung von PQ-Bäumen, die PQR-Bäume, vorstellen. Hierbei gibt es neben P- und Q-Knoten auch noch R-Knoten. Der Vorteil wird sein, dass es möglich ist, für jede Menge von Restriktionen einen PQR-Baum zu konstruieren. Dabei wird es genau dann R-Knoten geben, wenn die Menge von Restriktionen nicht die C1P erfüllt. Sollte die C1P nicht erfüllt sein, dann können uns die R-Knoten im PQR-Baum Hinweise liefern, an welchen „Stellen“ es Probleme bei der Erfüllung der C1P für die gegebene Menge von Restriktionen gibt.

1.3.1 Definition

Zuerst einmal wollen wir definieren, was wir formal unter einem PQR-Baum verstehen wollen.

Definition 1.22 *Sei Σ ein endliches Alphabet. Dann ist ein* PQR-Baum *über Σ induktiv wie folgt definiert:*

- *Jeder einelementige Baum (also ein Blatt), das mit einem Zeichen aus Σ markiert ist, ist ein PQR-Baum.*
- *Sind $T_1, \ldots, T_k$ PQR-Bäume, dann ist der Baum, der aus einem so genannten* P-Knoten *als Wurzel entsteht und dessen Kinder die Wurzeln der Bäume $T_1, \ldots, T_k$ sind, ebenfalls ein PQR-Baum.*
- *Sind $T_1, \ldots, T_k$ PQR-Bäume, dann ist der Baum, der aus einem so genannten* Q-Knoten *als Wurzel entsteht und dessen Kinder die Wurzeln der Bäume $T_1, \ldots, T_k$ sind, ebenfalls ein PQR-Baum.*
- *Sind $T_1, \ldots, T_k$ PQR-Bäume, dann ist der Baum, der aus einem so genannten* R-Knoten *als Wurzel entsteht und dessen Kinder die Wurzeln der Bäume $T_1, \ldots, T_k$ sind, ebenfalls ein PQR-Baum.*

Wie man leicht der Definition entnimmt, ist jeder PQ-Baum auch ein PQR-Baum.

In der Abbildung 1.26 ist skizziert, wie wir in Zukunft P-, Q- bzw. R-Knoten graphisch darstellen wollen. P-Knoten werden durch Kreise, Q-Knoten durch lange

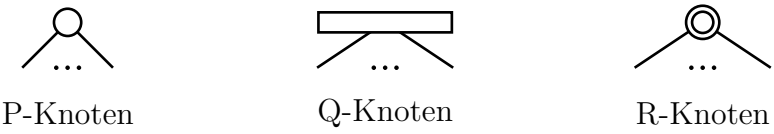


Abbildung 1.26: Skizze: Darstellung von P-, Q- und R-Knoten

Rechtecke und R-Knoten mit doppelt umrandeten Kreisen dargestellt. Für die Blätter führen wir keine besondere Konvention ein. In der Abbildung 1.27 ist ein Beispiel eines PQR-Baumes angegeben.

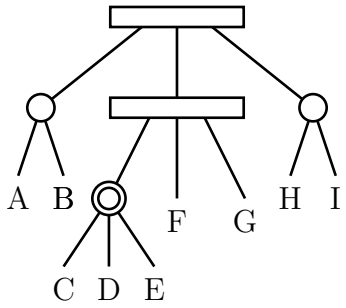


Abbildung 1.27: Beispiel: Ein PQR-Baum

Definition 1.23 *Ein PQR-Baum heißt* echt*, wenn folgende Bedingungen erfüllt sind:*

- *Jedes Element $a \in \Sigma$ kommt genau einmal als Blattmarkierung vor;*
- *Jeder P-Knoten hat mindestens zwei Kinder;*
- *Jeder Q-Knoten hat mindestens drei Kinder;*
- *Jeder R-Knoten hat mindestens drei Kinder.*

Der in Abbildung 1.27 angegebene PQ-Baum ist also ein echter PQR-Baum. Auch für echte PQR-Bäume gilt, dass die Anzahl der P-, Q- und R-Knoten kleiner als die Kardinalität des betrachteten Alphabets Σ ist.

Die R-Knoten werden innerhalb des PQR-Baumes die Stellen angeben, an denen bei der Einarbeitung neuer Restriktionen Widersprüche zur C1P aufgetreten sind.

Definition 1.24 *Sei T ein PQR-Baum über Σ. Die* Frontier *von T, kurz $f(T)$ ist die Permutation über Σ, die durch das Ablesen der Blattmarkierungen von links nach rechts geschieht (also die Reihefolge der Blattmarkierungen in einer Tiefensuche unter Berücksichtigung der Ordnung auf den Kindern jedes Knotens).*

Die Frontier des Baumes aus Abbildung 1.27 ist dann ABCDEFGHI.

Definition 1.25 *Zwei PQR-Bäume T und T' heißen* äquivalent*, kurz $T \cong T'$, wenn sie durch endliche Anwendung folgender Regeln ineinander überführt werden können:*

- *Beliebiges Umordnen der Kinder eines P- oder R-Knotens;*
- *Umkehren der Reihenfolge der Kinder eines Q-Knotens.*

Definition 1.26 *Sei T ein echter PQR-Baum, dann ist $cons(T)$ die Menge der konsistenten Frontiers von T, d.h.:*

$$cons(T) = \{f(T') \,:\, T \cong T'\}.$$

Beispielsweise befinden sich dann in der Menge $\text{cons}(T)$ für den Baum aus der Abbildung 1.27: BADCEFGIH, ABGFCDEHI oder HIDCEFGBA. Insgesamt hat dieser Baum 96 verschiedene Frontiers.

1.3.2 Eigenschaften von PQR-Bäumen

In diesem Abschnitt wollen wir zuerst ein paar elementare Begriffe und Eigenschaften von PQR-Bäumen festhalten.

Definition 1.27 *Sei $T = (V, E)$ ein echter PQR-Baum über Σ und $v \in V$. Dann ist die* Domain *des Knotens v, bezeichnet mit $D_T(v)$, als die Menge der Blattmarkierungen von Nachfolgern von v definiert. Formal gilt für ein Blatt $v \in V$ also $D_T(v) := \{v\}$ und für einen inneren Knoten $v \in V$ mit Kindermenge $S = \{w \in V \,:\, (v, w) \in E\}$ gerade $D_T(S) := \bigcup_{v \in S} D_T(v)$. Für $S \subseteq V$ ist allgemein:*

$$D_T(S) = \bigcup_{v \in S} D_T(v).$$

Im Beispiel in der Abbildung 1.27 ist die Domain für den Q-Knoten, der Kind der Wurzel ist, gerade $\{C, D, E, F, G\}$. Die Domain der Menge der P-Knoten ist $\{A, B, H, I\}$.

Definition 1.28 *Sei Σ ein Alphabet. Die* trivialen Teilmengen *von Σ, bezeichnet mit $\mathcal{T}(\Sigma)$, sind:*

$$\mathcal{T}(\Sigma) = \{\emptyset, \Sigma\} \cup \{\{a\} \,:\, a \in \Sigma\}.$$

Definition 1.29 *Sei Σ ein Alphabet und $\mathcal{F}$ eine Menge von Restriktionen über Σ. Eine Teilmenge $A \subset \Sigma$ heißt* implizite Restriktion*, wenn gilt:*

$$\Pi(\Sigma, \mathcal{F}) = \Pi(\Sigma, \mathcal{F} \cup \{A\}).$$

Lemma 1.30 *Sei Σ ein Alphabet und $\mathcal{F}$ eine beliebige Menge von Restriktionen über Σ, dann ist für beliebige Mengen $A, B \in \mathcal{F}$*

- *$A \cap B$ eine implizite Restriktion (*Durchschnitt*);*
- *$A \cup B$ eine implizite Restriktion, sofern $A \cap B \neq \emptyset$ (*nicht-disjunkte Vereinigung*);*
- *$A \setminus B$ eine implizite Restriktion, sofern $B \not\subset A$ (*Mengensubtraktion einer Nicht-Teilmenge*);*
- *jedes Element aus $\mathcal{T}(\Sigma)$ eine implizite Restriktion.*

Beweis: Übungsaufgabe. ■

Definition 1.31 *Eine Menge $\mathcal{F}$ von Restriktionen über einem Alphabet Σ heißt* vollständig*, wenn sie die trivialen Teilmengen enthält und unter Durchschnitt, nicht-disjunkter Vereinigung und Mengensubtraktion einer Nicht-Teilmenge abgeschlossen ist.*

Man kann zeigen, dass eine Menge genau dann vollständig ist, wenn sie bereits alle impliziten Restriktionen enthält. Dies ist jedoch technisch aufwendig und da wir diese Aussage im Folgenden so nicht benötigen, verzichten wir hier auf den Beweis.

Definition 1.32 *Sei Σ ein Alphabet und $\mathcal{F}$ eine Menge von Restriktionen über Σ. Mit $\overline{\mathcal{F}}$ bezeichnen wir die kleinste (bzgl. Mengeninklusion) Teilmenge von 2^Σ, die vollständig ist und $\mathcal{F}$ enthält.*

Das folgende Lemma zeigt, dass die vorhergehende Definition wohldefiniert ist und gibt eine andere Charakterisierung der kleinsten vollständigen Menge, die $\mathcal{F}$ enthält.

Lemma 1.33 *Es gilt*

$$\overline{\mathcal{F}} = \bigcap_{\substack{\mathcal{F} \subseteq \mathcal{F}' \subseteq 2^\Sigma \\ \mathcal{F}' \text{ ist vollständig}}} \mathcal{F}'.$$

Beweis: Übungsaufgabe. ■

Theorem 1.34 *Sei Σ ein Alphabet und $\mathcal{F}$ eine Menge von Restriktionen über Σ, dann gilt:*

$$\Pi(\Sigma, \mathcal{F}) = \Pi(\Sigma, \overline{\mathcal{F}}).$$

Beweis: Übungsaufgabe. ■

1.3.3 Beziehung zwischen PQR-Bäumen und C1P

In diesem Abschnitt geben wir einige Beziehungen zwischen PQR-Bäumen und Mengen von Restriktionen an, die die C1P erfüllen

Definition 1.35 *Sei T ein echter PQR-Baum über einem Alphabet Σ, dann ist die Menge Compl(T) induktiv wie folgt definiert:*

- $\mathcal{T}(\Sigma) \subseteq$ *Compl*(T)*;*
- $D_T(S) \in$ *Compl*(T)*, wenn S die Menge aller Kinder eines P-Knotens von T ist.*
- $D_T(S) \in$ *Compl*(T)*, wenn S eine beliebige Menge konsekutiver Kinder eines Q-Knotens von T ist.*
- $D_T(S) \in$ *Compl*(T)*, wenn S eine beliebige Menge von Kindern eines R-Knotens von T ist.*

Theorem 1.36 *Sei T ein echter PQR-Baum über einem Alphabet Σ, dann ist die Menge Compl(T) vollständig.*

Beweis: Offensichtlich gilt $\mathcal{T}(\Sigma) \subseteq \text{Compl}(T)$. Es muss also nur noch der Abschluss gegen Durchschnitt, nicht-disjunkte Vereinigung und Mengensubtraktion von Nicht-Teilmengen gezeigt werden.

Seien S und S' jeweils eine Menge von Kindern von Knoten in T. Nehmen wir zuerst an, dass der Elter v der Knoten aus S und der Elter w der Knoten aus S' verschieden sind. Ist v ein Nachfolger eines Kindes in S' von w, dann gilt $D_T(S) \subseteq D_T(S')$. Somit gilt also $D_T(S) \cap D_T(S') = \emptyset \in \text{Compl}(T)$, $D_T(S) \cup D_T(S') = D_T(S') \in \text{Compl}(T)$ sowie $D_T(S) \setminus D_T(S') = \emptyset \in \text{Compl}(T)$. Ist w ein Nachfolger eines Kindes in S von v, dann gilt $D_T(S') \subseteq D_T(S)$ und die obigen Fälle gelten analog. Andernfalls

ist $D_T(S) \cap D_T(S') = \emptyset$. In diesen Fällen erzeugen die Operationen Durchschnitt, nicht-disjunkte Vereinigung und Mengensubtraktion von Nicht-Teilmengen nur die leere Menge.

Seien also jetzt S und S' Mengen von Kindern desselben Knotens v. Ist v ein P-Konten, dann muss $S = S'$ sein und es ist nichts zu zeigen. Ist v ein Q-Knoten, dann müssen die Mengen S und S' konsekutive Kinder umfassen. Bei allen drei Operationen entstehen Mengen von konsekutiven Kindern von v, die sich wieder als eine Menge $D_T(S'')$ für eine geeignete konsekutive Kindermenge S'' von v schreiben lassen, namentlich $S'' := S \cap S'$ mit $D_T(S) \cap D_T(S') = D_T(S'') \in \mathrm{Compl}(T)$, $S'' := S \cup S'$ mit $D_T(S) \cup D_T(S') = D_T(S'') \in \mathrm{Compl}(T)$ (sofern $S \cap S' \neq \emptyset$) bzw. $S'' := S \setminus S'$ mit $D_T(S) \setminus D_T(S') = D_T(S'') \in \mathrm{Compl}(T)$ (sofern $S' \not\subset S$). Ist zuletzt v nun ein R-Knoten, dann sind S und S' beliebige Mengen von Kindern in v. Bei allen drei Operationen entstehen Mengen, die sich wieder als eine Menge $D_T(S'')$ für eine geeignete Menge S'' von Kindern von v schreiben lassen namentlich $S'' := S \cap S'$ mit $D_T(S) \cap D_T(S') = D_T(S'') \in \mathrm{Compl}(T)$, $S'' := S \cup S'$ mit $D_T(S) \cup D_T(S') = D_T(S'') \in \mathrm{Compl}(T)$ (sofern $S \cap S' \neq \emptyset$) bzw. $S'' := S \setminus S'$ mit $D_T(S) \setminus D_T(S') = D_T(S'') \in \mathrm{Compl}(T)$ (sofern $S' \not\subset S$). ■

Im Folgenden geben wir noch einige Sätze ohne Beweise an, um die Beziehung zwischen PQ- und PQR-Bäumen unter Berücksichtigung der Erfüllbarkeit der C1P zu beleuchten.

Definition 1.37 *Sei $\alpha = \alpha_1 \cdots \alpha_n \in \Sigma^n$, dann ist consec($\alpha$) die Menge aller von α* induzierten Restriktionen *über Σ, d.h. die Menge aller Mengen von in α konsekutiver Zeichen:*

$$\mathit{consec}(\alpha) = \{\{\alpha_i, \alpha_{i+1} \ldots, \alpha_{j-1}, \alpha_j\} \,:\, i, j \in [1:n]\}\,.$$

Beachte, dass mit $i > j \in [1:n]$ auch die leere Menge in consec(α) enthalten ist. Für $\alpha = acdb$ ist

$$\mathrm{consec}(\alpha) = \mathcal{T}(\{a,b,c,d\}) \cup \Big\{\{a,c\},\{c,d\},\{b,d\},\{a,c,d\},\{b,c,d\}\Big\}.$$

Definition 1.38 *Sei Σ ein Alphabet und $S \subseteq \Sigma^*$, dann ist*

$$\mathit{consec}(S) := \bigcap_{\alpha \in S} \mathit{consec}(\alpha).$$

Für $S = \{acdb, abcd\}$ ist

$$\mathrm{consec}(S) = \mathcal{T}(\{a,b,c,d\}) \cup \Big\{\{c,d\},\{b,c,d\}\Big\}.$$

Lemma 1.39 *Sei T ein echter PQR-Baum ohne R-Knoten, dann gilt*

$$\mathit{Compl}(T) = \mathit{consec}(\mathit{cons}(T)).$$

Beweis: Übungsaufgabe. ■

Ist ein PQR-Baum ein PQ-Baum, dann können wir die Menge der konsistenten Frontiers auch indirekt durch die vollständige Menge $\mathrm{Compl}(T)$ beschreiben.

Lemma 1.40 *Ein echter PQR-Baum T besitzt genau dann keinen R-Knoten, wenn $\Pi(\Sigma, \mathit{Compl}(T)) \neq \emptyset$.*

Beweis: Übungsaufgabe. ■

Damit haben wir eine Charakterisierung, wann ein PQR-Baum R-Knoten besitzt. Insbesondere erhalten wir genau dann wieder PQ-Bäume, wenn die gegebene Menge von Restriktionen die C1P erfüllt.

Lemma 1.41 *Sei T ein echter PQR-Baum ohne R-Knoten, dann gilt*

$$\Pi(\Sigma, \mathit{Compl}(T)) = \mathit{cons}(T).$$

Beweis: Übungsaufgabe. ■

Damit wissen wir, dass die Menge $\mathrm{Compl}(T)$ ebenfalls die Struktur der Restriktionen beschreibt, wenn die Menge von Restriktionen die C1P erfüllt.

Theorem 1.42 *Sei $\mathcal{F}$ eine Menge von Restriktionen über Σ, die die C1P besitzt, und sei T ein echter PQR-Baum mit $\mathit{Compl}(T) = \overline{\mathcal{F}}$, dann gilt*

$$\Pi(\Sigma, \mathcal{F}) = \mathit{cons}(T).$$

Beweis: Übungsaufgabe. ■

Man kann sich überlegen, dass der PQR-Baum aus dem vorhergehenden Theorem dann keinen R-Knoten besitzt.

Damit wissen wir, dass wir für eine Menge $\mathcal{F}$ von Restriktionen nur einen PQR-Baum T mit $\text{Compl}(T) = \overline{\mathcal{F}}$ konstruieren müssen, um festzustellen, dass die Menge die C1P erfüllt. Andernfalls erhalten wir einen PQR-Baum, der R-Knoten enthält und uns so auf Problemstellen aufmerksam macht, die für die Menge $\mathcal{F}$ die C1P verhindert.

1.3.4 Orthogonalität

Nun führen wir noch den Begriff der Orthogonalität ein. Diesen benötigen wir, um die Mengen, die die Knoten des PQR-Baumes beschreiben, besser verstehen und beschreiben zu können.

Definition 1.43 *Sei Σ ein Alphabet und $A, B \subseteq \Sigma$. Dann heißen A und B* orthogonal *zueinander, bezeichnet mit $A \perp B$, wenn eine der folgenden Bedingungen erfüllt ist:*

- $A \subseteq B$,
- $A \supseteq B$ *oder*
- $A \cap B = \emptyset$.

Sei $A \subseteq \Sigma$ und $\mathcal{F} \subseteq 2^\Sigma$, dann ist genau dann $A \perp \mathcal{F}$, wenn $A \perp F$ für alle $F \in \mathcal{F}$. Es bezeichne

$$\mathcal{F}^\perp := \{A \subseteq \Sigma \,:\, A \perp \mathcal{F}\}.$$

Sind $\mathcal{F}, \mathcal{G} \subseteq 2^\Sigma$, dann gilt $\mathcal{F} \perp \mathcal{G}$ genau dann, wenn $F \perp G$ für alle $F \in \mathcal{F}$ und alle $G \in \mathcal{G}$.

Beachte insbesondere, dass für jede Menge $F \subseteq \Sigma$ gilt: $F \perp \mathcal{T}(\Sigma)$. Damit gilt auch $\mathcal{F} \perp \mathcal{T}(\Sigma)$ für jedes $\mathcal{F} \subseteq 2^\Sigma$.

07.05.24

Theorem 1.44 *Sei T ein echter PQR-Baum über einem Alphabet Σ und $v \in V(T)$. Dann gilt $D_T(v) \in \mathit{Compl}(T) \cap \mathit{Compl}(T)^\perp$. Umgekehrt gibt es für jede nichtleere Menge $H \in \mathit{Compl}(T) \cap \mathit{Compl}(T)^\perp$ einen Knoten $v \in V(T)$ mit $D_T(v) = H$.*

Beweis: $\Rightarrow$: Wir zeigen zuerst, dass $D_T(v) \in \text{Compl}(T)$ gilt. Ist v ein Blatt, dann ist $D_T(v) = \{v\} \in \mathcal{T}(\Sigma) \subseteq \text{Compl}(T)$. Sei also im Folgenden v ein interner Knoten von T und sei S die Menge aller Kinder von v. Dann gilt unabhängig vom Typ des Knotens v, dass $D_T(S) \in \text{Compl}(T)$. Da $D_T(v) = D_T(S)$, folgt die Behauptung.

Wir zeigen jetzt, dass $D_T(v) \in \text{Compl}(T)^\perp$ gilt. Zuerst halten wir noch fest, dass nach Definition (bzw. wegen der Anmerkung nach der Definition der Orthogonalität)

$\mathcal{T}(\Sigma) \subseteq \text{Comp}(T)^\perp$ gilt. Sei $A \in \text{Compl}(T)$ beliebig. Es genügt also zu zeigen, dass $D_T(v) \perp A$ gilt. Für $A \in \mathcal{T}(\Sigma)$ gilt dies ja bereits. Da $A \in \text{Compl}(T)$, existiert ein Knoten $x \in V(T)$ und eine Menge S von Kindern von x mit $A = D_T(S)$.

Ist v ein Vorgänger von x, dann gilt nach Definition $D_T(S) \subseteq D_T(v)$ und damit $D_T(S) \perp D_T(v)$ und somit auch $A \perp D_T(v)$.

Ist v ein Nachfolger eines Kindes von x aus der Menge S, dann gilt nach Definition $D_T(v) \subseteq D_T(S)$ und damit $D_T(S) \perp D_T(v)$ und somit auch $A \perp D_T(v)$.

Ist v ein Nachfolger eines Kindes von x, das nicht zu S gehört oder ist v weder ein Vorgänger noch ein Nachfolger von x, dann gilt offensichtlich $D_T(v) \cap D_T(S) = \emptyset$. Auch in diesem Fall gilt dann wieder $D_T(S) \perp D_T(v)$ und somit auch $A \perp D_T(v)$.

In allen Fällen gilt also $D_T(v) \perp A$ für beliebige Mengen $A \in \text{Compl}(T)$. Also ist $D_T(v) \in \text{Compl}(T)^\perp$.

$\Leftarrow$: Sei $H \in \text{Compl}(T) \cap \text{Compl}(T)^\perp$ mit $H \neq \emptyset$. Ist $H \in \mathcal{T}(\Sigma)$, dann existiert offensichtlich ein Blatt v oder die Wurzel v mit $H = D_T(v)$. Beachte, dass nach Voraussetzung $H \neq \emptyset$. Sei also im Folgenden $H \notin \mathcal{T}(\Sigma)$.

Da $H \in \text{Compl}(T)$ ist, gibt es einen Knoten $v \in V(T)$ und eine Menge S von Kindern von v mit $H = D_T(S)$. Ohne Beschränkung der Allgemeinheit nehmen wir im Folgenden an, dass $|S| > 1$ gilt. Wäre $S = \{v\}$, dann ersetzen wir S durch die Menge aller Kinder von v, wovon es mindestens zwei geben muss. War v ein Blatt, dann ist die Behauptung sowieso trivial (da dann $H \in \mathcal{T}(\Sigma)$).

Ist v ein P-Knoten, dann muss S die Menge aller Kinder von v sein und somit gilt $D_T(S) = D_T(v)$. Ist v ein Q- oder R-Knoten und S die Menge aller Kinder, so gilt dasselbe Argument wie für einen P-Knoten.

Sei also jetzt v ein Q-Knoten und S eine echte konsekutive Teilmenge von Kindern von v mit $|S| > 1$. Dann gibt es jedoch eine andere konsekutive Teilmenge S' von Kindern von v mit $S \not\perp S'$. Dann ist jedoch auch $D_T(S) \not\perp D_T(S')$ und somit $H = D_T(S) \notin \text{Compl}(T)^\perp$. Dieser Fall braucht also nicht betrachtet zu werden.

Sei also v ein R-Knoten und S eine beliebige echte Teilmenge von Kindern von v mit $|S| > 1$. Dann gibt es jedoch eine andere Teilmenge S' von Kindern von v mit $S \not\perp S'$. Dann ist jedoch auch $D_T(S) \not\perp D_T(S')$ und somit $H = D_T(S) \notin \text{Compl}(T)^\perp$. Dieser Fall braucht also ebenfalls nicht betrachtet zu werden. ■

1.3.5 Konstruktion von PQR-Bäumen

In diesem Abschnitt wollen wir jetzt konkret beschreiben, wie man zu einer gegebenen Menge $\mathcal{F}$ von Restriktionen einen PQR-Baum T konstruiert, der $\text{Compl}(T) = \overline{\mathcal{F}}$ erfüllt.

Theorem 1.45 *Für jede Menge $\mathcal{F} \subseteq 2^\Sigma$ von Restriktionen über einem Alphabet Σ existiert ein echter PQR-Baum T mit $Compl(T) = \overline{\mathcal{F}}$.*

Sei T ein PQR-Baum für $\mathcal{F}$ mit $\text{Compl}(T) = \overline{\mathcal{F}}$ und $F \subseteq \Sigma$ mit $F \notin \mathcal{F}$. Dann konstruieren wir aus T einen neuen PQR-Baum T' mit $\text{Compl}(T') = \overline{\mathcal{F} \cup \{F\}}$. Wir bestimmen dazu wieder den relevanten reduzierten Teilbaum und gehen jedoch dann diesmal top-down von der Wurzel des relevanten reduzierten Teilbaumes aus. Die Strategie des Algorithmus ist in Abbildung 1.28 angegeben.

(1) Markiere alle Blätter von T, deren Marken in F enthalten sind;

(2) Finde die Wurzel des relevanten reduzierten Teilbaumes $T_{rr}(T, F)$;

(3) Solange die Wurzel des relevanten reduzierten Teilbaumes partielle Kinder besitzt, baue die Wurzel mit diesem partiellen Kind um; aktualisiere dabei gegebenenfalls die Wurzel;

(4) Passe die Wurzel an;

(5) Entferne alle Markierungen;

Abbildung 1.28: Algorithmus: Erweiterung eines PQR-Baumes um eine Restriktion

Analog wie im Falle von PQ-Bäume charakterisieren wir die Knoten des PQR-Baumes, um den relevanten reduzierten Teilbaum aus alle partiellen und vollen Knoten bestimmen zu können.

Definition 1.46 *Sei T ein echter PQR-Baum über einem Alphabet Σ und $F \subseteq \Sigma$ eine Restriktion. Ein Knoten v heißt*

- voll*, wenn $D_T(v) \subseteq F$;*
- partiell*, wenn $D_T(v) \not\subseteq F$ gilt;*
- leer*, wenn $D_T(v) \cap F = \emptyset$ oder $F \subsetneq D_T(v)$ gilt.*

Die Wurzel des relevanten reduzierten Teilbaumes werden wir immer als leer betrachten, da uns hier der Zustand nicht wirklich interessiert. In der obigen Definition ist die Wurzel des relevanten reduzierten Teilbaumes auch als leer definiert, außer wenn der relevante reduzierte Teilbaum voll ist. In diesem Fall ist jedoch sowieso nichts zu tun, da dann eine implizite Restriktion eingebaut werden soll.

Muster	Aktion
PP	Transformation P-Knoten in Q-Knoten (T1) Schablone für **PQ**
$\mathbf{P}^{\mathbf{Q}}_{\mathbf{R}}$	Vorbereiten der Wurzel (T2) (Orientieren des Q-Knoten) Entfernen der Kinder von der Wurzel (T4)
$^{\mathbf{Q}}_{\mathbf{R}}\mathbf{P}$	Transformation P-Knoten in Q-Knoten (T1) Schablone für $^{\mathbf{Q}}_{\mathbf{R}}\mathbf{Q}$
$^{\mathbf{Q}\ \mathbf{Q}}_{\mathbf{R}\ \mathbf{R}}$	(Orientieren des Q-Knotens) Mischen in die Wurzel (T3)

Abbildung 1.29: Skizze: PQR-Schablonen

In Abhängigkeit von der betrachteten Wurzel und eines seiner partiellen Kinder führen wir die Operationen wie in Abbildung 1.29 aus. Dabei unterscheiden wir verschiedenen Fälle, je nachdem, ob die Wurzel und das betrachtete partielle Kind ein P- oder Q- bzw. R-Knoten ist. Dabei verwenden wir einige Grundregeln die in den Abbildungen 1.30, 1.31, 1.32 und 1.33 dargestellt sind. Hierbei sind volle Teilbäume bzw. Knoten rot, partielle hellrot und leere weiß dargestellt. Teilbäume, bei denen der Zustand voll, partiell oder leer unwichtig ist, sind grau dargestellt.

Wir beschreiben im Folgenden die in der Skizze der PQR-Schablonen angegeben Transformationen T1 mit T4. Auf die angegebenen Operationen zur *Orientierung* gehen wir nicht im Detail ein. Hierbei werden nur Q-Knoten so gedreht, dass sie (sofern möglich) mit den anderen markierten Teilbaumes konsekutive Bereiche formen. Falls diese nicht möglich sein sollte, so passiert eigentlich nichts und die entsprechenden Q-Knoten werden im Schritt 4 des Algorithmus zu R-Knoten.

Wie man sich leicht überlegt, ist die Transformation T1 in Abbildung 1.30 offensichtlich korrekt. Man beachte hierbei, dass die P-Knoten unter dem Q-Knoten natürlich nur dann eingefügt werden, wenn dort jeweils mindestens zwei volle bzw. leere Teilbäume angehängt werden. Auch hier ist es wieder möglich, dass wir einen Q-Knoten mit nur zwei Kindern erzeugen. Da aber nach der Transformation T1 noch weitere Transformationen folgen, die den Q-Knoten entweder in einen anderen Q-Knoten mischen, einen anderen Q-Knoten hineinmischen oder weitere Teilbäume anhängen, ist dies auch hier wieder nur eine temporäre Erscheinung und letztendlich ist der resultierende PQR-Baum wieder echt.

Man sollte sich auch an dieser Stelle schon klar machen, dass die Kosten proportional zu der Anzahl der vollen und partiellen Kinder des ehemalige P-Knoten sind, da wir den ehemaligen P-Knoten mitsamt seiner leeren Teilbäume zu einem Kind des

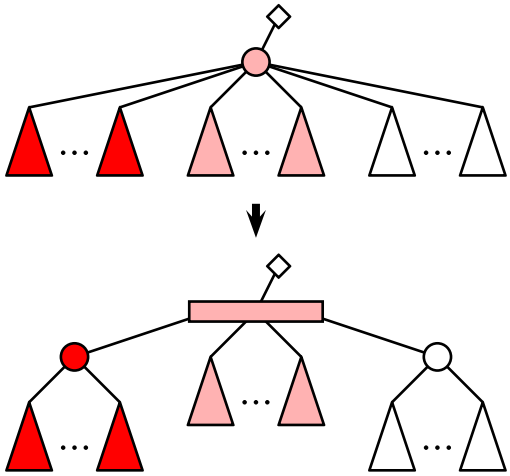


Abbildung 1.30: Skizze: Transformation P-Knoten in Q-Knoten (T1)

neuen Q-Knotens machen, um unbeteiligte (also leere) Teilbäume nicht anfassen zu müssen. Ansonsten könnten wir auch hier eine effiziente Implementierung wiederum nicht sicherstellen.

Auch die Transformation T2 in Abbildung 1.31 ist korrekt, da wir uns ja an der Wurzel des relevanten reduzierten Teilbaumes befinden und sich somit außerhalb dieses Teilbaumes keine markierten Blätter befinden können. Auch hier sind die Kosten wiederum proportional zur Anzahl der vollen und partiellen Kinder der betrachteten Wurzel.

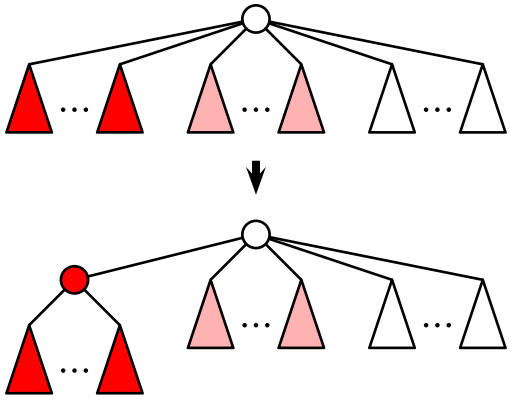


Abbildung 1.31: Skizze: Vorbereiten der Wurzel (T2)

In den folgenden Abbildungen werden Knoten, die entweder Q- oder R-Knoten sein können, durch breitgezogene Dreiecke symbolisiert.

Die Transformation T3 in Abbildung 1.32 ist ebenfalls korrekt, da wir einen partiellen Knoten in Q- bzw. R-Knoten mischen. Nur wenn der einzumischende Q-Knoten

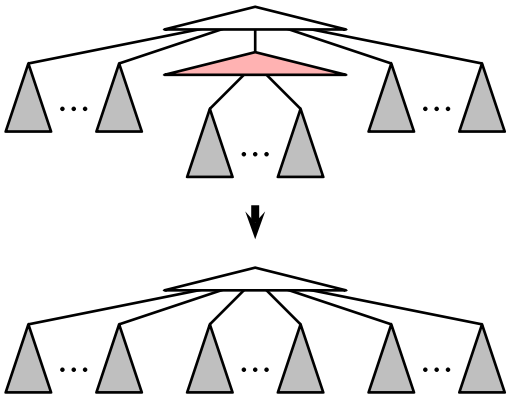


Abbildung 1.32: Skizze: Mischen in die Wurzel (T3)

voll wäre, könnte die Rotation weiterhin erlaubt bleiben. Ist einer der beteiligten Knoten ein R-Knoten, dann ist auch die resultierende Wurzel ein R-Knoten, andernfalls ein Q-Knoten.

Bei dieser Transformation könnte es passieren, dass die Kinder des resultierenden Q-Knotens nicht konsekutiv markiert sind. Dies wird aber in einer abschließenden Betrachtung geregelt, indem dann aus dem Q-Knoten ein R-Knoten wird. Hier sind die Kosten der Transformation sogar konstant. Dies können wir an der Stelle erst einmal nur behaupten, da dies von der konkreten Implementierung des PQR-Baums abhängt, aber diese wird dies unterstützen.

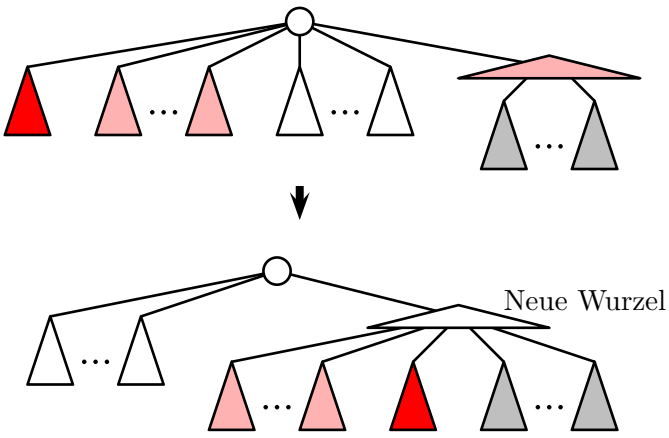


Abbildung 1.33: Skizze: Entfernen der Kinder von der Wurzel (T4)

Auch die Transformation T4 in Abbildung 1.33 ist korrekt, da die an der Wurzel hängenden vollen (es kann nur einer sein, da in den Schablonen T4 nur nach der Transformation T2 angewendet wird) und die partiellen konsekutiven zu den markierten Teilbäume des partiellen Q- bzw. R-Kindes gemacht werden. Falls die alte

Wurzel nur noch ein Kind hat, muss diese natürlich aus dem PQR-Baum eliminiert werden.

Auch hier sind die Kosten dieser Transformation offensichtlich proportional zur Anzahl der partiellen und vollen Kinder der aktuellen Wurzel.

Zum Schluss müssen wir noch die aktuelle Wurzel anpassen. Handelt es sich um einen P-Knoten, so werden wir noch die Transformation T2 anwenden. Handelt es sich um einen Q-Knoten, deren volle Kinder nicht konsekutiv sind oder die nicht geeignet konsekutive partielle Kinder besitzt, so wird dieser zu einem R-Knoten. Andernfalls tun wir natürlich nichts. Auch bei einem R-Knoten ist keine weitere Anpassung nötig.

Man überlege sich auch, dass wir mit diesem Vorgehen keinen R-Knoten mit nur zwei Kindern konstruieren.

14.05.24

In der Abbildung 1.34 auf Seite 52 ist ein Beispiel zur Konstruktion eines PQR-Baumes für die Restriktionsmenge

$$\mathcal{F} = \Big\{\{B, D, E, H\}, \{B, C, D, F\}, \{A, B, E, H\}, \{C, D, E, F\}\Big\}$$

angegeben.

1.3.6 Laufzeitanalyse

Im Gegensatz zur Konstruktion des PQ-Baumes, müssen wir auch bei der Ermittlung des relevanten reduzierten Teilbaumes jeweils die Eltern von Kindern von Q- bzw. R-Knoten kennen. Bei PQ-Bäumen konnte dies im Misserfolgsfall sehr teuer werden, was nur deshalb erträglich war, da dieser nur einmal eintreten konnte. Danach wurde die Konstruktion des PQ-Baumes abgebrochen. Im PQR-Baum kann dies jedoch bei der Einarbeitung jeder Restriktion passieren, da wir ja danach nicht abbrechen, sonder die Unverträglichkeit der Restriktionen mit Einführung von R-Knoten beheben. Wir werden im nächsten Abschnitt sehen, wie wir die Kindermengen organisieren müssen, damit der Elter jeweils im Mittel in Zeit $O(\log^*(n))$ ermittelt werden kann.

Wenn wir jetzt den gesamten PQR-Baum in linearer Zeit unter Nichtbeachtung der Elter-Bestimmung konstruieren können, erhalten wir als Gesamt-Laufzeit eine um den Faktor $O(\log^*(n))$ höheren Zeitaufwand, wobei n die Anzahl aller verwendeten Blätter und Knoten der konstruierten PQR-Bäume sind, also $n = \Theta(|\Sigma|)$.

Wir betrachten nun die Laufzeit der einzelnen Schritte gemäß des Algorithmus in Abbildung 1.28. Schritt 1 (Markierung der Blätter) geht offensichtlich in Zeit $O(\sum_{F\in\mathcal{F}} |F|)$. Schritt 2 können wir unter Nichtberücksichtigung der Kosten zur Elternermittlung von Q- und R-Knoten ähnlich wie bei PQ-Bäumen bestimmen.

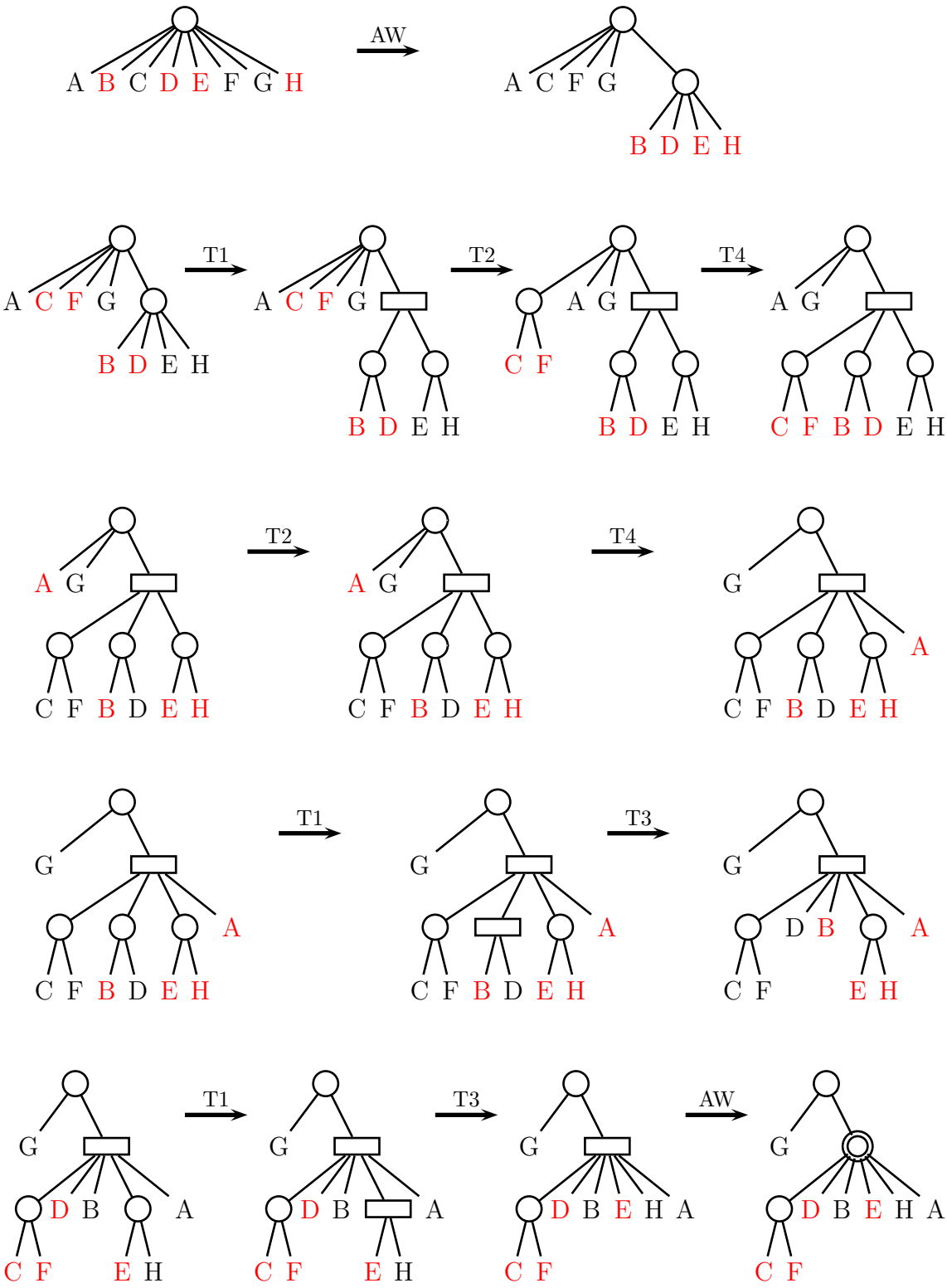


Abbildung 1.34: Beispiel: Konstruktion eines PQR-Baumes

Wir müssen hier nur berücksichtigen, dass es keine blockierten Knoten und somit auch keine Sektoren gibt. Auch hier kann es wieder passieren, dass wir nicht unbedingt die Wurzel des relevanten reduzierten Teilbaumes finden, sondern einen Knoten der Vorgänger dieser und Nachfolger der Wurzel des Gesamtbaumes ist. Wenn wir die freien Knoten wiederum in einer Queue aufbewahren, kann der besuchte Pfad von der Wurzel des relevanten reduzierten Teilbaumes aufwärts höchstens so lang sein wie der relevante reduzierte Teilbaum groß ist.

Die Kosten, den relevanten reduzierten Teilbaum zu finden, sind proportional zur Anzahl der nichtleeren Knoten (wenn wir die Zeit für die Bestimmung eines Elters eines Knoten nicht berücksichtigen). Die Anzahl der partiellen Knoten ist höchstens so groß wie die Laufzeit von Schritt 3, da dort ja jeder partielle Knoten besucht wird, also $O(\sum_{F\in\mathcal{F}}|F|)$, wie wir noch genauer sehen werden. Die Anzahl der vollen Knoten ist offensichtlich ebenfalls durch $O(\sum_{F\in\mathcal{F}}|F|)$ beschränkt.

Die Analyse von Schritt 3 ist am aufwendigsten und wir werden im Anschluss zeigen, dass dieser ebenfalls in Zeit $O(\sum_{F\in\mathcal{F}}|F|)$ durchführbar ist. Schritt 4 geht in konstanter Zeit plus der Kosten, die vollen Teilbäume abzuhängen, also ebenfalls insgesamt in Zeit $O(\sum_{F\in\mathcal{F}}|F|)$. Schritt 5 geht in Zeit $O(\sum_{F\in\mathcal{F}}|F|)$, wenn wir uns die markierten Knoten zur Entfernung der Markierung gemerkt haben.

Für die Analyse von Schritt 3 zeigen wir, dass die Kosten pro Anwendung einer Restriktion F plus der Veränderung der Norm durch $O(|F|)$ beschränkt ist. Hierfür definieren wieder einmal recht willkürlich die Norm eines PQR-Baumes wie folgt.

Definition 1.47 *Die* Norm eines PQR-Baumes T*, in Zeichen* $\|T\|$*, ist die Summe aus der Anzahl der Q- und R-Knoten plus der Anzahl der Kinder von P-Knoten.*

Man beachte, dass Q- und R-Knoten in der Norm zweimal gezählt werden können, nämlich genau dann, wenn sie ein Kind eines P-Knotens sind. Im Gegensatz zur Norm von PQ-Bäumen zählen wir hier auch Kinder von P-Knoten, selbst wenn diese Blätter sind.

Zuerst halten wir ein paar elementare Eigenschaften dieser Norm fest:

1. Es gilt $\|T\| \geq 0$ für alle PQR-Bäume T;
2. $\|T(\Sigma)\| = |\Sigma|$;
3. $\|T'\| - \|T\| \leq 1$, wenn T' aus T durch eine der Transformationen T_i hervorgeht.

Die ersten beiden Beziehungen sind offensichtlich, die dritte Beziehung folgt aus einen genauen Inspektion der vier Transformationen T1 mit T4 (wobei T3 und T4 die Norm tatsächlich erniedrigen).

Sei T ein PQR-Baum. Im Folgenden bezeichne $A(v)$ die Anzahl der vollen Kinder und $B(v)$ die Anzahl der partiellen Kinder eines Knotens $v \in V(T)$. Beachte, dass für einen inneren Knoten v im relevanten reduzierten Teilbaum immer $A(v) + B(v) \geq 1$ gilt.

Im Folgenden überlegen wir uns, welche Kosten bei einer Bearbeitung der aktuellen Wurzel r mit seinem partiellen Kind v entstehen. Zur Beweisführung verteilen wir die entstanden Kosten entweder auf volle Knoten, die dann anschließend zu einem Kind eines anderen vollen Knotens abgehängt werden, oder wir verrechnen die Kosten mit einer Normveränderung.

Wir werden die Norm quasi als ein Bankkonto verwenden, von dem wir etwas abheben, wenn eine Operation zu teuer wird (Normerniedrigung), oder etwas einzahlen (nach obigen Eigenschaften pro Transformation maximal 1 Einheit).

$\mathbf{P}_{\mathbf{R}}^{\mathbf{Q}}$-Schablone: Wir betrachten zuerst den Fall, dass $A(r) > 1$ gilt. Die Kosten zur Vorbereitung der Wurzel (T2) sind dann proportional zu $A(r) + 1$. Der Summand $+1$ kommt von der Erhöhung der Norm, für die wir eine Einheit auf das Bankkonto einzahlen müssen. Die Kosten zum Entfernen der Kinder von der Wurzel (T4) sind proportional zu $1 + (B(r) - 1) = B(r)$. Da jedes Umhängen in konstanter Zeit erledigt werden kann, sind somit die Gesamtkosten durch $O(A(r) + B(r))$ beschränkt. Darin sind auch die Sonderkosten bei einer Normerhöhung durch T2 enthalten.

Die Kosten $O(A(r))$ verteilen wir auf die Wurzeln der abgehängten vollen Teilbäume, wovon es ja gerade $A(r)$ viele gibt, so dass jede volle Wurzel konstante Kosten erhält. Die restlichen Kosten werden durch die Norm beglichen, da sich die Norm ja bei T4 gerade um mindestens $1 + (B(r) - 1) = B(r) \geq 1$ erniedrigt.

Wir merken an, dass die abgehängten Wurzeln von vollen Teilbäumen sich nun innerhalb von vollen Teilbäumen befinden können, und nach unserer Konstruktion jeder nicht-Wurzel-Knoten nur konstante Kosteneinheiten zugewiesen bekommen kann.

Betrachten wir jetzt den Fall, dass $A(r) = 1$ ist. Dann bleibt der Baum bei T2 unverändert und die Norm des Baumes erniedrigt sich bei T4 um mindestens $B(r) - 1 + 1 = B(r) \geq 1$. Somit können wir die Kosten von $O(B(r))$ mit der Normerniedrigung verrechnen.

Ist $A(r) = 0$, dann bleibt der Baum bei T2 gleich und die Norm des Baumes erniedrigt sich um $B(r) - 1 \geq 1$. Der Fall $B(r) = 1$ kann hier nicht eintreten, da dann r nicht die Wurzel des relevanten reduzierten Teilbaums sein kann. Somit können wir auch hier die Kosten von $O(B(r))$ mit der Normerniedrigung verrechnen.

Man beachte, dass selbst wenn sich in der Transformation T4 die Wurzel ändert, so hat die Wurzel immer mindestens zwei nichtleere (partielle oder volle) Bäume unter sich.

PP-Schablone: Hier wird vor der $\mathrm{P}^{\mathrm{Q}}_{\mathrm{R}}$-Schablone nur noch die Operation Transformiere P- in Q-Knoten (T1) am Kind v ausgeführt. Diese verursacht Kosten in Höhe von $O(A(v) + B(v))$. Ist $B(v) > 1$ und $A(v) > 1$, so können die Kosten von $O(B(v))$ mit der Normerniedrigung um $B(v) - 1 \geq 1$ und die Kosten von $O(A(v))$ über die Wurzeln der abgehängten vollen Knoten verrechnet werden.

Gilt nun $A(v) > 1$ und $B(v) \leq 1$, dann können wir die Kosten in Höhe von $O(A(v)+B(v)) = O(A(v))$ auf die $A(v)$ Wurzeln der abgehängten vollen Knoten verteilt werden. Man beachte, dass bei $A(v) > 1$ und $B(v) = 0$ die Norm auch wachsen kann. Dann müssen zusätzliche Kosten auch auf die abgehängten Wurzeln verteilt werden, um die Erhöhung des Kontostandes begleichen.

Gilt andererseits $A(v) \leq 1$ und $B(v) > 1$, dann können die Kosten in Höhe von $O(A(v) + B(v)) = O(B(v))$ mit der Normerniedrigung um $B(v) - 1 \geq 1$ verrechnet werden.

Es bleiben noch die Fälle $A(v), B(v) \in [0 : 1]$. Die Kosten sind dann in jedem Falle konstant und werden erst zusammen mit den Kosten der folgenden Schablone $\mathrm{P}^{\mathrm{Q}}_{\mathrm{R}}$ bezahlt.

16.05.24

$\mathbf{{}^{Q\,Q}_{R\,R}}$**-Schablone:** Das Mischen in die Wurzel (T3) ist in konstanter Zeit möglich. Da sich hierbei die Norm um genau 1 verringert, kann dies hiermit einfach verrechnet werden.

$\mathbf{{}^{Q}_{R}}$**P-Schablone:** Hier wird vor der ${}^{\mathrm{Q}\,\mathrm{Q}}_{\mathrm{R}\,\mathrm{R}}$-Schablone nur noch die Operation Transformiere P- in Q-Knoten (T1) ausgeführt. Die Verteilung der Kosten ist analog wie für die Schablone PP, da der Typ des Wurzelknotens bei der Argumentation überhaupt keine Rolle spielt.

Somit haben wir die Kosten für jede Transformation mit der Norm verrechnet oder auf einen Knoten verteilt, der anschließend innerhalb eines vollen Teilbaumes liegt.

Beachte, dass die Kosten bei der Schablone T1 auf die alten Enkel (also auf die neuen Urenkel) der Wurzel und bei Schablone T2 auf die alten Kinder (also die neuen Enkel) der Wurzel verteilt wurden. Da nach Zuweisung von Kosten die Wurzel jeweils um 1 weiter entfernt ist als vorher, können jedem vollen Knoten höchstens zweimal konstante Kosten zugewiesen werden, bevor er als Urenkel der Wurzel (oder Nachfahre hiervon) nie wieder Kosten zugewiesen bekommen kann.

Für das Folgende bezeichne work(T, F) die Anzahl der Umhänge-Operationen (auch leerer), die nötig sind, um in einen PQR-Baum T die Restriktion F einzubauen. Die

Kosten, um die Information über den Elter eines Kindes abzuholen, werden hierbei noch nicht berücksichtigt.

Lemma 1.48 *Sei T ein echter PQR-Baum über einem Alphabet Σ, $F \subseteq \Sigma$ eine Restriktion und T' der echte PQR-Baum, der durch die Einarbeitung der Restriktion F aus T entsteht. Dann existiert eine Konstante c, so dass gilt*

$$\mathit{work}(T, F) \leq 4c|F| + c(\|T\| - \|T'\|).$$

Dieses Lemma folgt unmittelbar aus der vorherigen Argumentation und der Beobachtung, dass die Summe aller voller Knoten (die einen Wald als Teilgraphen des resultierenden PQR-Baumes bilden) durch die doppelte Anzahl der markierten Knoten (also $|F|$) beschränkt ist. Weiterhin werden jedem vollen Knoten höchstens zweimal konstante Kosten zugewiesen.

Der zweite Term (die Normveränderung um $\|T\|-\|T'\|$) sind im negativen Fall (Normerhöhung des PQR-Baums T') die Einzahlung auf das Bankkonto, die ja mit Kosten auf die abgehängten vollen Knoten bezahlt wird und somit wieder korrigiert werden muss), und im positiven Fall (Normerniedrigung des PQR-Baums T') das Bezahlen der überschüssigen Kosten aus dem Bankkonto, die ja den Gesamtkosten hinzugefügt werden müssen.

Daraus erhalten wir sofort den folgenden Satz.

Theorem 1.49 *Sei Σ ein Alphabet und $\mathcal{F} \subseteq 2^\Sigma$ eine Menge von Restriktionen. Die Menge $\Pi(\Sigma, \mathcal{F})$ kann durch einen echten PQR-Baum T mit $\mathit{Compl}(T) = \overline{\mathcal{F}}$ dargestellt und mit $O\left(|\Sigma| + \sum_{F\in\mathcal{F}} |F|\right)$ Umhängeoperationen berechnet werden.*

Beweis: Sei $\mathcal{F} = \{F_1, \ldots, F_k\}$ und $T_0, T_1, \ldots, T_k$ die konstruierten echten PQR-Bäume mit $\mathrm{Compl}(T_i) = \overline{\{F_1, \ldots, F_i\}}$. Für die Anzahl durchgeführter Operationen gilt nach dem vorheriger Lemma

$$\begin{aligned}
\mathrm{work}(\mathcal{F}) := \sum_{j=1}^{k} \mathrm{work}(T_{j-1}, F_j) &\leq \sum_{j=1}^{k} 4c|F_j| + c\sum_{j=1}^{k} (\|T_{j-1}\| - \|T_j\|) \\
&\leq 4c\sum_{j=1}^{k} |F_j| + c\cdot\|T_0\| - c\cdot\|T_k\| \\
&\quad \text{da } \|T_k\| \geq 0 \text{ und } \|T_0\| = |\Sigma| \\
&\leq 4c\sum_{j=1}^{k} |F_j| + c|\Sigma|.
\end{aligned}$$

Da die Gesamtkosten $\mathrm{work}(\mathcal{F}) + c\cdot\|T_0\|$ betragen, folgt die Behauptung, ■

Für die Bestimmung der Elter-Information verwenden wir eine so genannte Union-Find-Datenstruktur, wie sie im folgenden Abschnitt näher erläutert wird. Diese stellt die folgenden zwei Operationen zur Verfügung:

Find-Operation: Für ein Kind in einer Menge wird ein ausgewähltes Kind dieser Menge bestimmt, von dem wir dann ausgehen, dass es seinen Elter kennt.

Union-Operation: Vereinigt zwei Mengen von Kindern (die dann Kinder eines Q- oder R-Knotens sein werden) und gibt dessen Index zurück, d.h. das ausgewählte Kind dieser Menge, das seinen Elter kennt.

Die Mengen werden dabei als Bäume dargestellt, wobei die Kanten von den Blättern zur Wurzel gerichtet sind. Die Knoten des Baumes sind die zur Menge gehörigen Kinder und die Wurzel ist das ausgewählte Kind.

Kinder eines P-Knotens werden dabei immer jeweils in einer ein-elementigen Menge gehalten (also in einem Baum aus einem Knoten). Die Kinder eines Q- oder R-Knotens werden in einer einzigen Menge gehalten. Das ist sinnvoll, da Kinder eines P-Knotens bei den Transformationen auch getrennt werden, während Kinder eines Q- oder R-Knotens immer nur mit Kindern eines anderen Knotens vereinigt werden.

Wie wir im nächsten Abschnitt sehen werden, können diese Operationen auf einer Menge von n Elementen in konstanter Zeit für eine Union-Operation und in Zeit $O(\log^*(n))$ für eine Find-Operation implementiert werden. Wie viele Kinder können wir insgesamt während der Erzeugung eines PQR-Baumes haben (dabei ist zu beachten, dass auch Kinder wieder verschwinden können)? Im schlimmsten Fall kann jede Umhänge-Operation eine konstante Anzahl neuer Kinder erzeugen. Somit sind in der Union-Find-Datenstruktur im schlimmsten Falle $n = O(|\Sigma| + \sum_{F\in\mathcal{F}} |F|)$ Kinder enthalten. Damit ergibt sich unmittelbar das folgende Theorem:

Theorem 1.50 *Sei Σ ein Alphabet und $\mathcal{F} \subseteq 2^\Sigma$ eine Menge von Restriktionen. Die Menge $\Pi(\Sigma, \mathcal{F})$ kann durch einen echten PQR-Baum T mit Compl$(T) = \overline{\mathcal{F}}$ dargestellt und in Zeit $O\left(\left(|\Sigma| + \sum_{F\in\mathcal{F}} |F|\right) \cdot \log^*\left(|\Sigma| + \sum_{F\in\mathcal{F}} |F|\right)\right)$ berechnet werden.*

Beweis: Wie wir im vorherigen Satz gesehen haben, können maximal

$$O\left(|\Sigma| + \sum_{F\in\mathcal{F}}^{k} |F|\right)$$

Umhänge-Operationen ausgeführt werden. In jedem Fall sind die Union-Operationen konstant und eine eventuelle Find-Operation kostet $O(\log^*(|\Sigma| + \sum_{F\in\mathcal{F}} |F|))$ Zeit.

Insgesamt erhalten wir für den Zeitbedarf also

$$O\left(\left(|\Sigma| + \sum_{F\in\mathcal{F}} |F|\right) \cdot \log^*\left(|\Sigma| + \sum_{F\in\mathcal{F}} |F|\right)\right).$$

Es bleiben noch die Kosten von Schritt 2. Man überlegt sich jedoch leicht, dass aus den vorhergehenden Diskussionen folgt, dass die Größe der relevanten reduzierten Teilbäume durch $O(|\Sigma| + \sum_{F\in\mathcal{F}} |F|)$ beschränkt ist. Bei der Abarbeitung verfolgen wir ja alle partiellen und vollen Knoten. In volle Teilbäume dringt unserer Prozedur zwar nicht ein, aber alle vollen Knoten werden bei der Abschätzung der Anzahl Umhängeoperationen berücksichtigt. Die Größe des relevanten reduzierte Teilbaumes entspricht jedoch genau der Anzahl der Umhänge-Operationen plus der Anzahl voller Knoten. Da wir jede Elterbestimmung in Zeit $O(\log^*(|\Sigma| + \sum_{F\in\mathcal{F}} |F|))$ durchführbar ist, ist die Zeitdauer von Schritt 2 genauso groß wie die von Schritt 3. ■

1.4 Exkurs: Union-Find-Datenstrukturen

In diesem Abschnitt machen wir einen kurzen Exkurs, um die im letzten Abschnitt erwähnten Methoden zu erläutern.

1.4.1 Problemstellung

Jetzt müssen wir noch genauer auf die so genannte Union-Find-Datenstruktur eingehen. Eine *Union-Find-Datenstruktur* für eine Grundmenge U beschreibt eine Partition $\mathcal{P} = \{P_1, \ldots, P_\ell\}$ für U mit $U = \bigcup_{i=1}^{\ell} P_i$ und $P_i \cap P_j = \emptyset$ für alle $i \neq j \in [1 : \ell]$. Dabei ist zu Beginn $\mathcal{P} = \{P_1, \ldots, P_{|U|}\}$ mit $P_u = \{u\}$ für alle $u \in U$. Weiterhin werden die beiden folgenden elementaren Operationen zur Verfügung gestellt:

Find-Operation: Gibt für ein Element $u \in U$ den Index der Menge in der Mengenpartition zurück, die u enthält.

Union-Operation: Vereinigt die beiden Menge mit Index i und Index j und vergibt für diese vereinigte Menge einen neuen Index.

Ziel ist es nun diese Datenstruktur effizient zu implementieren, so dass die beiden Operationen Union und Find jeweils mit minimalem Zeitaufwand durchführbar sind. Wir werden hierzu zwei verschiedene mögliche Implementationen vorstellen.

1.4.2 Realisierung durch Listen (*)

Die Realisierung erfolgt durch zwei Felder. Dabei nehmen wir der Einfachheit halber an, dass $U = [1 : n]$ ist. Ein Feld von ganzen Zahlen namens Index gibt für jedes Element $u \in U$ an, welchen Index die Menge besitzt, die u enthält. Ein weiteres Feld von Listen namens Liste enthält für jeden Mengenindex eine Liste von Elementen, die in der entsprechenden Menge enthalten sind.

Union-Find (int n)

```
Initialize(int n)
begin
    int Index[n];
    ⟨int⟩ Liste[n];
    for (i := 1; i ≤ n; i++) do
        Index[i] := i;
        Liste[i] := ⟨i⟩;
end

int Find(int u)
begin
    return Index[u];
end

int Union(int i, j)
begin
    // provided that i and j are different sets
    int Small, Big;
    if Liste[i].size() ≤ Liste[j].size() then
        Small := i;
        Big := j;
    else
        Small := j;
        Big := i;
    forall (u ∈ Liste[Small]) do
        Liste[Big].add(u);
        Index[u] := Big;
        Liste[Small].remove(u);
    return Big;
end
```

Abbildung 1.35: Algorithmus: Union-Find

Die Implementierung der Prozedur Find ist nahe liegend. Es wird einfach der Index zurückgegeben, der im Feld Index gespeichert ist. Für die Union-Operation werden wir die Elemente einer Menge in die andere kopieren. Die umkopierte Menge wird dabei gelöscht und der entsprechende Index der wiederverwendeten Menge wird dabei recycelt. Um möglichst effizient zu sein, werden wir die Elemente der kleineren Menge in die größere Menge kopieren. Die detaillierte Implementierung ist in Abbildung 1.35 angegeben.

Wir überlegen uns jetzt noch die Laufzeit dieser Union-Find-Datenstruktur. Hierbei nehmen wir an, dass wir k Find-Operationen ausführen und maximal $n-1$ Union-Operationen. Mehr Union-Operationen machen keinen Sinn, da sich nach $n-1$ Union-Operationen alle Elemente in einer Menge befinden.

Offensichtlich kann jede Find-Operation in konstanter Zeit ausgeführt werden. Für die Union-Operation ist der Zeitbedarf proportional zur Anzahl der Elemente in der kleineren Menge, die in der Union-Operation beteiligt ist. Somit ergibt sich für die maximal $n-1$ möglichen Union-Operationen:

$$\sum_{\sigma=(L,L')} \sum_{\substack{i \in L \\ |L| \leq |L'|}} O(1).$$

Um diese Summe jetzt besser abschätzen zu können vertauschen wir die Summationsreihenfolge. Anstatt die äußere Summe über die Union-Operation zu betrachten, summieren wir für jedes Element in der Grundmenge, wie oft es bei einer Union-Operation in der kleineren Menge sein könnte.

$$\sum_{\sigma=(L,L')} \sum_{\substack{i \in L \\ |L| \leq |L'|}} O(1) = \sum_{u \in U} \sum_{\substack{\sigma=(L,L') \\ u \in L \wedge |L| \leq |L'|}} O(1).$$

Was passiert mit einem Element, dass sich bei einer Union-Operation in der kleineren Menge befindet? Danach befindet es sich in einer Menge, die mindestens doppelt so groß wie vorher ist, da diese mindestens so viele neue Element in die Menge hinzubekommt, wie vorher schon drin waren. Damit kann jedes Element maximal $\log(n)$ Mal in einer kleineren Menge bei einer Union-Operation gewesen sein, da sich dieses Element dann in einer Menge mit mindestens n Elementen befinden muss.

Da die Grundmenge aber nur n Elemente besitzt, kann danach überhaupt keine Union-Operation mehr ausgeführt werden, da sich dann alle Elemente in einer Menge befinden. Somit ist die Laufzeit für ein Element durch $O(\log(n))$ beschränkt. Da es maximal n Elemente gibt, ist die Gesamtlaufzeit aller Union-Operationen durch $O(n \log(n))$ beschränkt.

Theorem 1.51 *Sei U mit $|U| = n$ die Grundmenge für die vorgestellte Union-Find-Datenstruktur. Die Gesamtlaufzeit von k Find- und maximal $n-1$ Union-Operationen ist durch $O(k + n\log(n))$ beschränkt.*

1.4.3 Darstellung durch Bäume

Als zweite Möglichkeit speichern wir die Mengen als Bäume ab, wobei die Kanten hier von den Kindern zu den Eltern gerichtet sind. Am Anfang bildet jede einelementige Menge einen Baum, der aus nur einem Knoten besteht. Bei der `union`-Operation wird dann die Wurzel des kleineren Baumes (also der kleineren Menge) zum Kind der Wurzel des größeren Baumes (also der größeren Menge).

Bei der `find`-Operation wandern wir von dem dem Element zugeordneten Knoten bis zur Wurzel des zugehörigen Baumes. Der Index der Wurzel gibt dann den Namen der Menge an, in der sich das gesuchte Element befindet. Die zweite Variante der Realisierung einer Union-Find-Datenstruktur ist im Bild 1.36 dargestellt, allerdings wird hier die zweite **while**-Schleife für die später erläuterte Pfadkompression noch nicht ausgeführt.

Offensichtlich hat die `union`-Operation nun Zeitkomplexität $O(1)$. Dafür ist die Zeitkomplexität der `find`-Operation gestiegen. Die Zeit für eine `find`-Operation ist durch die maximale Höhe der entstandenen Teilbäume beschränkt.

Lemma 1.52 *Ein in der zweiten Union-Find-Datenstruktur entstandener Baum mit Höhe h besitzt mindestens 2^h Knoten.*

Beweis: Wir beweisen diese Aussage mit vollständiger Induktion über die Höhe der Bäume.

Induktionsanfang ($h = 0$): Nach unserer Definition der Höhe ist ein Baum mit Höhe 0 gerade der Baum, der aus einem Knoten besteht. Damit ist der Induktionsanfang gelegt.

Induktionsschritt ($h \to h + 1$): Sei T ein Baum der Höhe $h + 1$. Wir betrachten den Prozess von Union-Operationen bei der Konstruktion von T. Sei dabei T' der erste entstandene Teilbaum (im Sinne eines Teilgraphen) von T, der eine Höhe $h+1$ besitzt (siehe auch Abbildung 1.37). Es gilt nach Konstruktion $|V(T)| \geq |V(T')|$. Der Baum T' der Höhe $h+1$ muss durch Anhängen eines Baumes T'' mit Höhe h an die Wurzel eines Baumes T''' entstanden sein. Nach Definition von T' hat der Baum T''' eine Höhe von maximal h. Nach Induktionsvoraussetzung hat der Baum T'' mit Höhe h mindestens 2^h Knoten. Nach Konstruktion wird die Wurzel des Baumes T'' nur dann ein Kind der Wurzel des Baumes T''', wenn dieser mindestens genauso

Union-Find-2 (int n)

```
Initialize (int n)
begin
    int size[n];
    int parent[n];
    for (int i := 0; i < n; i++) do
        size[i] := 1 ;                          /* permissible for roots only */
        parent[i] := i; ;                          /* a root points to itself */
end

int Find(int i)
begin
    int j := i;
    while (parent[j] ≠ j) do
        j := parent[j];
    int root := j;

    // path compression
    int p;
    j := i;
    while (parent[j] ≠ j;) do
        p := parent[j];
        parent[j] := root;
        j := p;
    // end of path compression

    return root;
end

int Union(int i, int j)
begin
    // provided that i and j are roots
    int root := (size[i] > size[j])?i : j;
    int child := (size[i] > size[j])?j : i;
    parent[child] := root;
    size[root] += size[child];
    return root;
end
```

Abbildung 1.36: Realisierung einer Union-Find-Datenstruktur mit Bäumen

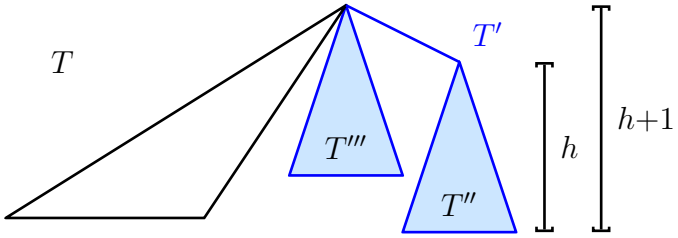


Abbildung 1.37: Skizze: Teilbaum der Höhe $h+1$

viele Knoten hat. Also hat der Baum T' mit Höhe $h+1$ mindestens $2^h + 2^h = 2^{h+1}$ Knoten. Somit hat auch der Baum T mindestens 2^{h+1} Knoten. ■

Damit kostet nun eine `find`-Operation $O(\log(n))$, da wegen Lemma 1.52 die Höhe der konstruierten Bäume maximal $O(\log(n))$ sein kann. Da bei unserem Algorithmus aber durchaus n `find`-Operationen auftauchen können, haben wir mit dieser zweiten Union-Find-Datenstruktur zuerst einmal nichts gewonnen. Wir werden im nächsten Abschnitt sehen, wie wir die `find`-Operation im Mittel doch noch beschleunigen können.

Theorem 1.53 *Unter Verwendung von Bäumen für eine Grundmenge von n Elementen benötigt jede `find`-Operation Zeit $O(\log(n))$ und jede `union`-Operationen Zeit $O(1)$.*

1.4.4 Pfadkompression

Wir zeigen jetzt, wie wir die `find`-Operation noch beschleunigen können. Jedesmal wenn wir bei einer `find`-Operation auf einem Pfad durch einen Baum laufen, werden alle besuchten Knoten zu Kindern der Wurzel. Dadurch wird diese `find`-Operation zwar nicht billiger, allerdings werden viele der folgenden `find`-Operationen erheblich billiger. Die oben genannte Umordnung des Baumes, in dem Knoten des Suchpfades zu Kindern der Wurzel werden, nennt man *Pfadkompression* (engl. *path compression*). Diese zweite Variante der Realisierung einer Union-Find-Datenstruktur mit Hilfe von Bäumen wurde ja bereits im Bild 1.36 dargestellt. Zu deren Analyse müssen wir noch zwei Funktionen definieren.

Definition 1.54 *Es ist $2 \uparrow\uparrow 0 := 1$ und $2 \uparrow\uparrow n := 2^{2\uparrow\uparrow(n-1)}$. Mit $\log^*$ bezeichnen wir die diskrete Umkehrfunktion von $2 \uparrow\uparrow (\cdot)$, also $\log^*(n) = \min\{k : 2 \uparrow\uparrow k \geq n\}$.*

Man beachte, dass $2 \uparrow\uparrow (\cdot)$ eine sehr schnell wachsende und damit $\log^*$ eine sehr langsam wachsende Funktion ist. Es gilt $2 \uparrow\uparrow 1 = 2$, $2 \uparrow\uparrow 2 = 4$, $2 \uparrow\uparrow 3 = 16$,

$2 \uparrow\uparrow 4 = 65536$, $2 \uparrow\uparrow 5 = 2^{65536}$. Für alle praktischen Zwecke ist $\log^*(n) \leq 5$, da $2^{65536} > (2^{10})^{6553} > (10^3)^{6553} = 10^{19659} \gg 10^{81}$ und damit $2 \uparrow\uparrow 5$ schon wesentlich größer als die vermutete Anzahl von Atomen im sichtbaren Weltall ist. Damit können Eingaben der Größe $2 \uparrow\uparrow 5$ schon gar nicht mehr direkt konstruiert werden!

Sei σ eine Folge von `union`- und `find`-Operationen. Sei F im Folgenden der Wald, der entstanden ist, wenn *nur* die `union`-Operationen in σ ausgeführt werden (also genau genommen keine Pfadkompression stattfindet). Mit dem *Rang* eines Knotens v wollen wir die Länge eines längsten Pfades von v zu einem Blatt seines Baumes in diesem Wald F bezeichnen.

Lemma 1.55 *Es gibt maximal $n/2^r$ Knoten mit Rang r im Wald F.*

Beweis: Zuerst bemerken wir, dass verschiedene Knoten mit demselben Rang verschiedene Vorgänger im Baum haben müssen. Man beachte, dass Vorgänger hier Knoten auf den Pfaden zu den Blättern sind, da die Kanten ja zur Wurzel hin gerichtet sind. Für eine Illustration des Folgenden siehe auch Abbildung 1.38. Nach

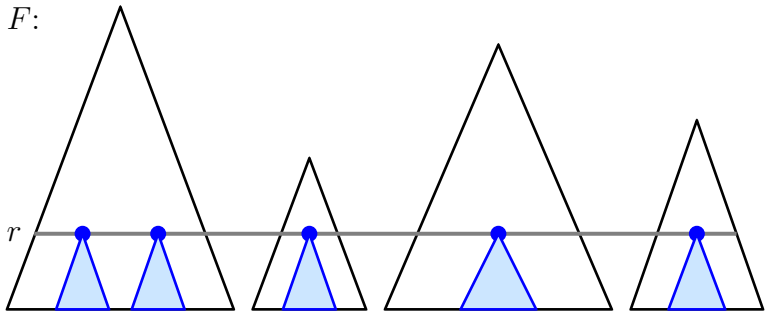


Abbildung 1.38: Skizze: Anzahl Knoten mit Rang R

Lemma 1.52 hat jeder Knoten mit Rang r mindestens 2^r verschiedene Vorgänger (sich selbst eingeschlossen). Für die Anwendung des Lemmas 1.52 ist zu beachten, dass ein Knoten, wenn er nicht mehr Wurzel eines Baumes ist, keine neuen Kinder mehr bekommen kann und dass kein Knoten ein Kind verlieren kann, da in F keine Pfadkompression ausgeführt wird. Des Weiteren überlegt man sich, dass Lemma 1.52 auch für jeden gewurzelten Teilbaum eines Baumes von F gilt. Gäbe es mehr als $n/2^r$ Knoten vom Rang r, so müsste der Wald mehr als n Knoten besitzen. ■

Damit können wir sofort folgern, dass kein Knoten einen Rang größer als $\log(n)$ besitzen kann.

Ist irgendwann bei der Abarbeitung der Folge σ von `union`- und `find`-Operationen v ein direkter Vorgänger von w, dann ist der Rang von v kleiner als der Rang von w.

Dies folgt aus der Tatsache, dass die Pfadkompression die Vorgängerrelation respektiert, d.h. ein Knoten wird bei der Pfadkompression nur dann ein direkter Vorgänger eines anderen Knotens, wenn er bereits ein Vorgänger gewesen war. Also ist v auch ein Vorgänger von w, wenn man die `find`-Operationen (samt Pfadkompression) aus σ nicht ausführt. Daraus folgt sofort die obige Aussage über die Ränge der Knoten von direkten Vorgängern.

Nun teilen wir die Knoten in Gruppen auf. Die Knoten mit Rang i ordnen wir der Gruppe mit Nummer $\log^*(i)$ zu. In der Gruppe g befinden sich nur Elemente, die einen Rang von aus dem Intervall zwischen $2 \uparrow\uparrow (g-1)+1$ und $2 \uparrow\uparrow g$ besitzen. Hierbei verwenden wir $2 \uparrow\uparrow (-1) := -\infty$. Zum Beispiel gelangen die Knoten mit Rängen zwischen 5 und 16 in die Gruppe mit Nummer 3 sowie alle mit Rang 0 oder 1 eins in die Gruppe 0. Wir verwenden auch hier wieder den *Buchhaltertrick* und nehmen dazu an, dass die Kosten der `find`-Operation genau der Anzahl der Knoten des Pfades entsprechen, der zum Auffinden der Wurzel durchlaufen wird. Wir belasten auch diesmal wieder die einzelnen Knoten mit den Kosten und berechnen hinterher die Gesamtschuld aller Knoten.

Zuerst belasten wir also jeden Knoten des Suchpfades mit einer Kosteneinheit. Offensichtlich bilden die Ränge der Knoten auf dem Suchpfad zur Wurzel eine streng monoton aufsteigende Folge. Nun begleichen die Wurzel, die Kinder der Wurzel und diejenigen Knoten, deren Elter in einer anderen Gruppe liegen, diese neue Schuld sofort. Da nach Lemma 1.52 der maximale Rang $\log(n)$ ist, ist die maximale Gruppe durch $\log^*(\log(n)) = \log^*(n) - 1$ gegeben. Da es dann maximal $\log^*(n)$ verschiedene Gruppen gibt (die kleinste Gruppennummer ist 0), kostet jede `find`-Operation maximal $\log^*(n) - 1 + 2 = \log^*(n) + 1$ (der Summand $+2$ steht für die Kinder der Wurzel und die Wurzel selbst).

Wir groß ist nun die verbleibende Schuld eines Knotens am Ende des Ablaufs? Immer wenn ein Knoten Schulden macht, erhält er wegen der Pfadkompression einen neuen Elter, nämlich die Wurzel des Baumes. Dabei erhält er also einen neuen Elter, dessen Rang um mindestens 1 größer ist. Nur Kinder einer Wurzel bilden hierbei eine Ausnahme (aber diese Knoten haben ihre Schulden ja sofort beglichen).

Wie bereits erwähnt, befinden sich in der Gruppe g nur Elemente, die einen Rang von aus dem Intervall zwischen $2 \uparrow\uparrow (g-1)+1$ und $2 \uparrow\uparrow g$ besitzen. Also macht ein Knoten maximal $2 \uparrow\uparrow g - 2 \uparrow\uparrow (g-1) \leq 2 \uparrow\uparrow g$ Schulden, bevor er einen Elter in einer höheren Gruppe zugewiesen bekommt. Nach der Zuweisung eines Elters einer höheren Gruppe macht der Knoten nach Konstruktion nie wieder Schulden, sondern zahlt die Kosten jeweils sofort.

Sei nun $G(g)$ die Anzahl der Knoten in Gruppe g. Dann erhalten wir mit der obigen Beobachtung, dass es nur $n/2^r$ Knoten mit demselben Rang geben kann (siehe

Lemma 1.55), folgende Abschätzung:

$$\begin{aligned} G(g) &\leq \sum_{r=2\uparrow\uparrow(g-1)+1}^{2\uparrow\uparrow g} \frac{n}{2^r} \\ &\leq \sum_{r=2\uparrow\uparrow(g-1)+1}^{\infty} \frac{n}{2^r} \\ &\leq \frac{n}{2^{2\uparrow\uparrow(g-1)}} \cdot \underbrace{\sum_{i=1}^{\infty} \frac{1}{2^i}}_{=1} \\ &\leq \frac{n}{2^{2\uparrow\uparrow(g-1)}} \\ &= \frac{n}{2 \uparrow\uparrow g}. \end{aligned}$$

Jeder Knoten in der Gruppe mit Nummer g macht wie bereits gesehen höchstens Schulden in Höhe von $2 \uparrow\uparrow g$. Damit machen alle Knoten in der Gruppe g Schulden in Höhe von $O(\frac{n}{2\uparrow\uparrow g}(2 \uparrow\uparrow g)) = O(n)$. Da es maximal $\log^*(n)$ verschiedene Gruppen gibt, ist die Gesamtschuld $O(n \log^*(n))$. Damit haben wir das folgende Theorem bewiesen.

Theorem 1.56 *Zur Ausführung von $f(n)$ `find`- und maximal n `union`- Operationen basierend auf der Union-Find-Datenstruktur mit Baumdarstellung und Pfadkompression wird maximal Zeit $O(n \log^*(n) + f(n) \log^*(n))$ benötigt.*

Beweis: Die Behauptung folgt unmittelbar aus der vorherigen Diskussion. Der erste Term entspricht den entstandenen Schulden, der zweite Term den sofort bezahlten Kosten. ■

Solange $f(n) = O(n)$ ist, gilt das folgende Korollar.

Korollar 1.57 *Zur Ausführung von $O(n)$ `union`- und `find`-Operationen basierend auf der Union-Find-Datenstruktur mit Baumdarstellung und Pfadkompression wird maximal Zeit $O(n \log^*(n))$ benötigt.*

Damit wird im Mittel Zeit $O(\log^*(n))$ pro `find`-Operation benötigt. Beim realen Einsatz von Union-Find-Datenstrukturen (auch bei den PQR-Bäumen) werden $\Theta(n)$ `union`-Operationen ausgeführt und daher auch $\Theta(n)$ `find`-Operationen ausgeführt (da einer `union`-Operation in der Regel zwei `find`-Operationen vorausgehen).

23.05.24

1.5 Weitere Varianten für die C1P

In diesem Abschnitt stellen wir noch kurz einige weitere Verfahren zur Erkennung der Consecutive Ones Property von Matrizen vor.

1.5.1 PC-Bäume (*)

Für die bereits in den Übungen kennen gelernte Circular Ones Property von 0-1-Matrizen kann man eigens einen direkten Algorithmus zur Feststellung dieser Eigenschaft angeben. Dazu benötigen wir die so genannten PC-Bäume. Doch geben wir zuerst noch einmal die Definition der Circular Ones Property an.

Definition 1.58 *Eine 0-1-Matrix M besitzt die* Circular Ones Property*, wenn es eine Permutation der Spalten gibt, so dass in jeder Zeile die Einsen oder die Nullen eine konsekutive Folge bilden.*

Die linke Matrix in Abbildung 1.39 erfüllt die Circular Ones Property. Dies sieht man, wenn man die erste und dritte Spalte vertauscht, wie in der rechten Matrix dargestellt.

$$\begin{pmatrix} 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 \end{pmatrix} \qquad \begin{pmatrix} 0 & 0 & 0 & 1 & 1 \\ 0 & 1 & 1 & 1 & 0 \\ 1 & 1 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 \end{pmatrix}$$

Abbildung 1.39: Beispiel: Circular Ones Property

Definition 1.59 *Sei Σ ein endliches Alphabet. Dann ist ein* PC-Baum *über Σ induktiv wie folgt definiert:*

- *Jeder einelementige Baum (also ein Blatt), das mit einem Zeichen aus Σ markiert ist, ist ein PC-Baum.*
- *Sind $T_1, \ldots, T_k$ PC-Bäume, dann ist der Baum, der aus einem so genannten* P-Knoten *als Wurzel entsteht und dessen Kinder die Wurzeln der Bäume $T_1, \ldots, T_k$ sind, ebenfalls ein PC-Baum.*
- *Sind $T_1, \ldots, T_k$ PC-Bäume, dann ist der Baum, der aus einem so genannten* C-Knoten *als Wurzel entsteht und dessen Kinder die Wurzeln der Bäume $T_1, \ldots, T_k$ sind, ebenfalls ein PC-Baum.*

Abbildung 1.40: Skizze: Darstellung von P- und C-Knoten

In der Abbildung 1.40 ist skizziert, wie wir P- bzw. C-Knoten graphisch darstellen wollen. P-Knoten werden durch Kreise, C-Knoten durch lange Rechtecke dargestellt. Für die Blätter führen wir keine besondere Konvention ein. In der Abbildung 1.41 ist das Beispiel eines PC-Baumes angegeben.

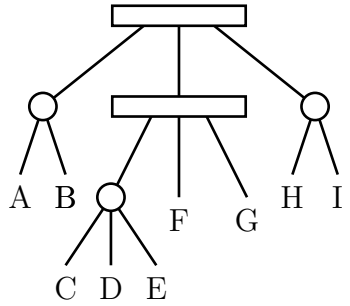


Abbildung 1.41: Beispiel: Ein PC-Baum

Im Folgenden benötigen wir spezielle PC-Bäume, die wir jetzt definieren wollen.

Definition 1.60 *Ein PC-Baum heißt* echt*, wenn folgende Bedingungen erfüllt sind:*

- *Jedes Element $a \in \Sigma$ kommt genau einmal als Blattmarkierung vor;*
- *Jeder P-Knoten hat mindestens zwei Kinder;*
- *Jeder C-Knoten hat mindestens drei Kinder.*

Der in Abbildung 1.41 angegebene PQ-Baum ist also ein echter PC-Baum. Auch für echte PC-Bäume gilt, dass die Anzahl der P- und C-Knoten kleiner als die Kardinalität des betrachteten Alphabets Σ ist.

Die P- und C-Knoten besitzen natürlich eine besondere Bedeutung, die wir jetzt erläutern wollen. Wir wollen PC-Bäume im Folgenden dazu verwenden, Permutation zu beschreiben. Daher wird die Anordnung der Kinder an P-Knoten willkürlich sein (d.h. alle Permutationen der Teilbäume sind erlaubt). An C-Knoten hingegen ist die Reihenfolge bis auf zyklische Rotationen und Umdrehen der Reihenfolge fest. Um dies genauer beschreiben zu können benötigen wir noch einige Definitionen.

Definition 1.61 *Sei T ein echter PC-Baum über Σ. Die* Frontier *von T, kurz $f(T)$ ist die Permutation über Σ, die durch das Ablesen der Blattmarkierungen von links nach rechts geschieht (also die Reihefolge der Blattmarkierungen in einer Tiefensuche unter Berücksichtigung der Ordnung auf den Kindern jedes Knotens).*

Die Frontier des Baumes aus Abbildung 1.4 ist dann ABCDEFGHI.

Definition 1.62 *Zwei echte PC-Bäume T und T' heißen* äquivalent*, kurz $T \cong T'$, wenn sie durch endliche Anwendung folgender Regeln ineinander überführt werden können:*

- *Beliebiges Umordnen der Kinder eines P-Knotens;*
- *zyklische Rotation der Reihenfolge der Kinder eines C-Knotens, eventuell gefolgt von Umkehrung der Reihenfolge.*

Damit kommen wir zur Definition konsistenter Frontiers eines PC-Baumes.

Definition 1.63 *Sei T ein echter PC-Baum, dann ist $cons(T)$ die Menge der konsistenten Frontiers von T, d.h.:*

$$cons(T) = \{f(T') \,:\, T \cong T'\}\,.$$

Beispielsweise befinden sich dann in der Menge cons(T) für den Baum aus der Abbildung 1.4: BADECFGHI, ABGDCEFIH oder FCDEGABHI.

Auch PC-Bäume lassen sich für eine Menge von Restriktionen in linearer Zeit konstruieren, sofern die Menge die Circular-Ones-Property besitzt. Für weitere Details verweisen wir auf die Originalliteratur von Hsu sowie Hsu und McConell.

1.5.2 Algorithmus von Hsu für die C1P (*)

Der Algorithmus von Hsu arbeitet direkt mit den gegebenen 0-1-Matrizen und versucht, für diese die Consecutive Ones Property festzustellen. Für eine effiziente Implementierung werden dünn besetzte Matrizen (also mit wenig 1en) durch verkettete Listen dargestellt. Dabei kann parallel zur Darstellung der erlaubten Permutationen ebenfalls wieder ein PQ-Baum mitkonstruiert werden.

Der Algorithmus versucht ebenfalls iterativ die verschiedenen Restriktionen (also Zeilen der Matrix) zu verarbeiten. Er verwendet dabei jeweils die kürzeste, d.h. diejenige mit den wenigsten Einsen.

Für eine kurze Beschreibung der Idee des Algorithmus von Hsu nehmen wir an, die Matrix wäre bereits so permutiert, dass sie die Consecutive Ones Property erfüllt. Sei v die kürzeste Zeile mit Einsen. Die Zeilen der Matrix lassen sich dann in (im Wesentlichen) vier Klassen (a) mit (e) einteilen:

(a) Die Zeilen, deren 1-Block mit dem 1-Block der Zeile v echt überlappt und nach links hinausragt.

(b) Die Zeilen, deren 1-Block mit dem 1-Block der Zeile v echt überlappt und nach recht hinausragt.

(c) Die Zeilen, deren 1-Block mit dem 1-Block der Zeile v überlappt und sowohl nach links als auch nach rechts hinausragt.

(d) Die Zeilen, deren 1-Block mit dem 1-Block der Zeile v identisch ist.

(e) Die Zeilen, deren 1-Block völlig außerhalb des 1-Blocks der Zeile v liegt.

Da wir jeweils die kürzeste Zeile wählen, kann es keine Zeile geben, deren 1-Block echt im 1-Block der Zeile v enthalten ist. In der folgenden Abbildung 1.42 ist diese Einteilung in diese (im Wesentlichen) vier Klassen noch einmal schematisch dargestellt. Die Klasse (d) ist hier keine echte Klasse, da sie bzgl. der Perutationen keine neuen Informationen liefert.

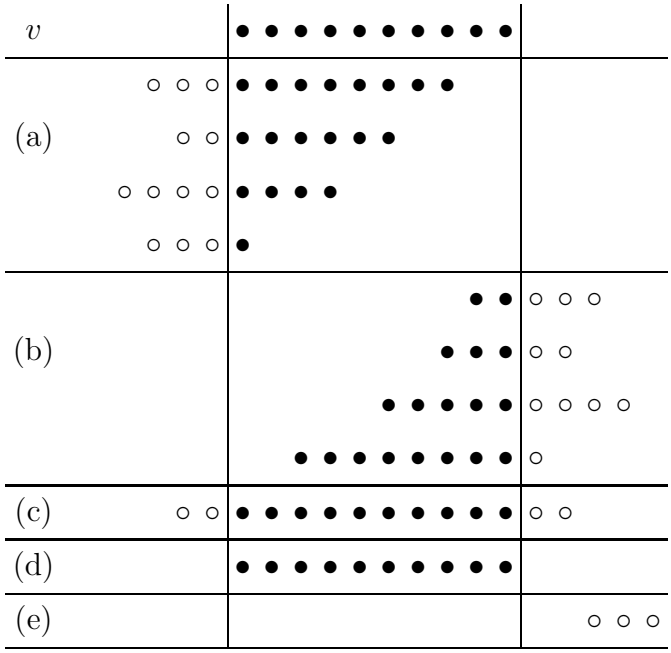


Abbildung 1.42: Skizze: Darstellung der verschiedenen Blöcke

Auch wenn die Matrix wild permutiert ist, können wir die Zeilen vom Typ (c), (d) und (e) leicht erkennen. Die restlichen Zeilen sind eine Mischung vom Typ (a) und (b) Da diese Zeilen außerhalb des 1-Blocks von Zeile v nach links oder rechts herausragen müssen und jeweils die erste Spalte links bzw. rechts mit einer 1 besetzt sein muss, suchen wir in der restlichen Menge von Zeilen nur noch die Spalte außerhalb des 1-Block von Zeile v mit den meisten 1-en. Diese ist dann entweder eine Spalte die unmittelbar links bzw. rechts vom 1-Block der Zeile v liegen muss. So können wir auch die Zeilen in die Klassen (a) und (b) einteilen.

Mit den Zeilen vom Typ (a) oder (b) können wir jetzt die Ordnung der Spalten im 1-Block von Zeile v festlegen. Da in diesem Block von Spalten die anderen Zeilen keinen Einfluss auf die Anordnung mehr haben, kontrahieren wir diese Spalten zu einer neuen Spalte und iterieren das Verfahren. Dabei kann parallel einen PQ-Baum bottom-up aufgebaut werden.

Man kann zeigen, dass diese Methode ebenfalls in linearer Zeit läuft und auch so modifiziert werden kann, dass er bei kleinen Fehlern, wie zu Beginn dieses Kapitels angegeben, eine sinnvolle Anordnung generieren kann. Für weitere Details verweisen wir auf die Originalliteratur.

1.6 Intervallgraphen

In diesem Abschnitt wollen wir eine andere Modellierung zur genomischen Kartierung vorstellen. Wie wir im nächsten Abschnitt sehen werden, hat diese Modellierung den Vorteil, dass wir Fehler und deren Korrekturen hier leichter mitmodellieren können.

1.6.1 Definition von Intervallgraphen

Zuerst geben wir die Definition eines Intervallgraphen an.

Definition 1.64 *Sei $\mathcal{I} = \{I_i : i \in [1:n]\}$ eine Multimenge reeller Intervalle, wobei $I_i = [\ell_i, r_i] \subset \mathbb{R}$ mit $\ell_i < r_i$ für alle $i \in [1:n]$. Der Graph $G(\mathcal{I}) = (V, E)$ ist gegeben durch*

- $V = [1:n] \cong \mathcal{I}$,
- $E = \{\{I_i, I_j\} : I_i, I_j \in \mathcal{I} \wedge i \neq j \wedge I_i \cap I_j \neq \emptyset\}$.

Die Menge $\mathcal{I}$ heißt Intervalldarstellung *von G. Ein Graph G heißt* Intervallgraph *(engl.* interval graph*), wenn er eine Intervalldarstellung besitzt.*

In Abbildung 1.43 ist ein Beispiel eines Intervallgraphen samt seiner zugehörigen Intervalldarstellung gegeben.

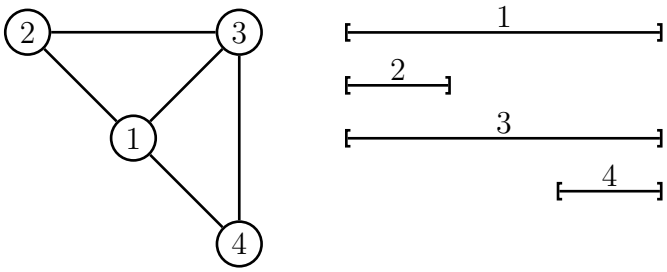


Abbildung 1.43: Beispiel: Ein Intervallgraph samt zugehöriger Intervalldarstellung

Zuerst bemerken wir, dass die Intervalle stets so gewählt werden können, dass die Intervallgrenzen paarweise verschieden sind, d.h. für einen Intervallgraphen G kann eine Intervalldarstellung $\mathcal{I} = \{[\ell_i, r_i] : [i \in 1 : n]\}$ gefunden werden, so dass $G \cong G(\mathcal{I})$ und $|\{\ell_i, r_i : i \in [1 : n]\}| = 2n$. Dazu müssen gleiche Intervallgrenzen nur um ein kleines Stück verschoben werden. Ferner merken wir hier noch an, dass die Intervallgrenzen der Intervalle einer Intervalldarstellung ohne Beschränkung der Allgemeinheit aus $\mathbb{N}$ gewählt werden können. Dazu müssen nur die Anfangs- und Endpunkte der Intervallgrenzen einer Intervalldarstellung nur aufsteigend durchnummeriert werden.

Wir definieren jetzt noch zwei spezielle Klassen von Intervallgraphen, die für die genomische Kartierung von Bedeutung sind.

Definition 1.65 *Ein Intervallgraph G heißt* echt *(engl.* proper interval graph*), wenn er eine Intervalldarstellung $\mathcal{I} = \{I_1, \ldots, I_k\}$ besitzt (d.h. $G \cong G(\mathcal{I})$), so dass*

$$\forall i \neq j \in [1 : k] : (I_i \not\subseteq Ij) \wedge (I_i \not\supseteq I_j).$$

Ein Intervallgraph G heißt Einheitsintervallgraph *(engl.* unit interval graph*), wenn er eine Intervalldarstellung $\mathcal{I}$ besitzt (d.h. $G \cong G(\mathcal{I})$), so dass $|I| = |I'|$ für alle $I, I' \in \mathcal{I}$.*

Zunächst halten wir fest, dass sich trotz unterschiedlicher Definition diese beiden Klassen gleich sind.

Lemma 1.66 *Ein Graph ist genau dann ein Einheitsintervallgraph, wenn er ein echter Intervallgraph ist.*

Den Beweis dieses Lemmas überlassen wir dem Leser als Übungsaufgabe.

1.6.2 Modellierung

Warum sind Intervallgraphen für die genomische Kartierung interessant. Schauen wir uns noch einmal unsere Aufgabe der genomischen Kartierung in Abbildung 1.44 an. Offensichtlich entsprechen die Fragmente gerade Intervallen, nämliche den Posi-

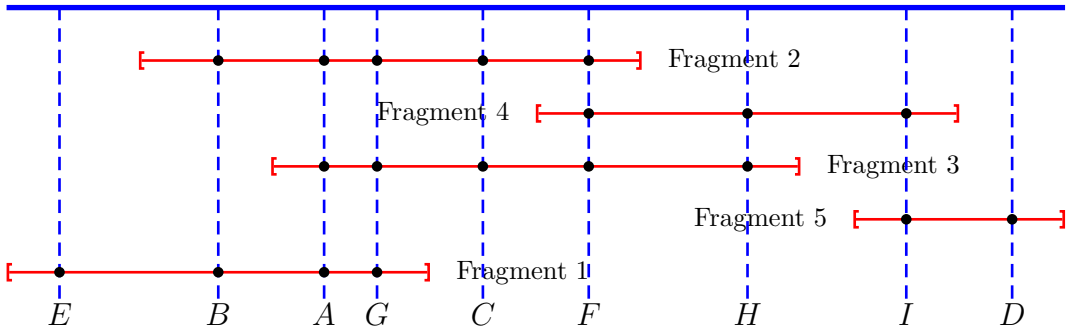


Abbildung 1.44: Skizze: Genomische Kartierung

tionen, die sie überdecken. Mit Hilfe unserer Hybridisierungsexperimente erhalten wir die Information, ob sich zwei Fragmente bzw. Intervalle überlappen, nämlich genau dann, wenn beide Fragmente dasselbe Landmark, also STS, enthalten. Somit bilden die Fragmente mit den Knoten und den Überschneidungen als Kanten einen Intervallgraphen. Was in der Aufgabe der genomischen Kartierung gesucht ist, ist die Anordnung der Fragmente auf dem Genom. Dies ist aber nichts anderes als eine Intervalldarstellung des Graphen, den wir über unsere biologischen Experimente erhalten. Aus diesem Grund sind oft auch Einheitsintervallgraphen von Interesse, da in den biologischen Experimenten die Fragmente im Wesentlichen dieselbe Länge besitzen und somit eine Intervalldarstellung durch gleich lange Intervalle erlauben sollte.

Wir formulieren nun einige Probleme für Intervallgraphen, die die Problemstellung bei der genomischen Kartierung widerspiegeln soll.

PROPER INTERVAL COMPLETION (PIC)

Eingabe: Ein Graph $G = (V, E)$ und $k \in \mathbb{N}$.
Gesucht: Ein (echter) Intervallgraph $G' = (V, E \cup F)$ mit $|F| \leq k$.

Mit PIC wird versucht, das Vorhandensein von False Negatives zu modellieren. Es wird angenommen, dass bei den Experimenten einige Überschneidungen von Fragmenten (maximal k) nicht erkannt wurden.

PROPER INTERVAL SELECTION (PIS)

Eingabe: Ein Graph $G = (V, E)$ und $k \in \mathbb{N}$.
Gesucht: Ein (echter) Intervallgraph $G' = (V, E \setminus F)$ mit $|F| \leq k$.

Mit PIS wird versucht, das Vorhandensein von False Positives zu modellieren. Es wird angenommen, dass bei den Experimenten einige Überschneidungen von Fragmenten (maximal k) zu Unrecht erkannt wurden.

INTERVAL SANDWICH (IS)

Eingabe: Ein Tripel (V, D, F') mit $D, F' \subset \binom{V}{2}$.
Gesucht: Ein Intervallgraph $G = (V, E)$ mit $D \subset E \subset F'$.

Mit IS soll in gewissen Sinne versucht werden, sowohl False Positives als auch False Negatives gleichzeitig zu modellieren. Hierbei repräsentiert die Menge D die Überschneidungen von Fragmenten, von denen man sich sicher ist, dass sich gelten. Diese werden eine Teilmenge der aus den experimentell gewonnen Überschneidungen sein. Mit der Menge F versucht ein Menge von Kanten anzugeben, die höchstens benutzt werden dürfen. Diese werden eine Obermenge der experimentellen Überschneidungen sein. Man kann dieses IS-Problem auch anders formulieren.

INTERVAL SANDWICH (IS)

Eingabe: Ein Tripel (V, M, F) mit $M, F \subset \binom{V}{2}$.
Gesucht: Ein Intervallgraph $G = (V, E)$ mit $M \subset E$ und $E \cap F = \emptyset$.

Hierbei bezeichnet M (wie vorher D) die Menge von Kanten, die in jedem Falle im Intervallgraphen auftreten sollen (engl. mandatory). Die Menge F bezeichnet jetzt die Menge von Kanten, die im zu konstruierenden Intervallgraphen sicherlich nicht auftreten dürfen (engl. forbidden), also das Komplement der vorherigen Menge F'. Wie man sich leicht überlegt, sind die beiden Formulierungen äquivalent.

Bevor wir unser letztes Problem für die Modellierung der genomischen Kartierung formalisieren, benötigen wir noch die Definition von Färbungen in Graphen.

Definition 1.67 *Sei $G = (V, E)$ ein Graph. Eine Abbildung $c : V \to [1 : k]$ heißt* k-Färbung. *Eine k-Färbung heißt* zulässig, *wenn $c(v) \neq c(w)$ für alle $\{v, w\} \in E$.*

Üblicherweise wird hier der Wertebereich $[1 : k]$ von c als Menge von Farben interpretiert, die die Knoten färben, d.h. jedes $i \in [1 : k]$ entspricht genau einer Farbe. Daher stammt der Name Färbung in der Definition.

Damit können wir das folgende Problem definieren.

Intervalizing Colored Graphs

Eingabe: Ein Graph $G = (V, E)$ und eine k-Färbung c für G.
Gesucht: Ein Intervallgraph $G' = (V, E')$ mit $E \subseteq E'$, so dass c eine zulässige k-Färbung für G' ist.

Die Motivation hinter dieser Formalisierung ist, dass man bei der Herstellung der Fragmente darauf achten kann, welche Fragmente aus einer Kopie des Genoms gleichzeitig generiert wurden. Damit weiß man, dass sich diese Fragmente sicherlich nicht überlappen können und gibt ihnen daher dieselbe Farbe.

Man beachte, dass ICG ist ein Spezialfall des Interval Sandwich Problems ist. Mit $F' = \{\{i, j\} : c(i) \neq c(j)\}$ bzw. $F = \{\{i, j\} : c(i) = c(j)\}$ können wir aus einer ICG-Instanz eine äquivalente IS-Instanz konstruieren.

28.05.24

1.6.3 Komplexitäten

In diesem Abschnitt wollen kurz auf die Komplexität der im letzten Abschnitt vorgestellten Probleme eingehen. Leider sind für diese Fehler tolerierenden Modellierungen die Entscheidungsproblem, ob es den gesuchten Graphen gibt oder nicht, in der Regel bereits $\mathcal{NP}$-hart.

PIC: Proper Interval Completion ist $\mathcal{NP}$-hart, wenn k Teil der Eingabe ist. Für feste k ist das Problem in polynomieller Zeit lösbar, aber die Laufzeit bleibt exponentiell in k.

ICG und IS: Intervalizing Colored Graphs ist ebenfalls $\mathcal{NP}$-hart. Somit ist auch das Interval Sandwich Problem, das ja ICG als Teilproblem enthält, ebenfalls $\mathcal{NP}$-hart. Selbst für ein festes $k \geq 4$ bleibt ICG $\mathcal{NP}$-hart. Für $k \leq 3$ lassen sich hingegen polynomielle Algorithmen für ICG finden. Der Leser sei dazu eingeladen, für die Fälle $k = 2$ und $k = 3$ polynomielle Algorithmen zu finden. Leider taucht in der Praxis doch eher der Fall $k \geq 4$ auf.

1.7 Interval Sandwich Problem

In diesem Abschnitt wollen wir das Interval Sandwich Problem vom algorithmischen Standpunkt aus genauer unter die Lupe nehmen. Wir wollen zeigen, wie man dieses Problem prinzipiell, leider mit einer exponentiellen Laufzeit löst, und wie man daraus für einen Speziallfall einen polynomiellen Algorithmus ableiten kann.

1.7.1 Allgemeines Lösungsprinzip

Wir wollen an dieser Stelle noch anmerken, dass wir im Folgenden ohne Beschränkung der Allgemeinheit annehmen, dass der Eingabe-Graph (V, M) zusammenhängend ist. Andernfalls bestimmen wir eine Intervalldarstellung für jede seine Zusammenhangskomponenten und hängen dieses willkürlich aneinander. Für praktische Eingaben in Bezug auf die genomische Kartierung können wir davon ausgehen, dass die Eingabe zusammenhängend ist, da andernfalls die Fragmente so dünn gesät waren, dass eine echte Kartierung sowieso nicht möglich ist.

Zuerst führen wir noch eine Notation für Intervalle ein.

Notation 1.68 *Es bezeichne* $\mathcal{J}(\mathbb{R}) = \{[a, b] : a < b \in \mathbb{R}\}$ *sowie* $\ell([a, b]) = a$ *und* $r([a, b]) = b$.

Zunächst definieren wir einige für unsere algorithmische Idee grundlegende, dennoch sehr einfache Begriffe.

Definition 1.69 *Sei* $S = (V, M, F)$ *eine Eingabe für IS. Eine Teilmenge* $X \subseteq V$ *heißt* Kern. *Der* Rand $\beta(X) \subseteq M$ *eines Kerns* X *ist definiert als*

$$\beta(X) = \{e \in M \ : \ |e \cap X| = 1\}.$$

Die aktive Region $\mathcal{A}(X) \subseteq V$ *eines Kerns* X *ist definiert als*

$$\mathcal{A}(X) = \{v \in X \ : \ \exists e \in \beta(X) : v \in e\}.$$

Diese Begriffe sind in Abbildung 1.45 noch einmal skizziert.

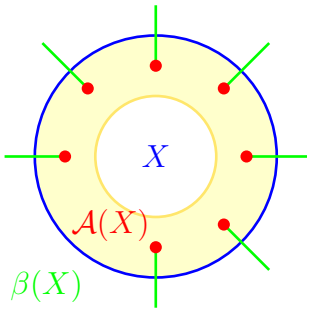


Abbildung 1.45: Skizze: Aktive Region $\mathcal{A}(X)$ und Rand $\beta(X)$ des Kerns X

Der Hintergrund für diese Definition ist der folgende. Der aktuell betrachtete Kern in unserem Algorithmus wird eine Knotenteilmenge sein, für die wir eine Intervalldarstellung bereits konstruiert haben. Die aktive Region beschreibt dann die Menge von

Knoten des Kerns, für die noch benachbarte Knoten außerhalb des Kerns existieren, die dann über die Kanten aus dem Rand verbunden sind.

Kommen wir nun dazu genauer zu formalisieren, was eine Intervalldarstellung eines Kerns ist.

Definition 1.70 *Sei V eine Knotenmenge und $M, F \subseteq \binom{V}{2}$. Ein* Layout *$L(X)$ eines Kerns $X \subseteq V$ ist eine Funktion $I : X \to \mathcal{J}(\mathbb{R})$ mit*

1. $\forall \{v, w\} \in M \cap \binom{X}{2} : I(v) \cap I(w) \neq \emptyset$,
2. $\forall \{v, w\} \in F \cap \binom{X}{2} : I(v) \cap I(w) = \emptyset$,
3. $\forall v \in \mathcal{A}(X) : r(I(v)) = \max\{r(I(w)) : w \in X\}$.

Ein Kern heißt zulässig, *wenn er ein Layout besitzt.*

Damit ist ein zulässiger Kern also der Teil der Knoten, für den bereits ein Layout bzw. eine Intervalldarstellung konstruiert wurde. Mit der nächsten Definition geben wir im Prinzip die algorithmische Idee an, wie wir zulässige Kerne erweitern wollen. Wir werden später sehen, dass diese Idee ausreichend sein wird, um für eine Eingabe des Interval Sandwich Problems eine Intervalldarstellung zu konstruieren.

Definition 1.71 *Ein zulässiger Kern $Y = X \cup \{v\}$* erweitert *genau dann einen zulässigen Kern X, wenn $L(Y)$ aus $L(X)$ durch Hinzufügen eines Intervalls $I(v)$ (sowie Vergrößerung der rechten Intervallgrenzen von Intervallen, die zu Knoten der aktiven Region gehören) entsteht, so dass*

1. $\forall w \in X \setminus \mathcal{A}(X) : r(I(w)) < \ell(I(v))$;
2. $\forall w \in \mathcal{A}(X) : r(I(w)) = r(I(v))$.

Ein Beispiel einer solchen Erweiterung des zulässigen Kerns $X = \{1, 2, 3, 4\}$ zu einem zulässigen Kern $Y = X \cup \{5\}$ ist in Abbildung 1.46 dargestellt. Nur wenn $3 \in \mathcal{A}(X)$ gilt, wird das in der Abbildung angegebene zugehörige graue Intervall verlängert.

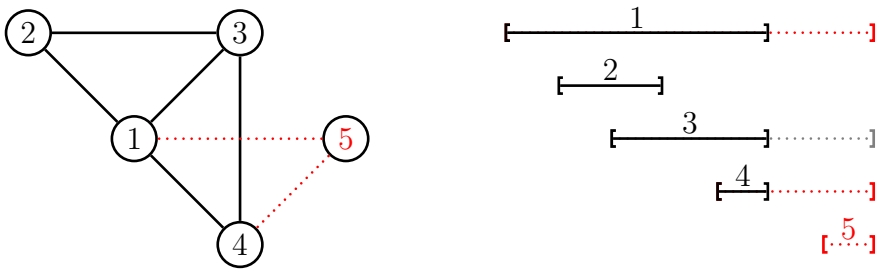


Abbildung 1.46: Beispiel: Erweiterung eines Layouts

Wir kommen im folgenden Lemma zu einer einfachen Charakterisierung, wann ein zulässiger Kern eine Erweiterung eines anderen zulässigen Kerns ist. Hierbei ist insbesondere wichtig, dass diese Charakterisierung völlig unabhängig von den zugrunde liegenden Layouts ist, die den Kernen ihre Zulässigkeit bescheinigen.

Lemma 1.72 *Sei X ein zulässiger Kern. $Y = X \cup \{v\}$ ist genau dann ein zulässiger Kern und erweitert X, wenn $\{v, w\} \notin F$ für alle $w \in \mathcal{A}(X)$.*

Beweis: $\Rightarrow$**:** Da Y eine Erweiterung von X ist, überschneidet sich das Intervall von v mit jedem Intervall aus $\mathcal{A}(X)$. Da außerdem $Y = X \cup \{v\}$ ein zulässiger Kern ist, gilt $\{v, w\} \notin F$ für alle $w \in \mathcal{A}(X)$.

$\Leftarrow$**:** Sei also X ein zulässiger Kern. Wir betrachten das Layout von X in Abbildung 1.47 Da $\{v, w\} \notin F$ für alle $w \in \mathcal{A}(X)$, können wir nun alle Intervalle der

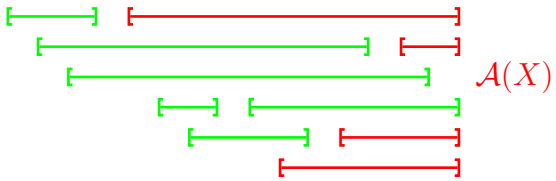


Abbildung 1.47: Skizze: Layout

Knoten aus $\mathcal{A}(X)$ verlängern und ein neues Intervall für v einfügen, das nur mit den Intervallen aus $\mathcal{A}(X)$ überlappt. ■

Wir zeigen jetzt noch, dass es zu jedem zulässigen Kern einen kleineren zulässigen Kern gibt, der sich zu diesem erweitern lässt.

Lemma 1.73 *Jeder zulässige Kern $Y \neq \emptyset$ erweitert mindestens einen zulässigen Kern $X \subsetneq Y$.*

Beweis: Sei $L(Y)$ mit $I : V \to \mathcal{J}(\mathbb{R})$ ein Layout für Y. Wir wählen jetzt $y \in Y$, so dass $\ell(I(y))$ maximal ist. Siehe dazu auch Abbildung 1.48. Beachte, dass y nicht notwendigerweise aus $\mathcal{A}(Y)$ sein muss.

Wir definieren jetzt ein Layout $L(X)$ für $X = Y \setminus \{y\}$ aus $L(Y)$ wie folgt um:

$$\forall \{x, y\} \in M \cap \binom{Y}{2} : \quad r(I(x)) := R + 2,$$

wobei $R := \max\{r(I(z)) : z \in \mathcal{A}(Y)\}$. Alle anderen Werte von I auf X bleiben unverändert und y wird aus dem Definitionsbereich von I entfernt (und für die Erweiterung von X auf Y ist dann $I(y) = [R{+}1, R{+}2]$).

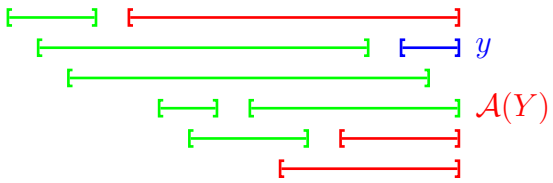


Abbildung 1.48: Skizze: Layout $L(Y)$ für Y

Wir müssen jetzt lediglich die drei Bedingungen aus der Definition eines zulässigen Layouts nachweisen. Offensichtlich gilt weiterhin $I(v) \cap I(w) \neq \emptyset$ für alle $\{v, w\} \in M \cap \binom{X}{2} \subseteq M \cap \binom{Y}{2}$, da dies bereits im Layout $L(Y)$ gegolten hat (und wir Intervalle nur verlängert haben). Außerdem gilt ebenfalls $I(v) \cap I(w) = \emptyset$ für alle $\{v, w\} \in F \cap \binom{X}{2} \subseteq F \cap \binom{Y}{2}$, da dies ebenfalls bereits im Layout $L(Y)$ gegolten hat und nur solche Intervalle verlängert werden, die $\ell(I(y))$ enthalten (und nach Wahl von y kein Intervall nach $\ell(I(y))$ beginnt). Letztendlich gilt nach unserer Konstruktion, dass $r(I(v)) = \max\{r(I(w)) : w \in X\}$ für alle $v \in \mathcal{A}(X)$. Da man sich leicht überlegt, dass

$$\mathcal{A}(X) = \{x \in X : \{x, y\} \in M\} \cup (\mathcal{A}(Y) \setminus \{y\})$$

gilt. Somit überlappten alle verlängerten Intervalle bereits mit y und es kann keine Kante aus F im Layout erzeugt werden, da nach Wahl von y kein Intervall weiter rechts als der Beginn von y beginnen kann.

Des Weiteren überlegt man sich leicht, dass mit dem nun neu festgelegten Intervall $I(y) = [R{+}1, R{+}2]$ zu y auch die Bedingungen 1. und 2. aus der Definition 1.71 für eine Erweiterung von X zu $Y = X \cup \{y\}$ erfüllt sind. Damit ist der Beweis abgeschlossen. ■

Als unmittelbare Folgerung erhält man das folgende Lemma, das die Basis für unseren Algorithmus sein wird.

Korollar 1.74 *Für eine Eingabe $S = (V, M, F)$ des Interval Sandwich Problems existiert genau dann eine Lösung, wenn V ein zulässiger Kern ist.*

Mit Hilfe dieses Lemmas wissen wir nun, dass es eine aufsteigende Folge

$$\emptyset = X_0 \subset X_1 \subset \cdots \subset X_{n-1} \subset X_n = V$$

mit $|X_{i+1} \setminus X_i| = 1$ gibt. Das bedeutet, dass wir ein Layout iterativ für unsere Eingabe konstruieren können. Nach dem Lemma 1.72 wissen wir auch, dass die Erweiterungen unabhängig vom betrachteten Layout möglich sind. Insbesondere folgt daraus, dass

wenn ein Layout eines zulässigen Kerns nicht erweitert werden kann, es auch kein anderes Layout dieses Kerns geben kann, das sich erweitern lässt.

Somit erhalten wir den in Abbildung 1.49 angegeben Algorithmus zur Konstruktion einer Intervalldarstellung für eine gegebene Eingabe S des Interval Sandwich Problems. Hierbei testen wir alle möglichen Erweiterungen der leeren Menge zu einem

```
Sandwich (S = (V, M, F))
begin
    Queue Q;
    Q.enqueue(∅);
    while (not Q.is_empty()) do
        X := Q.dequeue();
        foreach (v ∈ V \ X) do
            if (Y := X ∪ {v} is feasible and extends X) then
                if (Y = V) then output solution found;
                else if (not Q.contains(Y)) then Q.enqueue(Y);
    output no solutions found;
end
```

Abbildung 1.49: Algorithmus: Allgemeines Interval Sandwich Problem

zulässigen Kern V. Dabei werden im schlimmsten Falle alle möglichen Teilmengen von V aufgezählt. Da wir die zulässigen Kerne in einer Queue speichern und in diese jede Teilmenge nur einmal aufnehmen, sind die zulässigen Kerne in der Queue nach ihrer Größe sortiert. Somit kann jede Teilmenge von V höchstens einmal in die Queue aufgenommen und betrachtet werden.

Da es leider exponentiell viele Teilmengen gibt, nämlich genau 2^n, wenn $n = |V|$ ist, ist dieser Algorithmus sicherlich nicht praktikabel. Da der Test, ob sich ein Kern erweitern lässt, nach Lemma 1.72 in Zeit $O(|V|^2)$ implementiert werden kann, erhalten wir das folgende Theorem.

Theorem 1.75 *Für eine Eingabe $S = (V, M, F)$ des Interval Sandwich Problems lässt sich in Zeit $O(|V|^3 \cdot 2^{|V|})$ feststellen, ob es eine Lösung gibt, und falls ja, kann diese auch konstruiert werden.*

In unserem Algorithmus stellen wir zunächst nur fest, ob V zulässig ist. Wenn wir uns aber beim Aufbau merken, in welcher Reihenfolge die Knoten zu V hinzugefügt wurden, können wir das Layout von V ganz einfach durch den genau spezifizierten Prozess der Erweiterung rekonstruieren.

1.7.2 Lösungsansatz für Bounded Degree Interval Sandwich

Wir wollen nun zwei Varianten des Interval Sandwich Problems vorstellen, die sich in polynomieller Zeit lösen lassen, wenn wir gewisse (in unserem Falle sinnvolle) Restriktionen an die Eingabe bzw. an die Lösung des Problems stellen. Dazu wiederholen wir erst noch einmal kurz die Definition einer Clique.

Definition 1.76 *Sei $G = (V, E)$ ein Graph. Ein Teilgraph $G' = (V', E')$ von G heißt* Clique *oder k-*Clique*, wenn folgendes gilt:*

- $|V'| = k$,
- $E' = \binom{V'}{2}$ *(d.h. G' ist ein vollständiger Graph),*
- *Für jedes $v \in V \setminus V'$ ist $G'' = (V'', \binom{V''}{2})$ mit $V'' = V' \cup \{v\}$ kein Teilgraph von G (d.h. G' ist ein maximaler vollständiger Teilgraph von G).*

Die Cliquenzahl *$\omega(G)$ des Graphen G ist die Größe einer größten Clique von G.*

Mit Hilfe dieser Definition können wir folgende Spezialfälle des Interval Sandwich Problems definieren.

Bounded Degree and Width Interval Sandwich

Eingabe: Ein Tripel (V, M, F) mit $M, F \subset \binom{V}{2}$ sowie $d, k \in \mathbb{N}$ mit $\Delta((V, M)) \leq d$.
Gesucht: Ein Intervallgraph $G = (V, E)$ mit $M \subset E$, $E \cap F = \emptyset$ und $\omega(G) \leq k$.

Wir beschränken also hier die Eingabe auf Graphen mit beschränkten Grad und suchen nach Intervallgraphen mit einer beschränkten Cliquenzahl.

Bounded Degree Interval Sandwich (BDIS)

Eingabe: Ein Tripel (V, M, F) mit $M, F \subset \binom{V}{2}$ und $d \in \mathbb{N}$.
Gesucht: Ein Intervallgraph $G = (V, E)$ mit $M \subset E$, $E \cap F = \emptyset$ und $\Delta(G) \leq d$.

Beim Bounded Degree Interval Sandwich Problem beschränken wir den Suchraum nur dadurch, dass wir für die Lösungen nur gradbeschränkte Intervallgraphen zulassen. Wir werden jetzt für dieses Problem einen polynomiellen Algorithmus vorstellen. Für das erstgenannte Problem lässt sich mit ähnlichen Methoden ebenfalls ein polynomieller Algorithmus finden.

Die beiden hier erwähnten Probleme sind auch für die genomische Kartierung relevant, da wir bei den biologischen Experimenten davon ausgehen, dass die Überdeckung einer Position im Genom sehr gering ist und aufgrund der kurzen, in etwa gleichlangen Fragmente der resultierende Intervallgraph sowohl einen relativ kleinen Grad als auch eine relativ kleine Cliquenzahl besitzt.

04.06.24

Um unseren Algorithmus geeignet modifizieren zu können, müssen wir zunächst die grundlegende Definition der Layouts anpassen.

Definition 1.77 *Sei V eine Menge und $M, F \subseteq \binom{V}{2}$. Ein* d-Layout *(oder kurz* Layout*) $L(X)$ eines Kerns $X \subseteq V$ ist eine Funktion $I : X \to \mathcal{J}(\mathbb{R})$ mit*

1. $\forall \{v, w\} \in M \cap \binom{X}{2} : I(v) \cap I(w) \neq \emptyset$,
2. $\forall \{v, w\} \in F \cap \binom{X}{2} : I(v) \cap I(w) = \emptyset$,
3. $\forall v \in \mathcal{A}(X) : r(I(v)) = \max\{r(I(w)) : w \in X\}$,
4. $\forall v \in X \setminus \mathcal{A}(X) : I(v)$ *schneidet höchstens d andere Intervalle,*
5. $\forall v \in \mathcal{A}(X) : I(v)$ *schneidet höchstens $d - |E(v, X)|$ andere Intervalle, wobei $E(v, X) = \{\{v, w\} \in M \mid w \notin X\}$.*

Ein Kern X heißt d-zulässig *(oder kurz* zulässig*), wenn er ein d-Layout besitzt und $|\mathcal{A}(X)| \leq d - 1$.*

Im Folgenden werden wir meist die Begriffe Layout bzw. zulässig anstatt von d-Layout bzw. d-zulässig verwenden. Aus dem Kontext sollte klar sein, welches d wirklich gemeint ist.

Im Wesentlichen sind die Bedingungen 4 und 5 in der Definition neu hinzugekommen. Die Bedingung 4 ist klar, da wir ja nur Intervallgraphen mit maximalen Grad kleiner gleich d konstruieren wollen. Daher darf sich jeder fertig konstruierte Knoten mit maximal d anderen Intervalle schneiden.

Analog gibt bei Bedingung 5 die Kardinalität $|E(v, X)|$ gerade die Anzahl der Nachbarn an, die noch nicht in der aktuellen Intervalldarstellung bzw. im Layout realisiert sind. Daher darf ein Intervall der aktiven Region vorher maximal $d - |E(v, X)|$ andere Intervalle schneiden.

Bedingungen 4 und 5 kann man auch als äquivalent zu folgender Aussage sehen: Für alle $v \in X$ gilt, $I(v)$ schneidet höchstens $d - |E(v, X)|$ andere Intervalle. Im Falle $v \in \mathcal{A}(X)$ ist diese gerade 5, im Falle $v \in X \setminus \mathcal{A}$ ist dies gerade 4, da dann $E(v, X) = \emptyset$ gilt.

Es bleibt noch zu überlegen, warum man die Einschränkung gemacht hat, dass $|\mathcal{A}(X)| \leq d-1$ ist. Wäre $|\mathcal{A}(X)| \geq d+1$, dann würde jedes Intervall der aktiven Region bereits d andere Intervalle schneiden. Da die Knoten jedoch noch aktiv sind, gibt es noch nicht realisierte Nachbarn und der resultierenden Graph würde einen Grad von größer als d bekommen.

Warum verbieten wir auch noch $|\mathcal{A}(X)| = d$? Angenommen, wir hätten eine aktive Region mit d Knoten. Dann hätte jeder Knoten der aktiven Region bereits einen Grad von $d-1$, da sich alle Intervall der aktiven Region überschneiden. Die aktive Region bildet also eine d-Clique. Wenn nun ein Knoten hinzukommt, wird er zu allen Knoten der aktiven Region benachbart. Somit konstruieren wir eine $(d+1)$-Clique, in der jeder Knoten Grad d besitzt. Würde die aktive Region also einmal aus d Knoten bestehen, so müsste eine erfolgreiche Ausgabe des Algorithmus eine $(d+1)$-Clique sein. Da wir voraussetzen, dass der Eingabegraph (V, M) zusammenhängend ist und somit auch der zu konstruierende Ausgabegraph zusammenhängend sein muss, kann dies nur der Fall sein, wenn $|V| = d+1$ ist. Andernfalls gäbe es einen Knoten mit Grad größer als d. Wir können also vorher prüfen, ob der vollständige Graph auf V eine zulässige Ausgabe ist und hinterher diesen Fall ausschließen.

Wir definieren nun wieder die Erweiterung eines Layouts analog wie in Definition 1.71, nur jetzt für d-Layouts.

Definition 1.78 *Ein d-zulässiger Kern $Y = X \cup \{v\}$* erweitert *genau dann einen d-zulässigen Kern X, wenn $L(Y)$ aus $L(X)$ durch Hinzufügen eines Intervalls $I(v)$ (sowie Vergrößerung der rechten Intervallgrenzen von Intervallen, die zu Knoten der aktiven Region gehören) entsteht, so dass*

1. $\forall w \in X \setminus \mathcal{A}(X) : r(I(w)) < \ell(I(v))$;
2. $\forall w \in \mathcal{A}(X) : r(I(w)) = r(I(v))$.

Damit können wir die zu Lemma 1.72 analoge Aussage aufstellen.

Lemma 1.79 *Sei X ein d-zulässiger Kern. $Y = X \cup \{v\}$ ist genau dann ein d-zulässiger Kern und erweitert X, wenn $\{v, w\} \notin F$ für alle $w \in \mathcal{A}(X)$, X ein d-Layout L besitzt, so dass $I(u)$ für alle $u \in \mathcal{A}(X)$ mit $\{u, v\} \notin M$ höchstens $d - |E(u, X)| - 1$ andere Intervalle schneidet, $|\mathcal{A}(X)| \leq d - |E(v, Y)|$ und $|\mathcal{A}(Y)| \leq d - 1$.*

Beweis: $\Rightarrow$**:** Nach Lemma 1.72 wissen wir, dass $\{v, w\} \notin F$ für alle $w \in \mathcal{A}(X)$.

Sei $Y = X \cup \{v\}$ ein zulässiger Kern, der X erweitert. Sei $L(Y)$ ein Layout von Y, das durch eine Erweiterung aus einem Layout $L(X)$ für X entsteht. Sei weiter $u \in \mathcal{A}(X)$

mit $\{u, v\} \notin M$. Wir unterscheiden jetzt zwei Fälle, je nachdem, ob u auch in der aktiven Region von Y ist oder nicht.

Fall 1 ($u \notin \mathcal{A}(Y)$): Da u sich nicht mehr in der aktiven Region von Y befindet, obwohl es in der aktiven Region von X war, muss v der letzte verbliebene Nachbar von u außerhalb von X gewesen sein. Damit ist $\{u, v\} \in M$ und dieser Fall kann nach der Wahl von u gar nicht auftreten.

Fall 2 ($u \in \mathcal{A}(Y)$): Da $L(Y)$ ein Layout von Y ist und sich u in der aktiven Region von Y befindet, schneidet $I(u)$ maximal $d - |E(u, Y)|$ andere Intervalle in $L(Y)$ (Bedingung 5 des d-Layouts $L(Y)$). Durch Hinzunahme von v zu X bleibt die Menge der Nachbarn von u außerhalb von X bzw. Y unverändert, d.h. $E(u, X) = E(u, Y)$ (siehe auch Abbildung 1.50).

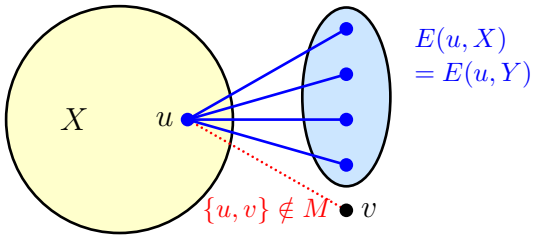


Abbildung 1.50: Skizze: Erweiterung von X um v

Nach Definition der Erweiterung müssen sich jedoch die Intervalle von u und v schneiden, obwohl $\{u, v\} \notin M$. Damit schneidet das Intervall $I(u)$ im Layout von Y maximal $d - |E(u, Y)| = d - |E(u, X)|$ andere Intervalle. Da im Layout von $L(X)$ nun das Intervall von v nicht mehr enthalten ist, das sich mit dem Intervall von u schneidet, gilt dann im Layout $L(X)$ von X, dass $I(u)$ maximal $d - |E(u, X)| - 1$ andere Intervalle schneidet.

Es bleibt noch zu zeigen, dass $|\mathcal{A}(X)| \leq d - |E(v, Y)|$ gilt. Da Y ein zulässiger Kern ist, schneidet das Intervall von v maximal $d - |E(v, Y)|$ andere Intervalle im Layout $L(Y)$ von Y. Die zu diesen Intervallen zugehörigen Knoten bilden gerade die aktive Region von X. Somit gilt $|\mathcal{A}(X)| \leq d - |E(v, Y)|$.

Nach Definition eines d-zulässigen Layouts für $L(Y)$ gilt $|\mathcal{A}(Y)| < d - 1$.

$\Leftarrow$: Nach Lemma 1.72 folgt aus $\{v, w\} \notin F$ für alle $w \in \mathcal{A}(X)$ bereits, dass Y eine Erweiterung von X und die Bedingungen 1 mit 3 für das Layout $L(Y)$ für Y gelten. Wir müssen also nur noch die Bedingungen 4 und 5 überprüfen.

Zum Nachweis der Bedingung 4 halten wir zunächst fest, dass Knoten aus X, die in X nicht mehr aktiv sind, sicherlich auch in Y nicht aktiv sind, d.h. es gilt $X \setminus \mathcal{A}(X) \subseteq Y \setminus \mathcal{A}(Y)$. Für alle Knoten aus $X \setminus \mathcal{A}(X)$ gilt also Bedingung 4.

Sei jetzt also $y \in Y \setminus \mathcal{A}(Y) \setminus (X \setminus \mathcal{A}(X))$ mit $y \neq v$. Daher wird y jetzt inaktiv und es muss daher $\{v, y\} \in M$ bzw. sogar $E(y, X) = \{\{v, y\}\}$ gelten. Aufgrund der Bedingung 5 für das Layout $L(X)$ von X schneidet das Intervall von y maximal $d - |E(v, X)| = d - 1$ andere Intervalle. Im Layout $L(Y)$ kann es also maximal $d - 1 + 1 = d$ andere Intervalle geben, die $I(y)$ schneiden, und die Bedingung 4 gilt.

Es bleibt noch der Fall, dass auch der neue Knoten $y = v$ in Y nicht mehr zur aktiven Region gehört. Dann schneidet v maximal $d - 1$ andere Intervalle, da nach Voraussetzung des Lemmas $|\mathcal{A}(Y)| \leq d - 1$ gilt

Zum Nachweis der Bedingung 5 für das Layout $L(Y)$ für Y machen wir wieder eine Fallunterscheidung und betrachten hierbei $y \in \mathcal{A}(Y) \subseteq (\mathcal{A}(X) \cup \{v\})$.

Fall 1 ($y \in \mathcal{A}(X) \wedge \{y, v\} \in M$): Dann schneidet das Intervall $I(y)$ im Layout $L(X)$ von X nach Bedingung 5 maximal $d - |E(y, X)|$ andere Intervalle. Da $E(y, X) = E(y, Y) \cup \{\{y, v\}\}$ und somit $|E(y, X)| = |E(y, Y)| + 1$ ist, kann $I(y)$ im erweiterten Layout $L(Y)$ maximal

$$d - |E(y, X)| + 1 = d - (|E(y, Y)| + 1) + 1 = d - |E(y, Y)|$$

andere Intervalle schneiden.

Fall 2 ($y \in \mathcal{A}(X) \wedge \{y, v\} \notin M$): Es gilt dann $E(y, Y) = E(y, X)$. Dann schneidet $I(y)$ im Layout $L(X)$ von X nach Voraussetzung maximal $d - |E(y, X)| - 1$ andere Intervalle. Somit kann $I(y)$ im erweiterten Layout $L(Y)$ maximal

$$(d - |E(y, X)| - 1) + 1 = d - |E(y, X)| = d - |E(y, Y)|$$

andere Intervalle schneiden.

Fall 3 ($y = v$): Da nach Voraussetzung $\mathcal{A}(X) \leq d - |E(v, Y)|$ gilt, kann $y = v$ im Layout $L(Y)$ maximal $d - |E(y, Y)|$ andere Intervalle schneiden.

Damit $L(X)$ ein zulässiges d-Layout $L(X)$ für X ist, muss noch $|\mathcal{A}(X)| \leq d - 1$ gelten, dies ist aber gerade eine der Voraussetzungen des Lemmas. ■

Somit haben wir auch wieder eine Charakterisierung gefunden, die eine Erweiterung von X zu Y beschreibt, ohne auf die konkreten Layouts einzugehen. Im Gegensatz zum allgemeinen Fall müssen wir hier jedoch die Grade der Knoten in der aktiven Region bzgl. des bereits konstruierten Intervallgraphen kennen.

Lemma 1.80 *Jeder d-zulässige Kern $Y \neq \emptyset$ erweitert mindestens einen d-zulässigen Kern $X \subsetneq Y$ oder beschreibt eine $(d+1)$-Clique.*

Beweis: Sei also Y ein d-zulässiger Kern und sei $L(Y)$ ein zugehöriges d-Layout. Sei $y \in Y$ so gewählt, dass $\ell(I(y))$ maximal ist. Weiter sei $L(X)$ das Layout für $X = Y \setminus \{y\}$, das durch Entfernen von $I(y)$ aus $L(Y)$ wie im Beweis von Lemma 1.73 entsteht. Aus dem Beweis von Lemma 1.73 folgt, dass die Bedingungen 1 mit 3 für das Layout $L(X)$ erfüllt sind. Wir müssen jetzt nur noch zeigen, dass auch die Bedingungen 4 und 5 gelten.

Zuerst zur Bedingung 4. Für alle $v \in X \setminus \mathcal{A}(X)$ gilt offensichtlich, dass $I(v)$ maximal d andere Intervalle schneidet. Ansonsten wäre $L(Y)$ schon kein Layout für Y, da dieses dann ebenfalls die Bedingung 4 verletzten würde.

Kommen wir jetzt zum Beweis der Gültigkeit von Bedingung 5. Zuerst stellen wir fest, dass $\mathcal{A}(X) \supseteq \mathcal{A}(Y) \setminus \{y\}$ gilt.

Fall 1 ($v \notin \mathcal{A}(Y)$): Damit gilt, dass $\{v, y\} \in M$ bzw. sogar $E(v, X) = \{\{v, y\}\}$ sein muss. In $L(Y)$ schneidet $I(v)$ dann wegen Bedingung 4 maximal d andere Intervalle. In $L(X)$ schneidet $I(v)$ dann maximal

$$d - 1 = d - |E(v, X)|$$

andere Intervalle, da sich $I(v)$ und $I(y)$ in $L(Y)$ schneiden und da, wie oben bereits angemerkt, $E(v, X) = \{\{v, y\}\}$.

Fall 2 ($v \in \mathcal{A}(Y)$): Nach Bedingung 5 des d-Layouts für $L(Y)$ schneidet $I(v)$ maximal $d - |E(v, Y)|$ andere Intervalle im Layout $L(Y)$. Es gilt weiterhin entweder $E(v, X) = E(v, Y) \cup \{\{v, y\}\}$ oder $E(v, X) = E(v, Y)$ (je nachdem, ob $\{v, y\}$ zu M gehört oder nicht). Somit gilt in jedem Fall $|E(v, Y)| \geq |E(v, X)| - 1$. Also schneidet $I(v)$ maximal

$$(d - |E(v, Y)|) - 1 \leq (d - (|E(v, X)| - 1)) - 1 \leq d - |E(v, X)|$$

andere Intervalle in $L(X)$.

Zum Schluss müssen wir noch zeigen, dass $|\mathcal{A}(X)| \leq d - 1$ gilt. Auch hier nehmen wir wieder eine Fallunterscheidung vor.

Fall 1 ($y \in \mathcal{A}(Y)$): Dann schneidet $I(y)$ in $L(Y)$ wegen Bedingung 5 maximal $d - 1$ andere Intervalle, da $|E(y, Y)| \geq 1$ ist. Nach Wahl von y, können nur Intervalle, die einen nichtleeren Schnitt mit $I(y)$ haben, zu $\mathcal{A}(X)$ gehören. Damit gilt $|\mathcal{A}(X)| \leq d - 1$.

Fall 2 ($y \notin \mathcal{A}(Y)$): Damit schneidet das Intervall $I(y)$ in $L(Y)$ wegen Bedingung 4 maximal d andere Intervalle. Sei zuerst $\mathcal{A}(Y) = \emptyset$ Nach Wahl von y, können nur Intervalle, die einen nichtleeren Schnitt mit $I(y)$ haben, zu $\mathcal{A}(X)$ gehören. Damit gilt $|\mathcal{A}(X)| \leq d$. Entweder ist $|\mathcal{A}(X)| < d$ und wir sind fertig oder $|\mathcal{A}(X)| = d$

und y schneidet d andere Intervalle. Im letzten Fall beschreibt Y dann genau eine $(d+1)$-Clique.

Es bleibt der Fall, dass $\mathcal{A}(Y) \neq \emptyset$. Dann gilt für jedes $x \in \mathcal{A}(Y)$, dass das Intervall $I(x)$ wegen Bedingung 5 maximal $d-1$ andere Intervalle in $L(Y)$ (einschließlich $I(y)$) schneidet, da $|E(x,Y)| \geq 1$ ist. Also schneidet auch $I(y)$ maximal $d-1$ andere Intervalle in $L(Y)$. Nach Wahl von y, können nur Intervalle, die einen nichtleeren Schnitt mit $I(y)$ haben, zu $\mathcal{A}(X)$ gehören (also maximal $d-1$ Intervalle). Damit gilt $|\mathcal{A}(X)| \leq d-1$. ■

Korollar 1.81 *Für die Eingabe $S = (V, M, F)$ des Bounded Degree Interval Sandwich Problems existiert genau dann eine Lösung, wenn V ein d-zulässiger Kern ist oder wenn $|V| = d+1$ und $F = \emptyset$.*

Wir merken hier noch an, dass man X durchaus zu Y erweitern kann, obwohl ein konkretes Layout für X sich nicht zu einem Layout für Y erweitern lässt. Dennoch haben wir auch hier wieder festgestellt, dass es eine bzgl. Mengeninklusion aufsteigende Folge von zulässigen Kernen gibt, anhand derer wir von der leeren Menge als zulässigen Kern einen zulässigen Kern für V konstruieren können, sofern das Problem überhaupt eine Lösung besitzt.

Da wir für die Charakterisierung der Erweiterbarkeit nun auf die Grade der der Knoten der aktiven Region bzgl. des bereits konstruierten Intervallgraphen angewiesen sind, ist die folgende Definition nötig.

Definition 1.82 *Für ein d-Layout $L(X)$ von X ist der* Grad *von $v \in \mathcal{A}(X)$ definiert als die Anzahl der Intervalle, die $I(v)$ schneiden.*

Ein Kern-Paar *(X, f) ist ein d-zulässiger Kern X zusammen mit einer Gradfolge $f : \mathcal{A}(X) \to \mathbb{N}$, die jedem Knoten in der aktiven Region von X ihren Grad zuordnet.*

Das vorherige Lemma impliziert, dass zwei Layouts mit demselben Grad für jeden Knoten $v \in \mathcal{A}(X)$ entweder beide erweiterbar sind oder keines von beiden. Damit können wir unseren generische Algorithmus aus dem vorigen Abschnitt wie folgt für das Bounded Degree Interval Sandwich Problem erweitern. Wenn ein Kern Paar (X, f) betrachtet wird, wird jedes mögliche Kern-Paar (Y, g) hinzugefügt, das ein Layout für Y mit Gradfolge g besitzt und ein Layout von X mit Gradfolge f erweitert. Aus den vorherigen Lemmata folgt bereits die Korrektheit. Wir wollen uns im nächsten Abschnitt nun noch um die leichte Modifikation des Algorithmus und dessen Laufzeit kümmern.

1.7.3 Laufzeitabschätzung

Für die Laufzeitabschätzung stellen wir zunächst einmal fest, dass wir im Algorithmus eigentlich nichts anderes tun, als einen so genannten *Berechnungsgraphen* z.B. per Tiefensuche zu durchlaufen. Die Knoten dieses Berechnungsgraphen sind die Kern-Paare und zwei Kern-Paare (X, f) und (Y, g) sind mit einer gerichteten Kante verbunden, wenn sich (X, f) zu (Y, g) erweitern lässt. Unser Startknoten ist dann die leere Menge mit einer leeren Gradfolge und unser Zielknoten ist die Menge V mit einer leeren Gradfolge (da $\mathcal{A}(V) = \emptyset$).

Wir müssen also nur noch (per Tiefen- oder Breitensuche oder einer anderen optimierten Suchstrategie) feststellen, ob sich der Zielknoten vom Startknoten aus erreichen lässt. Die Laufzeit ist dann proportional zur Anzahl der Knoten und Kanten im Berechnungsgraphen. Daher werden wir diese als erstes abschätzen.

Lemma 1.83 *Ein zulässiger Kern X ist durch das Paar $(\mathcal{A}(X), \beta(X))$ eindeutig charakterisiert.*

Beweis: Wir müssen jetzt nur feststellen, wie wir anhand des gegebenen Paares feststellen können, welche Knoten sich im zulässigen Kern befinden. Dazu stellen wir fest, dass genau dann $x \in X$ ist, wenn es einen Pfad von x zu einem Knoten $v \in \mathcal{A}(X)$ der aktiven Region im Graphen $(V, M \setminus \beta(X))$ gibt. Dies gilt natürlich nur, weil wir in diesem Kapitel (V, M) als zusammenhängenden Graphen vorausgesetzt haben. ■

Somit können wir jetzt die Anzahl der zulässigen Kerne abzählen, indem wir die Anzahl der oben beschriebenen charakterisierenden Paare abzählen, wobei wir diese der Einfachheit halber überschätzen werden.

Zuerst einmal stellen wir fest, dass es maximal

$$\sum_{i=0}^{d-1} \binom{n}{i} \leq \sum_{i=0}^{d-1} n^i \leq \frac{n^d - 1}{n - 1} = O(n^{d-1})$$

paarweise verschiedene Möglichkeiten gibt, eine aktive Region aus V auszuwählen, da $|\mathcal{A}(X)| \leq d - 1$ (mit $n = |V|$).

Für die möglichen Ränder der aktiven Region gilt, dass deren Anzahl durch $2^{d(d-1)}$ beschränkt ist. Wir müssen nämlich von jedem der maximal $(d-1)$ Knoten in $\mathcal{A}(X)$ jeweils festlegen, welche ihrer maximal d Nachbarn bzgl. M im Rand liegen.

Jetzt müssen wir noch die Anzahl möglicher Gradfolgen abschätzen. Diese ist durch $d^{d-1} < d^d \leq 2^{\varepsilon \cdot d^2}$ für ein $\varepsilon > 0$ beschränkt, da nur für jeden Knoten aus der aktiven Region ein Wert aus d möglichen Werten in $[0 : d-1]$ zu vergeben ist. Somit ist die Anzahl der Kern-Paare ist beschränkt durch $O(2^{(1+\varepsilon)d^2} n^{d-1})$.

Lemma 1.84 *Die Anzahl der Kern-Paare, deren aktive Region maximal $d-1$ Knoten besitzt, ist beschränkt durch $O(2^{(1+\varepsilon)d^2}n^{d-1})$ für ein $\varepsilon > 0$.*

Nun müssen wir noch die Anzahl von Kanten im Berechnungsgraphen ermitteln. Statt dessen werden wir jedoch den maximalen Ausgangsgrad der Knoten ermitteln, woraus sich die maximale Anzahl von Kanten abschätzen lässt.

Wir betrachten zuerst die Kern-Paare, deren aktive Region maximale Größe, also $d-1$ Knoten, besitzen. Wie viele andere Kernpaare können ein solches Kern-Paar erweitern? Zuerst bemerken wir, dass zu einem solchen Kern nur Knoten hinzugefügt werden können, die zu Knoten der aktiven Region benachbart sind. Andernfalls würde die aktive Region auf d Knoten anwachsen, was nicht zulässig ist. Dies folgt aus der Tatsache, dass wir (V, M) als zusammenhängend annehmen und somit jeder neu aufgenommen Knoten zusätzlich in die aktive Region aufgenommen wird (da es ja keine Kante zwischen dem neuen Knoten und eines Knotens der bisherigen aktiven Region gibt).

Wir müssen also nur einen Knoten aus der Nachbarschaft der aktiven Region auswählen. Da diese aus weniger als d Knoten besteht und jeder Knoten im Graphen (V, M) nur maximal d Nachbarn hat, kommen nur d^2 viele Knoten in Frage.

Nachdem wir einen dieser Knoten ausgewählt haben, können wir mit Hilfe des Graphen (V, M) und der aktuell betrachteten aktiven Region sofort die aktive Region sowie deren Rand bestimmen. Ebenfalls die Gradfolge der aktiven Region lässt sich leicht ermitteln. Bei allen Knoten, die in der aktiven Region bleiben, erhöht sich der Grad um 1. Alle, die aus der aktiven Region herausfallen, sind uninteressant, da wir uns hierfür den Grad nicht zu merken brauchen. Der Grad des neu hinzugenommen Knotens ergibt sich aus der Kardinalität der alten aktiven Region.

Somit kann der Ausgangsgrad der Kern-Paare mit einer aktiven Region von $d-1$ Knoten durch d^2 abgeschätzt werden. Insgesamt gibt es also $O(2^{(1+\varepsilon)d^2} \cdot n^{d-1})$ viele Kanten, die aus Kern-Paaren mit einer aktiven Region von $d-1$ Knoten herausgehen.

Jetzt müssen wir noch den Ausgangsgrad der Kern-Paare abschätzen, deren aktive Region weniger als $d-1$ Knoten umfasst. Hier kann jetzt jeder Knoten, der sich noch nicht im Kern befindet hinzugenommen werden. Wiederum können wir sofort die aktive Region und dessen Rand mithilfe des Graphen (V, M) berechnen. Auch die Gradfolge folgt unmittelbar.

Wie viele Kern-Paare, deren aktive Region maximal $d-2$ Knoten umfasst, gibt es denn überhaupt? Wir haben dies vorhin im Lemma 1.84 für $d-1$ angegeben. Also gibt es $O(2^{(1+\varepsilon)d^2}n^{d-2})$. Da von all diesen jeweils maximal n Kanten in unserem Berechnungsgraphen ausgehen, erhalten wir also insgesamt gibt es also $O(2^{(1+\varepsilon)d^2} \cdot n^{d-1})$ viele Kanten, die aus Kern-Paaren mit einer aktiven Region von maximal $d-2$ Knoten herausgehen.

Für ein gegebenes Kern-Paar (X, f) und ein $v \in V \setminus X$ kann in Zeit $O(d^2)$ leicht ermittelt werden, für welches g das Paar $(X \cup \{v\}, g)$ ein Kern-Paar ist (Übungsaufgabe!).

Damit erhalten wir zusammenfassend das folgende Theorem.

Theorem 1.85 *Das Bounded Degree Interval Sandwich Problem für die Eingabe (V, M, F, d) kann in Zeit $O(2^{(1+\varepsilon)d^2} n^{d-1})$ für ein $\varepsilon > 0$ gelöst werden.*

Da das Intervalizing Colored Graphs als Spezialfall des Interval Sandwich Problems aufgefasst werden kann, erhalten wir auch hier für eine gradbeschränkte Lösung eine polynomielle Laufzeit.

Korollar 1.86 *Das ICG Problem ist in $\mathcal{P}$, wenn der maximale Grad der Lösung beschränkt ist.*

06.06.24

Phylogenetische Bäume 2

2.1 Einleitung

In diesem Kapitel wollen wir uns mit *phylogenetischen Bäumen* bzw. *evolutionären Bäumen* beschäftigen. Wir wollen also die Entwicklungsgeschichte mehrerer verwandter Spezies anschaulich als Baum darstellen bzw. das Auftreten von Unterschieden in den Spezies durch Verzweigungen in einem Baum wiedergeben.

Definition 2.1 *Ein* phylogenetischer Baum *für eine Menge $S = \{s_1, \ldots, s_n\}$ von n Spezies bzw. Taxa ist ein ungeordneter gewurzelter Baum mit n Blättern und den folgenden Eigenschaften:*

- *Jeder innere Knoten hat mindestens zwei Kinder;*
- *Jedes Blatt ist mit genau einem Taxon $s \in S$ markiert;*
- *Jedes Taxon taucht nur einmal als Blattmarkierung auf.*

Ungeordnet bedeutet hier, dass die Reihenfolge der Kinder eines Knotens ohne Belang ist. Die bekannten und noch lebenden (zum Teil auch bereits ausgestorbenen) Spezies werden dabei an den Blättern dargestellt. Jeder (der maximal $n-1$) inneren Knoten entspricht dann einem Ahnen der Spezies, die in seinem Teilbaum die Blätter bilden. In Abbildung 2.1 ist ein Beispiel eines phylogenetischen Baumes angegeben.

Noch ein paar Anmerkungen zur Definition von phylogenetischen Bäumen: Manchmal werden wir statt gewurzelter auch ungewurzelte, also freie Bäume, als phylogenetische Bäume betrachten, wir werden dies aber dann jeweils explizit erwähnen. Da die Klassifikation mittels phylogenetischer Bäume auch für andere Arten als evolutionäre Stammbäume verwendet wird, wie etwa die Einteilung der Sprachfamilien in der Linguistik, aber auch bei der Analyse von Genen oder Gengruppen, spricht man oft auch ganz allgemein von *Taxa* anstatt von Spezies. Manchmal erlaubt man auch, dass die gegebene Spezies bzw. Taxa nicht nur an den Blättern, sondern auch an den inneren Knoten auftreten dürfen.

Wir wollen uns hier mit der mathematischen und algorithmischen Rekonstruktion von phylogenetischen Bäumen anhand der gegebenen (in der Regel) biologischen

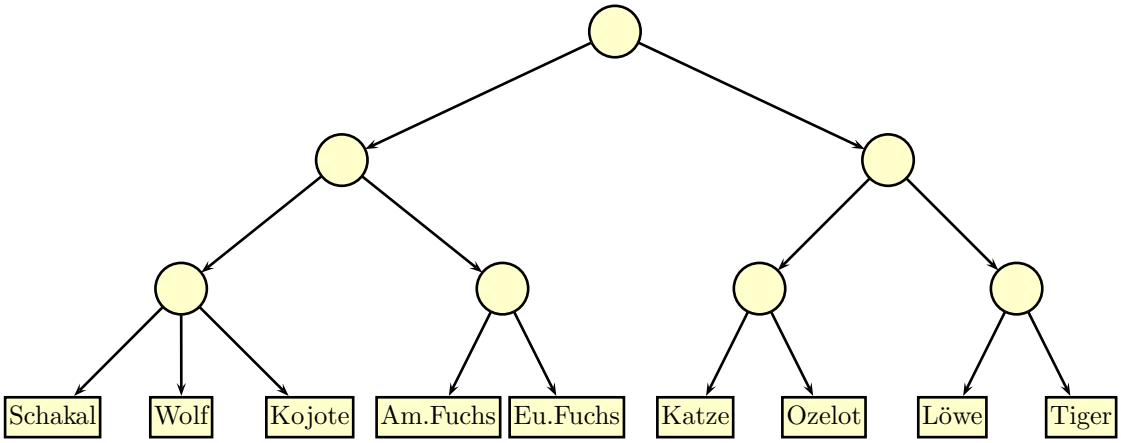


Abbildung 2.1: Beispiel: Ein phylogenetischer Baum

Daten beschäftigen. Die daraus resultierenden Bäume müssen daher nicht immer mit der biologischen Wirklichkeit übereinstimmen. Die rekonstruierten phylogenetischen Bäume mögen beispielsweise Ahnen vorhersagen, die niemals existiert haben.

Dies liegt zum einen daran, dass die biologischen Daten nicht in der Vollständigkeit und Genauigkeit vorliegen, die für eine mathematische Rekonstruktion nötig sind. Zum anderen liegt dies auch an den vereinfachenden Modellen, da in der Natur nicht nur Speziationen (d.h. Verzweigungen in den Bäumen) vorkommen können, sondern dass auch Vereinigungen bereits getrennter Spezies vorkommen können (beispielsweise durch lateralen Gen-Transfer).

Biologisch würde man daher eher nach einem gerichteten azyklischen Graphen statt eines gewurzelten Baumes suchen. Da diese Verschmelzungen aber eher vereinzelt vorkommen, bilden phylogenetischer Bäume einen ersten Ansatzpunkt, in die dann weiteres biologisches Wissen eingearbeitet werden kann.

In der Rekonstruktion unterscheidet man drei prinzipiell unterschiedliche Verfahren: distanzbasierte, charakterbasierte bzw. merkmalsbasierte und probabilistische Methoden, die wir in den folgenden Unterabschnitten genauer erörtern werden.

2.1.1 Distanzbasierte Verfahren

Bei den so genannten *distanzbasierten Verfahren* wird zwischen den Spezies bzw. Taxa ein Abstand bestimmt. Man kann diesen einfach als die Zeitspanne in die Vergangenheit interpretieren, vor der sich die beiden Spezies bzw. Taxa durch Speziation aus einem gemeinsamen Urahn auseinander entwickelt haben.

Für solche Distanzen, also evolutionäre Abstände, können beispielsweise die EDIT-Distanzen (oder andere komplexere Distanzmaße) von speziellen DNS-Teilsträngen oder Aminosäuresequenzen verwendet werden. Hierbei wird angenommen, dass sich die Sequenzen durch Mutationen auseinander entwickeln und dass die Anzahl der so genannten akzeptierten Mutationen (also derer, die einem Weiterbestehen der Art nicht im Wege standen) zur zeitlichen Dauer korreliert ist (beispielsweise aus nicht-kodierenden oder nicht-regulatorischen Bereichen). Hierbei muss man vorsichtig sein, da unterschiedliche Bereiche im Genom auch unterschiedliche Mutationsraten besitzen können.

Eine andere Möglichkeit aus früheren Tagen sind so genannte Hybridisierungsexperimente. Dabei werden durch vorsichtiges Erhitzen die DNS-Doppelstränge zweier Spezies voneinander getrennt. Bei der anschließenden Abkühlung hybridisieren die DNS-Stränge wieder miteinander. Da jetzt jedoch DNS-Einzelstränge von zwei Spezies vorliegen, können auch zwei Einzelstränge von zwei verschiedenen Spezies miteinander hybridisieren, vorausgesetzt, die Stränge waren nicht zu verschieden.

Beim anschließenden erneuten Erhitzen trennen sich diese gemischten Doppelstränge umso schneller, je verschiedener die DNS-Sequenzen sind, da dann entsprechend weniger Wasserstoffbrücken aufzubrechen sind. Aus den Temperaturen, bei denen sich dann diese gemischten DNS-Doppelstränge wieder trennen, kann man dann ein evolutionäres Abstandsmaß gewinnen.

Auch Distanzen aus der Untersuchung von so genannten Genome Rearrangements können als bei distanzbasierten Verfahren verwendet werden.

Ziel der evolutionären Verfahren ist es nun, einen Baum mit Kantengewichten zu konstruieren, so dass das Gewicht der Pfade von den zwei Spezies bzw. Taxa zu ihrem niedrigsten gemeinsamen Vorfahren dem Abstand entspricht. Ein solcher phylogenetischer Baum, der aufgrund von künstlichen evolutionären Distanzen konstruiert wurde, ist in der Abbildung 2.2 illustriert.

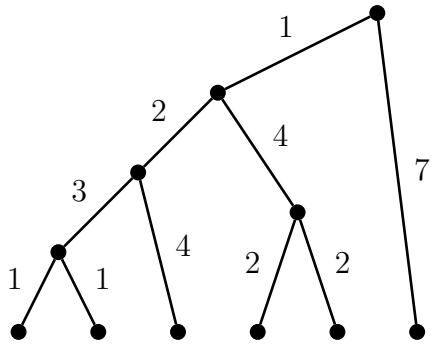


Abbildung 2.2: Beispiel: Ein distanzbasierter phylogenetischer Baum

2.1.2 Merkmalsbasierte Verfahren

Bei den so genannten *charakterbasierten Verfahren* bzw. *merkmalsbasierten Verfahren* verwendet man gewisse Eigenschaften, so genannte *Charaktere* bzw. *Merkmale*, der Spezies bzw. Taxa. Hierbei unterscheidet man *binäre Charaktere* bzw. *binäre Merkmale*, wie beispielsweise „ist ein Säugetier“, „ist ein Wirbeltier“, „ist ein Fisch“, „ist ein Vogel“, „ist ein Lungenatmer“, etc., *numerische Charaktere* bzw. *numerische Merkmale*, wie beispielsweise Anzahl der Extremitäten, Anzahl der Wirbel, (also morphologische Merkmale), copy number variation (genomische Merkmale), etc., und *zeichenreihige Charaktere* bzw. *zeichenreihige Merkmale*, wie beispielsweise ganz bestimmte Teilsequenzen in der DNS (also molekularbiologische Merkmale). Bei zeichenreihigen Merkmalen werden oft Teilsequenzen aus nicht-codierenden und nicht-regulatorischen Bereichen der DNS betrachtet, da diese bei Mutationen in der Regel unverändert weitergegeben werden und nicht durch Veränderung einer lebenswichtigen Funktion sofort aussterben.

Das Ziel ist auch hier wieder die Konstruktion eines phylogenetischen Baumes, wobei die Kanten mit Merkmalen und ihren Änderungen markiert werden. Eine Markierung einer Kante mit einem Merkmal bedeutet hierbei, dass alle Spezies bzw. Taxa in dem Teilbaum nun eine Änderung dieses Merkmals erfahren. Die genaue Änderung dieses Merkmals ist auch an der Kante erfasst.

Bei merkmalbasierten Verfahren verfolgt man das Prinzip der minimalen Mutationshäufigkeit bzw. der maximalen Parsimonie (engl. parsimony, auf dt. Geiz, Sparsamkeit). Das bedeutet, dass man einen Baum sucht, der so wenig Kantenmarkierungen wie möglich besitzt. Man geht hierbei von der Annahme aus, dass die Natur keine unnötigen Mutationen verwendet (Occam's Razor).

In der Abbildung 2.3 ist ein Beispiel für einen solchen merkmalbasierten phylogenetischen Baum angegeben, wobei hier nur binäre Merkmale verwendet wurden.

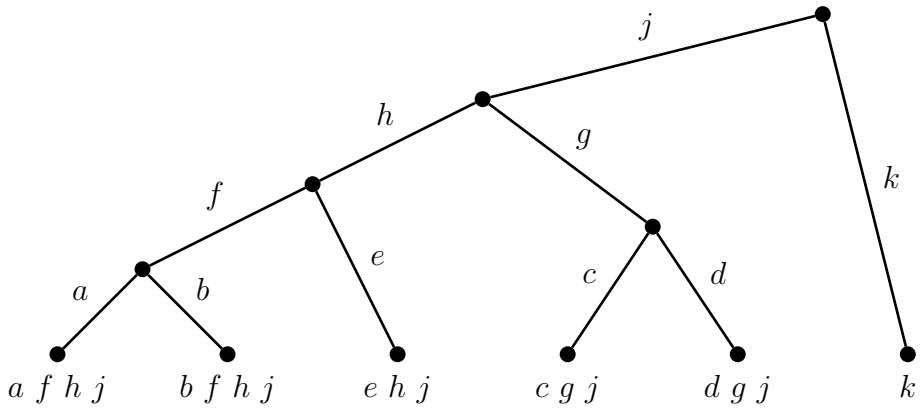


Abbildung 2.3: Beispiel: Ein merkmalbasierter phylogenetischer Baum

Außerdem werden die binären Merkmale hier so verwendet, dass sie nach einer Kantenmarkierung in den Teilbaum eingeführt und nicht gelöscht werden. Bei binären Merkmalen kann man dies immer annehmen, da man ansonsten das binäre Merkmal nur negieren muss.

2.1.3 Probabilistische Verfahren

Bei probabilistischen Verfahren wird ein Mutationsmodell für die verwendeten Daten benötigt. Hierbei wird für Mutationen bzw. Merkmalsänderungen eine Wahrscheinlichkeit festgelegt, mit der man annimmt, dass diese Mutationen auftreten. Basierend auf solchen Mutationswahrscheinlichkeiten und Wahrscheinlichkeitsverteilungen gibt es zwei Anwendungen, zum einen die Berechnung der Kantenlängen eines gegebenen phylogenetischen Baumes und zum anderen die Rekonstruktion der eigentlichen Topologie des phylogenetischen Baumes selbst.

In jedem Fall versucht man, die Wahrscheinlichkeit eines gegebenen phylogenetischen Baumes unter dem zugrunde liegenden Modell (im Wesentlichen die Wahrscheinlichkeitsverteilung der Mutationen, manchmal auch die Topologie) zu bestimmen. Derjenige phylogenetische Baum, der die größte Wahrscheinlichkeit seines Auftretens unter dem betrachteten Modell besitzt, wird als der phylogenetische Baum angesehen, der die Wirklichkeit am besten beschreibt (daher der Name *Maximum-Likelihood*). Oft wird auch ein Bayesscher Ansatz verwendet, also eine Maximum-a-posteriori-Methode. Zur numerischen Lösung werden dann oft Markov Chain Monte Carlo Algorithmen eingesetzt.

Problematisch wird es bei diesem Ansatz, wenn man versucht die Topologie eines phylogenetischen Baumes zu bestimmen. Da es für n Taxa exponentiell viele phylogenetische Bäume mit einer unterschiedlichen Topologie gibt, ist das Problem für große n so nicht lösbar. Daher werden oft aus den Daten inkrementell verschiedene phylogenetische Bäume aufgebaut und derjenige mit der größten Wahrscheinlichkeit bestimmt. Da hierbei nicht alle möglichen phylogenetischen Bäume betrachtet werden, kann die Lösung suboptimal sein. Oft bleibt man dabei in einem lokalen Minimum stecken.

2.2 Perfekte binäre Phylogenie

In diesem Abschnitt wollen wir uns jetzt um merkmalbasierte Methoden kümmern. Wir werden dazu allgemeine Kriterien angeben, wann eine Merkmalsmatrix überhaupt eine perfekte Phylogenie besitzt. Unter gewissen Umständen lassen sich daraus effiziente Algorithmen zur Rekonstruktion der perfekten Phylogenie ableiten.

2.2.1 Charakterisierung binärer perfekter Phylogenie

Wir beschränken uns hier auf den Spezialfall, dass die Merkmale binär sind, d.h. nur zwei verschiedene Werte annehmen können. Zunächst benötigen wir noch ein paar grundlegende Definitionen, um das Problem der Konstruktion phylogenetischer Bäume formulieren zu können.

Definition 2.2 *Eine* binäre Charaktermatrix *bzw.* binäre Merkmalsmatrix M *für* n *Taxa und* m *Merkmale ist eine binäre* $n \times m$*-Matrix. Ein* phylogenetischer Baum T *für eine binäre* $n \times m$*-Merkmalsmatrix ist ein ungeordneter gewurzelter Baum mit genau* n *Blättern, so dass:*

1. *Jedes der* n *Taxa aus* $[1:n]$ *markiert genau eines der Blätter und jedes Blatt ist mit einem Taxon markiert;*
2. *Jeder der* m *Merkmale aus* $[1:m]$ *markiert genau eine Kante;*
3. *Für jedes Taxon* $p \in [1:n]$ *gilt für die Menge* $C_T(p)$ *der Kantenmarkierungen auf dem Pfad von der Wurzel von* T *zu dem Blatt mit Markierung* p*, dass* $C_T(p) = \{c : M_{p,c} = 1\}$.

Man beachte, dass jedes Blatt aber nicht jede Kante markiert sein muss. Außerdem kann eine Kante mehrere Markierungen besitzen. In Abbildung 2.4 ist eine binäre Merkmalsmatrix samt seines zugehörigen phylogenetischen Baumes angegeben.

M	1	2	3	4	5	6	7
1	1	0	1	0	0	1	0
2	1	0	1	0	1	0	1
3	1	0	1	0	1	0	0
4	1	0	0	0	0	0	0
5	0	1	0	0	0	0	0
6	0	1	0	1	0	0	0

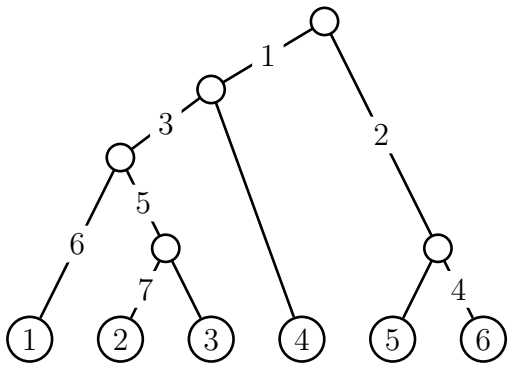


Abbildung 2.4: Beispiel: binäre Merkmalsmatrix und zugehöriger phylogenetischer Baum

Somit können wir das Problem der perfekten Phylogenie formulieren.

Perfekte binäre Phylogenie

Eingabe: Eine binäre $n \times m$-Merkmalsmatrix M.
Gesucht: Ein phylogenetischer Baum T für M.

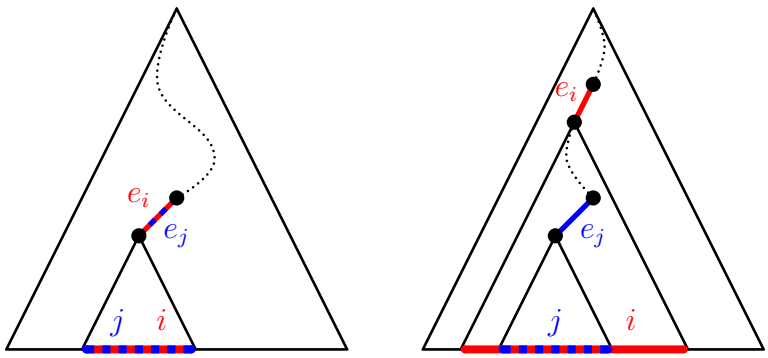
Zunächst benötigen wir noch eine weitere Notation bevor wir uns der Lösung des Problems zuwenden können.

Notation 2.3 *Sei M eine binäre $n \times m$-Merkmalsmatrix, dann umfasst die Menge $O_j = \{i \in [1:n] : M_{i,j} = 1\}$ die Objekte bzw. Taxa, die das Merkmal j besitzen.*

Wir geben zunächst eine Charakterisierung an, wann eine binäre Merkmalsmatrix überhaupt einen phylogenetischen Baum besitzt.

Theorem 2.4 *Eine binäre $n \times m$-Merkmalsmatrix M besitzt genau dann einen phylogenetischen Baum, wenn für jedes Paar $i, j \in [1:m]$ eine der folgenden Bedingungen $O_i \cap O_j = \emptyset$ oder $O_i \subseteq O_j$ oder $O_i \supseteq O_j$ gilt (also wenn $O_i \perp O_j$).*

Beweis: $\Rightarrow$**:** Sei T ein phylogenetischer Baum für M. Sei e_i bzw. e_j die Kante in T mit der Markierung i bzw. j. Wir unterscheiden jetzt vier Fälle, je nachdem, wie sich die Kanten e_i und e_j zueinander im Baum T verhalten.



Fall 1: $e_i = e_j$ Fall 2: $e_i > e_j$

Fall 3: $e_j > e_i$ Fall 4: $e_i \not\leqslant e_j$

Abbildung 2.5: Skizze: Phylogenetischer Baum für M

Fall 1: Wir nehmen zuerst an, dass $e_i = e_j$ ist (siehe auch Abbildung 2.5). In diesem Fall gilt offensichtlich $O_i = O_j$ und somit auch $O_i \subseteq O_j$ bzw. $O_j \subseteq O_i$.

Fall 2: Nun nehmen wir an, dass sich im Teilbaum vom unteren Knoten vom e_i die Kante e_j befindet (siehe auch Abbildung 2.5). Also sind alle Nachfahren des unteren Knotens von e_j auch Nachfahren des unteren Knoten von e_i und somit gilt $O_j \subseteq O_i$.

Fall 3: Nun nehmen wir an, dass sich im Teilbaum vom unteren Knoten vom e_j die Kante e_i befindet (siehe auch Abbildung 2.5). Also sind alle Nachfahren des unteren Knotens von e_i auch Nachfahren des unteren Knoten von e_j und somit gilt $O_i \subseteq O_j$.

Fall 4: Der letzte verbleibende Fall ist, dass keine Kante Nachfahre der jeweils anderen Kante ist (siehe auch Abbildung 2.5). Der Elterknoten der beiden Kanten e_i und e_j kann auch derselbe sein. In diesem Fall gilt offensichtlich $O_i \cap O_j = \emptyset$.

$\Leftarrow$: Wir müssen jetzt aus der Matrix M (mithilfe der Orthogonalitäts-Bedingungen an die Objektmengen O_i) einen phylogenetischen Baum konstruieren. Zuerst nehmen wir ohne Beschränkung der Allgemeinheit an, dass die Spalten der Matrix M interpretiert als Binärzahlen absteigend sortiert sind. Dabei nehmen wir an, dass jeweils in der ersten Zeile das höchstwertige Bit und in der letzten Zeile das niedriegstwertige Bit dieser Binärzahl steht. Wir machen uns dies noch einmal anhand unseres Beispiels aus der Einleitung klar und betrachten dazu Abbildung 2.6. Der besseren Übersichtlichkeit wegen, bezeichnen wir die Spalten jetzt mit a–g anstatt mit 1–7. Links ist die ursprüngliche Matrix und rechts die nach den Spalten absteigend sortierte Matrix M' angegeben.

M	a	b	c	d	e	f	g
1	1	0	1	0	0	1	0
2	1	0	1	0	1	0	1
3	1	0	1	0	1	0	0
4	1	0	0	0	0	0	0
5	0	1	0	0	0	0	0
6	0	1	0	1	0	0	0

M'	a	c	f	e	g	b	d
1	1	1	1	0	0	0	0
2	1	1	0	1	1	0	0
3	1	1	0	1	0	0	0
4	1	0	0	0	0	0	0
5	0	0	0	0	0	1	0
6	0	0	0	0	0	1	1

Abbildung 2.6: Beispiel: Sortierte Merkmalsmatrix

Wir betrachten jetzt zwei beliebige Objekte p und q. Sei $k := k_{p,q}$ das größte gemeinsame Merkmal, das sowohl p als auch q besitzt, d.h.

$$k := k_{p,q} := \max\{0, j \ : \ M(p,j) = M(q,j) = 1\}.$$

Hierbei gilt $k = 0$, wenn überhaupt kein solches j existiert. Wir stellen jetzt die folgende Behauptung auf.

Behauptung: Es gilt:

a) $\forall i \leq k : M(p,i) = M(q,i)$;

b) $\forall i > k : M(p,i) = M(q,i) \Rightarrow M(p,i) = 0 = M(q,i)$.

Teil b) und der Fall $k = i$ im Teil a) der Behauptung folgen unmittelbar aus der Definition (auch für $k = 0$).

Für Teil a) genügt es, die beiden folgenden Aussagen zu beweisen:

- $\forall i < k : M(p,i) = 1 \Rightarrow M(q,i) = 1$;
- $\forall i < k : M(p,i) = 1 \Leftarrow M(q,i) = 1$;

Dazu betrachten wir zuerst ein Merkmal $i < k$ von p mit $M(p,i) = 1$. Falls kein solches i existiert, ist nichts zu zeigen. Andernfalls gilt $p \in O_i \cap O_k$ und somit $O_i \cap O_k \neq \emptyset$. Nach Voraussetzung gilt dann $O_i \subseteq O_k$ oder $O_k \subseteq O_i$. Da die Spalten absteigend sortiert sind, muss die größere Zahl (also die zur Spalte i korrespondierende) mehr (zusätzliche) 1-Bits besitzen und es gilt somit $O_k \subseteq O_i$. Da $q \in O_k \subseteq O_i$ gilt, ist nun auch $q \in O_i$ und $M(q,i) = 1$. Die analoge Aussage gilt ebenso für ein $i < k$ mit $M(q,i) = 1$. □

Damit gilt für alle Merkmale $i \leq k$ (nach der Spaltensortierung), dass entweder beide das Merkmal i besitzen oder keiner. Da k der größte Index ist, so dass beide ein Merkmal gemeinsam besitzen, gilt für $i > k$, dass aus $M(p,i) = M(q,i)$ folgt, dass beide das Merkmal i nicht besitzen, d.h. $M(p,i) = M(q,i) = 0$.

Betrachten wir also zwei Objekte (also zwei Zeilen in der spaltensortierten Merkmalsmatrix), dann besitzen zu Beginn entweder beide ein Merkmal gemeinsam oder keines. Sobald wir ein Merkmal finden, das beide Objekte unterscheidet, können wir später kein gemeinsames Merkmal mehr finden.

Mit Hilfe dieser Eigenschaft können wir jetzt einen phylogenetischen Baum konstruieren. Dazu betrachten wir die Abbildung $p \mapsto s_{p,1} \cdots s_{p,\ell}\$$, wobei wir jedem Objekt eine Zeichenkette über $[1:m]$ zuordnen, so dass jedes Symbol aus $[1:m]$ maximal einmal auftaucht und ein $i \in [1:m]$ nur dann auftaucht, wenn p das entsprechende Merkmal besitzt, also wenn $M(p,i) = 1$. Die Reihenfolge ergibt sich dabei aus der Spaltensortierung der Merkmalsmatrix. Für unser Beispiel ist diese Zuordnung in der Abbildung 2.7 angegeben.

Wenn wir jetzt für diese Zeichenreihen einen Trie konstruieren (dies entspricht beispielsweise auch dem Suchmuster-Baum aus dem Aho-Corasick-Algorithmus), so erhalten wir den phylogenetischen Baum für unsere Merkmalsmatrix M. Wir müssen nur noch die Kantenmarkierungen mit dem \$ entfernen und Knoten mit nur

$$
\begin{aligned}
s_1 &= acf\$ \\
s_2 &= aceg\$ \\
s_3 &= ace\$ \\
s_4 &= a\$ \\
s_5 &= b\$ \\
s_6 &= bd\$
\end{aligned}
$$

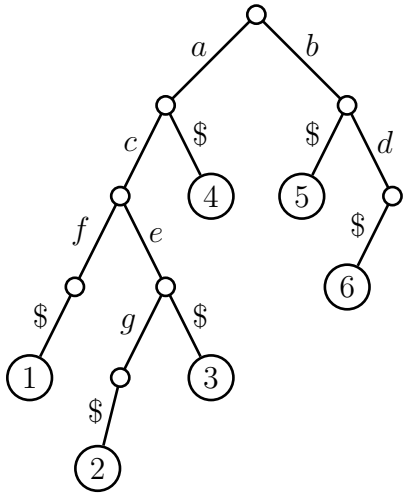


Abbildung 2.7: Skizze: Trie für die Zeichenreihen der Merkmalsmatrix

einem Kind kontrahieren. Das Terminalsymbol \$ hatten wir nur eingeführt, damit keine Taxa an inneren Knoten verbleiben. Für unser Beispiel ist diese Kompaktifizierung in der Abbildung 2.8 illustriert.

Aus der Konstruktion folgt, dass alle Blätter die benötigten Kantenmarkierungen auf dem Pfad von der Wurzel zum Blatt besitzen. Nach der Spaltensortierung und der daran anschließenden Diskussion kommt auch jedes Merkmal nur einmal als Kantenmarkierung vor. Somit haben wir nachgewiesen, dass der konstruierte Baum ein phylogenetischer Baum ist und der Beweis ist beendet. ■

$$
\begin{aligned}
s_1 &= acf\$ \\
s_2 &= aceg\$ \\
s_3 &= ace\$ \\
s_4 &= a\$ \\
s_5 &= b\$ \\
s_6 &= bd\$
\end{aligned}
$$

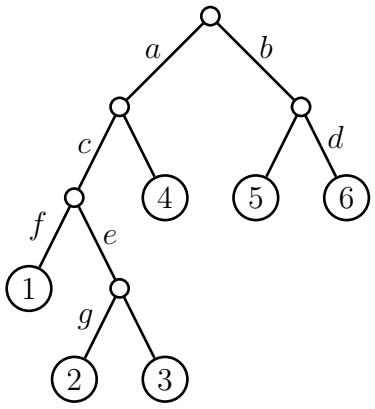


Abbildung 2.8: Skizze: Kompaktifizierter Trie ohne Sentenial für die Zeichenreihen der Merkmalsmatrix

2.2.2 Algorithmus zur perfekten binären Phylogenie

Aus dem vorherigen Beweis erhalten wir unmittelbar einen Algorithmus zur Berechnung einer perfekten binären Phylogenie, der in Abbildung 2.9 aufgelistet ist.

1. Sortieren der Spalten der binären Merkmalsmatrix. $O(nm)$
2. Konstruktion der Zeichenreihen $s_1, \ldots, s_n$. $O(nm)$
3. Konstruktion des Tries T für $s_1, \ldots, s_n$. $O(nm)$
4. Kompaktifiziere den Trie T und entferne \$. $O(nm)$
5. Teste, ob der kompaktifizierte Trie ein phylogenetischer Baum ist. $O(nm)$

Abbildung 2.9: Algorithmus: Konstruktion einer perfekten binäre Phylogenie

Anstatt die Bedingung aus dem Lemma zu überprüfen, konstruieren wir einfach einen kompaktifizierten Trie. Lässt sich dieser Trie in einen phylogenetischen Baum umwandeln, so muss dieser der gesuchte sein. Andernfalls gibt es keinen phylogenetischen Baum für die gegebene binäre Merkmalsmatrix.

Theorem 2.5 *Für eine binäre $n \times m$-Merkmalsmatrix lässt sich in Zeit $O(nm)$ entscheiden, ob sie eine perfekte Phylogenie besitzt. Der zugehörige phylogenetische Baum kann in Zeit $O(nm)$ konstruiert werden.*

Beweis: Wir müssen nur noch den Beweis zur Laufzeit führen. Zur Sortierung der Spalten verwenden wir einen Radix- oder Bucket-Sort, der sich in Zeit $O(nm)$ implementieren lässt. Die Zeichenreihen können sich anschließend trivialerweise in Zeit $O(nm)$ erzeugen lassen. Die Konstruktion des zugehörigen Tries ist durch die Anzahl der Zeichen in den Zeichenreihen, also $O(nm)$, beschränkt. Der Trie hat dann eine Größe von $O(nm)$. Die Kompaktifizierung lässt sich mittels einer Tiefensuche über den Trie in Zeit $O(nm)$ ausführen. Für die Entscheidung, ob der Trie einen phylogenetischen Trie darstellt, muss nur noch festgestellt werden, ob jede Kantenmarkierung maximal einmal vorkommt. Auch dies lässt sich mit Hilfe eines Booleschen Feldes für die Kantenmarkierungen mit einer Tiefensuche durch den Trie in Zeit $O(nm)$ bewerkstelligen. ■

Der eben gezeigte Algorithmus für die perfekte binäre Phylogenie hatte zur Voraussetzung, dass Übergänge eines binären Merkmals nur von 0 nach 1 erlaubt waren. Jetzt seien beide Richtungen eines Zustandwechsels erlaubt (wobei natürlich nur

eine der beiden Zustandänderungen eines Merkmals im zugehörigen phylogenetischen Baum auftreten darf).

Betrachte dazu die folgende Transformation einer binären Merkmalsmatrix M in eine binäre Merkmalsmatrix M': *In jeder Spalte, in der mehr 1en als 0en vorkommen, komplementiere die Einträge dieser Spalte.* Man kann jetzt zeigen, dass der Algorithmus aus der Vorlesung angewendet auf die transformierte Merkmalsmatrix M' eine perfekte binäre Phylogenie für M konstruiert. Die Details überlassen wir dem Leser als Übung.

11.06.24

2.3 Allgemeine perfekte Phylogenie

Im Folgenden wollen wir das Problem der perfekten Phylogenie näher untersuchen, wenn die Merkmale nicht nur binär sind, sondern jeweils beliebige Werte aus $[1 : r]$ annehmen können. Hierbei ist zu beachten, dass im Zustand nicht das Merkmal kodiert ist. Der Zustand 2 des Merkmals 1 kann also etwas völlig anderes sein als der Zustand 2 des Merkmals 2.

2.3.1 Charakterisierung allgemeiner perfekter Phylogenien

In der binären Phylogenie hatten wir zunächst nur angenommen, dass Zustandsübergänge der Form $0 \to 1$ auftreten. Später haben wir ohne Beweis erwähnt, dass es möglich ist Zustandsübergänge in beide Richtungen zuzulassen. Im allgemeinen Fall können die Zustandsübergänge auch gewissen Einschränkungen unterliegen. Meistens werden mögliche Übergänge in einen Diagramm, wie in Abbildung 2.10 dargestellt. Hier sind beispielsweise alle Übergänge möglich (wie im ersten Diagramm

Abbildung 2.10: Skizze: Mögliche Zustandsübergangsdiagramme

von links), die Übergänge bilden eine lineare Kette (wie im zweiten Diagramm), die Übergänge bilden ein partielle Ordnung (wie im dritten Diagramm), oder die Übergänge sind ohne weitere Struktur (wie im vierten Diagramm). Im Folgenden wollen wir annehmen, dass zwischen den Zuständen alle Übergänge möglich sind. Wir schränken nur ein, dass jeder Zustand nur einmal in der Evolution neu generiert wird. Dies führt zur folgenden Definition den allgemeinen perfekten Phylogenie.

PERFEKTE PHYLOGENIE

Eingabe: Eine $n \times m$-Merkmalsmatrix M mit Einträgen aus $[1 : r]$.
Gesucht: Ein ungewurzelter phylogenetischer Baum T für M, so dass alle Knoten mit Zustand $j \in [1 : r]$ des Merkmals $i \in [1 : m]$ einen Teilbaum von T bilden.

Ein Beispiel eines solchen phylogenetischen Baumes zu einer perfekten Phylogenie ist in der folgenden Abbildung 2.11 angegeben. Hierbei bezeichen wir der Über-

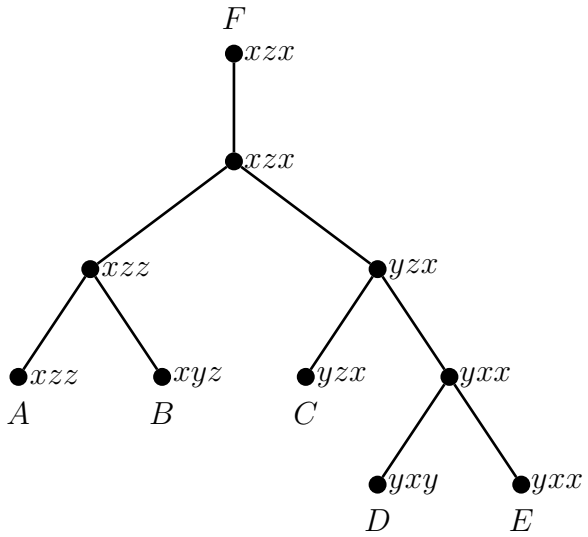


Abbildung 2.11: Beispiel: Phylogenetischer Baum mit mehrwertigen Merkmalen

sichtlichkeit die Zustände der verschiedenen Merkmale einfach mit x, y und z. Die Merkmalsmatrix für diesen Baum ist in der Tabelle in Abbildung 2.12 angegeben.

	c_1	c_2	c_3
A	x	z	z
B	x	y	z
C	y	z	x
D	y	x	y
E	y	x	x
F	x	z	x

Abbildung 2.12: Beispiel: Merkmalsmatrix des phylogenetischen Baumes

Diesen Baum können wir auch etwas anders darstellen, indem wir den Baum durch die Teilbäume darstellen, in denen sich ein Merkmal in einem festen Zustand befindet. Dies ist für den phylogenetischen Baum aus Abbildung 2.11 in Abbildung 2.13 dargestellt. Der Übersichtlichkeit haben wir hier die Zustände mit dem Merkmal

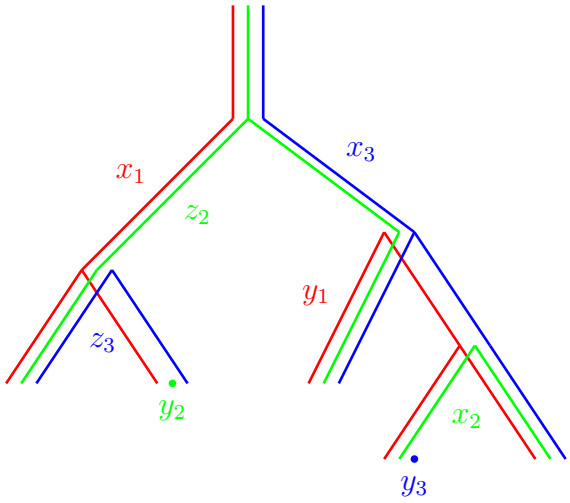


Abbildung 2.13: Beispiel: Phylogenetischer Baum mit Merkmalen als Pfade

indiziert, zu dem sie gehören. Gleichzeitig wurden die drei verschiedenen Merkmale auch noch farbig unterschienden.

Ziel wird es also sein, einen Baum zu rekonstruieren, wie beispielsweise in Abbildung 2.13 angegeben. Zu so dargestellten Bäumen (als Menge von Teilbäumen) können wir einen anderen Graphen definieren, der uns im Folgenden hilfreich sein wird.

Definition 2.6 *Sei $T = (W, F)$ ein Baum und $\mathcal{T} = \{T_1, \ldots, T_\ell\}$ eine Multimenge von Teilbäumen von T. $\mathcal{T}$ heißt* Baumdarstellung *des Graphen $G = (V, E)$ mit*

- $V = [1 : \ell] \cong \mathcal{T}$;
- $E = \{\{i, j\} : V(T_i) \cap V(T_j) \neq \emptyset\}$.

Ein Graph heißt Durchschnittsgraph von Bäumen *(bzw.* tree intersection graph*), wenn er eine Baumdarstellung besitzt.*

Wir wollen hier noch anmerken, dass diese tree intersection graphs eine Verallgemeinerung von interval graphs darstellen, in denen jedes Intervall als ein spezieller Baum aufgefasst werden kann, nämlich einem Pfad.

Wenn wir einen phylogenetischen Baum in einer Baumdarstellung haben, dann sind im zugeordneten Durchschnittsgraph von diesen Bäumen die Knoten gerade die verschiedenen Zustände der vorhandenen Merkmale.

Für die in Abbildung 2.13 angegebene Baumdarstellung ist der zugehörige Durchschnittsgraph von Bäumen in Abbildung 2.14 angegeben.

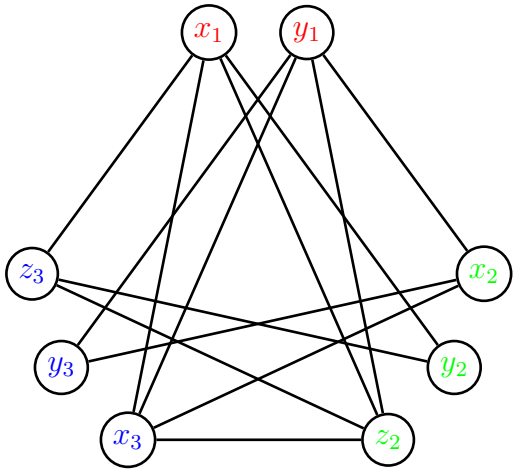


Abbildung 2.14: Beispiel: zugehöriger Durchschnittsgraph von Bäumen

Welchen Vorteil hat diese Darstellung? Wir können versuchen aus der gegebenen Merkmalsmatrix den Durchschnittsgraphen von Bäumen zu rekonstruieren. Wenn es eine Zuordnung zur Baumdarstellung gibt, haben wir einen phylogenetischen Baum rekonstruiert. Diese Idee werden wir im Folgenden weiter verfolgen. Zunächst geben wir noch eine Charakterisierung von Durchschnittsgraphen von Bäumen an, die für uns hilfreich sein wird. Dazu benötigen wir erst noch die Definition eine induzierten Teilgraphs und eines chordalen Graphen.

Definition 2.7 *Sei $G = (V, E)$ ein Graph. Ein Graph $G' = (V', E')$ heißt* Teilgraph, *bezeichnet mit $G' \subseteq G$, wenn $V' \subseteq V$ und $E' \subseteq E \cap \binom{V'}{2}$.*

Für eine Knotenmenge $V' \subseteq V$ bezeichnet $G[V'] = (V', E')$ den induzierten Teilgraph, *wenn $G[V']$ ein Teilgraph von G ist und wenn $E' = E \cap \binom{V'}{2}$*

Nun sind wir in der Lage chordale Graphen zu definieren.

Definition 2.8 *Ein Graph $G = (V, E)$ heißt* chordal *oder* trianguliert, *wenn für jeden einfachen Kreis $C \subseteq G$ mit $|V(C)| \geq 4$ gilt, dass $|E(G[V(C)])| > |E(C)|$ ist.*

Anschaulich bedeutet die obige Definition, dass mit jedem Kreis, der ein Teilgraph ist, auch eine Sehne (engl. *chord*) des Kreises im Graphen enthalten sein muss.

Iteriert man diese Beschreibung, so folgt, dass Kreise ohne Sehnen Dreiecke sein müssen und der Graph dann quasi aus Dreiecken bestehen muss, also trianguliert sein muss (insbesondere ist dies für planare Graphen ersichtlich). Anders interpretiert bedeutet dies, das jeder induzierte Kreis eines chordalen Graphen ein Dreieck sein muss. Daher stammt auch der Name triangulierter Graph.

Für den eigentlichen fundamentalen Satz fehlt uns noch eine weitere Definition (zur Erinnerung: Cliquen hatten wir in Definition 1.76 auf Seite 81 definiert).

Definition 2.9 *Sei $G = (V, E)$ ein Graph. Eine* Baumzerlegung in Cliquen *von G ist ein Baum $T = (W, F)$ mit*

- *$C \in W$ genau dann, wenn C eine Clique in G ist;*
- *für jedes $v \in V$ ist der induzierte Teilgraph $T[C_v]$ mit $C_v = \{C \in W : v \in C\}$ ein Baum.*

Anschaulich bedeutet dies, dass man den Graphen durch Aufzählung seiner Cliquen beschreiben kann. Damit sind alle Knoten und Kanten (eben die in den Cliquen) beschrieben. Darüber hinaus besitzen die einzelnen Knoten in den Cliquen noch eine baumartige Struktur. Letztendlich versucht man bei einer Baumzerlegung in Cliquen, die in dem Graphen vorhandene Baumstruktur zu extrahieren.

Wir geben in Abbildung 2.15 noch die Baumzerlegung in Cliquen für den Graphen aus Abbildung 2.14 an, wobei die sechs Cliquen hierbei $\{x_1, y_2, z_3\}$, $\{x_1, z_2, z_3\}$, $\{x_1, z_2, x_3\}$, $\{y_1, z_2, x_3\}$, $\{y_1, x_2, x_3\}$ und $\{y_1, x_2, y_3\}$ sind.

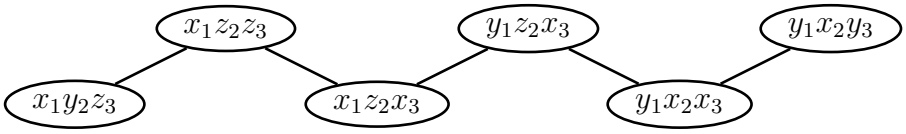


Abbildung 2.15: Beispiel: Baumzerlegung in Cliquen

Nun kommen wir für uns zu der zentralen Charakterisierung von Durchschnittgraphen von Bäumen (die ja als phylogenetische Bäume einer perfekten Phylogenie bei uns im Mittelpunkt des Interesses stehen).

Theorem 2.10 *Sei $G = (V, E)$ ein Graph, dann sind folgende Aussagen äquivalent:*

i) G ist chordal.

ii) G ist ein Durchschnittsgraph von Bäumen.

iii) G besitzt eine Baumzerlegung in Cliquen.

Bevor wir zum Beweis diese Satzes kommen, benötigen wir noch einige Definitionen und Lemmata.

Definition 2.11 *Sei $G = (V, E)$ ein Graph und $a, b \in V$ mit $\{a, b\} \notin E$. Ein* Knoten-Separator *(oder kurz* Separator*), der a und b trennt, ist eine Knotenmenge $S \subseteq V \setminus \{a, b\}$ mit der Eigenschaft, dass a und b in $G[V \setminus S]$ nicht durch einen Pfad verbunden sind.*

Damit können wir das folgende fundamentale Lemma formulieren.

Lemma 2.12 *Ein Graph $G = (V, E)$ ist genau dann chordal, wenn für beliebige $a, b \in V$ mit $\{a, b\} \notin E$ jeder inklusionsminimaler Knoten-Separator $S \subseteq V \setminus \{a, b\}$, der a und b trennt, ein vollständiger Teilgraph von G ist.*

Beweis: $\Rightarrow$: Sei S ein beliebiger inklusionsminimaler Knoten-Separator für beliebige Knoten a und b mit $\{a, b\} \notin E(G)$. Seien weiter V_a bzw. V_b die Zusammenhangskomponenten von $G[V \setminus S]$ mit $a \in V_a$ bzw. $b \in V_b$. Seien weiter $x, y \in S$ beliebige Knoten und $P_a = (x, a_1, \ldots, a_i, \ldots, a_m, y)$ bzw. $P_b = (y, b_1, \ldots, b_j, \ldots, b_n, x)$ zwei Pfade jeweils in V_a bzw. V_b mit $a_i = a$ bzw. $b_j = b$ (d.h. $a_1, \ldots, a_m \in V_a$ bzw. $b_1, \ldots, b_n \in V_b$), die jeweils x mit y verbinden. Man überlege sich, dass aufgrund des inklusionsminimaler Knoten-Separator für a und b diese Pfade existieren müssen (auch hier setzen wir wieder voraus, dass G zusammenhängend ist).

Da es nun Pfade P_a bzw. P_b gibt die x mit y verbinden und außer x und y nur Knoten aus V_a bzw. V_b enthalten, gibt es auch kürzeste Pfade $P'_a = (x, a'_1, \ldots a'_{m'}, y)$ bzw. $P'_b = (y, b'_1, \ldots b'_{n'}, x)$ mit dieser Eigenschaft. Dann ist $C := P'_a \cdot P'_b$ ein einfacher Kreis. Wenn $m' = 0$ oder $n' = 0$ ist, dann muss $\{x, y\} \in E(G)$ sein. Andernfalls hat C als Kreis mindestens die Länge 4. Da G chordal ist, muss C eine Sehne enthalten. Diese Sehne kann nicht $\{a'_i, a'_j\}$ bzw. $\{b'_i, b'_j\}$ sein, da dies der Minimalität der Pfadlängen von P'_a bzw. P'_b widerspricht (ebenso kann es nicht $\{a'_i, x\}$, $\{a'_i, y\}$ bzw. $\{b'_i, x\}$, $\{b'_i, y\}$ sein). Da S ein Separator ist, kann $\{a'_i, b'_j\}$ auch keine Sehne sein. Also muss $\{x, y\}$ die geforderte Sehne sein. Somit sind zwei beliebige Knoten $x, y \in S$ adjazent.

$\Leftarrow$: Sei $C = (a, x, b, y_1, \ldots, y_m)$ für $m \geq 1$ ein Kreis der Länge mindestens 4 in G. Ist $\{a, b\} \in E(G)$, dann ist nichts zu zeigen. Sei also $\{a, b\} \notin E(G)$ und S ein inklusionsminimaler Knoten-Separator für a und b. Dann ist nach Definition $\{x, y_i\} \subset S$ für ein $i \in [1 : m]$. Nach Voraussetzung ist dann $\{x, y_i\} \in E(G)$. Somit enthält jeder Kreis eine Sehne und der Graph ist chordal. ■

Kommen wir nun zu einer weiteren wichtigen Definition.

Definition 2.13 *Ein Knoten $v \in V(G)$ eines Graphen G heißt* simplizial*, wenn $N(v)$ ein vollständiger Teilgraph von G ist, wobei $N(v) = \{w \in V : \{v, w\} \in E\}$ die* Nachbarschaft *von v bezeichnet.*

Das folgende Lemma wird zentral für den Beweis unseres Satzes sein.

13.06.24

Lemma 2.14 *Jeder chordale Graph besitzt mindestens einen simplizialen Knoten. Jeder nicht vollständige chordale Graph besitzt mindestens zwei nicht adjazente simpliziale Knoten.*

Beweis: Wir beweisen den Satz durch Induktion nach der Knotenanzahl $n = |V|$.

Induktionsanfang ($n = 1$): Dann ist der Graph vollständig und die Behauptung ist trivialerweise erfüllt.

Induktionsschritt ($\rightarrow n$): Ist G ein vollständiger Graph, so ist jeder Knoten simplizial.

Sei also G ein unvollständiger Graph und seien daher $x, y \in V$ mit $\{x, y\} \notin E$. Sei weiter S ein inklusionsminimaler Separator, der x und y trennt. Da G chordal ist, ist nach dem vorherigen Lemma $G[S]$ vollständig. In $G[V \setminus S]$ seien $X, Y \subseteq V \setminus S$ die Zusammenhangskomponenten, die x bzw. y enthalten.

Sei zuerst $G[X \cup S]$ vollständig. Dann ist jeder Knoten aus $X \cup S$ simplizial in $G[X \cup S]$. Also gibt es insbesondere einen Knoten $x' \in X$ der simplizial im Graphen $G[X \cup S]$ ist. Da im Graphen G nach Eigenschaft des Separators S gilt, dass $N(x') \subseteq X \cup S$, ist x' auch simplizial in G.

Andernfalls ist $G[X \cup S]$ kein vollständiger Graph. Da $|Y| \geq 1$ ist, gilt $|X \cup S| < |V|$ und somit gilt nach Induktionsvoraussetzung, dass $G[X \cup S]$ zwei nichtadjazente simpliziale Knoten $x', z' \in X \cup S$ besitzt. Da nach dem vorherigen Lemma $G[S]$ ein vollständiger Teilgraph von G ist, muss also zumindest $x' \in X$ gelten. Analog können wir einen simplizialen Knoten $y' \in Y$ finden. Weil S ein Separator ist, muss $N(x') \subseteq X \cup S$ bzw. $N(y') \subseteq Y \cup S$ gelten und somit sind x' und y' zum einen nicht adjazent und zum anderen simplizial in G. ■

Mit Hilfe dieser Lemmata können wir nun den zentralen Satz 2.10 beweisen.

Beweis von Theorem 2.10: Wir führen den Beweis nur für den Fall, dass der Graph G zusammenhängend ist, ansonsten werden die Zusammenhangskomponenten einzeln betrachtet. Anschließend muss man nur die Bäume der Baumdarstellung bzw. Baumzerlegung mit zusätzlichen Kanten jeweils zu einem Baum verbinden

***iii)* $\Rightarrow$ *ii)*:** Der Baum T der Baumzerlegung wird als Baum für die Baumdarstellung verwendet. Für jeden Knoten des Graphen G werden alle Knoten des Baumes in den Teilbaum aufgenommen, deren Clique den entsprechenden Knoten enthält. Aufgrund der Baumzerlegung erhalten wir tatsächlich Teilbäume. Zwei Knoten des zugehörigen

Graphen sind ja genau dann adjazent, wenn sie in einer Clique enthalten sind und somit überlappen sich genau dann die zugehörigen Teilbäume.

Formal kann man aus der Baumzerlegung in Cliquen $T = (W, F)$ von G eine Baumdarstellung wie folgt definieren. Der Gesamtbaum ist T und zu jedem Knoten $v \in V$ ist der zugeordnete Teilbaum gerade $T[C_v]$, wobei wieder $C_v = \{C \in W : v \in C\}$ ist.

ii) ⇒ i): Wir betrachten einen einfachen Kreis $C = \{v_{i_1}, \ldots, v_{i_k}\}$ der Länge $k \geq 4$ im Graphen und die zugehörigen Teilbäume $\{T_{i_1}, \ldots, T_{i_k}\}$ in der Baumdarstellung. Wir wurzeln den Baum der Baumdarstellung beliebig an einem Blatt und nehmen ohne Beschränkung der Allgemeinheit an, dass wir bei einer Tiefensuche zuerst auf den Teilbaum T_{i_1} treffen. Die gemäß des Kreises mit diesen Baum benachbarten Teilbäume seien T_{i_k} und T_{i_2}. Überlappen sich T_{i_2} und T_{i_k}, so haben wir die gesuchte Sehne gefunden.

Seien also T_{i_2} und T_{i_k} in T disjunkt. Dann sind T_{i_2} und T_{i_k} durch einen einfachen Pfad verbunden (ohne T_{i_1} zu treffen), der keinen Knoten aus T_{i_2} und T_{i_k} enthält und vollständig unterhalb von T_{i_1}, und somit auch in T, verläuft.

Um den Kreis zwischen T_{i_2} und T_{i_k} durch überlappende Teilbäume schließen zu können, muss einer der Teilbäume, die zum Kreis C korrespondieren, sich aufgrund der Baumstruktur mit T_{i_1} überlappen und wir erhalten die gewünschte Sehne.

i) ⇒ iii): Wir führen den Beweis durch Induktion über $n = |V|$:

Induktionsanfang ($n = 1$): Die Aussage ist trivial.

Induktionsschritt ($n \to n + 1$): Sei $G = (V, E)$ ein zusammenhängender chordaler Graph auf $n + 1$ Knoten. Nach Lemma 2.14 besitzt dieser einen simplizialen Knoten $v \in V$. Da v simplizial ist, muss $N(v)$ ein vollständiger Teilgraph von G sein. Dann muss $G' := G[V \setminus \{v\}]$ zusammenhängend sein. Nach Induktionsvoraussetzung existiert für G' eine Baumzerlegung $T' = (W', F')$ in Cliquen.

Ist $N(v)$ eine Clique in G', dann ist $N(v)$ keine Clique in G, aber $N(v) \cup \{v\}$ ist eine Clique in G. Wir ersetzen in T den Knoten $N(v)$ durch $N(v) \cup \{v\}$ und erhalten somit die Baumzerlegung in Cliquen für G.

Ist $N(v)$ eine echte Teilmenge einer Clique C' in G', dann sind sowohl C' als auch $C := N(v) \cup \{v\}$ Cliquen in G. Wir erweitern T' um den Knoten C und verbinden ihn mit dem Knoten C' in T' und wir erhalten somit die Baumzerlegung in Cliquen $T = (W' \cup \{C\}, F' \cup \{C, C'\})$ für G. ■

Somit müssen wir aus der gegeben Merkmalsmatrix nur noch einen chordalen Graphen konstruieren. Aus diesem können wir nach dem Satz eine Baumzerlegung in

Cliquen konstruieren, die den gesuchten phylogenetischen Baum liefert. Für diesen chordalen Graph dient der so genannte State-Intersection-Graph als Grundlage.

Definition 2.15 *Sei M eine $n \times m$-Merkmalsmatrix M mit r Zuständen. Der* State-Intersection-Graph $G(M) = (V, E)$ *ist wie folgt definiert:*

- $V = \{(j,k) : \exists i \in [1:n] : j \in [1:m] \wedge k \in [1:r] \wedge M(i,j) = k\}$,
- $E = \{\{(j,k),(j',k')\} : \exists i \in [1:n] : M(i,j) = k \wedge M(i,j') = k'\}$.

Im Folgenden werden isolierte Knoten im State-Intersection-Graph nicht berücksichtigt, da diese keine Rolle spielen und auch an der Triangulierung des Graphen nichts ändern.

In Abbildung 2.16 ist der State-Intersection-Graph für die Merkmalsmatrix unseres Eingangsbeispiels für einen phylogenetischen Baum angegeben, wobei $(j,k) \hat{=} k_j$. Beispielsweise ist der isolierte Knoten $(1,z) \hat{=} z_1$ dort nicht angegeben (wobei dieser nach der Definition 2.15 auch nicht existiert).

	c_1	c_2	c_3
A	x_1	z_2	z_3
B	x_1	y_2	z_3
C	y_1	z_2	x_3
D	y_1	x_2	y_3
E	y_1	x_2	x_3
F	x_1	z_2	x_3

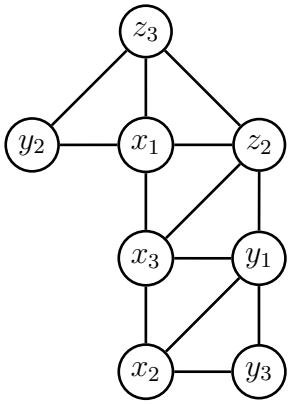


Abbildung 2.16: Beispiel: State-Intersection-Graph

In der Regel sind im State-Intersection-Graphen der Merkmalsmatrix nicht alle Kanten des Durchschnittsgraphen des zugehörigen phylogenetischen Baumes vorhanden. Somit muss der State-Intersection-Graphen der Merkmalsmatrix zunächst erst noch trianguliert werden. Dabei ist zu beachten, dass es keine Kanten zwischen verschiedenen Zuständen eines Merkmals geben darf, da sich diese im phylogenetischen Baum ja auch nicht überlappen können: In einem Knoten des phylogenetischen Baumes besitzt jedes Merkmal ja auch immer nur einen und nicht mehrere Zustände.

Somit ist der gegebene Eingabegraph, der State-Intersection-Graph der Merkmalsmatrix, gefärbt: Knoten, die verschiedene Zustände desselben Merkmals darstellen,

sind in derselben Farbe gefärbt. Für die Definition einer zulässigen Färbung verweisen wir auf Definition 1.67 auf Seite 74). Die Anzahl der Farben entspricht also der Anzahl der Merkmale. Daher geben wir nun noch folgende Definitionen an.

Definition 2.16 *Sei $G = (V, E)$ ein Graph. Ein Graph $G' = (V, E') \supseteq G$ heißt eine* Triangulierung *von G, wenn G' trianguliert ist.*

Definition 2.17 *Sei G ein Graph und c eine zulässige Färbung für G. G heißt c-*triangulierbar*, wenn G eine Triangulierung G' besitzt und c auch für G' eine zulässige Färbung ist.*

Damit und dem vorherigen Satz folgt unmittelbar der nächste Satz.

Theorem 2.18 *Eine $n \times m$-Merkmalsmatrix M besitzt genau dann einen phylogenetischen Baum, wenn der zugehörige State-Intersection-Graph $G(M)$ mit seiner zulässigen Färbung c auch c-triangulierbar ist.*

Leider ist das Problem der c-Triangulierbarkeit im Allgemeinen ein $\mathcal{NP}$-hartes Problem. Somit können wir im Allgemeinen nicht auf eine effiziente Lösung für die perfekte Phylogenie hoffen. Für feste Parameter m oder r gibt es wiederum fixed parameter solutions. Eine Übersicht der Komplexitäten für die verschiedenen Varianten ist in Abbildung 2.17 angegeben. Im folgenden Abschnitt werden auf die perfekte

	$m \leq 3$	m fest	m beliebig
$r = 2$			$O(nm)$
$r = 3$			$O(\min\{nm^2, n^2m\})$
$r = 4$			$O(\min\{nm^2, n^2m\})$
r fest			$O(2^{2r}m^2n)$
r bel.	$O(n)$	$O((rm)^{m+1} + nm^2)$	NPC

Abbildung 2.17: Übersicht: Komplexität perfekter Phylogenie

Phylogenie mit zwei Merkmalen noch genauer eingehen. Für Details verweisen wir für $m = 3$ auf die Originalarbeit von Kannan und Warnow aus dem Jahre 1992, für $r \in \{3, 4\}$ auf die Originalarbeit von Kannan und Warnow aus dem Jahre 1994, für festes r auf die Arbeit von Kannan und Warnow aus dem Jahre 1997, und für festes m auf die Arbeit von McMorris, Warnow und Wimer.

Für den allgemeinen Fall kehren wir noch einmal zu unserem Beispiel zurück. Der State-Intersection-Graph unserer Merkmalsmatrix in Abbildung 2.16 ist bereits c-trianguliert. Wie man leicht sieht, bestehen alle Cliquen dieses Graphen aus je drei

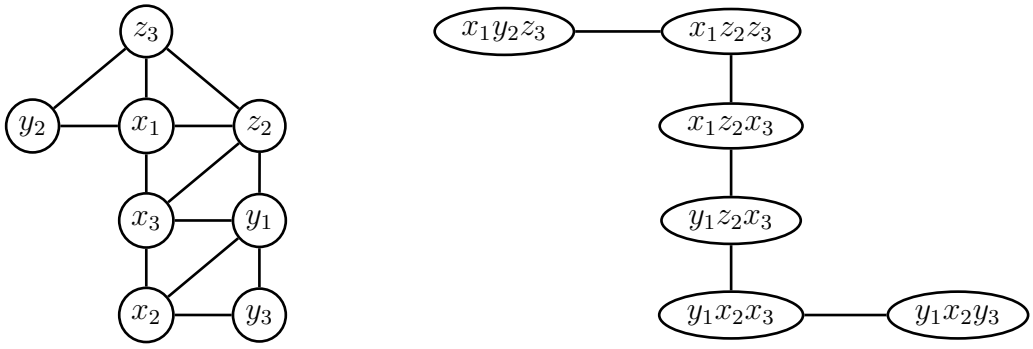


Abbildung 2.18: Beispiel: Baumzerlegung in Cliquen des c-triangulierten State-Intersection-Graphen

Knoten. Es gilt allgemein, dass jedes Taxon im State-Intersection-Graph eine m-Clique induziert. Allerdings muss nicht jeder triangulierte State-Intersection-Graph nur m-Cliquen enthalten, er kann auch kleinere enthalten

Die Baumzerlegung in Cliquen, wie sie in Abbildung 2.18 rechts angegeben ist, des State-Intersection-Graphen, wie in Abbildung 2.18 links angegeben, kann leicht konstruiert werden. Aus dieser Baumzerlegung in Cliquen erhalten wir unmittelbar die Baumdarstellung, wie sie in Abbildung 2.19 links angegeben ist.

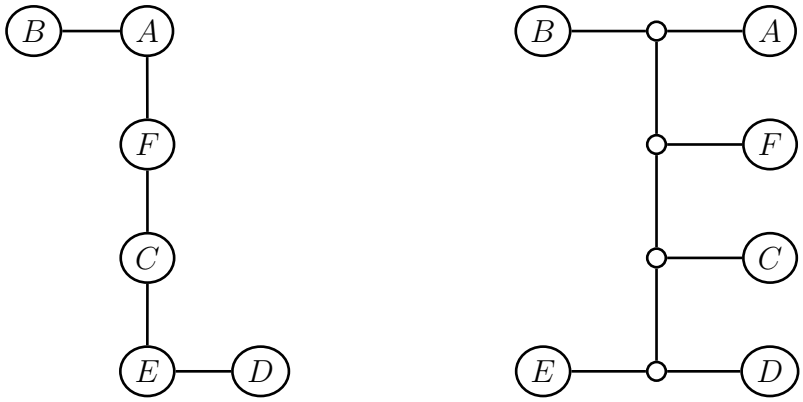


Abbildung 2.19: Beispiel: Baumdarstellung des c-triangulierten State-Intersection-Graphen

Wir haben hier bereits den zugehörigen phylogenetischen Baum angegeben, wobei wir das Zustandstripel durch das zugehörige Taxon angegeben haben, sofern möglich (hier im Beispiel entspricht jeder Knoten einem Taxon, was in der Regel nicht der Fall sein muss). Wir haben jetzt hier definitionswidrig einen phylogenetischen Baum konstruiert, in dem auch die inneren Knoten mit Taxa markiert sind. Um nun einen phylogenetischen Baum gemäß der Definition zu erhalten, werden die Taxa, die innere Knoten markieren, in daran adjazente Blätter ausgelagert, wie im rechten

Teil von Abbildung 2.19 zu sehen ist. Es sei dem Leser überlassen zu verifizieren, dass der rekonstruierte phylogenetische Baum in Abbildung 2.19 rechts isomorph zum ursprünglichen phylogenetischen Bild in Abbildung 2.11 auf Seite 103 ist.

Wir wollen jetzt noch ein zweites Beispiel betrachten. In Abbildung 2.20 ist rechts der State-Intersection-Graph für die links angegebene Merkmalsmatrix angegeben. In diesem Beispiel muss also noch die Kante $\{x_2, x_3\}$ zur Triangulierung eingezogen werden. Die Kante $\{x_1, y_1\}$ würde zwar auch zu einer Triangulierung des State-Intersection-Graphen führen, jedoch würde die Färbung, die durch die Merkmale induziert wird, dabei zerstört.

	c_1	c_2	c_3
A	x_1	x_2	y_3
B	x_1	y_2	x_3
C	y_1	x_2	z_3
D	y_1	z_2	x_3

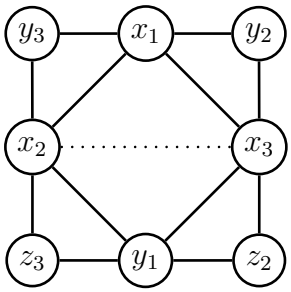


Abbildung 2.20: Beispiel: State-Intersection-Graph

Auch hier erkennt man nach der Triangulierung, dass alle Cliquen des triangulierten State-Intersection-Graphen aus jeweils drei Knoten bestehen. Somit ergibt sich die Baumzerlegung in Cliquen, die links in Abbildung 2.21 angegeben ist. Vergleicht man die Merkmale der einzelnen Knoten mit der ursprünglichen Merkmalsmatrix, so erhält man sofort den zugehörigen phylogenetischen Baum, der rechts in der Abbildung 2.21 angegeben ist.

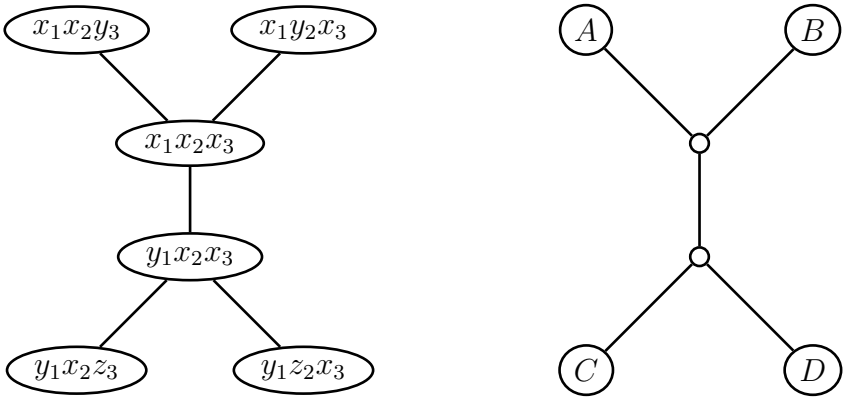


Abbildung 2.21: Beispiel: Baumzerlegung in Cliquen und zugehöriger phylogenetischer Baum

2.3.2 Perfekte Phylogenien mit zwei Merkmalen

Nun betrachten wir noch den Spezialfall, dass die Merkmalsmatrix nur zwei verschiedene Merkmale besitzt, die aber jeweils beliebig viele Zustände annehmen dürfen.

Theorem 2.19 *Eine $n \times 2$-Merkmalsmatrix M besitzt genau dann eine perfekte Phylogenie, wenn der zugehörige State-Intersection Graph $G(M)$ azyklisch ist.*

Beweis: $\Leftarrow$**:** Wenn der State-Intersection-Graph einer $n \times 2$-Merkmalsmatrix azyklisch ist, dann handelt es sich um einen Wald und dieser ist bereits trianguliert. Da ein Wald offensichtlich ein bipartiter Graph ist, ist dieser bereits auch schon zulässig auch 2-gefärbt. Mit dem Satz 2.18 folgt dann die Behauptung.

$\Rightarrow$**:** Mit Satz 2.18 folgt sofort, dass der zugehörige State-Intersection-Graph eine 2-Triangulierung besitzt. Somit ist dieser State-Intersection-Graph ein bipartiter Graph.

Für einen Widerspruchsbeweis betrachten wir jetzt einen kürzesten Kreis der Länge $k \geq 4$ in einem solchen bipartiten Graphen, der dann eine gerade Länge haben muss. Da der Graph ja chordal ist, muss dieser Graph eine Sehne dieses Kreises enthalten. Die Endpunkte müssen dabei eine unterschiedliche Farbe besitzen, da die Färbung des Graphen ja nach Voraussetzung eine 2-Färbung ist (somit kann der Kreis auch nicht die Länge 4 haben). Dann können wir den Kreis jedoch an dieser Kante in zwei Kreise der Länge r bzw. s mit $r + s = k + 2$ aufteilen. Somit muss mindestens einer der beiden Kreise eine Länge kürzer als k besitzen, was offensichtlich unserer Annahme widerspricht. ■

Somit können wir einfach einen Algorithmus zur Konstruktion einer perfekten Phylogenie für zwei Merkmale angeben, da sich Azyklität von Graphen in Zeit $O(|V(G)|)$ testen lässt. Für einen azyklischen Graphen $G = (V, E)$ gilt, dass $|E| < |V|$ ist. Somit können wir bei Graphen mit mindestens $|V|$ Kanten sofort sagen, dass er nicht azyklisch ist. Ansonsten können wir mit Hilfe einer Tiefen- oder Breitensuche in linearer Zeit feststellen, ob der Graph azyklisch ist.

Ist der Graph azyklisch, so ist er trivialerweise trianguliert und wir können gemäß dem Beweis von Satz 2.10 sofort eine Baumzerlegung in Cliquen (das sind hier 2-Cliquen, also die Kanten des Waldes) konstruieren und somit gleichzeitig die Baumdarstellung des Graphen angeben, die ja genau dem gesuchten phylogenetischen Baum entspricht.

Theorem 2.20 *Eine perfekte Phylogenie für eine $n \times 2$-Merkmalsmatrix kann in Zeit $O(n)$ bestimmt werden, sofern überhaupt eine existiert.*

2.3.3 Minimale Anzahl von Merkmalen perfekter Phylogenien

Am Ende dieses Abschnitts wollen wir noch festhalten, dass sich jeder binäre phylogenetische Baum mathematisch durch 5 verschiedene Merkmale eindeutig beschreiben lässt.

Theorem 2.21 *Jeder (binäre) phylogenetische Baum lässt sich durch eine $n \times 5$-Merkmalsmatrix eindeutig beschreiben.*

Für den Beweis verweisen wir auf die Originalarbeit von Semple und Steel aus dem Jahr 2002. Wir merken hier nur noch an, dass dabei die Anzahl der Zustände sehr groß werden kann. Daraus folgt natürlich nicht, dass sich jeder evolutionäre Baum aus der Biologie aus 5 verschiedenen Merkmalen rekonstruieren lässt (insbesondere dann nicht, wenn die Anzahl der Zustände sehr klein ist). Da es jedoch mathematisch prinzipiell möglich ist, besteht die Hoffnung, dass es prinzipiell möglich ist, dass man evolutionäre Bäume aus wenigen Merkmalen mit vielen verschiedenen Zuständen (also beispielsweise längeren DNS-Sequenzen) rekonstruieren kann.

Mittlerweile gibt es noch eine Verbesserung, die besagt, dass 4 Merkmale ausreichen. Man kann sich auch überlegen, dass 3 Merkmale zu wenig sind. Für Details verweisen wir auf die Originalliteratur von Huber, Moulton und Steel aus dem Jahr 2003.

Theorem 2.22 *Jeder (binäre) phylogenetische Baum lässt sich durch eine $n \times 4$-Merkmalsmatrix eindeutig beschreiben.*

2.4 Ultrametriken und ultrametrische Bäume

Wir wollen uns nun mit distanzbasierten Methoden beschäftigen. Dazu stellen wir zuerst einige schöne und einfache Charakterisierungen vor, ob eine gegebene Distanzmatrix einen phylogenetischen Baum besitzt oder nicht.

2.4.1 Metriken und Ultrametriken

Zuerst müssen wir noch ein paar Eigenschaften von Distanzen wiederholen und einige hier nützliche zusätzliche Definitionen angeben. Zuerst wiederholen wir die Definition einer Metrik.

Definition 2.23 *Eine Funktion $d : M^2 \to \mathbb{R}_+$ heißt* Metrik *auf M, wenn gilt:*

(M1) $\forall x, y \in M : d(x, y) = 0 \Leftrightarrow x = y$ *(Definitheit),*

(M2) $\forall x, y \in M : d(x, y) = d(y, x)$ *(Symmetrie),*

(M3) $\forall x, y, z \in M : d(x, z) \leq d(x, y) + d(y, z)$ *(Dreiecksungleichung).*

Im Folgenden werden wir auch die folgende verschärfte Variante der Dreiecksungleichung benötigen.

Definition 2.24 *Eine Metrik d auf M heißt* Ultrametrik*, wenn zusätzlich die so genannte* ultrametrische Dreiecksungleichung *gilt:*

$$\forall x, y, z \in M : d(x, z) \leq \max\{d(x, y), d(y, z)\}.$$

Eine andere Charakterisierung der ultrametrischen Ungleichung wird uns im Folgenden aus beweistechnischen Gründen nützlich sein.

Lemma 2.25 *Sei d eine Ultrametrik auf M. Dann sind für alle $x, y, z \in M$ die beiden größten Zahlen aus $d(x, y)$, $d(y, z)$ und $d(x, z)$ gleich.*

Beweis: Es gilt also die ultrametrische Dreiecksungleichung für alle $x, y, z \in M$:

$$d(x, z) \leq \max\{d(x, y), d(y, z)\}.$$

Ist $d(x, y) = d(y, z)$, dann ist nichts zu zeigen.

Sei nun also $d(x, y) < d(y, z)$. Dann ist auch $d(x, z) \leq d(y, z)$. Aufgrund der ultrametrischen Ungleichung gilt ebenfalls:

$$d(y, z) \leq \max\{d(y, x), d(x, z)\} = d(x, z).$$

Die Gleichung in der letzten Ungleichung folgt aus der oben gemachten Annahme, dass $d(y, z) > d(y, x)$. Zusammen gilt also $d(x, z) \leq d(y, z) \leq d(x, z)$. Also gilt $d(x, z) = d(y, z) > d(x, y)$.

Der Fall $d(x, y) < d(y, z)$ ist analog und das Lemma ist bewiesen. ■

Nun zeigen wir auch noch die umgekehrte Richtung.

Lemma 2.26 *Sei $d : M^2 \to \mathbb{R}_+$, wobei für alle $x, y \in M$ genau dann $d(x, y) = 0$ gilt, wenn $x = y$. Weiter gelte, dass für alle $x, y, z \in M$ die beiden größten Zahlen aus $d(x, y)$, $d(y, z)$ und $d(x, z)$ gleich sind. Dann ist d eine Ultrametrik.*

Beweis: Die Definitheit (M1) gilt nach Voraussetzung.

Für die Symmetrie (M2) betrachten wir beliebige $x, y \in M$. Aus der Voraussetzung folgt mit $z = x$, dass von $d(x, y)$, $d(y, x)$ und $d(x, x)$ die beiden größten Werte gleich sind. Da nach Voraussetzung $d(x, x) = 0$ sowie $d(x, y) \geq 0$ und $d(y, x) \geq 0$ gilt, folgt, dass $d(x, y)$ und $d(y, x)$ die beiden größten Werte sind und somit nach Voraussetzung gleich sein müssen. Also gilt $d(x, y) = d(y, x)$ für alle $x, y \in M$ und die Symmetrie ist gezeigt.

Für die ultrametrische Dreiecksungleichung ist Folgendes zu zeigen:

$$\forall x, y, z \in M : d(x, z) \leq \max\{d(x, y), d(y, z)\}.$$

Wir unterscheiden drei Fälle, je nachdem, welche beiden Werte der drei Distanzen die größten sind und somit nach Voraussetzung gleich sind.

Fall 1 ($d(x, z) \leq d(x, y) = d(y, z)$): Die Behauptung lässt sich sofort verifizieren.

Fall 2 ($d(y, z) \leq d(x, y) = d(x, z)$): Die Behauptung lässt sich sofort verifizieren.

Fall 3 ($d(x, y) \leq d(x, z) = d(y, z)$): Die Behauptung lässt sich sofort verifizieren.

Damit ist der Beweis abgeschlossen. ■

Da die beiden Lemmata bewiesen haben, dass in einer Ultrametrik von drei Abständen zwischen drei Punkten die beiden größten gleich sind, nennt man diese Eigenschaft auch *3-Punkte-Bedingung*.

18.06.24

2.4.2 Ultrametrische Bäume

Zunächst einmal definieren wir spezielle evolutionäre Bäume, für die sich, wie wir sehen werden, sehr effizient die gewünschten Bäume konstruieren lassen. Zuerst definieren wir hierfür noch so genannte Distanzmatrizen, aus denen wir im Folgenden die evolutionären Bäumen konstruieren wollen.

Definition 2.27 *Sei $D = (d_{i,j})$ eine symmetrische $n \times n$-Matrix mit $d_{i,i} = 0$ und $d_{i,j} > 0$ für alle $i \neq j \in [1 : n]$. Dann heißt D eine* Distanzmatrix.

Bevor wir nun die speziellen evolutionären Bäume definieren können, benötigen wir noch den Begriff des niedrigsten gemeinsamen Vorfahren von zwei Knoten in einem Baum.

Definition 2.28 *Sei $T = (V, E)$ ein gewurzelter Baum und seien $v, w \in V$ zwei Knoten von T. Der* niedrigste gemeinsame Vorfahr *von v und w, bezeichnet mit* $\mathrm{lca}(v, w)$ *(engl.* lowest common ancestor*, auch* least common ancestor *oder* nearest common ancestor*), ist der Knoten $u \in V$, so dass u sowohl ein Vorfahr von v als auch von w ist und es keinen echten Nachfahren von u gibt, der ebenfalls ein Vorfahr von v und w ist.*

Mit Hilfe des Begriffs des niedrigsten gemeinsamen Vorfahren können wir jetzt ultrametrische Bäume definieren.

Definition 2.29 *Sei D eine $n \times n$-Distanzmatrix. Ein* (strenger) ultrametrischer Baum *T für D ist ein Baum $T = T(D)$ mit*

1. *T besitzt n Blätter, die bijektiv mit $[1 : n]$ markiert sind;*
2. *Jeder innere Knoten von T ist mit Werten aus D markiert und besitzt mindestens 2 Kinder.*
3. *Entlang eines jeden Pfades von der Wurzel von T zu einem Blatt ist die Folge der Markierungen an den inneren Knoten (streng) monoton fallend;*
4. *Für je zwei Blätter i und j von T ist die Markierung des niedrigsten gemeinsamen Vorfahren gleich d_{ij}.*

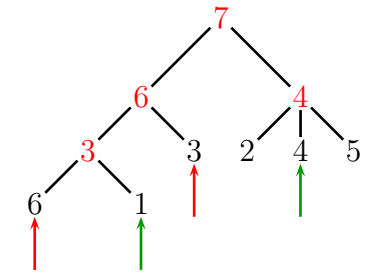
Besitzt D einen (streng) ultrametrischen Baum, so heißt D auch (streng) ultrametrisch.

In Folgenden werden wir hauptsächlich strenge ultrametrische Bäume betrachten. Wir werden jedoch der Einfachheit wegen immer von ultrametrischen Bäume sprechen. In Abbildung 2.22 ist ein Beispiel für eine 6×6-Matrix angegeben, die einen ultrametrischen Baum besitzt.

Wir wollen an dieser Stelle noch einige Bemerkungen zu ultrametrischen Bäumen festhalten.

- Nicht jede Matrix D besitzt einen ultrametrischen Baum. Dies folgt aus der Tatsache, dass jeder Baum, in dem jeder innere Knoten mindestens zwei Kinder besitzt, maximal $n - 1$ innere Knoten besitzen kann. Dies gilt daher insbesondere für ultrametrische Bäume. Also können in Matrizen, die einen ultrametrischen Baum besitzen, nur $n - 1$ von Null verschiedene Werte auftreten.

D	1	2	3	4	5	6
1	0	7	6	7	7	3
2		0	7	4	4	7
3			0	7	7	6
4				0	4	7
5					0	7
6						0



$\text{lca}(1,4) \rightsquigarrow 7 \quad \text{lca}(3,6) \rightsquigarrow 6$

Abbildung 2.22: Beispiel: ultrametrischer Baum

- Der Baum $T(D)$ für D heißt auch *kompakte Darstellung* von D, da sich eine Matrix mit n^2 Einträgen durch einen Baum der Größe $O(n)$ darstellen lässt.
- Die Markierung an den inneren Knoten können als Zeitspanne in die Vergangenheit interpretiert werden. Vor diesem Zeitraum haben sich die Spezies bzw. Taxa, die in verschiedenen Teilbäumen auftreten, auseinander entwickelt.

Unser Ziel wird es jetzt sein, festzustellen, ob eine gegebene Distanzmatrix einen ultrametrischen Baum besitzt oder nicht. Zuerst einmal überlegen wir uns, dass es es sehr viele gewurzelte Bäume mit n Blättern gibt. Somit scheidet ein einfaches Ausprobieren aller möglichen Bäume aus.

Lemma 2.30 *Die Anzahl der ungeordneten binären gewurzelten Bäume mit n verschieden markierten Blättern beträgt*

$$\prod_{i=2}^{n}(2i-3) = \frac{(2n-3)!}{2^{n-2}\cdot(n-2)!}.$$

Um ein besseres Gefühl für dieses Anzahl zu bekommen, rechnet man leicht nach, dass

$$\prod_{i=2}^{n}(2i-3) \geq (n-1)! \geq 2^{n-2}$$

gilt. Eine bessere Abschätzung lässt sich natürlich mit Hilfe der Stirlingschen Formel bekommen: $\Theta((\frac{2n}{e})^{n-1})$.

Beweis: Wir führen den Beweis durch vollständige Induktion über n.

Induktionsanfang ($n = 2$): Hierfür gilt die Formel offensichtlich, das es genau einem Baum mit zwei markierten Blättern gibt.

Induktionsschritt ($n-1 \to n \geq 3$): Sei T ein ungeordneter binärer gewurzelter Baum mit n Blättern. Der Einfachheit halber nehmen wir im Folgenden an, dass unser Baum noch eine Superwurzel besitzt, deren einziges Kind die ursprüngliche Wurzel von T ist.

Wir entfernen jetzt das Blatt v mit der Markierung n. Der Elter w davon hat jetzt nur noch ein Kind und wir entfernen es ebenfalls. Dazu wird das andere Kind von w jetzt ein Kind des Elters von w (anstatt von w). Den so konstruierten Baum nennen wir T'. Wir merken noch an, dass genau eine Kante eine „Erinnerung“ an das Entfernen von v und w hat. Falls w die Wurzel war, so bleibt die Superwurzel im Baum und die Kante von der Superwurzel hat sich das Entfernen „gemerkt“.

Wir stellen fest, dass T' ein ungeordneter binärer gewurzelter Baum ist. Davon gibt es nach Induktionsvoraussetzung $\prod_{i=2}^{n-1}(2i-3)$ viele. Darin kann an jeder Kante v mit seinem Elter w entfernt worden sein.

Wie viele Kanten besitzt ein binärer gewurzelter Baum mit $n-1$ Blättern? Ein binärer gewurzelter Baum mit $n-1$ Blättern besitzt genau $n-2$ innere Knoten plus die von uns hinzugedachte Superwurzel. Somit besitzt der Baum

$$(n-1)+(n-2)+1 = 2n-2$$

Knoten. Da in einem Baum die Anzahl der Kanten um eines niedriger ist als die Anzahl der Knoten, besitzt unser Baum $2n-3$ Kanten, die sich an einen Verlust eines Blattes „erinnern“ können. Somit ist die Gesamtanzahl der ungeordneten binären gewurzelten Bäume mit n Blättern genau

$$(2n-3)\cdot\prod_{i=2}^{n-1}(2i-3) = \prod_{i=2}^{n}(2i-3)$$

und der Induktionschluss ist vollzogen. ■

Wir fügen noch eine ähnliche Behauptung für die Anzahl ungewurzelter Bäume an. Der Beweis ist im Wesentlichen ähnlich zu dem vorherigen.

Lemma 2.31 *Die Anzahl der ungewurzelten (freien) Bäume mit n verschieden markierten Blättern, deren innere Knoten jeweils den Grad 3 besitzen, beträgt*

$$\prod_{i=3}^{n}(2i-5) = \frac{(2n-5)!}{2^{n-3}\cdot(n-3)!}.$$

Wiederum kann man mit der Stirlingschen Formel eine bessere Abschätzung erhalten: $\Theta((\frac{2n}{e})^{n-2})$.

2.4.3 Charakterisierung ultrametrischer Bäume

Bevor wir zu einer weiteren Charakterisierung ultrametrischer Matrizen kommen, benötigen wir noch eine kurze Definition, die Distanzmatrizen und Metriken in Beziehung setzen.

Definition 2.32 *Eine $n \times n$-Distanzmatrix D* induziert *eine Metrik bzw. Ultrametrik auf $[1:n]$, wenn die Funktion $d : [1:n]^2 \to \mathbb{R}_+$ mit $d(x,y) = D_{x,y}$ eine Metrik bzw. Ultrametrik auf $[1:n]$ ist.*

Wir geben jetzt eine weitere Charakterisierung ultrametrischer Matrizen an, deren Beweis sogar einen effizienten Algorithmus zur Konstruktion ultrametrischer Bäume erlaubt.

Theorem 2.33 *Eine $n \times n$-Distanzmatrix D besitzt genau dann einen (strengen) ultrametrischen Baum, wenn D eine Ultrametrik induziert.*

Beweis: $\Rightarrow$: Die Definitheit und Symmetrie folgt unmittelbar aus der Definition einer Distanzmatrix. Wir müssen also nur noch die ultrametrische Dreiecksungleichung zeigen:

$$\forall i,j,k \in [1:n] : \; d_{ij} \leq \max\{d_{ik}, d_{jk}\}.$$

Wir betrachten dazu den ultrametrischen Baum $T = T(D)$. Wir unterscheiden dazu drei Fälle in Abhängigkeit, wie sich die niedrigsten gemeinsamen Vorfahren von i, j und k zueinander verhalten. Sei dazu $x = \text{lca}(i,j)$, $y = \text{lca}(i,k)$ und $z = \text{lca}(j,k)$. In einem Baum muss dabei gelten, dass mindestens zwei der drei Knoten identisch sind. Die ersten beiden Fälle sind in Abbildung 2.23 dargestellt.

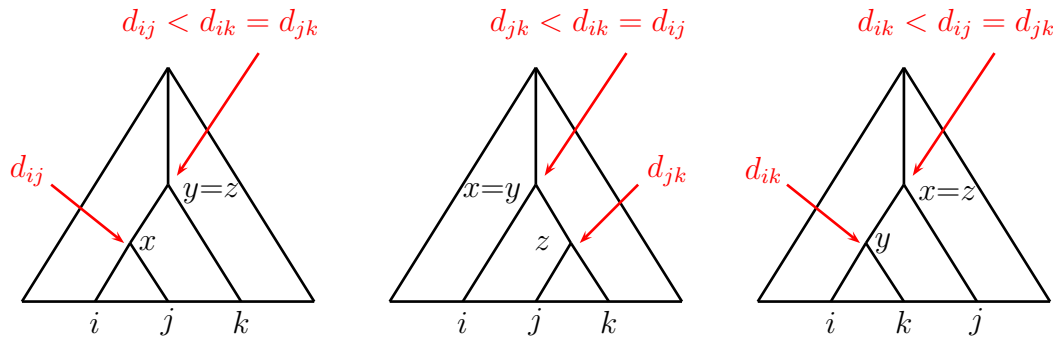


Abbildung 2.23: Skizze: Fall 1, Fall 2 und Fall 3

Fall 1 ($y = z \neq x$): Damit folgt aus dem rechten Teil der Abbildung 2.23 sofort, dass $d(i,j) \leq d(i,k) = d(j,k)$.

Fall 2 ($x = y \neq z$): Damit folgt aus dem linken Teil der Abbildung 2.23 sofort, dass $d(j,k) \leq d(i,k) = d(i,j)$.

Fall 3 ($x = z \neq y$): Dieser Fall ist symmetrisch zu Fall 2 (einfaches Vertauschen von i und j).

Fall 4 ($x = y = z$): Aus der Abbildung 2.24 folgt auch hier, dass die ultrametrische Dreiecksungleichung gilt, da alle drei Abstände gleich sind.

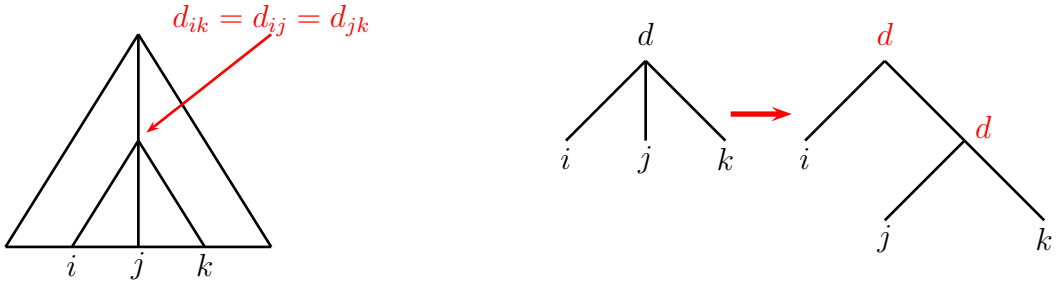


Abbildung 2.24: Skizze: Fall 4 und binäre Neukonstruktion

Wenn man statt eines strengen ultrametrischen Baumes lieber einen binären ultrametrischen Baum haben möchte, so kann man ihn beispielsweise wie im rechten Teil der Abbildung 2.24 umbauen.

$\Leftarrow$: Wir betrachten zuerst die Abstände von Blatt 1 zu allen anderen Blättern. Sei also $\{d_{12}, \ldots, d_{1n}\} = \{\delta_1, \ldots, \delta_k\}$, d.h. $\delta_1, \ldots, \delta_k$ sind die paarweise verschiedenen Abstände, die vom Blatt 1 aus auftreten. Ohne Beschränkung der Allgemeinheit nehmen wir dabei an, dass $\delta_1 < \cdots < \delta_k$. Wir partitionieren dann $[2:n]$ wie folgt:

$$D_i = \{\ell \in [2:n] \ : \ d_{1\ell} = \delta_i\}\,.$$

Es gilt dann offensichtlich $[2:n] = \uplus_{i=1}^{k} D_i$. Wir bestimmen jetzt für die Mengen D_i rekursiv die entsprechenden ultrametrischen Bäume $T(D_i')$, wobei D_i' aus D erzeugt wird, indem nur die Zeilen und Spalten aus D_i übrig bleiben. Anschließend konstruieren wir einen Pfad von der Wurzel zum Blatt 1 mit k inneren Knoten, an die die rekursiv konstruierten Teilbäume $T(D_i')$ angehängt werden. Dies ist in Abbildung 2.25 schematisch dargestellt.

Wir müssen jetzt nachprüfen, ob der konstruierte Baum ultrametrisch ist. Dazu müssen wir zeigen, dass die Knotenmarkierungen auf einem Pfad von der Wurzel zu einem Blatt streng monoton fallend sind und dass der Abstand von den Blättern i und j gerade die Markierung von $\mathrm{lca}(i,j)$ ist.

Für den ersten Teil überlegen wir uns, dass die Monotonie der Knotenmarkierungen sowohl auf dem Pfad von der Wurzel zu Blatt 1 gilt als auch auf allen Pfaden

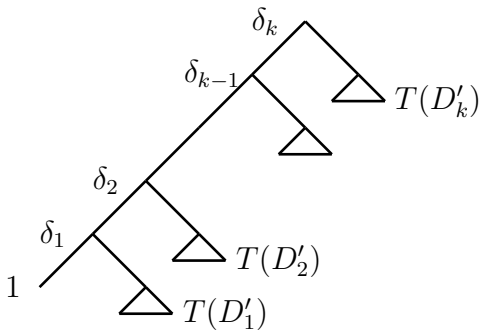


Abbildung 2.25: Skizze: Rekursive Konstruktion ultrametrischer Bäume

innerhalb der Teilbäume $T(D'_i)$ von der jeweiligen Wurzel zu eine beliebigen Blatt von $T(D'_i)$. Wir müssen also nur noch die Verbindungspunkte von einem Konten des Pfades von der Wurzel zum Blatt 1 zu jedem Teilbaum $T(D'_i)$ überprüfen. Dies ist in Abbildung 2.26 illustriert. Hier sind x und y zwei Blättern in $T(D'_i)$, deren niedrigster gemeinsamer Vorfahre gerade die Wurzel von $T(D'_i)$ ist (falls $T(D'_i)$ nur aus einem Blatt besteht, ist nichts zu zeigen).

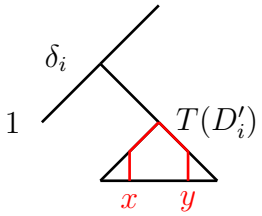


Abbildung 2.26: Skizze: Abstände von x und y und geforderte Monotonie

Wir müssen als zeigen, dass $d_{x,y} < \delta_i$ gilt. Es gilt zunächst aufgrund der ultrametrischen Dreiecksungleichung:

$$d_{xy} \leq \max\{d_{1x}, d_{1y}\} = d_{1x} = d_{1y} = \delta_i.$$

Gilt jetzt $d_{xy} < \delta_i$, dann ist alles gezeigt. Andernfalls gilt $\delta_{xy} = \delta_i$ und wir werden den Baum noch ein wenig umbauen, wie in der folgenden Abbildung 2.27 illustriert. Dabei wird die Wurzel des Teilbaums $T(D'_i)$ mit dem korrespondierenden Knoten

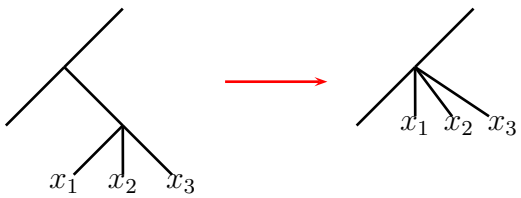


Abbildung 2.27: Skizze: Umbau im Falle nicht-strenger Monotonie

des Pfades von der Wurzel des Gesamtbaumes zum Blatt 1 miteinander identifiziert und die Kante dazwischen gelöscht. Damit haben wir die strenge Monotonie der Knotenmarkierungen auf den Pfaden von der Wurzel zu den Blättern nachgewiesen.

Es ist jetzt noch zu zeigen, dass die Abstände von zwei Blättern x und y den Knotenmarkierungen entsprechen. Innerhalb der Teilbäume $T(D'_i)$ gilt dies nach Konstruktion. Ebenfalls gilt dies nach Konstruktion für das Blatt 1 mit allen anderen Blättern.

Wir müssen diese Eigenschaft nur noch nachweisen, wenn sich zwei Blätter in unterschiedlichen Teilbäumen befinden. Sei dazu $x \in V(T(D'_i))$ und $y \in V(T(D'_j))$, wobei wir ohne Beschränkung der Allgemeinheit annehmen, dass $\delta_i > \delta_j$ gilt. Dies ist in der Abbildung 2.28 illustriert.

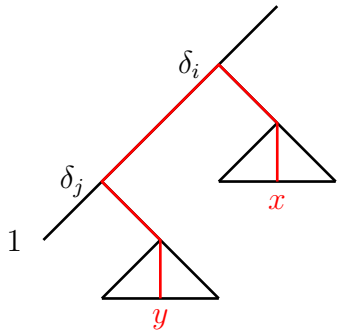


Abbildung 2.28: Skizze: Korrektheit der Abstände zweier Blätter in unterschiedlichen rekursiv konstruierten Teilbäumen

Nach Konstruktion gilt: $\delta_i = d_{1x}$ und $\delta_j = d_{1y}$. Mit Hilfe der ultrametrischen Dreiecksungleichung folgt:

$$\begin{aligned} d_{xy} &\leq \max\{d_{1x}, d_{1y}\} \\ &= \max\{\delta_i, \delta_j\} \\ &= \delta_i \\ &= d_{1x} \\ &\leq \max\{d_{1y}, d_{yx}\} \\ &\quad \text{da } d_{1y} = \delta_j < \delta_i = d_{1x}\text{, kann } d_{1y} \text{ nicht das Maximum sein} \\ &= d_{xy}. \end{aligned}$$

Daraus folgt also $d_{xy} \leq \delta_i \leq d_{xy}$ und somit $\delta_i = d_{xy}$. Damit ist der Beweis abgeschlossen. ■

Aus dem Beweis folgt weiter unter Annahme der strengen Monotonie (d.h. für strenge ultrametrische Bäume), dass der konstruierte ultrametrische Baum eindeu-

tig ist. Die Konstruktion des ultrametrischen Baumes ist ja bis auf die Umordnung der Kinder eines Knoten eindeutig festgelegt.

Korollar 2.34 *Sei D eine ultrametrische Matrix, dann ist der zugehörige strenge ultrametrische Baum eindeutig.*

Für nicht-strenge ultrametrische Bäume kann man sich überlegen, wie die Knoten mit gleicher Markierung umgeordnet werden können, so dass der Baum ein ultrametrischer bleibt. Bis auf diese kleinen Umordnungen sind auch nicht-streng ultrametrische Bäume im Wesentlichen eindeutig. Des Weiteren ist ein Baum genau dann ultrametrisch, wenn er streng ultrametrisch ist. Dazu muss man nur die Endknoten von Kanten, die dieselbe Markierung besitzen, miteinander identifizieren.

2.4.4 Konstruktion ultrametrischer Bäume

Mit Hilfe der bereits gezeigten erwähnten Charakterisierung einer Ultrametrik, dass von den drei Abständen zwischen drei Punkten die beiden größten gleich sind, können wir sofort einen einfachen Algorithmus zur Erkennung ultrametrischer Matrizen angeben. Wir müssen dazu nur alle dreielementigen Teilmengen aus $[1:n]$ untersuchen, ob von den drei verschiedenen Abständen, die beiden größten gleich sind. Falls ja, ist die Matrix ultrametrisch, ansonsten nicht.

Dieser Algorithmus hat jedoch eine Laufzeit $O(n^3)$. Wir wollen im Folgenden einen effizienteren Algorithmus zur Erkennung ultrametrischer Matrizen angeben, der auch gleichzeitig noch die zugehörigen ultrametrischen Bäume mitberechnet.

Wir versuchen jetzt aus dem Beweis der Existenz eines ultrametrischen Baumes einen effizienten Algorithmus zur Konstruktion eines ultrametrischen Baumes zu entwerfen und dessen Laufzeit zu analysieren.

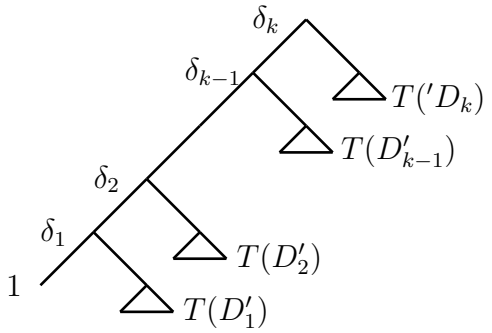


Abbildung 2.29: Skizze: Konstruktion eines ultrametrischen Baumes

(1) Sortiere die Menge $\{d_{12}, \ldots, d_{1n}\}$ und bestimme anschließend $\delta_1 < \cdots < \delta_k$ mit $\{\delta_1, \ldots, \delta_k\} = \{d_{12}, \ldots, d_{1n}\}$ und die Partition $[2:n] = \uplus_{i=1}^{k} D_i$. $O(n \log(n))$

(2) Bestimme für $D_1, \ldots D_k$ ultrametrische Bäume $T(D'_1), \ldots, T(D'_k)$ mittels Rekursion. $\sum_{i=1}^{k} T(n_i)$

(3) Setze die Teillösungen und den Pfad von der Wurzel zu Blatt 1 zur Gesamtlösung zusammen. $O(k) = O(n)$

Abbildung 2.30: Algorithmus: Naive Konstruktion eines ultrametrischen Baumes

Wir erinnern noch einmal an die Partition von $[2:n]$ durch $D_i = \{\ell : d_{1\ell} = \delta_i\}$ mit $n_i := |D_i|$. Dabei war $\{d_{12}, \ldots, d_{1n}\} = \{\delta_1, \ldots, \delta_k\}$, wobei $\delta_1 < \cdots < \delta_k$. Wir erinnern hier auch noch einmal an Skizze der Konstruktion des ultrametrischen Baumes, wie in Abbildung 2.29.

Daraus ergibt sich der erste einfache Algorithmus, der in Abbildung 2.30 aufgelistet ist. Da jeder Schritt Zeitbedarf $O(n \log(n))$ und es maximal n rekursive Aufrufe geben kann (mit jedem rekursiven Aufruf wird ein Knoten des ultrametrischen Baumes explizit konstruiert), ist die Laufzeit im worst-case höchsten $O(n^2 \log(n))$. Man kann zeigen, dass es Distanzmatrizen gibt, die tatsächlich eine Laufzeit von $\Theta(n^2 \log(n))$ besitzen.

Wir werden jetzt noch einen leicht modifizierten Algorithmus vorstellen, der eine Laufzeit von nur $O(n^2)$ besitzt. Dies ist optimal, da die Eingabe, die gegebene Distanzmatrix, bereits eine Größe von $\Theta(n^2)$ besitzt. Dazu beachten wir, dass der aufwendige Teil des Algorithmus das Sortieren der Elemente in $\{d_{12}, \ldots, d_{1n}\}$ ist. Insbesondere ist dies sehr teuer, wenn es nur wenige verschieden Elemente in dieser Menge gibt, da dann auch nur entsprechend wenig rekursive Aufrufe folgen, die wiederum auf relativ großen Matrizen stattfinden.

Daher werden wir zuerst feststellen, wie viele verschiedene Elemente es in der Menge $\{d_{11}, \ldots, d_{1n}\}$ gibt und bestimmen diese. Die paarweise verschiedenen Elemente dieser Menge können wir in einer linearen Liste (oder auch in einem balancierten Suchbaum) aufsammeln. Dies lässt sich in Zeit $O(k \cdot n)$ (bzw. in Zeit $O(n \log(k))$ bei Verwendung balancierter Bäume) implementieren. Das kann zwar teurer als mittels Sortieren sein, jedoch ist es insbesondere dann billig, wenn es wenige paarweise verschiedene Elemente gibt.

Um die verschiedenen Elemente $\delta_1 < \cdots < \delta_k$ der Menge $\{d_{11}, \ldots, d_{1n}\}$ zu ermitteln, verwenden wir eine lineare Liste, die die verschiedenen Element beinhalten wird, zu Beginn ist diese leer. Für jedes neue Element d_{1i} suchen wir in der Liste in Zeit $O(k)$, ob es bereits darin enthalten ist. Falls nicht, fügen wir es ans Ende

(1) Bestimme zuerst $k = |\{d_{12}, \ldots, d_{1n}\}|$ und $\{\delta_1, \ldots, \delta_k\} = \{d_{12}, \ldots, d_{1n}\}$.

Dies kann mit Hilfe linearer Listen in Zeit $O(k \cdot n)$ erledigt werden. Mit Hilfe balancierter Bäume können wir dies sogar in Zeit $O(n \log(k))$ realisieren.

(2) Sortiere die k paarweise verschiedenen Werte $\{\delta_1, \ldots, \delta_k\}$.

Dies kann in Zeit $O(k \log(k))$ erledigt werden.

(3) Bestimme für $D_1, \ldots D_k$ ultrametrische Bäume $T(D'_1), \ldots, T(D'_k)$ mittels Rekursion.

(4) Setze die Teillösungen und den Pfad von der Wurzel zu Blatt 1 zur Gesamtlösung zusammen.

Dies lässt sich wiederum in Zeit $O(k)$ realisieren.

Abbildung 2.31: Algorithmus: Effiziente Konstruktion eines ultrametrischen Baumes

der Liste an, andernfalls tun wir nichts weiter. Die Laufzeit hierzu ist offensichtlich $O(k \cdot n)$. Zum Schluss kopieren wir die Elemente der Liste in ein Feld in Zeit $O(k)$ und sortieren dieses in Zeit $O(k \log(k)) = O(k \cdot n)$. Der aus diesen Gedanken resultierende Algorithmus selbst ist in Abbildung 2.31 aufgelistet.

Damit erhalten wir als Rekursionsgleichung für diesen modifizierten Algorithmus:

$$T(n) = d \cdot k \cdot n + \sum_{i=1}^{k} T(n_i)$$

mit $\sum_{i=1}^{k} n_i = n - 1$, wobei $n_i \geq 1$ für $i \in [1 : n]$, und einer geeignet gewählten Konstanten d. Hierbei ist zu beachten, dass $O(k \log(k)) = O(k \cdot n)$ ist, da ja $k < n$ ist.

20.06.24

Lemma 2.35 *Ist D eine ultrametrische $n \times n$-Matrix, dann kann der zugehörige ultrametrische Baum in Zeit $O(n^2)$ konstruiert werden.*

Beweis: Es ist nur noch zu zeigen, dass $T(n) \leq c \cdot n^2$ für eine geeignet gewählte Konstante c gilt. Wir wählen c jetzt so, dass zum einen $c \geq 2d$ und zum anderen $T(n) \leq c \cdot n^2$ für alle $n \leq 2$ gilt. Den Beweis selbst führen wir mit vollständiger Induktion über n.

Induktionsanfang ($n \leq 2$): Nach Wahl von c gilt dies offensichtlich.

Induktionsschritt ($\rightarrow n$): Es gilt dann nach Konstruktion des Algorithmus mit $n-1 = \sum_{i=1}^{k} n_i$:

$$T(n) \leq d \cdot k \cdot n + \sum_{i=1}^{k} T(n_i).$$

Dies können wir nun wie folgt abschätzen:

$$\begin{aligned}
T(n) &\leq d \cdot k \cdot n + \sum_{i=1}^{k} T(n_i) \\
&\quad \text{nach Induktionsvoraussetzung ist } T(n_i) \leq c \cdot n_i^2 \\
&\leq d \cdot k \cdot n + \sum_{i=1}^{k} c \cdot n_i^2 \\
&\quad \text{mit } n_i^2 = n_i(n - n + n_i) \\
&= d \cdot k \cdot n + c \cdot \sum_{i=1}^{k} n_i(n - n + n_i) \\
&= d \cdot k \cdot n + c \cdot \sum_{i=1}^{k} n_i \cdot n - c \cdot \sum_{i=1}^{k} n_i(n - n_i) \\
&\quad \text{da } x(n-x) > 1(n-1) \text{ für } x \in [1 : n-1], \text{ siehe auch Abbildung 2.32} \\
&\leq d \cdot k \cdot n + c \cdot n^2 - c \cdot \sum_{i=1}^{k} 1(n-1) \\
&= d \cdot k \cdot n + c \cdot n^2 - c \cdot k(n-1) \\
&\quad \text{da } c \geq 2d \\
&\leq d \cdot k \cdot n + c \cdot n^2 - 2d \cdot k(n-1) \\
&\quad \text{da } 2(n-1) \geq n \text{ für } n \geq 2 \\
&\leq d \cdot k \cdot n + c \cdot n^2 - d \cdot k \cdot n \\
&= c \cdot n^2.
\end{aligned}$$

Damit ist der Induktionsschluss vollzogen und der Beweis beendet. ■

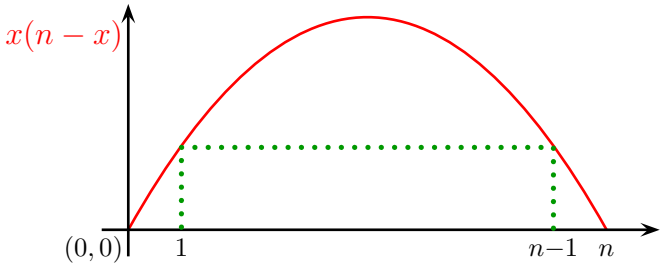


Abbildung 2.32: Skizze: Die Funktion $[0, n] \rightarrow \mathbb{R}_+ : x \mapsto x(n-x)$

Korollar 2.36 *Es kann in Zeit $O(n^2)$ entschieden werden, ob eine gegebene $n \times n$-Distanzmatrix (streng) ultrametrisch ist oder nicht.*

Beweis: Wir wenden einfach den Algorithmus zu Rekonstruktion ultrametrischer Bäume an. Wird dabei ein ultrametrischer Baum erzeugt, so ist nach Definition die gegebene Distanzmatrix ultrametrisch. Ist er hingegen nicht ultrametrisch, so kann die Distanzmatrix nicht ultrametrisch sein, denn sonst hätten wir nach dem vorherigen Satz einen ultrametrischen Baum konstruieren müssen.

Wir müssen also nur noch prüfen, ob der konstruierte Baum ultrametrisch ist, d.h. ob die Monotonie-Bedingung erfüllt ist und ob für zwei Taxa am zugehörigen niedrigsten gemeinsamen Vorfahren der entsprechende Abstand markiert ist.

Für die Monotonie testen wir dabei beim Anhängen des rekursiv konstruierten Baumes $T(D'_i)$, ob die Markierung seiner Wurzel kleiner (gleich) der Markierung des Knotens ist, an den $T(D'_i)$ angehängt wird. Ist dies nicht der Fall, so ist die Matrix nicht (streng) ultrametrisch, da wir ja sonst nach Lemma 2.35 einen (streng) ultrametrischen Baum konstruieren müssen.

Für die Abstände bestimmen wir zunächst für jeden Knoten v im konstruierten Baum T seine Domain $D_T(v)$, die als einfach verkettete Liste gespeichert wird. Der einfacheren Beschreibung wegen gehen wir ohne Beschränkung der Allgemeinheit von einem binären Baum aus, die Erweiterung auf den allgemeineren Fall ist technisch etwas aufwendiger.

Der Baum wird mit einer einfachen Tiefensuche durchlaufen und bei der Rückkehr von den Kinder u und w des Knotens v kann durch Konkatenation der Domains $D_T(u)$ und $D_T(w)$ die Domain $D_T(v)$ konstruiert werden. Bevor wir die Domain $D_T(v)$ konstruieren, betrachten wir alle Paare (i, j) mit $i \in D_T(u)$ und $j \in D_T(w)$ (und somit $i \neq j$). Für diese Paare (i, j) gilt dann $\text{lca}(i, j) = v$. Somit muss die Markierung des Knotens v mit all solchen $d_{ij} = d_{ji}$ übereinstimmen. Man mache sich klar, dass man so alle Paare $(i, j) \in [1 : n]^2$ genau einmal aufsucht.

Die Laufzeit für die Tiefensuche und die Konstruktion der Domains (durch Konkatenation der Listen) beträgt offensichtlich $O(n)$. Da jedes Paar $(i, j) \in [1 : n]^2$ (für $i \neq j$) genau einmal betrachtet wird, kostet die Überprüfung der Markierungen an den inneren Knoten $O(n^2)$. Somit lässt sich der ganze Test in Zeit $O(n^2)$ zusätzlich zur Zeit $O(n^2)$ zur Konstruktion des Baumes T durchführen. ■

Zur Feststellung, ob eine Distanzmatrix ultrametrisch ist, und für die Konstruktion des zugehörigen ultrametrischen Baumes gibt es auch ein anderes bekanntes Verfahren durch Neighbor-Joining: UPGMA. Hierbei ist zu beachten, dass die nahe liegende Implementierung nicht in quadratischer Zeit durchführbar ist und Zeit $O(n^2 \log(n))$

benötigt (auch wenn dies in vielen Büchern und Skripten fälschlicherweise behauptet wird). Es gibt jedoch auch mehrere aufwendige Implementierungen von UPGMA in quadratischer Zeit, die aber mit der eigentlichen Beschreibung des Algorithmus nur entfernt etwas zu tun hat (siehe die Originalarbeiten von F. Murtagh, D. Eppstein bzw. I. Gronau und S. Moran).

2.5 Additive Distanzen und additive Bäume

Leider sind nicht alle Distanzmatrizen ultrametrisch. Wir wollen jetzt eine größere Klasse von Matrizen vorstellen, zu denen sich evolutionäre Bäume konstruieren lassen, sofern wir auf eine explizite Wurzel in diesen Bäumen verzichten.

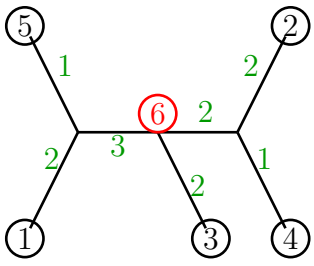
2.5.1 Additive Bäume

Zunächst einmal definieren wir, was wir unter additiven Bäumen verstehen wollen.

Definition 2.37 *Sei D eine $n \times n$-Distanzmatrix. Sei T ein Baum mit mindestens n Knoten und positiven Kantengewichten, wobei einige Knoten bijektiv mit Werten aus $[1 : n]$ markiert sind. Dann ist T ein* additiver Baum *für D, wenn der Pfad vom Knoten mit Markierung i zum Knoten mit Markierung j in T das Gewicht d_{ij} besitzt.*

In der folgenden Abbildung 2.33 ist noch einmal ein Beispiel einer Matrix samt ihres zugehörigen additiven Baumes angegeben. Wie bei den ultrametrischen Bäumen besitzt auch nicht jede Distanzmatrix einen additiven Baum. Des Weiteren sind nicht markierte Blätter in einem additiven Baum überflüssig, da jeder Pfad innerhalb

D	1	2	3	4	5	6
1	0	9	7	8	3	5
2		0	6	3	8	4
3			0	5	6	2
4				0	7	3
5					0	4
6						0



Gewicht des Pfades zwischen 5 und 4: $\Rightarrow 1 + 3 + 2 + 1 = 7$
Gewicht des Pfades zwischen 1 und 5: $\Rightarrow 2 + 1 = 3$

Abbildung 2.33: Beispiel: Eine Matrix und der zugehörige additive Baum

eines Baum nur dann ein Blatt berührt, wenn dieses ein Endpunkt des Pfades ist. Daher nehmen wir im Folgenden ohne Beschränkung der Allgemeinheit an, dass ein additiver Baum nur markierte Blätter besitzt.

Definition 2.38 *Sei D eine Distanzmatrix. Besitzt D einen additiven Baum, so heißt D eine* additive Matrix. *Ein additiver Baum heißt* kompakt*, wenn alle Knoten markiert sind (insbesondere also auch alle inneren Knoten). Die zugehörige additive Matrix heißt dann auch* kompakt additiv. *Ein additiver Baum heißt* extern*, wenn nur Blätter markiert sind. Die zugehörige additive Matrix heißt dann auch* extern additiv.

In der Abbildung 2.33 ist der Baum ohne Knoten 6 (und damit die Matrix ohne Zeile und Spalte 6) ein externer additiver Baum. Durch Hinzufügen von zwei Markierungen (und zwei entsprechenden Spalten und Zeilen in der Matrix) könnte dieser Baum zu einem kompakten additiven Baum gemacht werden.

Lemma 2.39 *Sei D eine additive Matrix, dann induziert D eine Metrik.*

Beweis: Da D eine Distanzmatrix ist, gelten die Definitheit und Symmetrie unmittelbar. Die Dreiecksungleichung sieht man leicht, wenn man sich die entsprechenden Pfade im zu D gehörigen additiven Baum betrachtet. ■

Die Umkehrung gilt übrigens nicht, wie wir später noch zeigen werden.

2.5.2 Charakterisierung additiver Bäume

Wir wollen nun eine Charakterisierung additiver Matrizen angeben, mit deren Hilfe sich auch gleich ein zugehöriger additiver Baum konstruieren lässt. Zunächst einmal zeigen wir, dass ultrametrische Bäume spezielle additive Bäume sind.

Lemma 2.40 *Sei D eine additive Matrix. D ist genau dann ultrametrisch, wenn es einen additiven Baum T für D gibt, so dass es in T einen Knoten v (zentraler Knoten) gibt, der zu allen markierten Knoten denselben Abstand besitzt.*

Beweis: $\Rightarrow$**:** Sei T ein ultrametrischer Baum für D mit Knotenmarkierungen μ. Wir nehmen hierbei ohne Beschränkung der Allgemeinheit an, dass T sogar streng

ultrametrisch ist. Wir erhalten daraus einen additiven Baum T', indem wir als Kantengewicht $\gamma(v, w)$ einer Kante (v, w) folgendes wählen (da T streng ultrametrisch ist, gilt die Behauptung $\gamma(v, w) > 0$):

$$\gamma(v, w) = \frac{1}{2}(\mu(v) - \mu(w)) > 0.$$

Man rechnet jetzt leicht nach, dass für zwei Blätter i und j sowohl das Gewicht des Weges von i nach $\text{lca}(i, j)$ als auch das Gewicht von j nach $\text{lca}(i, j)$ gerade $\frac{1}{2} \cdot d_{ij}$ beträgt. Die Länge des Weges von i nach j beträgt daher im additiven Baum wie gefordert d_{ij}.

Allerdings gibt es in einem solchen additiven Baum Knoten mit Grad 2 (wie sie beispielweise von der Wurzel konstruiert werden, siehe Abbildung 2.34). Man kann sich überlegen, dass man solche Konten mit Grad 2 nicht eliminieren kann (außer man verzichtet auf zentrale Knoten).

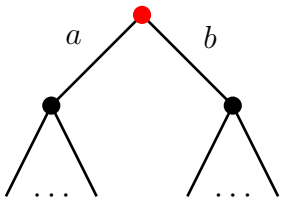


Abbildung 2.34: Skizze: Zentraler Knoten mit Grad 2

$\Leftarrow$: Sei D additiv und sei v der Knoten des additiven Baums T, der von allen Blättern den gleichen Abstand hat. Man überlegt sich leicht, dass T dann extern additiv sein muss.

Betrachtet man v als Wurzel, so gilt für ein beliebiges Paar von Blättern i und j, dass der Abstand zwischen dem kleinsten gemeinsamen Vorfahr ℓ für beide Blätter gleich ist. Da der Abstand $d_{\ell v}$ fest ist und die Kantengewichte positiv sind, wäre andernfalls der Abstand der Blätter zu v nicht gleich. Wir zeigen jetzt, dass für drei beliebige Blätter i, j, k die von den drei möglichen Distanzen d_{ij}, d_{ik} und d_{jk} die beiden größten gleich sind. Dann ist die Matrix D nach Lemma 2.26 ultrametrisch.

Falls $\text{lca}(i, j) = \text{lca}(i, k) = \text{lca}(j, k)$ die gleichen Knoten sind, so ist $d_{ik} = d_{ij} = d_{jk}$ und dann sind die beiden größten Distanzen offensichtlich gleich.

Daher sei jetzt ohne Beschränkung der Allgemeinheit $\text{lca}(i, j) \neq \text{lca}(i, k)$ und dass $\text{lca}(i, k)$ näher an der Wurzel liegt. Da $\text{lca}(i, j)$ und $\text{lca}(i, k)$ auf dem Weg von i zur Wurzel liegen, gilt entweder $\text{lca}(j, k) = \text{lca}(j, i)$ oder $\text{lca}(j, k) = \text{lca}(i, k)$. Weiterhin muss dann $\text{lca}(j, k) = \text{lca}(i, k) = \text{lca}(i, j, k)$ gelten (siehe auch Abbildung 2.35).

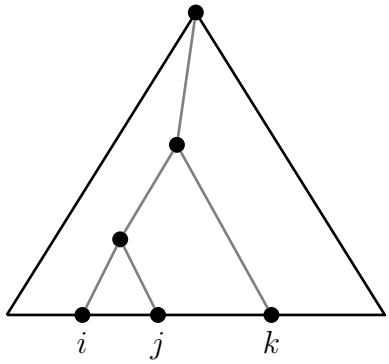


Abbildung 2.35: Skizze: Die Knoten i, j, k, $\mathrm{lca}(i,j)$ und $\mathrm{lca}(i,k) = \mathrm{lca}(j,k)$

Dann gilt:

$$\begin{aligned} d_{ij} &= w(i, \mathrm{lca}(i,j)) + w(j, \mathrm{lca}(i,j)) \\ &= 2 \cdot w(j, \mathrm{lca}(i,j)), \end{aligned}$$

$$\begin{aligned} d_{ik} &= w(i, \mathrm{lca}(i,j)) + w(\mathrm{lca}(i,j), \mathrm{lca}(i,j,k)) + w(\mathrm{lca}(i,j,k), k) \\ &= 2 \cdot w(\mathrm{lca}(i,j,k), k), \end{aligned}$$

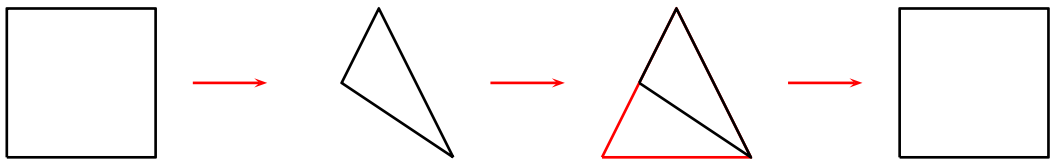
$$\begin{aligned} d_{jk} &= w(j, \mathrm{lca}(i,j)) + w(\mathrm{lca}(i,j), \mathrm{lca}(i,j,k)) + w(\mathrm{lca}(i,j,k), k) \\ &= 2 \cdot w(\mathrm{lca}(i,j,k), k) \end{aligned}$$

Daher gilt unter anderem $d_{ij} \leq d_{ik}$, $d_{ik} \leq d_{jk}$ und $d_{jk} \leq d_{ik}$. Es gilt also insbesondere $d_{ij} \leq d_{ik} = d_{jk}$ und die Behauptung ist bewiesen. ■

Wie können wir nun für eine Distanzmatrix entscheiden, ob sie additiv ist, und falls ja, beschreiben, wie der zugehörige additive Baum aussieht? Im vorherigen Lemma haben wir gesehen, wie wir das Problem auf ultrametrische Matrizen zurückführen können.

Wir wollen dies noch einmal mir einer anderen Charakterisierung tun. Wir betrachten zuerst die gegebene Matrix D. Diese ist genau dann additiv, wenn sie einen additiven Baum T_D besitzt. Wenn wir diesen Baum wurzeln und die Kanten zu den Blättern so verlängern, dass alle Pfade von der Wurzel zu den Blättern gleiches Gewicht besitzen, dann ist T'_D ultrametrisch. Daraus können wir dann eine Distanzmatrix $D' := D(T'_D)$ ablesen, die ultrametrisch ist, wenn D additiv ist. Dies ist in der folgenden Abbildung 2.36 schematisch dargestellt.

Wir wollen also die gegebene Matrix D so modifizieren, dass daraus eine ultrametrische Matrix wird. Wenn diese Idee funktioniert, können wir eine Matrix auf



add. Matrix D | add. Baum T_D | ultrametrischer Baum T'_D | ultrametrische Matrix $D' = D(T'_D)$

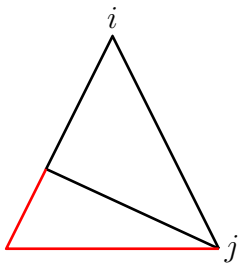
Abbildung 2.36: Skizze: Transformation einer additiven in eine ultrametrische Matrix

Additivität hin testen, indem wir die zugehörige, neu konstruierte Matrix auf Ultrametrik hin testen. Für Letzteres haben wir ja bereits einen effizienten Algorithmus kennen gelernt.

Sei im Folgenden D eine additive Matrix und d_{ij} ein maximaler Eintrag, d.h.

$$d_{ij} = \max\{d_{k\ell} \ : \ k, \ell \in [1:n]\}\,.$$

Weiter sei T_D der additive Baum zu D. Wir wurzeln jetzt diesen Baum am Knoten i. Man beachte, dass i ein Blatt ist. Wir kommen später noch darauf zurück, wie wir dafür sorgen, dass i ein Blatt bleibt. Dies ist in der folgenden Abbildung 2.37 illustriert.



Alle Blätter auf denselben Abstand bringen

Abbildung 2.37: Skizze: Wurzeln von T_D am Blatt i

Unser nächster Schritt besteht jetzt darin, alle Blätter (natürlich außer i) auf denselben Abstand zur neuen Wurzel i zu bringen. Wir betrachten dazu jetzt ein Blatt k des nun gewurzelten Baumes. Wir versuchen die zu k inzidente Kante jetzt so zu verlängern, dass der Abstand von k zur Wurzel i auf d_{ij} anwächst. Dazu setzen wir das Kantengewicht auf d_{ij} und verkürzen es um das Gewicht des restlichen Pfades von k zu i, d.h. um $(d_{ik} - d)$, wobei d das Gewicht der zu k inzidenten Kante ist. Dies ist in Abbildung 2.38 noch einmal illustriert.

War der Baum vorher additiv, so ist er jetzt ultrametrisch, wenn wir als Knotenmarkierung jetzt das Gewicht des Pfades von diesem Knotens zu einem seiner Blätter (die alle gleich sein müssen) wählen.

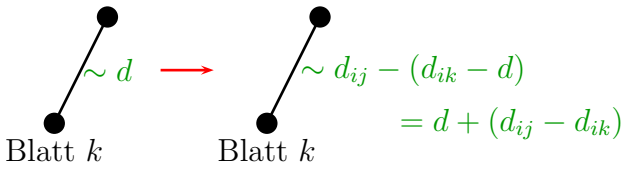


Abbildung 2.38: Skizze: Verlängern der zu Blättern inzidenten Kanten

Somit haben wir aus einem additiven einen ultrametrischen Baum gemacht. Jedoch haben wir dazu den additiven Baum benötigt, den wir eigentlich erst konstruieren wollen. Es stellt sich nun die Frage, ob wir die entsprechende ultrametrische Matrix aus der additiven Matrix direkt ohne Kenntnis des zugehörigen additiven Baumes berechnen können. Betrachten wir dazu noch einmal zwei Blätter im additiven Baum und versuchen den entsprechenden Abstand im ultrametrischen Baum zu berechnen. Dazu betrachten wir die Abbildung 2.39.

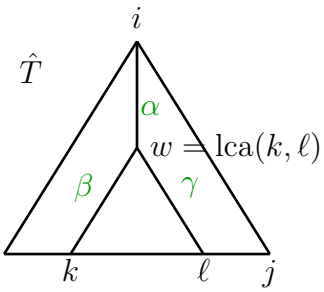


Abbildung 2.39: Skizze: Abstände im gewurzelten additiven Baum $\hat{T}$

Seien k und ℓ zwei Blätter, für die wir den Abstand in der entsprechenden ultrametrischen Matrix bestimmen wollen. Sei w der niedrigste gemeinsame Vorfahre von k und ℓ im gewurzelten additiven Baum $\hat{T}$ und α, β bzw. γ die Abstände im gewurzelten additiven Baum $\hat{T}$ zwischen der Wurzel und w, w und k bzw. w und ℓ. Es gilt dann

$$\begin{aligned} \beta &= d_{ik} - \alpha, \\ \gamma &= d_{i\ell} - \alpha. \end{aligned}$$

Im ultrametrischen Baum $T' = T(D')$ gilt dann gemäß der vorherigen Überlegung, wie man die Gewichte zu den Blättern inzidenter Kanten erhöht:

$$\begin{aligned} d_{T'}(w,k) &= \beta + (d_{ij} - d_{ik}) \\ d_{T'}(w,\ell) &= \gamma + (d_{ij} - d_{i\ell}). \end{aligned}$$

Damit gilt:

$$\begin{aligned} d_{T'}(w,k) &= \beta + d_{ij} - d_{ik} \\ &= d_{ik} - \alpha + d_{ij} - d_{ik} \\ &= d_{ij} - \alpha \end{aligned}$$

und analog:

$$\begin{aligned} d_{T'}(w,\ell) &= \gamma + d_{ij} - d_{i\ell} \\ &= d_{i\ell} - \alpha + d_{ij} - d_{i\ell} \\ &= d_{ij} - \alpha. \end{aligned}$$

Offensichtlich haben wir dann auch $d_{T'}(w,k) = d_{T'}(w,\ell)$ nachgewiesen, wie wir in unserer Konstruktion behauptet haben. Also ist T' tatsächlich ultrametrisch. Wir müssen jetzt nur noch α bestimmen. Aus der Skizze in Abbildung 2.39 folgt sofort, wenn wir die Gewichte der Pfade von i nach k sowie ℓ addieren und davon das Gewicht des Pfades von k nach ℓ subtrahieren:

$$2\alpha = d_{ik} + d_{i\ell} - d_{k\ell}.$$

Daraus ergibt sich:

$$\begin{aligned} d_{T'}(w,k) &= d_{ij} - \frac{1}{2}(d_{ik} + d_{i\ell} - d_{k\ell}), \\ d_{T'}(w,\ell) &= d_{ij} - \frac{1}{2}(d_{ik} + d_{i\ell} - d_{k\ell}). \end{aligned}$$

Somit können wir jetzt die ultrametrische Matrix D' direkt aus der additiven Matrix D berechnen:

$$d'_{k\ell} := d_{T'}(w,k) = d_{T'}(w,\ell) = d_{ij} - \frac{1}{2}(d_{i\ell} + d_{ik} - d_{k\ell}).$$

Wir müssen uns jetzt nur noch überlegen, dass dies wirklich eine ultrametrische Matrix ist, da wir den additiven Baum ja am Blatt i gewurzelt haben. Damit würden wir sowohl ein Blatt verlieren, nämlich i, als auch keinen echten ultrametrischen Baum generieren, da dessen Wurzel nur ein Kind anstatt mindestens zweier besitzt. Wir machen das Blatt i nun zur echten Wurzel und hängen an diese Wurzel ein weiteres Blatt mit Markierung i. Da alle Blätter zu i den Abstand d_{ij} hatten, markieren wir die neue Wurzel mit d_{ij}. Die Monotonie bleibt dadurch erhalten und die Abstände von allen Blättern $\ell \neq i$ zu i ist dann auch d_{ij}. Damit erhalten wir einen echten ultrametrischen Baum, wie in Abbildung 2.40 illustriert.

Damit haben wir das folgende Lemma bewiesen.

Lemma 2.41 *Sei D eine additive Matrix, deren maximaler Eintrag d_{ij} ist, dann ist D' mit*

$$d'_{kl} = \begin{cases} d_{ij} - \frac{1}{2}(d_{i\ell} + d_{ik} - d_{k\ell}) & \text{für } k \neq \ell \\ 0 & \text{sonst} \end{cases}$$

eine ultrametrische Matrix.

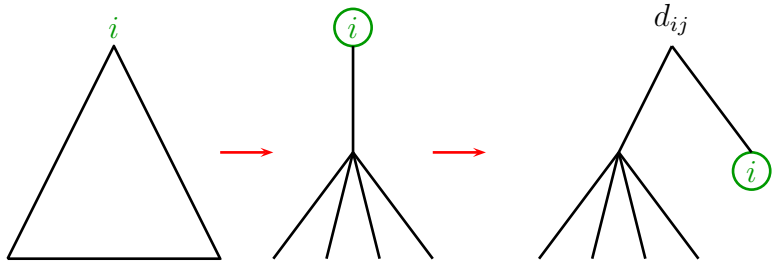


Abbildung 2.40: Skizze: Wirkliches Wurzeln des additiven Baumes

Es wäre schön, wenn auch die umgekehrte Richtung gelten würde, nämlich, dass wenn D' ultrametrisch ist, dass dann bereits D additiv ist. Dies ist leider nicht der Fall, auch wenn dies in vielen Lehrbüchern und Skripten fälschlicherweise behauptet wird. Wir geben hierfür ein Gegenbeispiel in Abbildung 2.41. Man sieht leicht, dass D' ultrametrisch ist. D ist hingegen nicht additiv, weil $d(1,2) = 8$, aber

$$d(1,3) + d(3,2) = 4 + 2 = 6 < 8 = d(1,2)$$

gilt und somit keine Metrik ist (siehe Lemma 2.39).

D	1	2	3
1	0	8	4
2		0	2
3			0

D'	1	2	3
1	0	8	8
2		0	3
3			0

$d'_{12} = 8 - \frac{1}{2}(0 + 8 - 8) = 8$
$d'_{13} = 8 - \frac{1}{2}(0 + 4 - 4) = 8$
$d'_{23} = 8 - \frac{1}{2}(8 + 4 - 2) = 3$

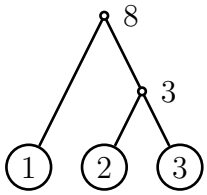


Abbildung 2.41: Gegenbeispiel: D nicht additiv, aber D' ultrametrisch

Im Baum muss also der Umweg über 3 größer sein als der direkte Weg, was nicht sein kann (siehe auch Abbildung 2.42), denn im additiven Baum gilt die normale Dreiecksungleichung (siehe Lemma 2.39).

25.06.24

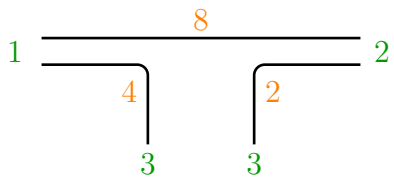


Abbildung 2.42: Gegenbeispiel: „Unmöglicher“ additiver Baum

Dennoch können wir mit einer weiteren Zusatzbedingung dafür sorgen, dass das Lemma in der von uns gewünschten Weise gerettet werden kann.

Lemma 2.42 *Sei D eine Distanzmatrix und sei d_{ij} ein maximaler Eintrag von D. Weiter sei D' durch $d'_{k\ell} = d_{ij} - \frac{1}{2}(d_{ik} + d_{i\ell} - d_{k\ell})$ für $k \neq \ell \in [1:n]$ und $d'_{kk} = 0$ für $k \in [1:n]$ definiert. Wenn D' eine ultrametrische Matrix ist und wenn für jedes Blatt b im zugehörigen ultrametrischen Baum $T(D')$ für das Gewicht γ der zu b inzidenten Kante gilt: $\gamma \geq (d_{ij} - d_{bi})$, dann ist D additiv.*

Beweis: Sei $T' := T(D')$ ein ultrametrischer Baum für D'. Weiter sei $(v, w) \in E(T')$ und es seien p, q, r, s Blätter von T' (nicht notwendigerweise paarweise verschieden), so dass $\text{lca}(p, q) = v$ und $\text{lca}(r, s) = w$. Dies ist in Abbildung 2.43 illustriert. Hierbei ist q zweimal angegeben, da a priori nicht klar ist, ob q ein Nachfolger von w ist oder nicht.

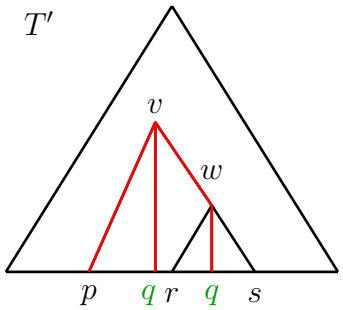


Abbildung 2.43: Skizze: Kante (v, w) in T'

Um nun T' in einen additiven Baum zu transformieren, definieren wir dann das Kantengewicht von (v, w) durch:

$$\gamma(v, w) = d'_{pq} - d'_{rs}.$$

Damit ergibt sich für den Abstand $d_{T'}(k, \ell)$ zwischen k und ℓ im dann so konstruierten additiven Baum T' (wir verwenden weiterhin die Bezeichnung T'):

$$\begin{aligned} d_{T'}(k, \ell) &= 2 \cdot d'_{k,\ell} \\ &= 2 \cdot (d_{ij} - \frac{1}{2}(d_{ik} + d_{i\ell} - d_{k\ell})) \\ &= 2d_{ij} - d_{ik} - d_{i\ell} + d_{k\ell}. \end{aligned}$$

Beachte dabei, dass die Distanzen im Baum T' mit Kantengewichten zwischen zwei Blättern nun doppelt so groß wie im ursprünglichen ultrametrischen Baum T' sind. Wenn wir jetzt für jedes Blatt b das Gewicht der inzidenten Kante um $d_{ij} - d_{bi}$

erniedrigen, erhalten wir einen neuen additiven Baum T. Da nach Voraussetzung, das Gewicht einer solchen zu b inzidenten Kante größer als oder gleich $d_{ij} - d_{bi}$ ist, bleiben die Kantengewichte von T nichtnegativ. Weiterhin gilt in T:

$$\begin{aligned} d_T(k,\ell) &= d_{T'}(k,\ell) - (d_{ij} - d_{ik}) - (d_{ij} - d_{i\ell}) \\ &= (2d_{ij} - d_{ik} - d_{i\ell} + d_{k\ell}) - d_{ij} + d_{ik} - d_{ij} + d_{i\ell} \\ &= d_{k\ell}. \end{aligned}$$

Somit ist T ein additiver Baum für D und das Lemma ist bewiesen. ■

Gehen wir noch einmal zurück zu unserem Gegenbeispiel, das besagte, dass die Ultrametrik von D' nicht ausreicht, um die Additivität von D zu zeigen. Wir schauen einmal was hier beim Kürzen der Gewichte von zu Blättern inzidenten Kanten passieren kann. Dies ist in Abbildung 2.44 illustriert. Wir sehen hier, dass der Baum zwar die gewünschten Abstände besitzt, jedoch negative Kantengewichte erzeugt, die bei additiven Bäumen nicht erlaubt sind.

D	1	2	3
1	0	8	4
2		0	2
3			0

D'	1	2	3
1	0	8	8
2		0	3
3			0

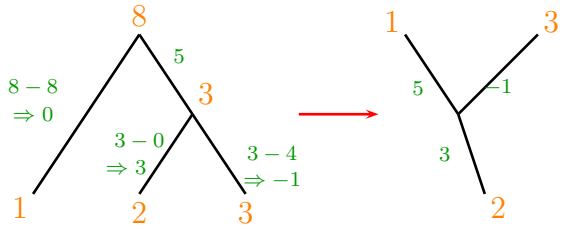


Abbildung 2.44: Gegenbeispiel: D nicht additiv und D' ultrametrisch (Fortsetzung)

Damit können wir nun im folgenden Korollar eine exakte Charakterisierung additiver Bäume angeben.

Korollar 2.43 *Sei D eine Distanzmatrix und sei d_{ij} ein maximaler Eintrag von D. Weiter sei D' durch $d'_{k\ell} = d_{ij} - \frac{1}{2}(d_{ik} + d_{i\ell} - d_{k\ell})$ für $k \neq \ell \in [1:n]$ und $d'_{kk} = 0$ für $k \in [1:n]$ definiert. Wenn D' eine ultrametrische Matrix ist und wenn für jedes Blatt b im zugehörigen ultrametrischen Baum $T(D')$ für das Gewicht γ der zu b inzidenten Kante gilt: $\gamma \geq (d_{ij} - d_{bi})$, dann und nur dann ist D additiv.*

Beweis: Wir müssen im Vergleich zum Lemma 2.42 nur die Rückrichtung zeigen. Nach unserer Konstruktion eines ultrametrischen Baumes aus einer additiven Matrix D ist aber klar, dass die Bedingung an die Kantengewichte der zu den Blättern inzidenten Kanten eingehalten wird. ■

Wir geben jetzt noch eine andere Formulierung des Korollars an, die ohne eine explizite Konstruktion des zugehörigen ultrametrischen Baumes auskommt. Diese

Fassung ist für den Entwurf eines Algorithmus zur Erkennung additiver Matrizen viel angenehmer und leichter zu implementieren.

Korollar 2.44 *Sei D eine Distanzmatrix und sei d_{ij} ein maximaler Eintrag von D. Weiter sei D' durch $d'_{k\ell} = d_{ij} - \frac{1}{2}(d_{ik} + d_{i\ell} - d_{k\ell})$ für $k \neq \ell \in [1:n]$ und $d'_{kk} = 0$ für $k \in [1:n]$ definiert. Wenn D' eine ultrametrische Matrix ist und wenn $d_{i\ell} \leq d_{ib} + d_{b\ell}$ für alle $b, \ell \in [1:n]$ gilt, dann und nur dann ist D additiv.*

Beweis: Wir müssen nur zeigen, dass für jedes Blatt b die Bedingung, dass im zugehörigen ultrametrischen Baum $T(D')$ für das Gewicht γ der zu b inzidenten Kante $\gamma \geq (d_{ij} - d_{bi})$ gilt, äquivalent zu $d_{i\ell} \leq d_{ib} + d_{b\ell}$ für alle $b, \ell \in [1:n]$ ist. Es gilt für jedes Blatt b im ultrametrischen Baum $T(D')$:

$$\gamma = \min\left\{d'_{b\ell} \ : \ \ell \in [1:n] \wedge \ell \neq b\right\},$$

da der Elter eines Blattes im ultrametrischen Baum mit dem minimalem Abstand zu einem anderen Taxa markiert ist.

Mit der Gleichung $d'_{b\ell} = d_{ij} - \frac{1}{2}(d_{ib} + d_{i\ell} - d_{b\ell})$ folgt, dass die folgende äquivalente Ungleichung erfüllt sein muss

$$(d_{ij} - d_{bi}) \leq \min\left\{d_{ij} - \frac{1}{2}(d_{ib} + d_{i\ell} - d_{b\ell}) \ : \ \ell \in [1:n] \wedge \ell \neq b\right\}.$$

Das ist äquivalent zu

$$0 \leq \min\left\{\frac{1}{2}\left(d_{ib} - d_{i\ell} + d_{b\ell}\right) \ : \ \ell \in [1:n] \wedge \ell \neq b\right\}.$$

Da für $\ell = b$ immer $d_{ib} - d_{i\ell} + d_{b\ell} = 0$ gilt, ist dies weiterhin äquivalent zu

$$0 \leq \min\left\{d_{ib} + d_{b\ell} - d_{i\ell} \ : \ \ell \in [1:n]\right\}.$$

Dies ist aber äquivalent dazu, dass für alle b, ℓ gilt: $d_{i\ell} \leq d_{ib} + d_{b\ell}$. Somit ist das Korollar bewiesen. ■

Wir wollen hier noch anmerken, dass wir im Beweis der Einfachheit halber im additiven Baum Kanten mit Gewicht 0 zugelassen haben. Diese Kanten müssen nachher kontrahiert werden, d.h. die Endpunkte von einer Kante mit Gewicht 0 werden als ein Knoten identifiziert.

Wir wollen noch explizit anmerken, dass diese Charakterisierung für gewöhnliche additive und nicht nur für extern additive Matrizen funktioniert.

2.5.3 Algorithmus zur Erkennung additiver Matrizen

Aus der Charakterisierung der additiven Matrizen mit Hilfe ultrametrischer Matrizen lässt sich der folgende Algorithmus zur Erkennung additiver Matrizen und der Konstruktion zugehöriger additiver Bäume herleiten. Dieser ist in Abbildung 2.45 aufgelistet. Es lässt sich leicht nachrechnen, dass dieser Algorithmus eine Laufzeit von $O(n^2)$ besitzt, da im Wesentlichen jeder der 5 Punkte diese Laufzeit hat.

1. Bestimme den maximalen Eintrag d_{ij} von D sowie die zugehörigen Indizes.
2. Teste, ob $d_{i\ell} \leq d_{ib} + d_{bl}$ für alle $b, \ell \in [1:n]$. Falls nicht, ist D nicht additiv.
3. Konstruiere aus der gegebenen $n \times n$-Matrix D eine neue $n \times n$-Matrix D' mittels $d'_{k\ell} = d_{ij} - \frac{1}{2}(d_{ik} + d_{i\ell} - d_{k\ell})$ für alle $k \neq \ell$.
4. Teste, ob D' ultrametrisch ist (und konstruiere ggf. den zugehörigen ultrametrischen Baum T'). Ist D' ultrametrisch, dann ist D additiv, ansonsten nicht.
5. Konstruiere aus T' einen additiven Baum für D, sofern gewünscht.

Abbildung 2.45: Algorithmus: Erkennung additiver Matrizen

Wir müssen uns nur noch kurz um die Korrektheit kümmern. Nach Korollar 2.44 wissen wir, dass in Schritt 2 die richtige Entscheidung getroffen wird. Nach Lemma 2.42 wird auch im Schritt 4 die richtige Entscheidung getroffen. Also ist der Algorithmus korrekt. Fassen wir das Ergebnis noch zusammen.

Theorem 2.45 *Es kann in Zeit $O(n^2)$ entschieden werden, ob eine gegebene $n \times n$-Distanzmatrix additiv ist oder nicht. Falls die Matrix additiv ist, kann der zugehörige additive Baum in Zeit $O(n^2)$ konstruiert werden.*

Wenn wir den additiven Baum rekonstruieren wollen, können wir die Überprüfung gemäß Korollar 2.44 im Schritt 2 auch auslassen. Stattdessen wird dann in Schritt 5 die Überprüfung gemäß Korollar 2.43 durchgeführt. Wir überprüfen dann dabei, ob die Erniedrigung des Gewichts γ einer zum einem Blatt b inzidenten Kanten möglich ist, d.h. ob $\gamma \geq (d_{ij} - d_{ib})$ gilt. Anders gesagt, wenn wir bei der Erniedrigung des Gewichts einer zu einem Blatt inzidenten Kante feststellen, dass diese negativ werden würde, brechen wir den Algorithmus ab, da dann die gegebene Distanzmatrix nicht additiv sein kann.

Zum Schluss wenden wir diesen Algorithmus noch einmal explizit auf die additive Distanzmatrix für unser Beispiel aus Abbildung 2.33 an (für die Distanzmatrix siehe

auch Abbildung 2.46). Hier werden wir explizit Schritt 2 auslassen (der Leser sei eingeladen diesen selbst einmal zu überprüfen) und testen stattdessen die Bedingung gemäß Korollar 2.43 in Schritt 5.

Zuerst bestimmen wir den maximalen Eintrag von D samt zugehöriger Indizes, also $d_{12} = 9$. Dann berechnen wir für die Distanzmatrix D den relevanten Wert $\delta_{k\ell} := d_{ik} + d_{i\ell} - d_{k\ell}$ für die Umrechnung zu D', dann ergibt sich:

$$d'_{k\ell} = d_{ij} - \frac{1}{2}(d_{ik} + d_{i\ell} - d_{k\ell}) = d_{ij} - \frac{\delta_{k\ell}}{2}.$$

Konkret gilt in unserem Beispiel dann $\delta_{k\ell} = d_{1k} + d_{1\ell} - d_{k\ell} = \delta_{\ell k}$ und somit $d'_{k\ell} = 9 - \delta_{k\ell}/2$. Die Berechnung von $\delta_{k\ell}$ und damit auch D' ist in Abbildung 2.46 angegeben.

D	1	2	3	4	5	6
1	0	9	7	8	3	5
2		0	6	3	8	4
3			0	5	6	2
4				0	7	3
5					0	4
6						0

δ	1	2	3	4	5	6
1		$0{+}9-9=0$	$0{+}7-7=0$	$0{+}8-8=0$	$0{+}3-3=0$	$0{+}5-5=0$
2			$9{+}7-6=10$	$9{+}8-3=14$	$9{+}3-8=4$	$9{+}5-4=10$
3				$7{+}8-5=10$	$7{+}3-6=4$	$7{+}5-2=10$
4					$8{+}3-7=4$	$8{+}5-3=10$
5						$3{+}5-4=4$
6						

D'	1	2	3	4	5	6
1	0	$9{-}0/2=9$	$9{-}0/2=9$	$9{-}0/2=9$	$9{-}0/2=9$	$9{-}0/2=9$
2		0	$9{-}10/2=4$	$9{-}14/2=2$	$9{-}4/2=7$	$9{-}10/2=4$
3			0	$9{-}10/2=4$	$9{-}4/2=7$	$9{-}10/2=4$
4				0	$9{-}4/2=7$	$9{-}10/2=4$
5					0	$9{-}4/2=7$
6						0

Abbildung 2.46: Beispiel: Bestimmung von D' aus D

Damit ergibt sich dann die finale Distanzmatrix D' wie in Abbildung 2.47 links angegeben. Dann berechnen wir den zugehörigen ultrametrischen Baum $T' = T(D')$ gemäß dem Algorithmus aus Abbildung 2.31. Der daraus entstandene ultrametrische Baum ist in Abbildung 2.47 rechts angegeben.

D'	1	2	3	4	5	6
1	0	9	9	9	9	9
2		0	4	2	7	4
3			0	4	7	4
4				0	7	4
5					0	7
6						0

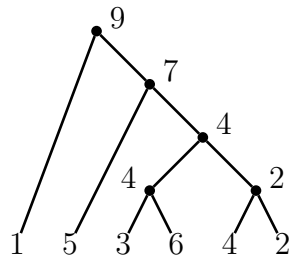


Abbildung 2.47: Beispiel: Konstruktion von T' aus D'

Nun rechnen wir Knotenmarkierungen in Kantenmarkierungen um, wie in der vorherigen Beschreibung angegeben. Im Wesentlichen ergibt sich die Kantemarkierung aus der Differenz der beiden Markierungen der inzidenten Knoten (dabei haben Blätter hierfür die Markierung 0). Dieser additive Baum für die ultrametrische Matrix D' ist in Abbildung 2.48 links angegeben (hierbei ist die additive Distanz jedoch noch doppelt so groß wie in D' angegeben).

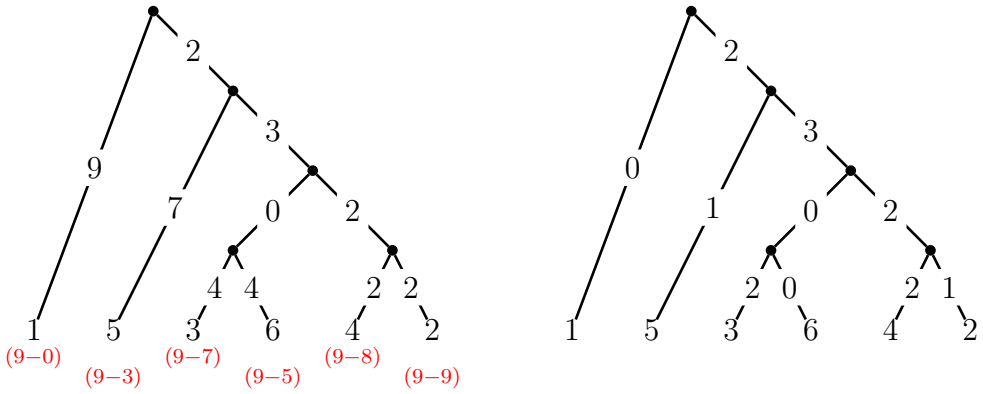


Abbildung 2.48: Beispiel: Konstruktion von T aus T'

Nun müssen noch die Gewichte der Blättern inzidenten Kanten erniedrigt werden. Im Beispiel muss also für das Blatt b mit $b \in [1:6]$ das zugehörige Gewicht um den Wert $(d_{ij} - d_{ib}) = (9 - d_{1b})$ erniedrigt werden. Diese Werte sind in rot unter den zugehörigen Blätter in Abbildung 2.48 links angegeben. Beachte dabei, dass jeder dieser Werte höchstens so groß wie das Gewicht der zu dem Blatt gehörigen inzidenten Kante ist, so dass die Gewichte nach der Erniedrigung weiteren nichtnegativ bleiben. Nach Subtraktion dieser Werte von den zu den Blättern inzidenten Kanten ergibt sich der additive Baum, der in Abbildung 2.48 rechts angegeben ist.

Dieser Baum in Abbildung 2.48 rechts besitzt jedoch noch Kanten mit Gewicht 0. Diese müssen nun kontrahiert werden, damit wird Blatt 6 zu einem internen Knoten als Nachbar von Blatt 3. Wenn man diesen kontrahierten additiven Baum mit dem additiven Baum in Abbildung 2.33 vergleicht, sieht man, dass diese identisch sind.

2.5.4 4-Punkte-Bedingung (+)

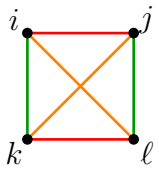
Zum Abschluss wollen wir noch eine andere Charakterisierung von additiven Matrizen angeben, die so genannte *4-Punkte-Bedingung von Buneman*.

Lemma 2.46 (Bunemans 4-Punkte-Bedingung) *Eine $n \times n$-Distanzmatrix D ist genau dann additiv, wenn für je vier Punkte $i, j, k, \ell \in [1:n]$ mit*

$$d_{i\ell} + d_{jk} = \min\{d_{ij} + d_{k\ell}, d_{ik} + d_{j\ell}, d_{i\ell} + d_{jk}\}$$

gilt, dass $d_{ij} + d_{k\ell} = d_{ik} + d_{j\ell}$.

Diese 4-Punkte-Bedingung ist in Abbildung 2.49 noch einmal illustriert. Sie besagt informal, dass bei gegebenen vier Punkten, die beiden größten Summen der Abstände von je zwei Punkten gleich sein müssen. Es lässt sich eine Analogie zur Drei-Punkte-Bedingung bei Ultrametriken erkennen.



$$d_{i\ell} + d_{jk} = \min\{d_{ij} + d_{k\ell}, d_{ik} + d_{i\ell}, d_{i\ell} + d_{jk}\}$$
$$\Rightarrow \quad d_{ij} + d_{k\ell} = d_{ik} + d_{j\ell}$$

Abbildung 2.49: Skizze: 4-Punkte-Bedingung

Beweis: $\Rightarrow$**:** Sei T ein additiver Baum für D. Wir unterscheiden jetzt drei Fälle, je nachdem, wie die Pfade in T verlaufen.

Fall 1: Im ersten Fall nehmen wir an, die Pfade von i nach j und k nach ℓ knotendisjunkt sind. Dies ist in Abbildung 2.50 illustriert. Dann ist aber $d_{i\ell} + d_{jk}$ nicht minimal und somit ist nichts zu zeigen.

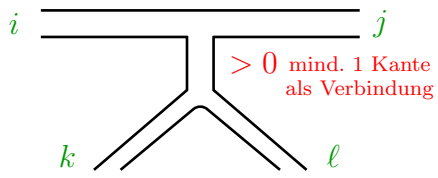


Abbildung 2.50: Skizze: Die Pfade $i \to j$ und $k \to \ell$ sind knotendisjunkt

Fall 2: Im zweiten Fall nehmen wir an, die Pfade von i nach k und j nach ℓ knotendisjunkt sind. Dies ist in Abbildung 2.51 illustriert. Dann ist jedoch ebenfalls $d_{i\ell} + d_{jk}$ nicht minimal und somit ist nichts zu zeigen.

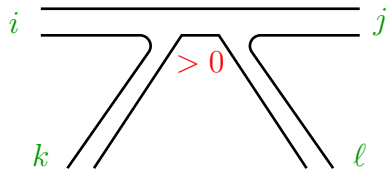


Abbildung 2.51: Skizze: Die Pfade $i \to k$ und $j \to \ell$ sind knotendisjunkt

Fall 3: Im dritten und letzten Fall nehmen wir an, die Pfade von i nach ℓ und j nach k kantendisjunkt sind und sich daher höchstens in einem Knoten schneiden. Dies ist in Abbildung 2.52 illustriert. Nun gilt jedoch, wie man leicht der Abbildung 2.52 entnehmen kann: $d_{ij} = d_{ik} + d_{j\ell} - d_{k\ell}$ und somit $d_{ij} + d_{k\ell} = d_{ik} + d_{j\ell}$.

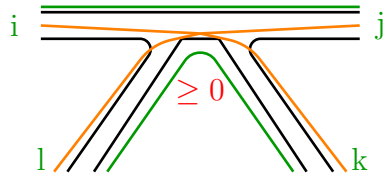


Abbildung 2.52: Skizze: Die Pfade $i \to \ell$ und $j \to k$ sind kantendisjunkt

$\Leftarrow$: Betrachte D' mit $d'_{k\ell} := d_{ij} - \frac{1}{2}(d_{ik} + d_{i\ell} - d_{k\ell})$, wobei d_{ij} ein maximaler Eintrag von D ist. Es genügt zu zeigen, dass D' ultrametrisch ist und dass gilt:

$$d_{bi} \geq \max\{d_{i\ell} - d_{b\ell} \ : \ \ell \in [1:n]\}.$$

Betrachte $p, q, r \in [1:n]$ mit $d'_{pq} \leq d'_{pr} \leq d'_{qr}$. Es ist dann zu zeigen: $d'_{pr} = d'_{qr}$. Aus $d'_{pq} \leq d'_{pr} \leq d'_{qr}$ folgt dann:

$$2d_{ij} - d_{ip} - d_{iq} + d_{pq} \ \leq \ 2d_{ij} - d_{ip} - d_{ir} + d_{pr} \ \leq \ 2d_{ij} - d_{iq} - d_{ir} + d_{qr}.$$

Daraus folgt:

$$d_{pq} - d_{ip} - d_{iq} \ \leq \ d_{pr} - d_{ip} - d_{ir} \ \leq \ d_{qr} - d_{iq} - d_{ir}$$

und weiter

$$d_{pq} + d_{ir} \ \leq \ d_{pr} + d_{iq} \ \leq \ d_{qr} + d_{ip}.$$

Aufgrund der 4-Punkt-Bedingung folgt unmittelbar

$$d_{pr} + d_{iq} = d_{qr} + d_{ip}.$$

Nach Addition von $(2d_{ij} - d_{ir})$ folgt:

$$(2d_{ij} - d_{ir}) + d_{pr} + d_{iq} = (2d_{ij} - d_{ir}) + d_{qr} + d_{ip}.$$

Nach kurzer Rechnung folgt:

$$2d_{ij} - d_{ir} - d_{ip} + d_{pr} = 2d_{ij} - d_{ir} - d_{iq} + d_{qr}$$

und somit gilt

$$2d'_{pr} = 2d'_{qr}.$$

Also ist D' nach Lemma 2.25 ultrametrisch. Wir müssen jetzt noch zeigen, dass $d_{i\ell} \leq d_{ib} + d_{b\ell}$ für alle $b, \ell \in [1 : n]$ gilt. Setzen wir in der Voraussetzung für $j = k := b$ ein, dann folgt, dass das Maximum der drei folgenden Werte

$$d_{ib} + d_{b\ell}, \qquad d_{ib} + d_{b\ell}, \qquad d_{i\ell} + d_{bb} = d_{i\ell}$$

nicht eindeutig ist. Das kann nur sein, wenn $d_{i\ell} \leq d_{ib} + d_{b\ell}$ gilt. Somit ist sowohl D' ultrametrisch als auch die in Lemma 2.42 geforderte Bedingung erfüllt. Also ist D additiv. ■

Mit Hilfe dieser Charakterisierung werden wir noch zeigen, dass es Matrizen gibt, die eine Metrik induzieren, aber keinen additiven Baum besitzen. Dieses Gegenbeispiel ist in Abbildung 2.53 angegeben. Man sieht leicht, dass die 4-Punkte-Bedingung nicht gilt und somit die Matrix nicht additiv ist. Man überprüft leicht, dass für jedes Dreieck die Dreiecksungleichung gilt, die Abstände also der Definition einer Metrik genügen.

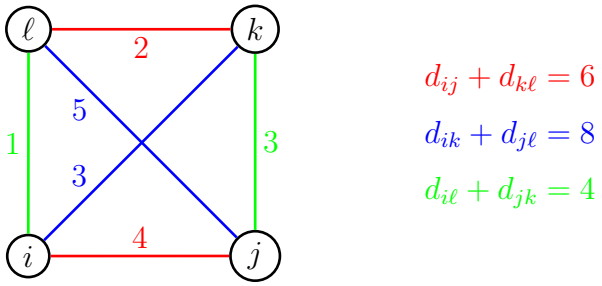


Abbildung 2.53: Gegenbeispiel: Metrische Matrix, die nicht additiv ist

2.5.5 Charakterisierung kompakter additiver Bäume

In diesem Abschnitt wollen wir eine schöne Charakterisierung kompakter additiver Bäume angeben. Zunächst benötigen wir noch einige grundlegende Definitionen aus der Graphentheorie.

Definition 2.47 *Sei G ein ungerichteter Graph. Ein Teilgraph $G' \subset G$ heißt* aufspannend*, wenn $V(G') = V(G)$ und G' zusammenhängend ist.*

Der aufspannende Teilgraph enthält also alle Knoten des aufgespannten Graphen. Nun können wir den Begriff eines Spannbaumes definieren.

Definition 2.48 *Sei G ein ungerichteter Graph. Ein Teilgraph $G' \subset G$ heißt* Spannbaum*, wenn G' ein aufspannender Teilgraph von G ist und G' ein Baum ist.*

Für gewichtete Graphen brauchen wir jetzt noch das Konzept eines minimalen Spannbaumes.

Definition 2.49 *Sei $G = (V, E, \gamma)$ ein gewichteter ungerichteter Graph und T ein Spannbaum von G. Das* Gewicht des Spannbaumes *T von G ist definiert durch*

$$\gamma(T) := \sum_{e \in E(T)} \gamma(e).$$

Ein Spannbaum T für G heißt minimal*, wenn er unter allen möglichen Spannbäumen für G minimales Gewicht besitzt, d.h.*

$$\gamma(T) \leq \min \{\gamma(T') \,:\, T' \text{ ist ein Spannbaum von } G\}.$$

Im Folgenden werden wir der Kürze wegen einen minimalen Spannbaum oft auch mit MST (engl. *minimum spanning tree*) abkürzen.

Definition 2.50 *Sei D eine $n \times n$-Distanzmatrix. Dann ist $G(D) = (V, E)$ der zu D* gehörige gewichtete Graph*, wobei*

$$\begin{aligned} V &= [1:n], \\ E &= \binom{V}{2} := \{\{v, w\} \,:\, v \neq w \in V\}, \\ \gamma(v, w) &= D(v, w). \end{aligned}$$

27.06.24

Nach diesen Definitionen kommen wir zu dem zentralen Lemma dieses Abschnittes, der kompakte additive Bäume charakterisiert.

Theorem 2.51 *Sei D eine $n \times n$-Distanzmatrix. Besitzt D einen kompakten additiven Baum T, dann ist T der minimale Spannbaum von $G(D)$ und ist eindeutig bestimmt.*

Beweis: Sei T ein kompakter additiver Baum für D (dieser muss natürlich nicht gewurzelt sein). Wir betrachten ein Knotenpaar $\{x, y\} \notin E(T)$, das durch keine Kante in T verbunden ist. Sei $p = (v_0, v_1, \ldots, v_k)$ mit $v_0 = x$ und $v_k = y$ sowie $k \geq 2$ der Pfad von x nach y in T. Dann gilt $\gamma(p) = D(x, y)$, weil T ein additiver Baum für D ist (siehe auch Abbildung 2.54).

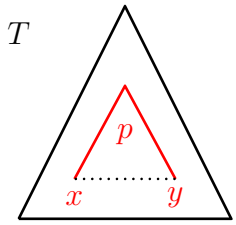


Abbildung 2.54: Skizze: Pfad $p = (v_0, \ldots, v_k)$ von x nach y in T

Da nach Definition einer Distanzmatrix alle Kantengewichte positiv sind und der Pfad aus mindestens zwei Kanten besteht, ist $\gamma(v_{i-1}, v_i) < D(x, y) = \gamma(x, y)$ für alle Kanten (v_{i-1}, v_i) des Pfades p mit $i \in [1 : k]$.

Sei jetzt T' ein minimaler Spannbaum von $G(D)$. Wir nehmen jetzt für einen Widerspruchsbeweis an, dass die Kante $\{x, y\}$ eine Kante des minimalen Spannbaumes ist, d.h. $\{x, y\} \in E(T')$. Somit verbindet die Kante $\{x, y\}$ zwei Teilbäume T'_x und T'_y von T'. Dies ist in Abbildung 2.55 illustriert.

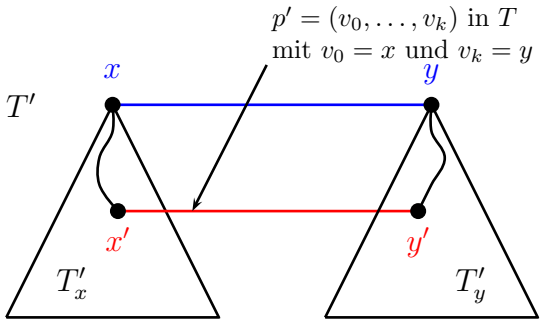


Abbildung 2.55: Skizze: Spannbaum T' von $D(G)$

Da $\{x, y\} \notin E(T)$ keine Kante im Pfad von x nach y im kompakten additiven Baum von T ist, verbindet der Pfad p in T die beiden durch Entfernen der Kante $\{x, y\}$ separierten Teilbäume T'_x und T'_y des minimalen Spannbaumes T'. Sei $\{x', y'\}$ die erste Kante auf dem Pfad p in $G(D)$ von x nach y, die nicht mehr innerhalb des Teilbaums T'_x des Spannbaums von T' verläuft. Wie wir oben gesehen haben, gilt also $\gamma(x', y') < \gamma(x, y)$.

Wir konstruieren jetzt einen neuen Spannbaum T'' für $G(D)$ wie folgt:

$$\begin{aligned} V(T'') &= V(T') = V \\ E(T'') &= (E(T') \setminus \{\{x,y\}\}) \cup \{\{x',y\}\} \end{aligned}$$

Für das Gewicht des Spannbaumes T'' gilt dann:

$$\begin{aligned} \gamma(T'') &= \sum_{e \in E(T'')} \gamma(e) \\ &= \sum_{e \in E(T')} \gamma(e) - \gamma(x,y) + \gamma(x',y') \\ &= \sum_{e \in E(T')} \gamma(e) + \underbrace{(\gamma(x',y') - \gamma(x,y))}_{<0} \\ &< \gamma(T'). \end{aligned}$$

Somit ist $\gamma(T'') < \gamma(T)$ und dies liefert den gewünschten Widerspruch zu der Annahme, dass T' ein minimaler Spannbaum von $G(D)$ ist. Also folgt aus $\{x,y\} \notin T$, dass $\{x,y\} \notin T'$.

Somit gilt für jedes Knotenpaar (x,y) mit $(x,y) \notin E(T)$, dass dann die Kante (x,y) auch nicht im minimalen Spannbaum von $G(D)$ sein kann, d.h. $(x,y) \notin T'$. Also ist eine Kante, die sich nicht im kompakten additiven Baum befindet, auch keine Kante des Spannbaums. Da ein Spannbaum jedoch genau $n-1$ Kanten besitzen muss, muss als $T = T'$ sein.

Dies gilt sogar für zwei verschiedene kompakte additive Bäume (und daher minimale Spannbäume) für $G(D)$. Somit kann es nur einen minimalen Spannbaum und also auch nur einen kompakt additiven Baum geben. ■

2.5.6 Konstruktion kompakter additiver Bäume

Die Information aus dem vorherigen Theorem kann man dazu ausnutzen, um einen effizienten Algorithmus zur Erkennung kompakter additiver Matrizen zu entwickeln. Sei D die Matrix, für den der kompakte additive Baum konstruiert werden soll, und somit $G(D) = (V, E, \gamma)$ der gewichtete Graph, für den der minimale Spannbaum berechnet werden soll.

Wir beginnen mit der Knotenmenge $V' = \{v\}$ für eine beliebiges $v \in V$ und der leeren Kantenmenge $E' = \emptyset$. Der Graph (V', E') soll letztendlich der gesuchte minimale Spannbaum werden. Wir verwenden hier Prims Algorithmus zur Konstruktion eines minimalen Spannbaumes, der wieder ein Greedy-Algorithmus sein wird. Wir

versuchen die Menge V' in jedem Schritt um einen Knoten aus $V \setminus V'$ zu erweitern. Von allen solchen Kandidaten wählen wir eine Kante minimalen Gewichtes. Dies ist schematisch in Abbildung 2.56 dargestellt.

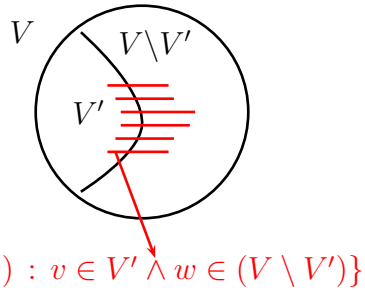


Abbildung 2.56: Skizze: Erweiterung von V'

Den Beweis, dass wir hiermit einen minimalen Spannbaum konstruieren, wollen wir an dieser Stelle nicht führen. Wir verweisen hierzu auf die einschlägige Literatur. Der formale Algorithmus ist in Abbildung 2.57 angegeben. Bis auf die blauen Zeilen ist dies genau der Pseudo-Code für Prims Algorithmus zur Konstruktion eines minimalen Spannbaums.

Die blaue for-Schleife ist dazu da, um zu testen, ob der konstruierte minimale Spannbaum wirklich ein kompakter additiver Baum für D ist. Zum einen setzen wir den

Modified_Prim (V, E, γ)

```
begin
    E' := ∅;
    V' := {v} // for some v ∈ V
    // Finally, (V', E') will be the minimal spanning tree

    while (V' ≠ V) do
        let e = (x, y), s.t. γ(x, y) := min {γ(v, w) : v ∈ V' ∧ w ∈ V \ V'};
        // Wlog let x ∈ V'
        E' := E' ∪ {e};
        V' := V' ∪ {y};
        forall (v ∈ V' \ {y}) do
            d_T(v, y) := d_T(v, x) + γ(x, y);
            if (d_T(v, y) ≠ γ(v, y)) then
                reject;
    return (V', E');
end
```

Abbildung 2.57: Algorithmus: Konstruktion eines kompakten additiven Baumes

konstruierten Abstand d_T für den konstruierten minimalen Spannbaum. Zum anderen überprüft der nachfolgende Test, ob dieser Abstand d_T mit der vorgegebenen Distanzmatrix γ übereinstimmt.

Wir müssen uns jetzt nur noch die Laufzeit überlegen. Die while-Schleife wird genau $n = |V|$ Mal durchlaufen. Danach ist $V' = V$. Die Minimumsbestimmung lässt sich sicherlich in Zeit $O(n^2)$ erledigen, da maximal n^2 Kanten zu durchsuchen sind. Daraus folgt eine Laufzeit von $O(n^3)$.

Theorem 2.52 *Sei D eine $n \times n$-Distanzmatrix. Dann lässt sich in Zeit $O(n^3)$ entscheiden, ob ein kompakter additiver Baum für D existiert. Falls dieser existiert, kann dieser ebenfalls in Zeit $O(n^3)$ konstruiert werden.*

Implementiert man Prims-Algorithmus ein wenig geschickter, z.B. mit Hilfe von Priority-Queues, so lässt sich die Laufzeit auf $O(n^2)$ senken. Für die Details verweisen wir auch hier auf die einschlägige Literatur. Auch die blaue Schleife kann insgesamt in Zeit $O(n^2)$ implementiert werden, da jede for-Schleife maximal n-mal durchlaufen wird und die for-Schleife selbst maximal n-mal ausgeführt wird.

Theorem 2.53 *Sei D eine $n \times n$-Distanzmatrix. Dann lässt sich in Zeit $O(n^2)$ entscheiden, ob ein kompakter additiver Baum für D existiert. Falls dieser existiert, kann dieser ebenfalls in Zeit $O(n^2)$ konstruiert werden.*

2.6 Exkurs: Priority Queues & Fibonacci-Heaps (*)

In diesem Abschnitt gehen wir in einen kurzen Exkurs auf eine Realisierung von Priority Queues mittels Fibonacci-Heaps ein.

2.6.1 Priority Queues

Eine *Priority Queue* ist eine Datenstruktur auf einer total geordneten Menge von Schlüsseln mit den folgenden Operationen:

empty() erzeugt eine leere Priority Queue.

insert(elt, key) fügt das Element *elt* mit dem Schlüssel *key* in die Priority Queue ein.

int size() gibt die Anzahl der Elemente in der Priority Queue zurück.

elt delete_min() liefert ein Element mit dem kleinsten Schlüssel, das in der Priority Queue enthalten ist, und entfernt dieses aus der Priority Queue.

decrease_key(elt, key) erniedrigt den Schlüssel des Elements *elt* in der Priority Queue auf den Wert *key*, sofern der gespeicherte Schlüssel größer war.

delete(elt) löscht das Element *elt* aus der Priority Queue.

key(elt) liefert den zu dem Element *elt* gehörigen Schlüssel aus der Priority Queue.

Meist wird die Operation delete nicht explizit gefordert und auch nicht benötigt. Wir werden sie im Folgenden daher auch außer Acht lasen. Die Operationen decrease_key und key operieren meistens nicht direkt auf dem Element, sondern auf einem Verweis, den insert zurückliefert, damit man die entsprechenden Elemente in der Priority Queue schnell wieder finden kann. Auf diese Details wollen wir im Folgenden auch nicht eingehen.

2.6.2 Realisierung mit Fibonacci-Heaps

Zunächst einmal wiederholen wir den Begriff eines Heaps.

Definition 2.54 *Ein* Heap *ist ein Baum, in dem in den Knoten Elemente mit Schlüssel gespeichert sind, wobei auf den Schlüsseln eine totale Ordnung gegeben ist. Dabei muss ein Heap die* Heap-Bedingung *erfüllen, die besagt, dass für jeden Knoten des Baumes mit Ausnahme der Wurzel gilt, dass der Schlüssel des Elements im Elter-Knoten kleiner gleich dem Schlüssel des betrachteten Knotens sein muss.*

Man beachte, dass es sich hier bei Heaps um beliebige gewurzelte Bäume handeln kann, die als werde eine fest begrenzte Anzahl von Kindern besitzen muss noch irgendwie balanciert sein muss

Definition 2.55 *Ein* Fibonacci-Heap *ist ein Wald, deren Bäume Heaps sind.*

Wir definieren nun noch den Rang eines Knotens.

Definition 2.56 *Der* Rang *eines Knoten in einem Fibonacci-Heap ist die Anzahl seiner Kinder.*

Für die Implementierung von Fibonacci-Heaps vereinbaren wir das Folgende (siehe Abbildung 2.58 für ein Beispiel).

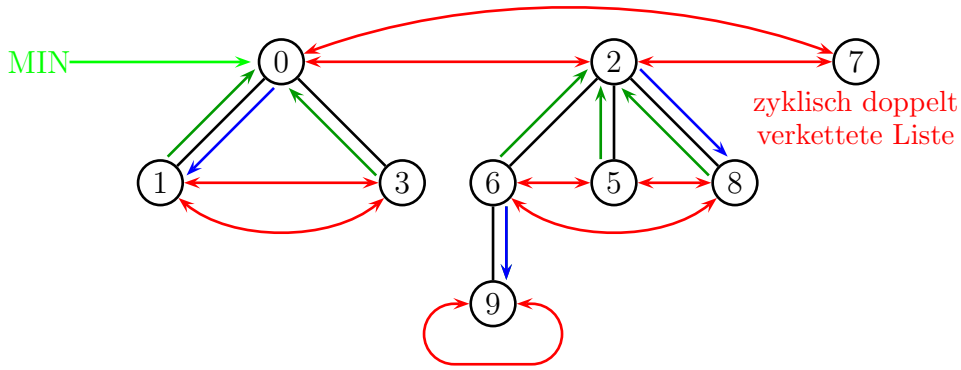


Abbildung 2.58: Beispiel: Fibonacci-Heap

- Die Wurzeln der Heaps eines Fibonacci-Heaps werden in einer zyklisch doppelt verketteten Liste gehalten. Diese Liste wird im Folgenden auch als *Wurzelliste* bezeichnet (rote Doppelpfeile im Beispiel).
- Ebenso werden die Kinder eines Knotens in einer zyklisch doppelt verketten Liste gehalten (rote Doppelpfeile im Beispiel).
- Die Wurzel eines Heaps mit dem Element mit einem kleinsten Schlüssel wird mit dem so genannten Min-Zeiger memoriert (grüner Pfeil im Beispiel).
- Jeder Knoten mit Ausnahme der Wurzeln von Heaps haben einen Verweis auf ihren Elter (dunkelgrüne Pfeile im Beispiel).
- Jeder Knoten besitzt einen Verweis auf eines seiner Kinder (nicht auf alle, denn die sind ja dann über die doppelt verkettete zyklische Kette der Geschwister-Knoten zugänglich, blaue Pfeile im Beispiel).
- Die Knoten eines Fibonacci-Heaps können markiert sein (die Wurzeln werden dabei niemals markiert sein).

Nun geben wir eine Realisierung der Operationen einer Priority Queue basierend auf einem Fibonacci-Heap an.

size: Die Anzahl der Elemente der Priority Queue wird in einer Variablen gespeichert. Diese wird zu Beginn mit 0 initialisiert und wird bei insert's um 1 erhöht und bei delete_min's um 1 erniedrigt.

insert: Ein neues Element wird als einelementiger Baum in die Wurzelliste eingefügt, dabei wird der Min-Pointer aktualisiert.

empty: Es wird eine leere Wurzelliste zurückgegeben.

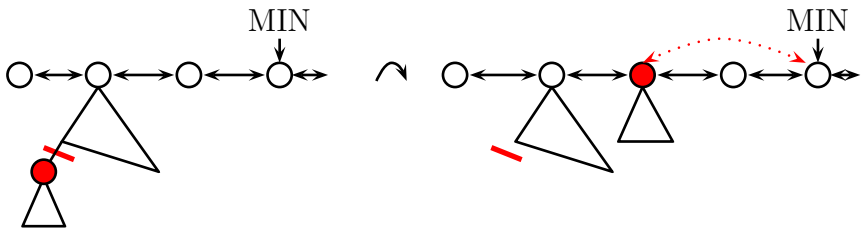


Abbildung 2.59: Skizze: Operation decrease_key

decrease_key: Die Realisierung wird im Folgenden beschrieben (siehe auch Abbildung 2.59). Man lasse sich momentan nicht dadurch verwirren, dass wir noch gar nicht wissen, wie überhaupt Bäume in der Wurzellisten entstehen können.

- Der am gegebenen Element gewurzelte Teilbaum wird abgehängt und in die Wurzelliste eingefügt.
- Der Wert in der Wurzel dieses Teilbaumes wird wie angegeben erniedrigt. (dabei bleibt die Heap-Bedingung offensichtlich erfüllt). Gegebenenfalls wird eine vorhandenen Markierung der Wurzel entfernt.
- Dann wird der Min-Pointer aktualisiert.
- Der Elter des abgehängten Elements wird markiert (außer es war eine Wurzel).
- War dieser Knoten bereits markiert, so wiederholt sich dieses Verfahren des Abhängens mit diesem Knoten bis wir auf einen unmarkierten Knoten treffen. Man beachte, dass die Wurzeln der Heaps ja prinzipiell unmarkiert sind.

  Dies kann ein mehrfaches Abhängen von Teilbäumen zur Folge haben (siehe hierzu auch Abbildung 2.60) und wird als *kaskadenartiger Schnitt* bezeichnet.

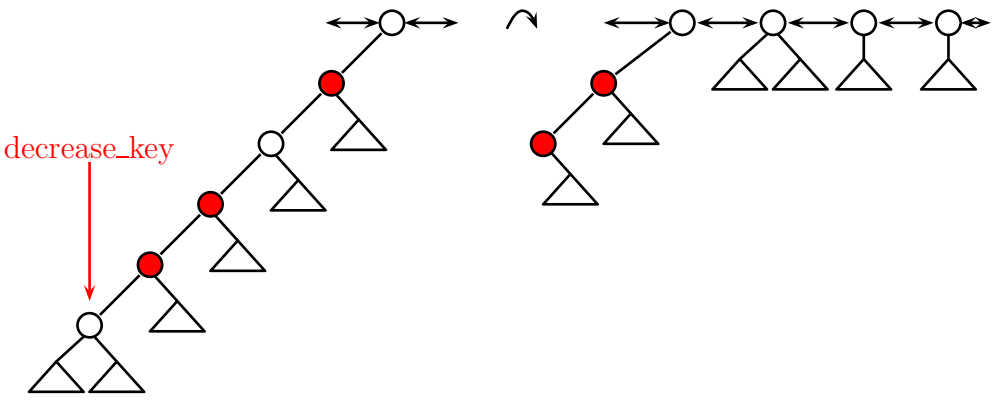


Abbildung 2.60: Skizze: kaskadenartiger Schnitt bei einer decrease_key-Operation

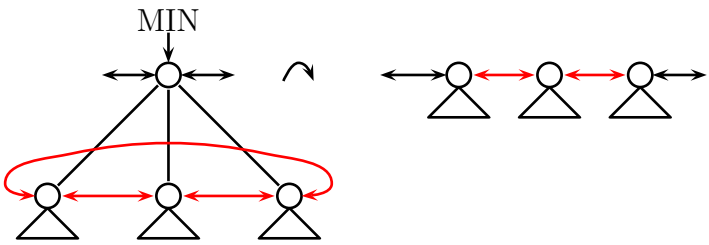


Abbildung 2.61: Skizze: Operation delete_min

delete_min: Die Realisierung erfolgt wie folgt (siehe auch Abbildung 2.61):

- Das Element, auf das der Min-Pointer zeigt, wird ausgegeben und aus seinem Heap gelöscht.
- Die doppelt verkettet Liste der Kinder des gelöschten Knotens wird in die Wurzelliste eingefügt.
- Nun folgt das so genannte Aufräumen der Wurzelliste (siehe auch Abbildung 2.62): Bäume, deren Wurzel gleichen Rang besitzen, werden nun ver-

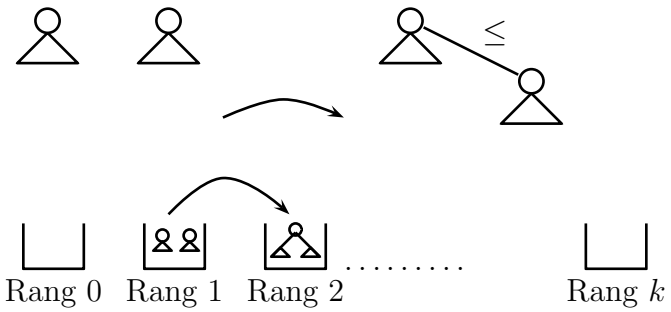


Abbildung 2.62: Skizze: Aufräumen der Wurzelliste

schmolzen bis alle Knoten der Wurzelliste einen unterschiedlichen Rang besitzen. Beim Verschmelzen ist dabei zu beachten, dass die Wurzel mit dem kleineren Schlüssel zum Elter der Wurzel des anderen Baumes wird. Somit bleibt die Heap-Bedingung für den neu entstandenen Baum erhalten.

Die Aufräumaktion selbst wird mit Hilfe eines Feldes der Größe $O(\log(n))$ geführt, das in jedem Eintrag einen Heap (bzw. einen Verweis darauf) aufnehmen kann. Die Heaps werden der Reihe nach in das Feld eingebracht, wobei ein Heap, deren Wurzel Rang k hat, in den Index k des Feldes geschrieben wird. Ist bereits ein Heap in einem Index, so werden diese beiden Heaps gemäß der Heap-Bedingung verschmolzen. Dann wird wieder versucht den neuen Heap im Feld an der richtigen Indexposition abzuspeichern.

Zum Schluss wird das Feld geleert und gleichzeitig sukzessive eine neue Wurzelliste aufgebaut.

- Beim sukzessiven Aufbau der neuen Wurzelliste wird nebenbei der neuen Min-Zeiger mitbestimmt.

2.6.3 Worst-Case Analyse

Zunächst einmal wollen wir die worst-case Kosten der einzelnen Priority Queue Operationen abschätzen.

Lemma 2.57 *Sei v ein Knoten des Fibonacci-Heaps und seien $v_1, \ldots, v_k$ seine Kinder in der Reihenfolge, wie sie Kinder von v wurden. Dann ist der Rang von v_i mindestens $i-2$.*

Beweis: Wir führen den Bewies durch Induktion nach i.

Induktionsanfang ($i \leq 2$): Es ist nichts zu zeigen

Induktionsschritt ($\rightarrow i$): Betrachte den Zeitpunkt als v_i ein Kind von v wurde. Dann hatte v bereits die Kinder $v_1, \ldots, v_{i-1}$ (eventuell auch mehr). Somit war der Rang von v zu diesem Zeitpunkt mindestens $i-1$. Da v_i ein Kind von v wurde, mussten beide Knoten denselben Rang gehabt haben. Somit musste zu diesem Zeitpunkt der Rang von v_i ebenfalls mindestens $i-1$ gewesen sein.

Wie viele Kinder kann v_i seitdem verloren haben? Maximal eines, da v_i nach einem decrease_key auf einem Kind von v_i markiert wird. Jedes weitere decrease_key hätte v_i von seinem Elter v getrennt, was ja nach Voraussetzung nicht sein kann.

Somit ist der Rang von v_i mindestens $(i-1)-1 = i-2$. ■

Für die weitere Analyse müssen wir zunächst die Fibonacci-Zahlen definieren.

Definition 2.58 *Die* Fibonacci-Zahlen *sind wie folgt definiert: $f_0 = 0$, $f_1 = 1$ und $f_{n+2} = f_{n+1} + f_n$ für $n \geq 2$.*

Für die folgende Analyse benötigen wir noch die folgenden bemerkenswerte Eigenschaft der Fibonacci-Zahlen.

Lemma 2.59 *Es gilt für alle $k \in \mathbb{N}$:*

$$f_{k+2} = 1 + \sum_{i=1}^{k} f_i.$$

Beweis: Wir führen den Beweis durch Induktion nach k.

Induktionsanfang ($k = 0$): Es gilt $f_2 = 1 = 1 + 0 = 1 + \sum_{i=1}^{0} f_i$.

Induktionsschritt ($k - 1 \to k$): Es gilt

$$1 + \sum_{i=1}^{k} f_i = f_k + 1 + \sum_{i=1}^{k-1} f_i \overset{\text{IV}}{=} f_k + f_{k+1} = f_{k+2}.$$

Damit ist der Induktionsschluss vollzogen. ■

Mit Hilfe dieser Eigenschaft der Fibonacci-Zahlen und dem vorhergehenden Lemma über Fibonacci-Heaps können wir die folgende Eigenschaft von Fibonacci-Heaps beweisen, die ihnen ihren Namen gegeben haben.

Lemma 2.60 *Sei v ein Knoten eines Fibonacci-Heaps mit Rang k, dann ist die Größe des an v gewurzelten Teilbaumes mindestens f_{k+2}.*

Beweis: Wir führen den Beweis durch Induktion nach k.

Induktionsanfang ($k \in \{0, 1\}$): Ist der Rang des Knotens gleich 0, dann ist die Größe des zugehörigen Teilbaumes genau $1 = f_2$. Ist der Rang des Knotens gleich 1, dann ist die Größe des zugehörigen Teilbaumes mindestens $2 = f_3$ (der Knoten selbst und sein Kind).

Induktionsschritt ($\to k$): Sei v ein Knoten mit Rang k. Nach Lemma 2.57 hat das i-te Kind Rang mindestens $i - 2$. Nach Induktionsvoraussetzung ist die Größe des Teilbaumes von i-ten Kind mindestens $f_{(i-2)+2} = f_i$. Somit gilt für die Größe des Teilbaumes von v:

$$|T(v)| \geq \underbrace{1}_{\text{Wurzel}} + \underbrace{\sum_{i=2}^{k} f_{(i-2)+2}}_{2.,\ldots,k.\text{ Kind}} + \underbrace{1}_{\text{1.Kind}} = 1 + \sum_{i=1}^{k} f_i = f_{k+2}$$

Damit ist der Induktionsschluss vollzogen. ■

Operation	Aufwand (worst-case)
empty	$O(1)$
size	$O(1)$
insert	$O(1)$
decrease_key	$O(\#\text{cuts}) = O(n)$
delete_min	$O(\#\text{heaps}) = O(n)$

Abbildung 2.63: Tabelle: Worst-Case Laufzeiten der Priority-Queue Operationen

Daraus folgt für die Analyse: $f(n) = \Theta(\varphi^n)$ mit $\varphi = \frac{1+\sqrt{5}}{2}$. Damit ist der Rang der Knoten durch $O(\log(n))$ beschränkt. Für eine Zusammenfassung der worst-case Laufzeiten siehe Abbildung 2.63.

2.6.4 Amortisierte Kosten bei Fibonacci-Heaps

Die worst-case Kosten sind also sehr teuer. Man überlegt sich jedoch leicht, dass teure Operationen (wie delete_min's mit großen kaskadenartigen Schnitten) nicht zu oft hintereinander vorkommen können. Daher analysieren wir noch die amortisierten Kosten der Priority Queue Operationen. Dies sind die Kosten im Mittel, wenn wir den gesamten Lebenszyklus einer Priority Queue betrachten.

Ziel: Wir versuchen mit Hilfe von zwei Sparkonten die Kosten abzuschätzen. Wir werden bei billigen Operationen schon etwas vorweg sparen, um später teure Operationen vom Sparkonto bezahlen zu können.

Als *amortisierte Kosten* bezeichnen wir dabei die Kosten, die für jeden Schritt bezahlt werden. Das sind zum einen die Kosten, die wir direkt aus der Geldbörse bezahlen, und das, was wir auf das Sparkonto einzahlen. Das Abheben vom Sparkonto betrachten wir nicht, da wir diese Kosten bereits beim Einzahlen berücksichtigt haben.

Im Folgenden beschreiben wir, bei welchen Operationen wir wieviel sparen und versuchen eine intuitive Beschreibung zu geben, warum. Dabei ist das Sparkonto A für Kosten, die beim Aufräumen der Wurzelliste entstehen, und Sparkonto B für Kosten, die bei kaskadenartigen Schnitten anfallen.

insert: 1 Kosteneinheit auf Sparkonto A.

Wir sparen schon einmal eine Kosteneinheit für das Aufräumen der Wurzelliste, da das neue Element ja als einelementiger Baum in dieses eingefügt wird und dann bei einer Aufräumaktion beteiligt ist.

decrease_key: 2 Kosteneinheiten auf Sparkonto A und 1 Kosteneinheiten auf Sparkonto B.

Wir sparen eine Kosteneinheit für das Aufräumen der Wurzelliste und zwar für die Wurzel des Teilbaumes, den wir in die Wurzelliste einfügen (siehe auch Abbildung 2.64). Darüber hinaus sparen wir zwei Kosteneinheiten für den neu

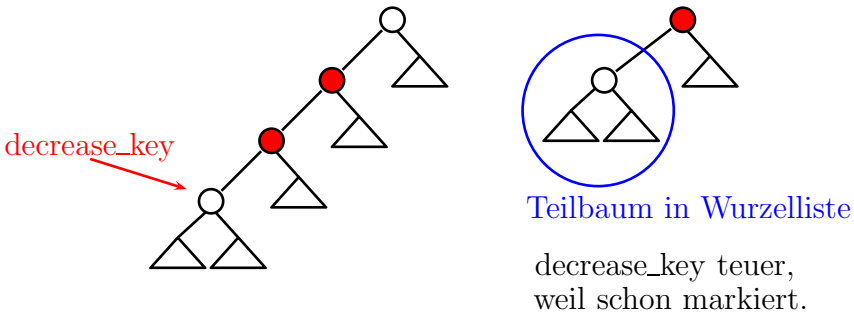


Abbildung 2.64: Skizze: Vorsorge-Kosten bei decrease_key

markierten Knoten. Eine Kosteneinheit wird auf Sparkonto B gespart, damit wir die Kosten eines zukünftigen Schnittes am markierten Knoten im Rahmen eines kaskadenartigen Schnittes begleichen können. Eine weitere Einheit wird auf Konto A gespart, da nach einem solchen Schnitt am markierten Knoten im Rahmen eines kaskadenartigen Schnittes der markierte Knoten dann Mitglied der Wurzelliste wird.

delete_min() m Kosteneinheiten auf Sparkonto A, wobei m die Anzahl der Heaps in der neu konstruierten Wurzelliste ist (dabei gilt $m = O(\log(n))$).

Wir sparen also für jeden neu konstruierten Heap der neuen Wurzelliste jeweils eine Kosteneinehit auf dem Sparkonto A, um für zukünftige Aufräumarbeiten gewappnet zu sein.

Zusammenfassend können wir sagen, dass zu jedem Zeitpunkt Folgendes gilt:

Für jeden Knoten in der Wurzelliste haben wir eine Einheit auf dem Sparkonto A und für jeden markierten Knoten haben wir je eine Einheit auf dem Sparkonto A und B.

Kostenanalyse: Wir untersuchen jetzt die anfallenden amortisierten Kosten für die einzelnen Operationen (außer für size und empty, da diese trivialerweise konstante Kosten haben).

insert: Für die insert-Operation gilt:

Einfügen in die Wurzelliste und Aktualisieren MIN-Pointer	$O(1)$
Einzahlung aufs Sparkonto A	$O(1)$
amortisierte Kosten	$O(1)$

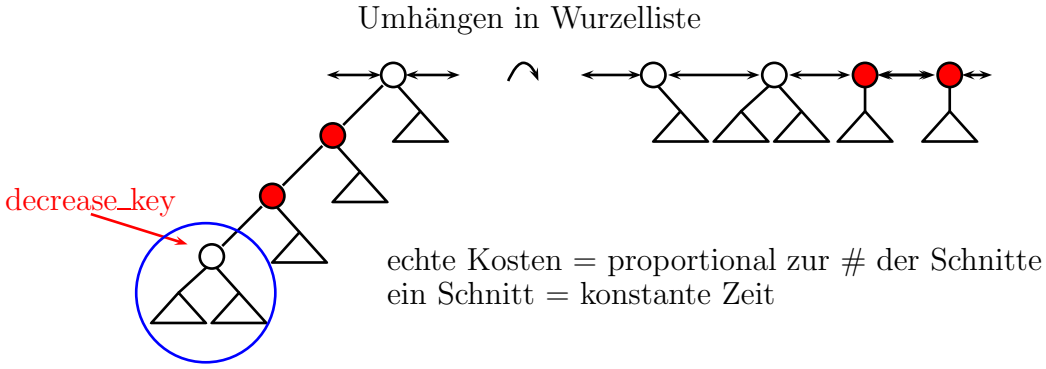


Abbildung 2.65: Skizze: Kosten für decrease_key

decrease_key: Für die decrease_key Operation gilt (siehe Abbildung 2.65):

Erster Schnitt	$O(1)$
kaskadenartiger Schnitt (durch Abheben vom Sparkonto B)	0
Einzahlung aufs Sparkonto A	$O(2)$
Einzahlung aufs Sparkonto B	$O(1)$
amortisierte Kosten	$O(1)$

delete_min: Verschmelzen von Bäumen mit gleichem Rang (siehe auch Abbildung 2.66):

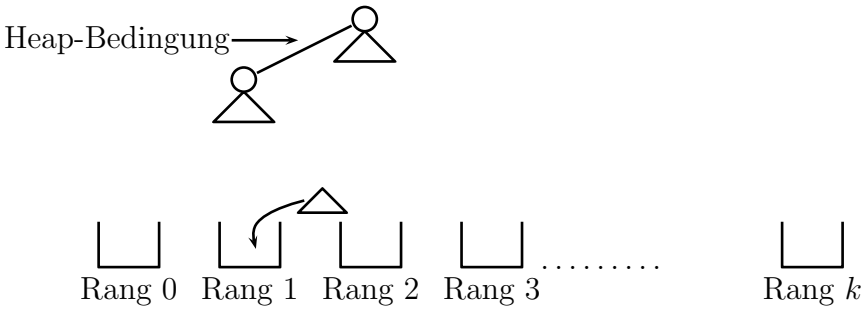


Abbildung 2.66: Skizze: Kosten für das Aufräumen der Wurzelliste

Bezahle Verschmelzen vom Sparkonto A (jeder Knoten der Wurzelliste hat ja eine Einheit eingezahlt und es sind maximal so viele Verschmelzungen möglich, wie es Knoten in der Wurzelliste gibt).

Die Kosten für die Rekonstruktion der neuen Wurzelliste müssen direkt aus der Geldbörse bezahlt werden.

Verschmelzen der Heaps (Abheben vom Sparkonto A)	0
Konstruktion der neuen Wurzelliste)	$O(\log(n))$
Einzahlung auf das Sparkonto A	$O(\log(n))$
amortisierte Kosten	$O(\log(n))$

Damit erhalten wir die in Tabelle 2.67 angegebenen amortisierten Laufzeiten für die einzelnen Operationen einer Priority Queue, die mit Hilfe eines Fibonacci-Heaps implementiert wurden.

Operation	Aufwand (worst case)	Aufwand (amortisiert)
empty	$O(1)$	$O(1)$
size	$O(1)$	$O(1)$
insert	$O(1)$	$O(1)$
decrease_key	$O(n)$	$O(1)$
delete_min	$O(n)$	$O(\log(n))$

Abbildung 2.67: Tabelle: Amortisierte Laufzeiten der Priority-Queue Operationen

Zusammenfassend können wir den folgenden Satz über die amortisierte Laufzeit einer Folge von Operationen einer Priority Queue basierend auf einem Fibonacci-Heap angeben.

Theorem 2.61 *Beginnend mit einem leeren Fibonacci-Heap kann eine Folge von ℓ Operationen (insert, decrease_key, delete_min, size) in Zeit $O(\ell + k \cdot \log(m))$ ausgeführt werden, wobei m die maximale Anzahl von Elementen im Fibonacci-Heap während der Ausführung der Operationen ist und $k \leq \ell$ die Anzahl der delete_min-Operationen.*

2.6.4.1 Potentialmethode (*)

Der Vollständigkeit halber erwähnen wir an dieser Stelle noch eine andere Vorgehensweise zur Bestimmung der amortisierten Kosten (die allerdings nicht Bestandteil der Vorlesung war).

Bei Verwendung der so genannten *Potentialmethode* zur amortisierten Analyse einer Datenstruktur wird für jeden aktuelle Zustand einer Datenstruktur ein so genanntes Potential definiert. Sei dazu im Folgenden $O_1, \ldots, O_\ell$ eine Folge von Operationen auf der Datenstruktur mit Kosten c_i für die Operation O_i. Der Zustand der Datenstruktur zu Beginn sei D_0 und nach Ausführung der Operationen $O_1, \ldots, O_i$ eben D_i. Mit $\Phi(D_i)$ wird jetzt das *Potential* der Datenstruktur zum Zeitpunkt i definiert, d.h. nach Abarbeitung der Operationen $O_1, \ldots, O_i$.

Ziel ist es nun die Potentialfunktion Φ so zu wählen, dass gilt:

$$\overline{c}_i = c_i + \Phi(D_i) - \Phi(D_{i-1}), \tag{2.1}$$

wobei mit $\overline{c}_i$ die amortisierten Kosten der i-ten Operation bezeichnet werden.

Für die Ausführung der ersten i Operationen gilt dann offensichtlich

$$\sum_{j=1}^{i} \overline{c}_i = \sum_{j=1}^{i} c_i + \Phi(D_i) - \Phi(D_0).$$

Gilt immer $\Phi(D_i) \geq \Phi(D_0)$, dann sind die $\overline{c}_i$ tatsächlich die amortisierten Kosten (oder sogar mehr). $\Phi(D_i)$ kann dabei als unser Sparkonto interpretiert werden.

Für die Fibonacci-Heaps wählt man als Potential-Funktion $\Phi(F) = w(F) + 2 \cdot m(F)$, wobei $w(F)$ die Anzahl der Heaps in der Wurzelliste und $m(F)$ die Anzahl markierter Knoten im Fibonacci-Heaps F und m sind. Damit ist zu Beginn $\Phi(F_0) = 0$. Wir überlassen es dem Leser, die Gleichung 2.1 mit dieser Potentialfunktion zu beweisen.

In der Regel ist es schwierig diese Potentialfunktion zu finden. Wir haben in unserer Analyse mit den Sparkonten versucht, diese Potentialfunktion intuitiv verständlich zu machen.

2.6.5 Laufzeit des Prim-Algorithmus mit Priority Queues

Die erweiterte Version des Prim-Algorithmus basierend auf Priority-Queues ist in Abbildung 2.68 auf Seite 163 angegeben. Dabei ist der blaue Teil wiederum nur die Erweiterung zur Erkennung kompakt additiver Matrizen bzw. zur Konstruktion eines kompakt additiven Baumes, der ja dann dem minimalen Spannbaum entspricht.

Zur Analyse des Algorithmus von Prim basierend auf Priority Queues stellen wir zunächst für die Anzahl der Aufrufe von Priority Queue Operationen fest:

- Anzahl insert's: $= n - 1 \leq n$
- Anzahl delete_min's: $= n - 1 \leq n$
- Anzahl decrease_key's: $\leq n^2$

Somit ist die Laufzeit $O(n^2 + 2n + n \log(n)) = O(n^2)$. (Die Eingabegröße ist ja schon n^2, somit ist die Laufzeit eigentlich linear). Wir rezitieren hier der Vollständigkeit halber noch einmal das Gesamtergebnis.

Theorem 2.62 *Sei D eine $n \times n$-Distanzmatrix. Dann lässt sich in Zeit $O(n^2)$ entscheiden, ob ein kompakter additiver Baum für D existiert. Falls dieser existiert, kann dieser ebenfalls in Zeit $O(n^2)$ konstruiert werden.*

PQ-Prim (set V, int $\gamma[][]$)

```
begin
    set E' := ∅;
    set V' := {v};                                   // for some node v ∈ V
    node pred[V];
    real d_T[V][V];
    real key[V];
    PQ q := new PQ.empty();

    forall (w ∈ V \ {v}) do
        q.insert(w, γ(v, w));
        pred[w] := v;
        key[w] := γ(v, w);

    while (V' ≠ V) do
        y := q.delete_min();
        x := pred[y];
        E' := E' ∪ {x, y};
        V' := V' ∪ {y};

        forall (v ∈ V' \ {y}) do
            d_T(v, y) := d_T(v, x) + γ(x, y);
            if (d_T(v, y) ≠ γ(v, y)) then
                reject;

        forall (w ∈ V \ V') do
            if (γ(y, w) < key[w]) then
                q.decrease_key(w, γ(y, w));
                pred[w] := y; key[w] := γ(y, w);
end
```

Abbildung 2.68: Algorithmus: Algorithmus von Prim mit Priority Queues und der Erweiterung zur Erkennung kompakt additiver Matrizen

2.7 Sandwich Probleme

Hauptproblem bei den bisher vorgestellten Verfahren war, dass wir dazu die Distanzen genau wissen mussten, da beispielsweise eine leicht modifizierte ultrametrische Matrix in der Regel nicht mehr ultrametrisch ist. Aus diesem Grund werden wir in diesem Abschnitt einige Problemestellungen vorstellen und lösen, die Fehler in den Eingabedaten modellieren.

2.7.1 Fehlertolerante Modellierungen

Bevor wir zur Problemformulierung kommen, benötigen wir noch einige Notationen, wie wir Matrizen zwischen zwei anderen Matrizen einschachteln können.

Notation 2.63 *Seien M und M' zwei $n \times n$-Matrizen, dann gilt $M \leq M'$, wenn $M_{i,j} \leq M'_{i,j}$ für alle $i, j \in [1 : n]$ gilt. Für drei Matrizen M, M' und M'' gilt $M \in [M', M'']$, wenn $M' \leq M \leq M''$ gilt.*

Nun können wir die zu untersuchenden Sandwich-Probleme angeben.

Additives Sandwich Problem

Eingabe: Zwei $n \times n$-Distanzmatrizen D_ℓ und D_h.
Gesucht: Eine additive Distanzmatrix $D \in [D_\ell, D_h]$, sofern eine existiert.

Ultrametrisches Sandwich Problem

Eingabe: Zwei $n \times n$-Distanzmatrizen D_ℓ und D_h.
Gesucht: Eine ultrametrische Distanzmatrix $D \in [D_\ell, D_h]$, sofern eine existiert.

Im Folgenden bezeichnet $\|M\|$ eine Norm einer Matrix. Hierbei wird diese Norm jedoch nicht eine Abbildungsnorm der durch M induzierten Abbildung sein, sondern wir werden Normen verwenden, die eine $n \times n$-Matrix als einen Vektor mit n^2 Einträgen interpretieren. Beispielsweise können wir die so genannten p-Normen verwenden:

$$\begin{aligned} \|D\|_p &= \left(\sum_{i=1}^{n} \sum_{j=1}^{n} |D_{i,j}|^p \right)^{1/p}, \\ \|D\|_\infty &= \max \{ |D_{i,j}| \ : \ i, j \in [1 : n] \} . \end{aligned}$$

Damit können wir die folgenden Approximationsprobleme definieren.

Additives Approximationsproblem

Eingabe: Eine $n \times n$-Distanzmatrix D.
Gesucht: Eine additive Distanzmatrix D', die $\|D - D'\|$ minimiert.

Ultrametrisches Approximationsproblem

Eingabe: Eine $n \times n$-Distanzmatrix D.
Gesucht: Eine ultrametrische Distanzmatrix D', die $\|D - D'\|$ minimiert.

Für die weiteren Untersuchungen benötigen wir noch einige Notationen.

Notation 2.64 *Sei M eine $n \times n$-Matrix, dann ist* $\mathrm{MAX}(M)$ *der maximale Eintrag von M, d.h.* $\mathrm{MAX}(M) = \max\{M_{i,j} : i, j \in [1:n]\}$.

Sei T ein additiver Baum mit n markierten Knoten (mit Markierungen aus $[1:n]$) und $d_T(i,j)$ der durch den additiven Baum induzierte Abstand zwischen den Knoten mit Markierung i und j, dann ist $\mathrm{MAX}(T) = \max\{d_T(i,j) : i, j \in [1:n]\}$.

Sei M eine $n \times n$-Matrix und T ein additiver Baum mit n markierten Knoten, dann schreiben wir $T \leq M$ bzw. $T \geq M$, wenn $d_T(i,j) \leq M_{i,j}$ bzw. $d_T(i,j) \geq M_{i,j}$ für alle $i, j \in [1:n]$ gilt.

Für zwei Matrizen M und M' sowie einen additiven Baum T gilt $T \in [M, M']$, wenn $M \leq T \leq M'$ gilt.

Wir wollen noch einmal kurz daran erinnern, wie man aus einem ultrametrischen Baum $T = (V, E, \mu)$ mit der Knotenmarkierung $\mu : V \to \mathbb{R}_+$ einen additiven Baum $T = (V, E, \gamma)$ mit der Kantengewichtsfunktion $\gamma : E \to \mathbb{R}_+$ erhält, indem man γ wie folgt definiert:

$$\forall (v,w) \in E : \gamma(v,w) := \frac{\mu(v) - \mu(w)}{2} \geq 0.$$

Somit können wir ultrametrische Bäume immer auch als additive Bäume auffassen (allerdings ggf. mit Kantengwichten 0, was im Folgenden nicht weiter stört).

2.7.2 Minimale Spannbäume und ultrametrische Sandwichs

In diesem Abschnitt wollen wir zeigen, wie wir mit Hilfe eines minimalen Spannbaumes das ultrametrischen Sandwich-Problem effizient lösen kann.

Definition 2.65 *Sei D_h eine obere Schrankenmatrix. Ein ultrametrischer Baum $T \leq D_h$ heißt* höchster *ultrametrischer Baum für D_h, wenn für alle ultrametrischen Bäume T' mit $T' \leq D_h$ gilt, dass $T' \leq T$.*

Für den folgenden Beweis ist die folgende Charakterisierung einer ultrametrischen Matrix hilfreich.

Lemma 2.66 *Eine $n \times n$-Distanzmatrix D ist genau dann ultrametrisch, wenn es in jedem Kreis in $G(D)$ mindestens zwei Kanten maximalen Gewichtes gibt.*

Beweis: Übungsaufgabe. ■

Wir geben in Abbildung 2.69 einen Algorithmus zur Konstruktion eines höchsten ultrametrischen Baumes für eine gegebene obere Schrankenmatrix D_h an.

1. Erzeuge einen minimalen Spannbaum S für $G(D_h)$.
2. Sortiere die Kanten in $E(S)$ absteigend nach ihrem Gewicht.
3. Konstruiere rekursiv aus S einen ultrametrischen Baum T wie folgt:
   a) Ist $|V(S)| = 1$, dann konstruiere einen Baum mit genau einem Knoten, der mit dem Label $s \in V(S)$ markiert ist.
   b) Sonst entferne die Kante $e = \{a, b\}$ mit höchstem Gewicht aus S und bestimme (beispielsweise mit Hilfe einer Tiefensuche) die beiden Zusammenhangskomponenten in $(V(S), E(S) \setminus \{e\})$; diese seien S_1 und S_2.
   c) Konstruiere rekursiv ultrametrische Teilbäume T_i für $V(S_i)$ mit Hilfe von S_i für $i \in [1:2]$.
   d) Konstruiere T, indem an eine neue Wurzel r die beiden Wurzeln von T_1 und T_2 als Kinder angehängt werden. Das Gewicht der Kante zur Wurzel des Baumes T_i ist dabei $\frac{1}{2}D_h(a, b) - h(T_i)$, wobei $h(T_i)$ die Summe der Kantengewichte auf einem einfachen Pfad von der Wurzel von T_i zu einem beliebigen Blatt von T_i ist.

Abbildung 2.69: Algorithmus: Konstruktion eines höchsten ultrametrischen Baumes

In Abbildung 2.70 ist die rekursive Konstruktion des ultrametrischen Baumes aus T_1 und T_2 wie im Schritt 3d) vom Algorithmus aus Abbildung 2.69 noch einmal illustriert (Kantengewichte in blau, Kontenmarkierung in rot).

Für die Rekursion erwähnen wir hier noch das wichtige Resultat, das wir im Folgenden stillschweigend verwenden werden.

Lemma 2.67 *Sei $G = (V, E, \gamma)$ ein gewichteter ungerichteter Graph und T ein minimaler Spannbaum für G. Sei $V' \subseteq V$ und $G' = G[V']$ der durch V' induzierte Teilgraph von G. Wenn $T[V']$ zusammenhängend ist, dann ist auch $T[V']$ ein minimaler Spannbaum für G'.*

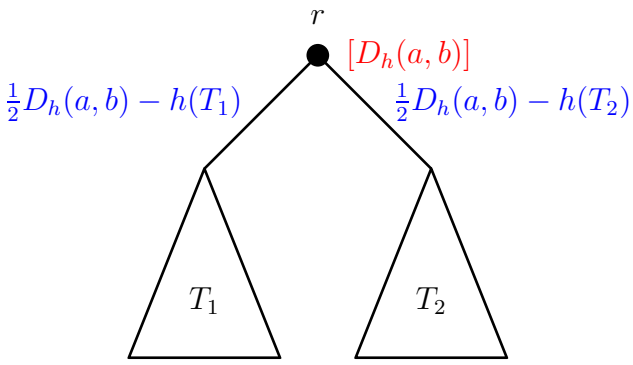


Abbildung 2.70: Skizze: Rekursive Konstruktion eines ultrametrischen Baumes mit Hilfe eine minimalen Spannbaumes

Beweis: Übungsaufgabe. ■

Nun können wir uns an den Korrektheitsbeweis des Algorithmus heranwagen, der im folgenden Satz formalisiert ist.

Theorem 2.68 *Für eine gegebene $n \times n$-Distanzmatrix D konstruiert der angegebene Algorithmus in Zeit $O(n^2)$ einen höchsten ultrametrischen Baum T für D.*

Beweis: Wir beweisen zunächst die Aussage über die Laufzeit des Algorithmus. Schritt 1 benötigt mit dem Algorithmus von Prim basierend auf Priority Queues realisiert mit Fibonacci-Heaps Zeit $O(n^2)$. Das Sortieren der Kantengewichte des minimalen Spannbaums in Schritt 2 lässt sich in Zeit $O(n \log(n))$ realisieren, da ein Spannbaum auf n Knoten genau $n-1$ Kanten besitzt. In Schritt 3 lässt sich jeder rekursive Aufruf in Zeit $O(n)$ mit Hilfe einer Tiefensuche durchführen. Da maximal $n-1$ rekursive Aufrufe nötig sind, benötigt Schritt 3 also insgesamt auch nur Zeit $O(n^2)$.

Als erstes beweisen wir, dass der Algorithmus überhaupt einen ultrametrischen Baum konstruiert. Dazu bemerken wir, dass die entsprechende Markierung an der Wurzel des rekursiv konstruierten ultrametrischen Baumes gerade $D(a,b)$ ist, wobei $\{a,b\}$ die entfernte Kante aus dem minimalen Spannbaum ist. Da wir die Kanten absteigend nach Gewicht entfernen, konstruieren wir somit einen (nicht notwendigerweise strengen) ultrametrischen Baum.

Wir merken noch an, dass wir nicht nachweisen müssen, dass am niedrigsten gemeinsamen Vorfahren der Blätter von i und j ein bestimmter Wert stehen muss. Das folgt aus der Tatsache, dass wir nur einen ultrametrischen Baum für eine beliebige ultrametrische Matrix konstruieren müssen. Diese Matrix hat als Eintrag an der Stelle

(i, j) gerade den Wert, der am niedrigsten gemeinsamen Vorfahren von i und j steht, so dass diese Bedingung nach Konstruktion erfüllt ist.

Als nächstes beweisen wir, dass der konstruierte ultrametrische Baum T die Bedingung $T \leq D$ erfüllt. Wir betrachten dazu zwei beliebige Blätter x und y in T. Sei $z = \text{lca}(x, y)$ der niedrigste gemeinsame Vorfahre von x und y in T. Dies ist in Abbildung 2.71 schematisch dargestellt.

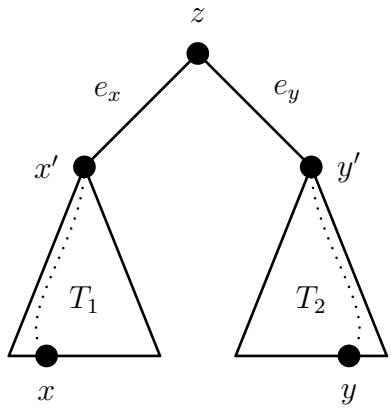


Abbildung 2.71: Skizze: Abstand zwischen x und y in T

Im Folgenden bezeichne $d_T(x, y)$ den Abstand zwischen zwei Knoten x und y im Baum T, den wir hier wieder anstelle eines ultrametrischen Baumes als additiven Baum mit entsprechenden Kantengewichten interpretieren. Also ist $d_T(x, y)$ die Summe der Kantengewichte der Kanten auf dem einfachen Pfad, der x und y in T verbindet. Sei e die Kante, die aus dem minimalen Spannbaum entfernt wird, um die Kantengewichte von e_x und e_y zu bestimmen (bzw. mit der z im ultrametrischen Baum markiert wird), d.h. $\gamma(e_x) = \frac{1}{2}D(e) - h(T_1)$ und $\gamma(e_y) = \frac{1}{2}D(e) - h(T_2)$, wobei $D(e) := D(a, b)$ für die Kante $e = \{a, b\} \in E(S)$ ist. Dann gilt:

$$\begin{aligned} d_T(x, y) &= d_{T_1}(x, x') + \gamma(e_x) + \gamma(e_y) + d_{T_2}(y', y) \\ &= h(T_1) + \frac{1}{2}D(e) - h(T_1) + \frac{1}{2}D(e) - h(T_2) + h(T_2) \\ &= D(e) \\ &= D(a, b) \\ &\leq D(x, y). \end{aligned}$$

Die letzte Ungleichung lässt sich wie folgt begründen. Sind x und y durch eine Kante im Spannbaum S verbunden, d.h. es gilt $\{x, y\} = \{a, b\}$, so ist diese Ungleichung eine triviale Gleichung. Andenfalls muss in einem minimalen Spannbaum S der Abstand von zwei nicht in S benachbarten Knoten x und y mindestens so groß sein muss, wie das Gewicht jeder Kante des einfachen Pfades, der x und y in S verbindet.

Sonst könnte man einen Spannbaum mit geringerem Gewicht als dem des minimalen Spannbaums konstruieren, indem man eine Kante mit größerem Gewicht als $\gamma(x, y)$ durch die Kante $\{x, y\}$ ersetzt. Somit konstruiert also der in Abbildung 2.69 angegebene Algorithmus einen ultrametrischen Baum konstruiert, der die obere Schrankenmatrix D einhält.

Wir müssen also nur noch zeigen, dass ein höchster ultrametrischer Baum T mit $T \leq D$ konstruiert wird. Seien wieder x und y beliebige Blätter des Baumes T und $z = \text{lca}(x, y)$ der niedrigste gemeinsame Vorfahre von x und y in T. Sei $e = \{a, b\}$ die Kante, die aus dem minimalen Spannbaum S entfernt wird, um die Kantengewichte von e_x und e_y zu bestimmen, d.h. durch Entfernen von e kommen x und y erstmals in zwei verschiedene Zusammenhangskomponenten. Somit gilt $\gamma(e_x) = \frac{1}{2}D(e) - h(T_1)$ und $\gamma(e_y) = \frac{1}{2}D(e) - h(T_2)$.

Es gilt dann also, wie eben gezeigt, $d_T(x, y) = D(e) = D(a, b)$. Weiterhin betrachten wir im minimalen Spannbaum S den einfachen Pfad p von x nach y, in dem nach Annahme die Kante $e = \{a, b\}$ enthalten sein muss. Nach Konstruktion des Algorithmus, wird auf diesem Pfad die Kante $\{a, b\}$ als erste entfernt, also muss $D(r, s) \leq D(a, b)$ für alle Kanten $\{r, s\}$ im Pfad p gelten.

Sei jetzt U ein beliebiger ultrametrischen Baum (bzw. Matrix) mit $U \leq D$. Wir müssen also zeigen, dass $U \leq T$ gilt. Weiterhin sei $p = (v_0, \ldots, v_k)$ der einfache Pfad im Spannbaum S, der $v_0 = x$ und $v_k = y$ verbindet. Da mit U auch d_U ultrametrisch ist, gilt:

$$
\begin{aligned}
d_U(x, y) &= d_U(v_0, v_k) \\
&\quad \text{wegen der ultrametrischen Dreiecksungleichung} \\
&\leq \max\{d_U(v_0, v_1), d_U(v_1, v_k)\} \\
&\quad \text{wegen der ultrametrischen Dreiecksungleichung} \\
&\leq \max\{d_U(v_0, v_1), \max\{d_U(v_1, v_2), d_U(v_2, v_k)\}\} \\
&= \max\{d_U(v_0, v_1), d_U(v_1, v_2), d_U(v_2, v_k)\} \\
&\quad \text{wegen der ultrametrischen Dreiecksungleichung} \\
&\leq \max\{d_U(v_0, v_1), d_U(v_1, v_2), \max\{d_U(v_2, v_3), d_U(v_3, v_k)\}\} \\
&= \max\{d_U(v_0, v_1), d_U(v_1, v_2), d_U(v_2, v_3), d_U(v_2, v_k)\} \\
&\vdots \\
&= \max\{d_U(v_0, v_1), d_U(v_1, v_2), \ldots, d_U(v_2, v_k)\} \\
&\leq \max\{d_U(r, s) \,:\, \{r, s\} \in p\}
\end{aligned}
$$

Durch das sukzessive Einfügen der Knoten dieses Pfades erhalten wir also mithilfe der ultrametrischen Dreiecksungleichung eine Abschätzung des Abstands eines Kno-

tenspaares durch das Maximum der Kantengewichte auf dem zugehörigen einfachen Pfad, der im Spannbaum diese beiden Knoten verbindet.

Weiterhin sei $p = (v_0, \ldots, v_k)$ der einfache Pfad im Spannbaum S, der $v_0 = x$ und $v_k = y$ verbindet. Dann gilt weiter:

$$\begin{aligned} d_U(x,y) &= \max\{d_U(r,s) \,:\, \{r,s\} \in p\} \\ &\quad \text{da nach Annahme } U \leq D \\ &\leq \max\{D(r,s) \,:\, \{r,s\} \in p\} \\ &\quad \text{da } D(a,b) \text{ auf } p \text{ maximales Gewicht hat} \\ &\leq D(a,b) \\ &\quad \text{wie wir vorhin gezeigt haben} \\ &= d_T(x,y). \end{aligned}$$

Man überlegt sich leicht, dass sie obige Argumentation auch dann stimmt, falls der Pfad p nur aus der Kante $\{x, y\}$ besteht.

Damit gilt also $U \leq T$ und der Satz ist bewiesen. ■

Man überlege sich, dass es zu jeder Distanzmatrix D mindestens einen ultrametrischen Baum T mit $T \leq D$ gibt.

02.07.24

Korollar 2.69 *Seien $D_\ell \leq D_h$ zwei $n \times n$-Distanzmatrizen, dann lässt sich in Zeit $O(n^2)$ entscheiden, ob es einen ultrametrischen Baum $T \in [D_\ell, D_h]$ gibt. Falls es einen solchen gibt, so kann dieser ebenfalls in Zeit $O(n^2)$ konstruiert werden.*

Beweis: Zuerst konstruieren wir einen höchsten ultrametrischen Baum T für D_h. Nach dem vorhergehenden Satz kann dies in Zeit $O(n^2)$ geschehen. Dann überprüfen wir, ob $T \geq D_\ell$ gilt. Falls ja, dann ist T der gesuchte ultrametrische Baum. Falls nicht, dann kann es keinen ultrametrischen Baum für das ultrametrische Sandwich-Problem mit der gegeben Distanzmatrix geben. Angenommen es gäbe einen ultrametrischen Baum $U \in [D_\ell, D_h]$, d.h.$D_\ell \leq U \leq D_h$. Da T ein höchster ultrametrischer Baum für D_h ist, gilt also $U \leq T \leq D_h$ und somit $D_\ell \leq U \leq T \leq D_h$ und somit $T \in [D_\ell, D_h]$. Dies ist der gewünschte Widerspruch zu $D_\ell \not\leq T$. ■

In Abbildung 2.72 ist noch einmal ein Beispiel zur Konstruktion eines höchsten ultrametrischen Baumes angegeben. Der minimale Spannbaum wurde mit Dem Algorithmus von Prim konstruiert, die kleinen blauen Zahlen geben die Reihenfolge an, in dem die Knoten hinzugefügt wurden.

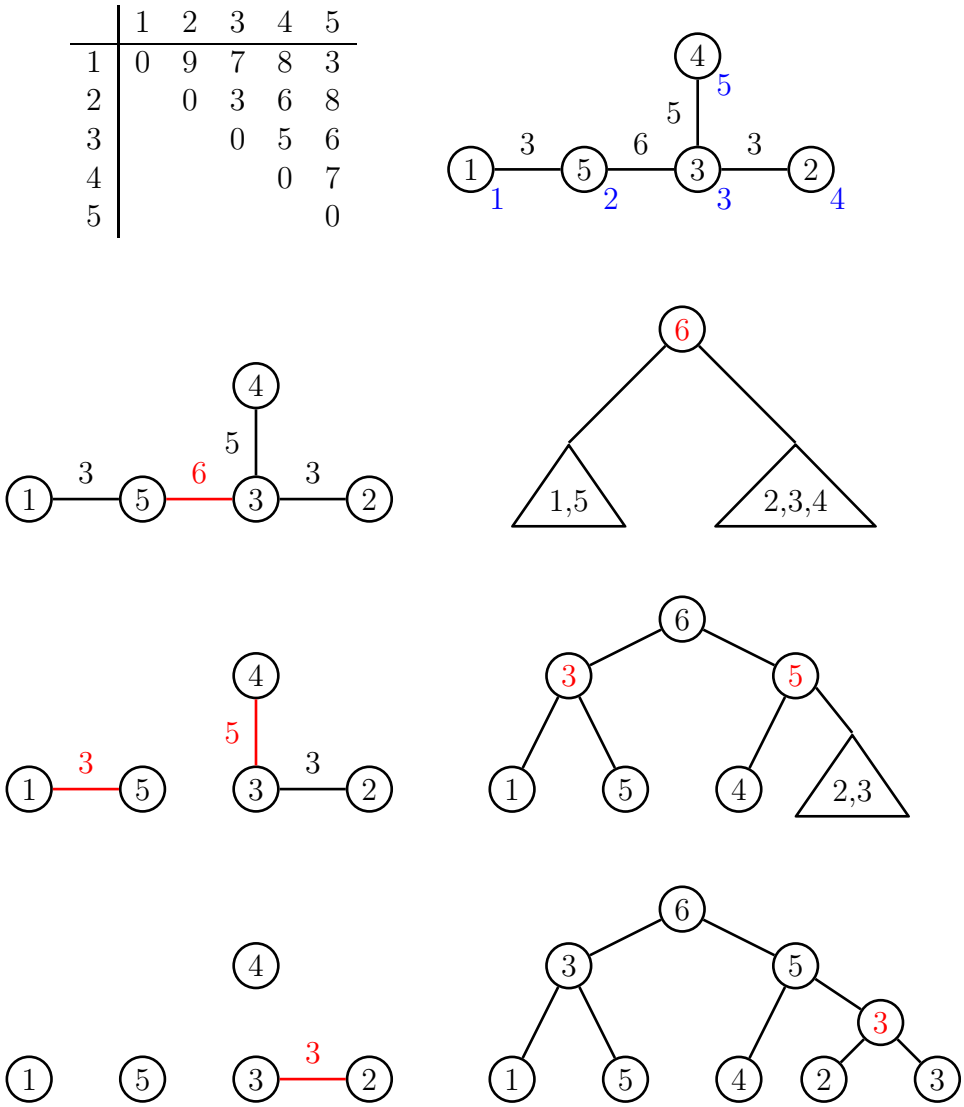


Abbildung 2.72: Beispiel: Konstruktion eines höchsten ultrametrischen Baumes

2.7.3 Asymmetrie zwischen oberer und unterer Schranke

In diesem Abschnitt wollen wir noch kurz darauf eingehen, dass man nicht sinnvollerweise analog zu höchsten ultrametrischen Bäumen, die eine obere Schrankenmatrix erfüllen, auch niedrigste ultrametrische Bäume, die eine untere Schrankenmatrix erfüllen, definieren kann.

Definition 2.70 *Sei D_ℓ eine untere Schrankenmatrix. Ein ultrametrischer Baum $T \geq D_\ell$ heißt* niedrigster *ultrametrischer Baum für D_ℓ, wenn für alle ultrametrischen Bäume T' mit $T' \geq D_\ell$ gilt, dass $T' \geq T$.*

Wir werden jetzt zeigen, dass es eine (und damit auch unendlich viele) untere Schrankenmatrizen gibt, die keinen niedrigsten ultrametrischen Baum erlauben. Dies steht völlig im Gegensatz zum Ergebnis des vorherigen Abschnittes, dass es zu jeder oberen Schrankenmatrix einen höchsten ultrametrischen Baum gibt.

L	a	b	c
a	0	x	y
b		0	z
c			0

H_1	a	b	c
a	0	x	x
b		0	z
c			0

H_2	a	b	c
a	0	x	y
b		0	x
c			0

Abbildung 2.73: Beispiel: Distanzmatrizen zum Beweis der Nichtexistenz niedrigster ultrametrischer Bäume ($x > y > 0$ und $x > z > 0$ sowie $y \neq z$)

Dazu sind in Abbilddung 2.73 drei 3×3-Distanzmatrizen definiert. Hier gilt sowohl $x > y > 0$ als auch $x > z > 0$ sowie $y \neq z$. Wie man sofort sieht, gilt $L \leq H_1$ und $L \leq H_2$ und dass L nicht ultrametrisch ist. Des Weiteren lassen sich für H_1 und H_2 sofort ultrametrische Bäume angeben, d.h. H_1 und H_2 sind ultrametrische Matrizen.

Wegen Lemma 2.66 muss für jede ultrametrische Matrix $U \geq L$ gelten, dass im Kreis (a, b, c, a) das Maximum auf mindestens zwei verschiedenen Kanten angenommen wird. Da die Kante $\{a, b\}$ das Gewicht $\gamma \geq x > \max\{y, z\}$ besitzt, muss in U entweder $U(a, c) = \gamma \geq x$ oder $U(b, c) = \gamma \geq x$ gelten. Die kleinsten solchen Matrizen mit $U \geq L$ sind dann aber gerade H_1 und H_2. Da aber offensichtlich weder $H_1 \leq H_2$ noch $H_1 \geq H_2$ gilt, kann es keine niedrigste ultrametrische Matrix U mit $U \geq L$ geben. Alternativ kann auch $U(a, c) = U(b, c) = \gamma \geq x$ gelten. Auch dann ist $U \geq H_1$ und $U \geq H_2$.

2.7.4 Ultrametrisches Approximationsproblem

Wir wollen nun noch das ultrametrische Approximationsproblem für die Maximumsnorm lösen. Zunächst einmal zeigen wir das folgende Resultat.

Lemma 2.71 *Seien $G = (V, E, \gamma)$ bzw. $G' = (V, E, \gamma + c)$ gewichtete ungerichtete Graphen, wobei $\gamma + c : E \to \mathbb{R}_+ : e \mapsto \gamma(e) + c$ für ein $c \in \mathbb{R}_+$ ist. T ist genau dann ein minimaler Spannbaum G, wenn T ein minimaler Spannbaum von G' ist.*

Beweis: Übungsaufgabe. ■

Definieren wir nun noch für die folgenden Überlegungen die so genannte Translation einer Distanzmatrix.

Definition 2.72 *Sei D eine $n \times n$-Distanzmatrix und sei $c > -\mathrm{MIN}(D)$, wobei $\mathrm{MIN}(D) := \min\{D(i,j) : i \neq j \in [1:n]\}$, dann ist die* Translation *einer Distanzmatrix um c wie folgt definiert:*

$$D^{(c)}(i,j) = \begin{cases} D(i,j) & \text{falls } i = j, \\ D(i,j) + c & \text{falls } i \neq j. \end{cases}$$

Man möge sich überlegen, dass nach der obigen Definitionen jede Translation einer Distanzmatrix wieder eine Distanzmatrix liefert.

Wir zeigen nun, dass sich durch eine Translation die zugehörigen ultrametrischen Bäume kaum ändern.

Lemma 2.73 *Sei D eine Distanzmatrix und $c \in \mathbb{R}_+$ Der höchste ultrametrische Baum für $D^{(c)}$ lässt sich aus dem höchsten ultrametrischen Baum T für D konstruieren, indem die Kantengewichte zu den zu den Blättern inzidenten Kanten um $c/2$ erhöht werden.*

Beweis: Nach Lemma 2.71 ist die Struktur eines minimalen Spannbaumes für die verschobene Distanzmatrix unverändert. Einzig und allein die Kantengewichte ändern sich.

Im Schritt 3d) des Algorithmus ändern sich nur die Kantengewichte zu neu hinzukonstruierten Wurzel. Diese wurden mittels $\gamma(e_i) := \frac{1}{2}D(e) - h(T_i)$ gesetzt. An den Gewichten der inneren Kanten (also zwischen inneren Knoten) ändert sich daher nichts, sondern nur die Gewichte der zu Blättern inzidenten Kanten. Diese Gewichte der zu den Blättern inzidenten Kanten werden dann jeweils um den Wert $c/2$ erhöht. ■

Für das ultrametrische Approximationsproblem in der Maximumsnorm müssen wir jetzt nur eine translatierte Distanzmatrix D' finden, so dass $\|D - D'\|_\infty = \varepsilon$ minimal wird und D' ultrametrisch ist. Das ist äquivalent dazu ein minimales ε zu finden, so dass $D' \in [D^{(-\varepsilon)} : D^{(+\varepsilon)}]$ gilt. Wir konstruieren also weiterhin einen höchsten ultrametrischen Baum T für D. Dann bestimmen wir den maximalen Abstand von T und D und setzen

$$\varepsilon := \frac{\|D - d_T\|_\infty}{2}.$$

Dann ist der zu $D^{(\varepsilon)}$ gehörige höchste ultrametrische Baum der gesuchte.

Theorem 2.74 *Sei D eine $n \times n$-Distanzmatrix, dann lässt sich eine ultrametrische Matrix D' mit minimalen Abstand zu D in der Maximumsnorm in Zeit $O(n^2)$ konstruieren, d.h. eine ultrametrische Matrix D', die $\|D - D'\|_\infty$ minimiert.*

2.7.5 Komplexitätsresultate

Fassen wir die positiven Ergebnisse noch einmal in der folgenden Abbildung 2.74 zusammen und erwähnen, dass die nicht behandelten Probleme alle $\mathcal{NP}$-Hart sind.

Problem	ultrametrisch	additiv
Sandwich	$O(n^2)$	NPC
Approximation ($\|\cdot\|_\infty$)	$O(n^2)$	NPC
Approximation ($\|\cdot\|_p$)	NPC	NPC

Abbildung 2.74: Übersicht: Komplexitäten der Approximationsprobleme

Für die Beweise zu den Resultaten zur $\mathcal{NP}$-Vollständigkeit verweisen wir auf die Originalliteratur von M. Farach, S. Kannan und T. Warnow.

2.8 Alternative Lösung für das Sandwich-Problem(*)

Dieser Abschnitt wurde nicht in der Vorlesung behandelt und ist nur der Vollständigkeit halber aufgenommen worden.

2.8.1 Eine einfache Lösung

In diesem Abschnitt wollen wir noch eine andere Lösung angeben, um das ultrametrische Sandwich-Problem zu lösen. Wir beginnen mit einer einfachen, nicht allzu effizienten Lösung, um die Ideen hinter der Lösung zu erkennen. Im nächsten Abschnitt werden wir dann eine effizientere Lösung angeben, die von der Idee her im Wesentlichen gleich sein wird, aber dort nicht so leicht bzw. kaum zu erkennen sein wird.

Zuerst zeigen wir, dass, wenn es einen ultrametrischen Baum gibt, der der Sandwich-Bedingung genügt, es auch einen ultrametrischen Baum gibt, dessen Wurzelmarkierung gleich dem maximalem Eintrag der Matrix D_ℓ ist. Dies liefert einen ersten Ansatzpunkt für einen rekursiven Algorithmus.

Lemma 2.75 *Seien D_ℓ und D_h zwei $n \times n$-Distanzmatrizen. Wenn ein ultrametrischer Baum T mit $T \in [D_\ell, D_h]$ existiert, dann gibt es einen ultrametrischen Baum T' mit $T' \in [D_\ell, D_h]$ und $\mathrm{MAX}(T') = \mathrm{MAX}(D_\ell)$.*

Beweis: Wir beweisen die Behauptung durch Widerspruch. Wir nehmen also an, dass es keinen ultrametrischen Baum $T' \in [D_\ell, D_h]$ mit $\mathrm{MAX}(T') = \mathrm{MAX}(D_\ell)$ gibt.

Sei $T' \in [D_\ell, D_h]$ ein ultrametrischer Baum, der $s := \mathrm{MAX}(T') - \mathrm{MAX}(D_\ell)$ minimiert. Nach Voraussetzung gibt es mindestens einen solchen Baum und nach unserer Annahme für den Widerspruchsbeweis ist $s > 0$.

Wir konstruieren jetzt aus T' einen neuen Baum T'', indem wir nur die Kantengewichte der Kanten in T'' ändern, die zur Wurzel von T' bzw. T'' inzident sind. Zuerst definieren wir $\alpha := \frac{1}{2}\min\{\gamma(e), s\} > 0$. Wir bemerken, dass dann $\alpha \leq \frac{\gamma(e)}{2}$ und $\alpha \leq \frac{s}{2}$ gilt.

Sei also e ein Kante, die zur Wurzel von T'' inzident ist. Dann setzen wir

$$\gamma''(e) = \gamma'(e) - \alpha.$$

Hierbei bezeichnet γ' bzw. γ'' die Kantengewichtsfunktion von T' bzw. T''. Zuerst halten wir fest, dass die Kantengewichte der Kanten, die zur Wurzel inzident sind, weiterhin positiv sind, da

$$\gamma(e) - \alpha \;\geq\; \gamma(e) - \frac{\gamma(e)}{2} \;=\; \frac{\gamma(e)}{2} \;>\; 0.$$

Da wir Kantengewichte nur reduzieren, gilt offensichtlich weiterhin $T'' \leq D_h$. Wir müssen also nur noch zeigen, dass auch $D_\ell \leq T''$ weiterhin gilt. Betrachten wir hierzu zwei Blätter v und w in T'' und die Wurzel $r(T'')$ von T''. Wir unterscheiden jetzt zwei Fälle, je nachdem, ob der kürzeste Weg von v nach w über die Wurzel führt oder nicht.

Fall 1 ($\mathbf{lca}(v, w) \neq r(T'')$): Dann wird der Abstand von $d_{T''}$ gegenüber $d_{T''}$ für diese Blätter nicht verändert und es gilt:

$$d_{T''}(v, w) = d_{T'}(v, w) \geq D_\ell(v, w).$$

Fall 2 ($\mathbf{lca}(v, w) = r(T'')$): Dann werden die beiden Kantengewichte der Kanten auf dem Weg von v zu w die inzident zur Wurzel sind um jeweils α erniedrigt und wir erhalten:

$$\begin{aligned}
d_{T''}(v, w) &= d_{T'}(v, w) - 2\alpha \\
&\quad \text{da } d_{T'}(v, w) = s + \mathrm{MAX}(D_\ell) \\
&= s + \mathrm{MAX}(D_\ell) - 2\alpha \\
&\quad \text{da } 2\alpha \leq s \\
&\geq s + \mathrm{MAX}(D_\ell) - s \\
&= \mathrm{MAX}(D_\ell) \\
&\geq D_\ell(v, w).
\end{aligned}$$

Also ist $D_\ell \leq T''$. Somit haben wir einen ultrametrischen Baum $T'' \in [D_\ell, D_h]$ konstruiert, für den

$$\begin{aligned} \mathrm{MAX}(T'') - \mathrm{MAX}(D_\ell) &\leq \mathrm{MAX}(T') - 2\alpha - \mathrm{MAX}(D_\ell) \\ &\quad \text{da } \alpha > 0 \\ &< \mathrm{MAX}(T') - \mathrm{MAX}(D_\ell) \\ &= s \end{aligned}$$

gilt. Dies ist offensichtlich ein Widerspruch zur Wahl von T' und das Lemma ist bewiesen. ■

Das vorherige Lemma legt die Definition niedriger ultrametrische Bäume nahe.

Definition 2.76 *Seien $D_\ell \leq D_h$ zwei Distanzmatrizen. Ein ultrametrischer Baum $T \in [D_\ell, D_h]$ heißt* niedrig, *wenn* $\mathrm{MAX}(T) = \mathrm{MAX}(D_\ell)$.

Um uns im weiteren etwas leichter zu tun, benötigen wir noch einige wichtige Notationen.

Notation 2.77 *Sei $T = (V, E)$ ein gewurzelter Baum. Dann bezeichnet $\mathcal{L}(T)$ die Menge der Blätter von T. Für $v \in \mathcal{L}(T)$ bezeichnet*

$$\mathcal{L}(T, v) := \{w \in \mathcal{L}(T) \,:\, \mathrm{lca}(v, w) \neq r(T)\}$$

die Menge aller Blätter die sich im selben Teilbaum der Wurzel von T befinden wie v selbst.

Aus dem vorherigen Lemma und den Notationen folgt jetzt unmittelbar das folgende Korollar.

Korollar 2.78 *Für jeden niedrigen ultrametrischen Baum $T \in [D_\ell, D_h]$ gilt:*

$$\forall x, y \in \mathcal{L}(T) : (D_\ell(x, y) = \mathrm{MAX}(D_\ell)) \Rightarrow (d_T(x, y) = \mathrm{MAX}(D_\ell)).$$

Bevor wir zum zentralen Lemma kommen, müssen wir noch eine weitere grundlegende Definition festlegen.

Definition 2.79 *Seien $D_\ell \leq D_h$ zwei $n \times n$-Distanzmatrizen und seien $k, \ell \in [1:n]$, dann ist der Graph $G_{k,\ell} = (V, E_{k,\ell})$ wie folgt definiert:*

$$\begin{aligned} V &:= [1:n], \\ E_{k,l} &:= \{(i,j) \,:\, D_h(i,j) < D_\ell(k,\ell)\}. \end{aligned}$$

Mit $C(G_{k,\ell}, v)$ bezeichnen wir die Zusammenhangskomponente von $G_{k,\ell}$, die den Knoten v enthält.

Nun kommen wir zu dem zentralen Lemma für unseren einfachen Algorithmus, das uns beschreibt, wie wir mit Kenntnis des Graphen $G_{k\ell}$ (oder besser dessen Zusammenhangskomponenten) den gewünschten ultrametrischen Baum konstruieren können.

Lemma 2.80 *Seien $D_\ell \leq D_h$ zwei $n \times n$-Distanzmatrizen und seien $k, \ell \in [1:n]$ so gewählt, dass $D_\ell(k,\ell) = \text{MAX}(D_\ell)$. Wenn ein niedriger ultrametrischer Baum $T \in [D_\ell, D_h]$ existiert, dann gibt es auch einen niedrigen ultrametrischen Baum $T' \in [D_\ell, D_h]$ mit*

$$\mathcal{L}(T', v) = V(C(G_{k,\ell}, v))$$

für alle $v \in \mathcal{L}(T)$.

Beweis: Wir beweisen zuerst die folgende Behauptung:

$$\forall T \in [D_\ell, D_h] : V(C(G_{k,l}, v)) \subseteq \mathcal{L}(T, v).$$

Wir führen diesen Beweis durch Widerspruch. Sei dazu $x \in V(C(G_{k,\ell}, v)) \setminus \mathcal{L}(T, v)$. Da $x \in V(C(G_{k,\ell}, v))$ gibt es einen Pfad p von v nach x in $G_{k\ell}$ (siehe dazu auch Abbildung 2.75). Sei $(y, z) \in p$ die erste Kante des Pfades von v nach x in $G_{k\ell}$, so dass $y \in \mathcal{L}(t, v)$, aber $z \notin \mathcal{L}(T, v)$ gilt. Da $(y, z) \in E(C(G_{k,\ell}, v)) \subseteq E_{k,\ell}$ ist, gilt $D_h(y, z) < D_\ell(k, \ell)$.

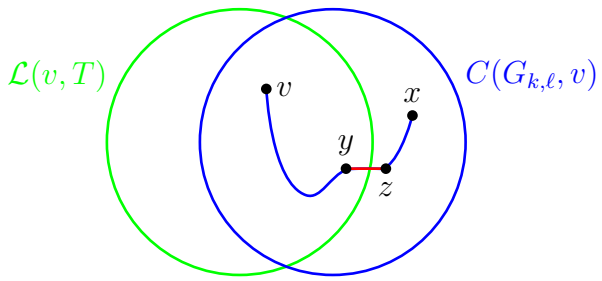


Abbildung 2.75: Skizze: $\mathcal{L}(T, v)$ und $C(G_{k,\ell}, v)$

Da $y \in \mathcal{L}(T, v)$, aber $z \notin \mathcal{L}(T, v)$ ist, gilt $\text{lca}(y, z) = r(T)$. Somit ist

$$d_T(y, z) \geq \text{MAX}(D_\ell) = D_\ell(k, \ell).$$

Daraus folgt unmittelbar, dass

$$d_T(y, z) \geq D_\ell(k, \ell) > D_h(y, z).$$

Dies ist aber offensichtlich ein Widerspruch zu $T \in [D_\ell, D_h]$ und somit ist die Behauptung gezeigt.

Wir zeigen jetzt, wie ein Baum T umgebaut werden kann, so dass er die gewünschte Eigenschaft des Lemmas erfüllt. Sei also T ein niedriger ultrametrischer Baum mit $T \in [D_\ell, D_h]$. Sei weiter $S := \mathcal{L}(T, v) \setminus V(C(G_{k,\ell}, v))$. Ist S leer, so ist nichts mehr zu zeigen. Sei also $s \in S$ und $x \in V(C(G_{k,\ell}, v))$. Nach Wahl von s und x gibt es in $G_{k,\ell}$ keinen Pfad von s nach x. Somit gilt

$$D_h(x, s) \geq D_\ell(k, \ell) = \text{MAX}(D_\ell) = \text{MAX}(T) \geq d_T(x, s) \geq D_\ell(x, s).$$

Wir bauen jetzt T zu T' wie folgt um. Betrachte den Teilbaum T_1 der Wurzel $r(T)$ von T der sowohl x als auch s enthält. Wir duplizieren T_1 zu T_2 und hängen an die Wurzel von T'' sowohl T_1 als auch T_2 an. In T_1 entfernen wir alle Blätter aus S und in T_2 entfernen wir alle Blätter aus $V(C(G_{k,\ell}, v))$ (siehe auch Abbildung 2.76). Anschließend räumen wir in den Bäumen noch auf, indem wir Kanten entfernen, die zu keinen echten Blättern mehr führen. Ebenso löschen wir Knoten, die nur noch ein Kind besitzen, indem wir dieses Kind zum Kind des Elters des gelöschten Knoten machen. Letztendlich erhalten wir einen neuen ultrametrischen Baum T''.

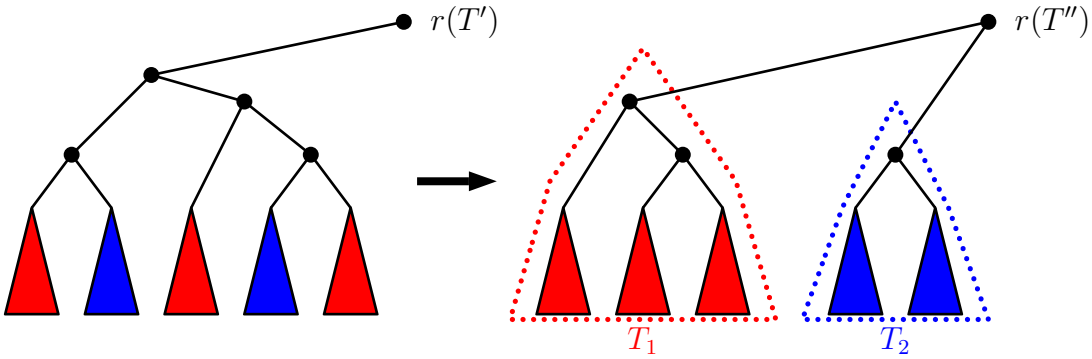


Abbildung 2.76: Skizze: Umbau des niedrigen ultrametrischen Baumes

Wir zeigen jetzt, dass $T'' \in [D_\ell, D_h]$ ist. Da wir die Knotenmarkierung der überlebenden Knoten nicht ändern, bleibt der ultrametrische Baum niedrig. Betrachten

wir zwei Blätter x und y , die nicht zu T_1 oder T_2 gehören. Da der Pfad in T'' derselbe wie in T' ist, gilt weiterhin

$$D_\ell(x,y) \leq d_{T''}(x,y) \leq D_h(x,y).$$

Gehört ein Knoten x zu T_1 oder T_2 und der andere Knoten y nicht zu T_1 und T_2, so ist der Pfad nach die Duplikation bezüglich des Abstands derselbe. Es gilt also wiederum

$$D_\ell(x,y) \leq d_{T''}(x,y) \leq D_h(x,y).$$

Gehören beide Knoten v und w entweder zu T_1 oder zu T_2, dann hat sich der Pfad nach der Duplikation bzgl. des Abstands auch wieder nicht geändert, also gilt

$$D_\ell(x,y) \leq d_{T''}(x,y) \leq D_h(x,y).$$

Es bleibt der Fall zu betrachten, dass ein Knoten zu T_1 und einer zu T_2 gehört. Hier hat sich der Pfad definitiv geändert, da er jetzt über die Wurzel von T'' führt. Sei s der Knoten in T_1 und x der Knoten in T_2. Wir haben aber bereits gezeigt, dass für solche Knoten gilt:

$$D_h(x,s) \geq d_{T''}(x,s) \geq D_\ell(x,s).$$

Somit ist der Satz bewiesen. ■

Aus diesem Beweis ergibt sich unmittelbar die folgende Idee für unseren Algorithmus. Aus der Kenntnis des Graphen $G_{k\ell}$ für eine größte untere Schranke $D_\ell(k,\ell)$ für zwei Spezies bzw. Taxa k und ℓ beschreiben uns die Zusammenhangskomponenten die Partition der Spezies bzw. Taxa, wie diese in den verschiedenen Teilbäumen an der Wurzel des ultrametrischen Baumes hängen müssen, sofern es überhaupt einen gibt. Somit können wir die Spezies bzw. Taxa partitionieren und für jede Menge der Partition einen eigenen ultrametrischen Baum rekursiv konstruieren, deren Wurzeln dann die Kinder der Gesamtwurzel des zu konstruierenden ultrametrischen Teilbaumes werden. Damit ergibt sich der folgende, in Abbildung 2.77 angegebene Algorithmus.

Für die Korrektheit müssen wir nur noch zeigen, dass die Partition nicht trivial ist, d.h., dass der Graph $G_{k,\ell}$ nicht zusammenhängend ist.

Lemma 2.81 *Seien $D_\ell \leq D_h$ zwei $n \times n$-Distanzmatrizen und seien $k,\ell \in [1:n]$ so gewählt, dass $D_\ell(k,\ell) = \text{MAX}(D_\ell)$ gilt. Wenn $G_{k,\ell}$ zusammenhängend ist, dann kann es keine ultrametrische Matrix $U \in [D_\ell, D_h]$ geben.*

Beweis: Angenommen, es gäbe eine ultrametrische Matrix $U \in [D_\ell, D_h]$. Da $G_{k,\ell}$ zusammenhängend ist, gibt es einen Pfad $p = (v_1, \ldots, v_m)$ mit $v_1 = k$ und $v_m = \ell$ in

- Bestimme $k, \ell \in [1:n]$, so dass $D_\ell(k,\ell) = \text{MAX}(D_\ell)$. $O(n^2)$
- Konstruiere $G_{k,\ell}$. $O(n^2)$
- Bestimme die Zusammenhangskomponenten $C_1, \ldots, C_m$ von $G_{k,\ell}$. $O(n^2)$
- Konstruiere rekursiv ultrametrische Bäume für die einzelnen Zusammenhangskomponenten. $\sum_{i=1}^{m} T(|C_i|)$
- Baue aus den Teillösungen $T_1, \ldots, T_m$ für $C_1, \ldots, C_m$ einen ultrametrischen Baum, indem man die Wurzeln der $T_1, \ldots, T_m$ als Kinder an eine neue Wurzel hängt, die als Knotenmarkierung $\text{MAX}(D_\ell)$ erhält. $O(n)$

Abbildung 2.77: Algorithmus: Algorithmus für das ultrametrische Sandwich-Problem

$G_{k,\ell}$. Aufgrund der ultrametrischen Matrix U gilt dann (da diese ja die ultrametrische Dreiecksungleichung erfüllt):

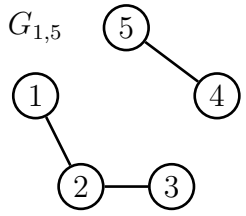
$$\begin{aligned}
U(k,\ell) &\leq \max\{U(k,v_2), U(v_2,\ell)\} \\
&\leq \max\{U(k,v_2), \max\{U(v_2,v_3), U(v_3,\ell)\}\} \\
&= \max\{U(k,v_2), U(v_2,v_3), U(v_3,\ell)\} \\
&\leq \max\{U(k,v_2), U(v_2,v_3), \max\{U(v_3,v_4), U(v_4,\ell)\}\} \\
&= \max\{U(k,v_2), U(v_2,v_3), U(v_3,v_4), U(v_4,\ell)\} \\
&\leq \max\{U(k,v_2), U(v_2,v_3), U(v_3,v_4), \max\{U(v_4,v_5), U(v_6,\ell)\}\} \\
&= \max\{U(k,v_2), U(v_2,v_3), U(v_3,v_4), U(v_4,v_5), U(v_5,\ell)\} \\
&\leq \vdots \\
&\leq \max\{U(k,v_2), U(v_2,v_3), \ldots, U(v_{m-1},\ell)\} \\
&\quad \text{da ja } k = v_1 \text{ und } \ell = v_m \\
&= \max\{U(v_1,v_2), U(v_2,v_3), \ldots, U(v_{m-1},v_m)\} \\
&\quad \text{da } U \in [D_\ell, D_h] \\
&\leq \max\{D_h(v_1,v_2), D_h(v_2,v_3), \ldots, D_h(v_{m-1},v_m)\} \\
&\quad \text{nach Konstruktion von } G_{k,\ell} \\
&< D_\ell(k,\ell)
\end{aligned}$$

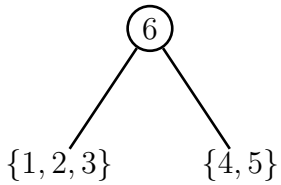
Damit erhalten wir also $U(k,\ell) < D_\ell(k,\ell)$. Dies ist aber offensichtlich ein Widerspruch dazu, dass $U \in [D_\ell, D_h]$. ■

Damit ist die Korrektheit bewiesen. In Abbildung 2.78 ist ein Beispiel zur Illustration der Vorgehensweise des einfachen Algorithmus angegeben.

D_ℓ	1	2	3	4	5
1	0	1	2	3	6
2		0	4	5	5
3			0	4	5
4				0	1
5					0

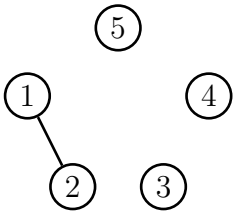
D_h	1	2	3	4	5
1	0	3	6	8	8
2		0	5	6	8
3			0	6	8
4				0	3
5					0

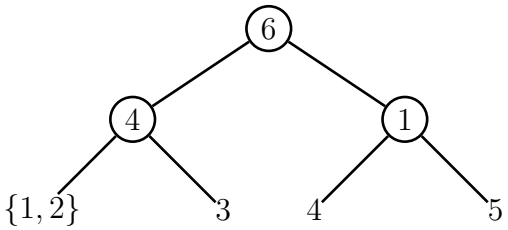




D_ℓ	1	2	3	4	5
1	0	1	2	3	6
2		0	4	5	5
3			0	4	5
4				0	1
5					0

D_h	1	2	3	4	5
1	0	3	6	8	8
2		0	5	6	8
3			0	6	8
4				0	3
5					0





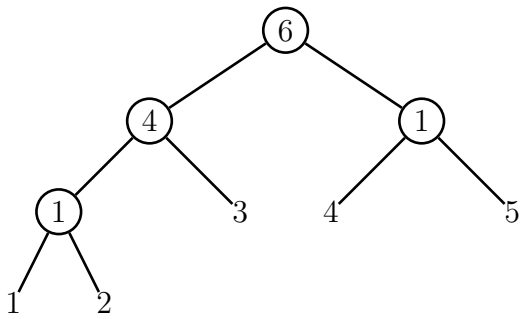


Abbildung 2.78: Beispiel: Einfache Lösung des ultrametrischen Sandwich-Problems

Für die Laufzeit erhalten wir

$$T(n) = O(n^2) + \sum_{i=1}^{m} T(n_i)$$

mit $\sum_{i=1}^{m} n_i = n$. Wir überlegen uns, was bei einem rekursiven Aufruf geschieht. Bei jedem rekursiven Aufruf wird eine Partition der Menge $[1:n]$ verfeinert. Da wir mit der trivialen Partition $\{[1:n]\}$ starten und mit $\{\{1\},\ldots,\{n\}\}$ enden, kann es also maximal $n-1$ rekursive Aufrufe geben. Somit ist die Laufzeit durch $O(n \cdot n^2)$ beschränkt.

Theorem 2.82 *Seien $D_\ell \leq D_h$ zwei $n \times n$-Distanzmatrizen. Ein ultrametrischer Baum $T \in [D_\ell : D_h]$ kann in Zeit $O(n^3)$ konstruiert werden, sofern überhaupt einer existiert.*

2.8.2 Charakterisierung einer effizienteren Lösung

In diesem Abschnitt wollen wir zeigen, dass wir das ultrametrische Sandwich Problem sogar in Zeit $O(n^2)$ lösen können. Dies ist optimal, da ja bereits die Eingabematrizen die Größe $\Theta(n^2)$ besitzen. Dazu benötigen wir erst noch die Definition der Kosten eines Pfades.

Definition 2.83 *Sei $G = (V, E, \gamma)$ ein gewichteter Graph. Die* Kosten $c(p)$ *eines Pfades $p = (v_0, \ldots, v_n)$ ist definiert durch*

$$c(p) := \max\{\gamma(v_{i-1}, v_i) \ : \ i \in [1:n]\}.$$

Mit $D(G, v, w)$ bezeichnen wir die minimalen Kosten eines Pfades von v nach w in G.

$$D(G, v, w) := \min\{c(p) \ : \ p \text{ ist ein Pfad von } v \text{ nach } w\}.$$

Warum interessieren wir uns überhaupt für die Kosten eines Pfades? Betrachten wir einen Pfad $p = (v_0, \ldots, v_n)$ in einem gewichteten Graphen $G(D)$, der von einer ultrametrischen Matrix D induziert wird. Seien dabei $\gamma(v, w)$ die Kosten der Kante (v, w). Wie groß ist nun der Abstand $d_D(v_0, v_n)$ von v_0 zu v_n? Unter Berücksichtigung der ultrametrischen Dreiecksungleichung erhalten wir:

$$\begin{aligned} d_D(v_0, w_n) &\leq \max\{d_D(v_0, v_{n-1}), \gamma(v_{n-1}, v_n)\} \\ &\leq \max\{\max\{d_D(v_0, v_{n-2}), \gamma(v_{n-2}, v_{n-1})\}, \gamma(v_{n-1}, v_n)\} \end{aligned}$$

$$
\begin{aligned}
&= \max\{d_D(v_0, v_{n-2}), \gamma(v_{n-2}, v_{n-1}), \gamma(v_{n-1}, v_n)\} \\
&\vdots \\
&\leq \max\{\gamma(v_0, v_1), \ldots, \gamma(v_{n-2}, v_{n-1}), \gamma(v_{n-1}, v_n)\} \\
&= c(p)
\end{aligned}
$$

Die letzte Gleichung folgt nur für den Fall, dass wir einen Pfad mit minimalen Kosten gewählt haben. Somit sind die Kosten eines Pfades in dem zur ultrametrischen Matrix gehörigem gewichteten Graphen eine obere Schranke für den Abstand der Endpunkte dieses Pfades.

Im folgenden Lemma erhalten wir dann eine weitere Charakterisierung, wann zwei Knoten k und ℓ im zugehörigen Graphen $G_{k\ell}$ durch einen Pfad verbunden sind. Man beachte, dass wir hier nicht beliebige Knotenpaare betrachten, sondern genau das Knotenpaar, dessen maximaler Abstand gemäß der unteren Schranke den Graphen $G_{k\ell}$ definiert.

Lemma 2.84 *Seien $D_\ell \leq D_h$ zwei Distanzmatrizen. Zwei Knoten k und ℓ befinden sich genau dann in derselben Zusammenhangskomponente von $G_{k,\ell}$, wenn $D_\ell(k,\ell) > D(G(D_h), k, \ell)$.*

Beweis: $\Rightarrow$**:** Wenn sich k und ℓ in derselben Zusammenhangskomponente von $G_{k,\ell}$ befinden, dann gibt es einen Pfad p von k nach ℓ. Für alle Kanten $(v,w) \in p$ gilt daher (da sie Kanten in $G_{k,\ell}$ sind): $\gamma(v,w) < D_\ell(k,l)$. Somit ist auch das Maximum der Kantengewichte durch $D_\ell(k,\ell)$ beschränkt und es gilt $D(G(D_h), k, \ell) < D_\ell(k,\ell)$.

$\Leftarrow$**:** Gelte nun $D(G(D_h), k, \ell) < D_\ell(k,\ell)$. Dann existiert nach Definition von $D(\cdot,\cdot)$ und $G_{k,\ell}$ ein Pfad p in $G(D_h)$, so dass das Gewicht jeder Kante durch $D_\ell(k,\ell)$ beschränkt ist. Somit ist p auch ein Pfad in $G_{k,\ell}$ von v nach w. Also befinden sich v und w in derselben Zusammenhangskomponente von $G_{k,\ell}$. ■

Notation 2.85 *Sei T ein Baum. Den eindeutigen einfachen Pfad von $v \in V(T)$ nach $w \in V(T)$ bezeichnen wir mit $p_T(v,w)$.*

Im Folgenden sei T ein minimaler Spannbaum von $G(D_h)$. Mit obiger Notation gilt dann, dass $D(T,v,w) = c(p_T(v,w))$. Wir werden jetzt zeigen, dass wir die oberen Schranken, die eigentlich durch die Matrix D_h gegeben sind, durch den zugehörigen minimalen Spannbaum von $G(D_h)$ mit viel weniger Speicherplatz darstellen können.

Lemma 2.86 *Sei D_h eine Distanzmatrix und sei T ein minimaler Spannbaum des Graphen $G(D_h)$. Dann gilt $D(T,v,w) = D(G(D_h),v,w)$ für alle Knoten $v,w \in V(T) = V(G(D_h))$.*

Beweis: Zuerst halten wir fest, dass jeder Pfad in T auch ein Pfad in $G(D_h)$ ist. Somit gilt in jedem Falle

$$D(G(D_h), v, w) \leq D(T, v, w).$$

Für einen Widerspruchsbeweis nehmen wir jetzt an, dass es zwei Knoten v und w mit

$$D(G(D_h), v, w) < D(T, v, w)$$

gibt. Dann existiert ein Pfad p in $G(D_h)$ von v nach w mit $c(p) < D(T, v, w)$.

Wir betrachten jetzt den eindeutigen Pfad $p_T(v, w)$ im minimalen Spannbaum T. Sei jetzt (x, y) eine Kante in $p_T(v, w)$mit maximalem Gewicht, also $\gamma(x, y) = c(p_T)$. Wir entfernen jetzt diese Kante aus T und erhalten somit zwei Teilbäume T_1 und T_2, die alle Knoten des Graphen $G(D_h)$ beinhalten. Dies ist in der Abbildung 2.79 illustriert.

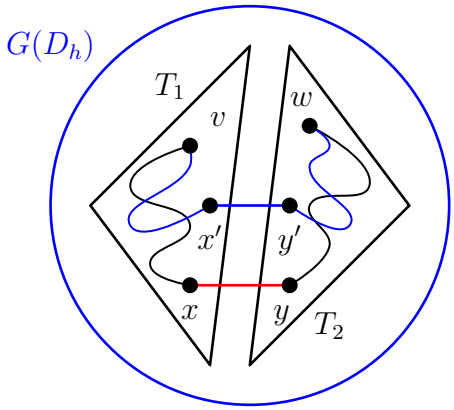


Abbildung 2.79: Skizze: Spannender Wald $\{T_1, T_2\}$ von $G(D_h)$ nach Entfernen von $\{x, y\}$ aus dem minimalen Spannbaum T

Sei (x', y') die erste Kante auf dem Pfad p in $G(D_h)$, die die Bäume T_1 und T_2 verbindet, d.h. $x' \in V(T_1)$ und $y' \in V(T_2)$. Nach Voraussetzung gilt, dass

$$D_h(x', y') < D_h(x, y),$$

da jede Kante des Pfades p nach Widerspruchsannahme leichter sein muss als die schwerste Kante in p_T und da (x, y) ja eine schwerste Kante in p_T war.

Dann könnten wir jedoch einen neuen Spannbaum T' mittels

$$E(T') = (E(T) \setminus \{(x, y)\}) \cup \{(x', y')\}$$

konstruieren. Dieser hat dann Gewicht

$$\gamma(T') \;=\; \gamma(T) + \underbrace{\gamma(x', y' - \gamma(x, y)}_{<0} \;<\; \gamma(T).$$

Somit hätten wir einen Spannbaum mit einem kleineren Gewicht konstruiert als der des minimalen Spannbaumes, was offensichtlich ein Widerspruch ist. ■

Damit haben wir gezeigt, dass wir im Folgenden die Informationen der Matrix D_h durch die Informationen im minimalen Spannbaum T von $G(D_h)$ ersetzen können.

Definition 2.87 *Seien $D_\ell \leq D_h$ zwei $n \times n$-Distanzmatrizen. Im Graph $G(D)$ heißen zwei Knoten $v, w \in [1:n]$* separabel*, wenn $D_\ell(v,w) \leq D(G(D_h), v, w)$ gilt.*

Die Separabilität von zwei Knoten ist eine nahe liegende Eigenschaft, die wir benötigen werden, um zu zeigen, dass es möglich ist einen ultrametrischen Baum zu konstruieren, in dem der verbindende Pfad sowohl die untere als auch die obere Schranke für den Abstand einhält. Wir werden dies gleich formal beweisen, aber wir benötigen dazu erst noch ein paar Definitionen und Lemmata. Zuerst halten wir aber noch das folgende Korollar fest, das aus dem vorherigen Lemma und der Definition unmittelbar folgt.

Korollar 2.88 *Seien $D_\ell \leq D_h$ zwei $n \times n$-Distanzmatrizen und sei T ein minimaler Spannbaum von $G(D_h)$. Zwei Knoten $v, w \in [1:n]$ sind genau dann separabel, wenn $D_\ell(v,w) \leq D(T,v,w)$ gilt.*

Bevor wir den zentralen Zusammenhang zwischen paarweiser Separabilität und der Existenz ultrametrischer Sandwich-Bäume zeigen, benötigen wir noch die folgende Definition.

Definition 2.89 *Seien $D_\ell \leq D_h$ zwei $n \times n$-Distanzmatrizen und sei T ein minimaler Spannbaum von $G(D_h)$. Eine Kante (x,y) des Pfades $p_T(v,w)$ im minimalen Spannbaum, der v und w verbindet, heißt* link-edge*, wenn sie eine Kante maximalen Gewichtes in $p_T(v,w)$ ist, d.h. wenn*

$$D_h(x,y) = c(p_T(v,w)) = D(T,v,w)$$

gilt. Mit $\mathrm{Link}(v,w)$ bezeichnen wir die Menge der Link-edges für das Knotenpaar (v,w).

Anschaulich ist die Link-Edge eines Pfades im zur oberen Schrankenmatrix gehörigen Graphen diejenige, die den maximalen Abstand von zwei Knoten bestimmt, die durch diesen Pfad verbunden werden. Aus diesem Grund werden diese eine besondere Rolle spielen.

Definition 2.90 *Seien $D_\ell \leq D_h$ zwei $n \times n$-Distanzmatrizen und sei T ein minimaler Spannbaum von $G(D_h)$. Für jede Kante $(x, y) \in E(T)$ im minimalen Spannbaum ist die* cut-weight $\mathrm{CW}(x, y)$ *wie folgt definiert:*

$$\mathrm{CW}(x, y) := \max \{D_\ell(v, w) \ : \ (v, w) \in \mathrm{Link}(x, y)\} .$$

Die cut-weight einer Kante ist der maximale Mindestabstand zweier Knoten, deren Verbindungspfad diese Kante als link-edge besitzt. Um ein wenig mehr Licht in die Bedeutung der cut-weight zu bringen, betrachten wir jetzt nur die maximale auftretende cut-weight eines minimalen Spannbaumes des zu den Maximalabständen gehörigen Graphen.

Für diese maximale cut-weight c^* gilt dann $c^* = \mathrm{MAX}(D_\ell)$, wie man sich leicht überlegt. Wie im einfachen Algorithmus werden wir jetzt versuchen alle schwereren Kanten in $G_{k\ell}$ (der ja ein Teilgraph von $G(D_h)$ ist) zu entfernen. Statt $G_{k\ell}$ betrachten wir jedoch den minimalen Spannbaum T von $G(D_h)$ (der ja nach Definition auch ein Teilgraph von $G(D_h)$ ist).

In $G_{k\ell}$ werden alle schwereren Kanten entfernt, damit $G_{k\ell}$ in mehrere Zusammenhangskomponenten zerfällt. Zuerst überlegt man sich, dass es auch genügen würde, so viele von den schwersten Kanten zu entfernen, bis zwei Zusammenhangskomponenten entstehen. Dies wäre jedoch algorithmisch sehr aufwendig und würde in der Regel keinen echten Zeitvorteil bedeuten. Im minimalen Spannbaum T lässt sich dies hingegen sehr leicht durch das Entfernen der Kante mit der maximalen cut-weight erzielen. Im Beweis vom übernächsten Lemma werden wir dies noch wesentlich genauer sehen.

Das folgende Lemma stellt noch einen weiteren fundamentalen Zusammenhang zwischen Kantengewichten in Kreisen zu additiven Matrizen gehörigen gewichteten Graphen und der Eigenschaft einer Ultrametrik dar, die wir im Folgenden benötigen, um leicht von einer additiven Matrix nachweisen zu können, dass sie bereits ultrametrisch ist.

Lemma 2.91 *Eine additive Matrix M ist genau dann ultrametrisch ist, wenn für jede Folge $(i_1, \ldots, i_k) \in \mathbb{N}^k$ paarweise verschiedener Werte mit $k \geq 3$ gilt, dass in der Folge $(M(i_1, i_2), M(i_2, i_3), \ldots, M(i_{k-1}, i_k), M(i_k, i_1))$ das Maximum mehrfach angenommen wird.*

Beweis: $\Leftarrow$**:** Sei M eine additive Matrix und es gelte für alle $k \geq 3$ und für alle Folgen $(i_1, \ldots, i_k) \subset \mathbb{N}^k$ mit paarweise verschiedenen Folgengliedern, dass

$$\max\{M(i_1, i_2), M(i_2, i_3), \ldots, M(i_{k-1}, i_k), M(i_k, i_1)\} \tag{2.2}$$

nicht eindeutig ist.

Wenn man in (2.2) $k = 3$ einsetzt, gilt insbesondere für beliebige $a, b, c \in \mathbb{N}$, dass

$$\max\{M(a,b), M(b,c), M(c,a)\}$$

nicht eindeutig ist und damit ist M also ultrametrisch.

⇒: Sei M nun ultrametrisch, dann ist M auch additiv. Wir müssen nur noch (2.2) zu zeigen. Sei $S = (i_1, \ldots, i_k) \in \mathbb{N}^k$ gegeben. Wir betrachten zunächst i_1, i_2 und i_3. Aus der fundamentalen Eigenschaft einer Ultrametrik wissen wir, dass das Maximum von $(M(i_1, i_2), M(i_2, i_3), M(i_3, i_1))$ nicht eindeutig ist. Wir beweisen (2.2) mittels vollständiger Induktion.

Gilt für j, dass das Maximum von $(M(i_1, i_2), M(i_2, i_3), \ldots, M(i_{j-1}, i_j), M(i_j, i_1))$ nicht eindeutig ist, so gilt dies auch für $j + 1$. Dazu betrachten wir i_1, i_j und i_{j+1}. Weiter wissen wir, dass $\max\{M(i_1, i_j), M(i_j, i_{j+1}), M(i_{j+1}, i_1)\}$ nicht eindeutig ist, d.h. einer der Werte ist kleiner als die anderen beiden. Betrachten wir wie sich das Maximum ändert, wenn wir $M(i_j, i_1)$ herausnehmen und dafür $M(i_j, i_{j+1})$ und $M(i_{j+1}, i_1)$ dazugeben.

Fall 1: $M(i_1, i_j)$ ist der kleinste Wert. Durch das Hinzufügen von $M(i_j, i_{j+1})$ und $M(i_{j+1}, i_1)$ kann das Maximum nicht eindeutig werden, denn beide Werte sind gleich. Liegen sie unter dem alten Maximum ändert sich nichts, liegen sie darüber, bilden beide das nicht eindeutige neue Maximum. Das Entfernen von $M(i_1, i_j)$ spielt nur eine Rolle, falls $M(i_1, i_j)$ vorher ein Maximum war. In dem Fall sind aber $M(i_j, i_{j+1})$ und $M(i_{j+1}, i_1)$ das neue, nicht eindeutige Maximum.

Fall 2: $M(i_{j+1}, i_1)$ ist der kleinste Wert. Dann ist $M(i_1, i_j) = M(i_j, i_{j+1})$. Das Entfernen von $M(i_1, i_j)$ wird durch das Hinzufügen von $M(i_j, i_{j+1})$ wieder ausgeglichen. $M(i_{j+1}, i_1)$ wird auch hinzugefügt, spielt aber keine Rolle.

Fall 3: $M(i_j, i_{j+1})$ ist der kleinste Wert. Dann ist $M(i_1, i_j) = M(i_1, i_{j+1})$. Dieser Fall ist analog zu Fall 2. ■

Nun kommen wir zum Beweis unseres zentralen Lemmas, dass es genau dann einen ultrametrischen Baum für eine gegebene Sandwich-Bedingung gibt, wenn alle Knoten paarweise separabel sind. Der Bewies in die eine Richtung wird konstruktiv sein und wird uns somit einen effizienten Algorithmus an die Hand geben.

Lemma 2.92 *Seien $D_\ell \leq D_h$ zwei $n \times n$-Distanzmatrizen. Ein ultrametrischer Baum $U \in [D_\ell, D_h]$ existiert genau dann, wenn jedes Paar $v, w \in [1 : n]$ von Knoten separabel ist.*

Beweis: $\Rightarrow$**:** Für einen Widerspruchsbeweis nehmen wir an, dass v und w nicht separabel sind. Dann ist $D_\ell(v,w) > D_h(x,y)$ für alle $\{x,y\} \in \text{Link}(v,w)$. Für jede Kante $\{a,b\}$ in $p_T(v,w)$ gilt dann $D_h(a,b) \leq D_h(x,y)$ für alle $\{x,y\} \in \text{Link}(v,w)$. Also ist $\{v,w\} \notin p_T(v,w)$. Damit ist $D_\ell(x,y) > D_h(a,b)$ für alle $\{a,b\} \in \text{Link}(v,w)$. Damit muss die Kante $\{x,y\}$ im Kreis, gebildet aus $p_T(x,y)$ und der Kante $\{x,y\}$, in einer ultrametrischen Matrix das eindeutige Maximum sein. Dies steht jedoch im Widerspruch zu Lemma 2.91 und diese Implikation ist bewiesen.

$\Leftarrow$**:** Sei T ein minimaler Spannbaum von $G(D_h)$ und sei $E(T) = \{e_1, \ldots, e_{n-1}\}$, wobei $\text{CW}(e_1) \geq \cdots \geq \text{CW}(e_{n-1})$. Wir entfernen jetzt sukzessive die Kanten aus T gemäß der cut-weight der Kanten. Dabei wird durch jedes Entfernen ein Baum in zwei neue Bäume zerlegt.

Betrachten wir jetzt nachdem Entfernen der Kanten $e_1, \ldots, e_{i-1}$ den entstandenen Wald und darin den Baum T', der die Kante $e_i = \{v,w\}$ enthält. Das Entfernen der Kante $e_i = \{v,w\}$ zerlegt den Baum T' in zwei Bäume T'_1 und T'_2. Wir setzten dann $d_U(v,w) := \text{CW}(e_i)$ für alle $v \in V(T'_1)$ und $w \in V(T'_2)$.

Wir zeigen als erstes, dass $d_U(v,w) \geq D_\ell(v,w)$ für alle $v,w \in [1:n]$. Wir betrachten die Kante e, nach deren Entfernen die Knoten v und w im Rest des minimalen Spannbaums nicht mehr durch einen Pfad verbunden sind. Ist e eine link-edge in $p_T(v,w)$, dann gilt nach Definition der cut-weight: $\text{CW}(e) \geq D_\ell(v,w)$. Ist e hingegen keine link-edge in $p_T(v,w)$, dann gilt $\text{CW}(e) \geq \text{CW}(e')$ für jede link-edge $e' \in \text{Link}(v,w)$, da wir die Kanten gemäß ihrer absteigenden cut-weight aus dem minimalen Spannbaum T entfernen. Somit gilt nach Definition der cut-weight:

$$\text{CW}(e) \geq \text{CW}(e') \geq D_\ell(v,w).$$

Wir zeigen jetzt, dass ebenfalls $d_U(v,w) \leq D_h(v,w)$ für alle $v,w \in [1:n]$ gilt. Nach Definition der cut-weight gilt, dass ein Knotenpaar $\{x,y\}$ existiert, so dass für alle $\{a,b\} \in \text{Link}(x,y)$ gilt: $D_\ell(x,y) = \text{CW}(a,b)$. Da x und y nach Voraussetzung separabel sind, gilt

$$\text{CW}(a,b) = D_\ell(x,y) \leq D(T,x,y) = D_h(a,b).$$

Die letzte Gleichheit folgt aus der Tatsache, dass $\{a,b\} \in \text{Link}(x,y)$. Aufgrund der Konstruktion des minimalen Spannbaumes gilt weiterhin $D_h(a,b) \leq D_h(v,w)$. Somit gilt $d_U(v,w) = \text{CW}(a,b) \leq D_h(v,w)$.

Damit haben wir gezeigt, dass $U \in [D_\ell, D_h]$. Wir müssen zum Schluss nur noch zeigen, dass U ultrametrisch ist. Nach Lemma 2.91 genügt es zu zeigen, dass in jedem Kreis in $G(U)$ das Maximum nicht eindeutig ist. Sei also C ein Kreis in $G(U)$ und (v,w) eine Kante maximalen Gewichtes in C. Falls es mehrere davon geben sollte, wählen wir eine solche, deren Endpunkte in unserem Konstruktionsverfahren durch Entfernen von Kanten im minimalen Spannbaum T zuerst separiert wurden.

Wir betrachten jetzt den Zeitpunkt in unserem Konstruktionsverfahren von d_U, als $d_U(v, w)$ gesetzt wurde. Zu diesem Zeitpunkt wurde v und w in zwei Bäume T' und T'' aufgeteilt. Ferner sind die Knoten dieses Kreises nach Wahl der Kante $\{v, w\}$ alle Knoten des Kreises in diesen beiden Teilbäumen enthalten. Da es in C noch einen anderen Weg von v nach w gibt, muss zu diesem Zeitpunkt auch für eine andere Kante $\{v', w'\}$ der Wert $d_U(v', w')$ ebenfalls festgelegt worden sein. Nach unserer Konstruktion gilt dann natürlich $d_U(v, w) = d_U(v', w')$ und in C ist das Maximum nicht eindeutig. ■

2.8.3 Algorithmus für das ultrametrische Sandwich-Problem

Der Beweis des vorherigen Lemmas liefert unmittelbar den folgenden Algorithmus, der in Abbildung 2.80 angegeben ist.

1. Bestimme einen minimalen Spannbaum T für $G(D_h)$. $O(n^2)$
2. Bestimme dabei den zugehörigen Kartesischen Baum R für den Zusammenbau von T $O(n^2)$
3. Bestimme die cut-weights der einzelnen Kanten aus T mit Hilfe des Kartesischen Baumes. $O(n^2)$
4. Baue den minimalen Spannbaum durch Entfernen der Kanten absteigend nach den cut-weights der Kanten ab und baue parallel den ultrametrischen Baum U wieder auf. $O(n)$

Abbildung 2.80: Algorithmus: Effiziente Lösung des ultrametrischen Sandwich-Problems

Die Korrektheit des Algorithmus folgt im Wesentlichen aus dem Beweis des vorherigen Lemmas (wir gehen später noch auf ein paar implementierungstechnische Besonderheiten ein).

Wir müssen uns nur noch um die effiziente Implementierung kümmern. Einen Algorithmus zur Bestimmung minimaler Spannbäume haben wir bereits kennen gelernt. Wir werden hier noch eine andere Möglichkeit darstellen, die für unsere Zwecke besser geeignet ist. Ebenso müssen wir uns um die effiziente Berechnung der cut-weights kümmern. Ferner müssen wir noch erklären, was kartesische Bäume sind und wie wir sie konstruieren können.

2.8.3.1 Kartesische Bäume

Kommen wir nun zur Definition eines kartesischen Baumes.

Definition 2.93 *Sei M eine Menge von Objekten, $E \subset \binom{M}{2}$ eine Menge von Paaren, so dass der Graph (M, E) ein Baum ist und $\gamma : E \to \mathbb{R}$ eine Gewichtsfunktion auf E. Ein* kartesischer Baum *für (M, E) ist rekursiv wie folgt definiert.*

- *Ist $|M| = 1$, dann ist der einelementige Baum mit der Wurzelmarkierung $m \in M$ ein kartesischer Baum.*
- *Ist $|M| \geq 2$ und $\{v_1, v_2\} \in E$ mit $\gamma(\{v_1, v_2\}) = \max\{\gamma(e) : e \in E\}$. Sei weiter T' der Wald, der aus T durch Entfernen von $\{v_1, v_2\}$ entsteht, d.h. $V(T') = V(T)$ und $E(T') = E(T) \setminus \{v_1, v_2\}$. Sind T_1 bzw. T_2 kartesische Bäume für $C(T', v_1)$ bzw. $C(T', v_2)$, dann ist der Baum mit der Wurzel, die mit $\{v_1, v_2\}$ markiert ist und deren Teilbäume der Wurzel gerade T_1 und T_2 sind, ebenfalls ein kartesischer Baum.*

Wir merken hier noch explizit an, dass die Elemente aus M nur als Blattmarkierungen auftauchen und dass alle inneren Knoten mit den Elementen aus E markiert sind.

Betrachtet man auf der Menge $[1 : n]$ als Baum eine lineare Liste mit

$$E = \{\{i, i+1\} \,:\, i \in [1 : n-1]\} \quad \text{mit} \quad \gamma(i, i+1) = \max\{F[i], F[i+1]\},$$

wobei F ein Feld von Zahlen ist, so erhält man im Wesentlichen einen Heap, in dem die Werte an den Blättern stehen und die inneren Knoten den maximalen Wert ihres Teilbaumes besitzen, wenn man, wie hier üblich, anstatt der Kante in den inneren Knoten das Gewicht der Kante einträgt. Diese Struktur wird manchmal auch als kartesischer Baum bezeichnet.

Der kartesische Baum für einen minimalen Spannbaum lässt sich jetzt sehr einfach rekursiv konstruieren. Wir entfernen die Kanten maximalen Gewichtes und konstruieren für die beiden entstehenden Spannbäume rekursiv einen kartesischen Baum. Anschließend fügen wir einen Wurzel ein und die beiden Kinder der neuen Wurzel sind die Wurzeln der rekursiv konstruierten kartesischen Bäume.

Lemma 2.94 *Sei $G = (V, E, \gamma)$ ein gewichteter Graph und T ein minimaler Spannbaum von G. Der kartesische Baum für T kann mit einem rekursiven Algorithmus aus dem minimalen Spannbaumes T in Zeit $O(n^2)$ konstruiert werden.*

2.8.3.2 Berechnung der cut-weights

Wir wollen jetzt zeigen, dass wir die cut-weights einer Kante $e \in E$ im minimalen Spannbaum T mit Hilfe von so genannten lca-Anfragen im kartesischen Baum R einfach bestimmen können. Dazu gehen wir durch die Matrix D_ℓ und bestimmen für jedes Knotenpaar (i, j) den niedrigsten gemeinsamen Vorfahren $\text{lca}(i, j)$ im kartesischen Baum R.

Die dort gespeicherte Kante e ist nach Konstruktion des kartesischen Baumes eine Kante mit größtem Gewicht auf dem Pfad von i nach j im minimalen Spannbaum T, d.h. $e \in \text{Link}(i, j)$. Daher werden wir dort die cut-weight $\text{CW}(e)$ dieser Kante mittels $\text{CW}(e) = \max\{\text{CW}(e), D_\ell(i, j)\}$ aktualisieren. Wir bemerken an dieser Stelle, dass es durchaus noch andere link-edges auf dem Pfad von i nach j im minimalen Spannbaum geben kann. Daher wird die cut-weight nicht für jede Kante korrekt berechnet. Wir gehen auf diesen „Fehler“ am Ende diese Abschnittes noch ein.

Lemma 2.95 *Sei $G = (V, E, \gamma)$ ein gewichteter Graph und T ein minimaler Spannbaum von G. Der kartesische Baum für T kann mit einem rekursiven Algorithmus aus dem minimalen Spannbaumes T in Zeit $O(n^2)$ konstruiert werden.*

Nachdem wir jetzt die wesentlichen Schritte unseres effizienten Algorithmus weitestgehend verstanden haben, können wir uns einem Beispiel zuwenden. In der Abbildung 2.81 ist ein Beispiel angegeben, wie unser effizienter Algorithmus vorgeht.

2.8.3.3 Least Common Ancestor Queries

Wir müssen uns nun nur noch überlegen, wie wir $O(n^2)$ lca-Anfragen in Zeit $O(n^2)$ beantworten können. Dazu werden wir das lca-Problem auf das Range Minimum Query Problem reduzieren, das wie folgt definiert ist.

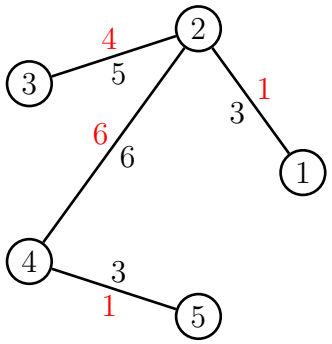
RANGE MINIMUM QUERY

Eingabe: Eine Feld F der Länge n von reellen Zahlen und $i \leq j \in [1 : n]$.
Gesucht: Ein Index k mit $F[k] = \min\{F[\ell] \,:\, \ell \in [i : j]\}$.

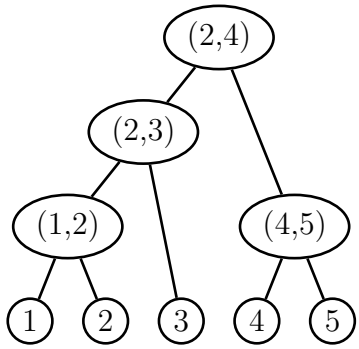
Wir werden später zeigen, wie wir mit einem Preprocessing in Zeit $O(n^2)$ jede Anfrage in konstanter Zeit beantworten können. Für die Reduktion betrachten wir die so genannte *Euler-Tour* eines Baumes.

D_ℓ	1	2	3	4	5
1	0	1	2	3	6
2		0	4	5	5
3			0	4	5
4				0	1
5					0

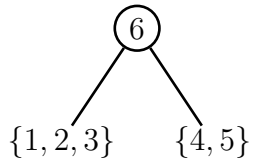
D_h	1	2	3	4	5
1	0	3	6	8	8
2		0	5	6	8
3			0	6	8
4				0	3
5					0



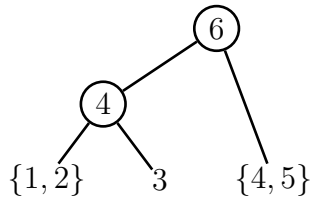
Minimaler Spannbaum T



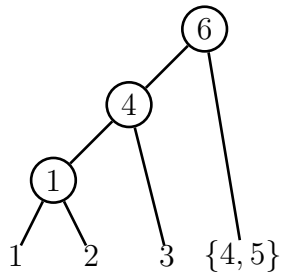
Kartesischer Baum R



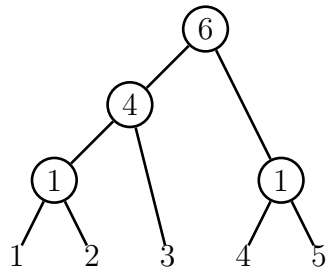
U_1 nach Entfernen $\{2, 4\}$



U_2 nach Entfernen $\{2, 3\}$



U_3 nach Entfernen $\{1, 2\}$



$U = U_4$ nach Entfernen $\{4, 5\}$

Abbildung 2.81: Beispiel: Lösung eines ultrametrischen Sandwich-Problems

Definition 2.96 *Sei $T = (V, E)$ ein gewurzelter Baum mit Wurzel r und seien $T_1, \ldots, T_\ell$ die Teilbäume, die an der Wurzel hängen. Die* Euler-Tour *durch T ist eine Liste von $2n-1$ Knoten, die wie folgt rekursiv definiert ist:*

- *Ist $\ell = 0$, d.h. der Baum besteht nur aus dem Blatt r, dann ist diese Liste durch (r) gegeben.*
- *Für $\ell \geq 1$ seien $L_1, \ldots, L_\ell$ mit $L_i = (v_1^{(i)}, \ldots, v_{n_i}^{(i)})$ für $i \in [1:\ell]$ die Euler-Touren von $T_1, \ldots, T_\ell$. Die Euler-Tour von T ist dann durch*

$$(r, v_1^{(1)}, \ldots, v_{n_1}^{(1)}, r, v_1^{(2)}, \ldots, v_{n_2}^{(2)}, r, \ldots, r, v_1^{(\ell)}, \ldots, v_{n_\ell}^{(\ell)}, r)$$

definiert.

Der Leser sei dazu aufgefordert zu verifizieren, dass die oben definierte Euler-Tour eines Baumes mit n Knoten tatsächlich eine Liste mit $2n-1$ Elementen ist.

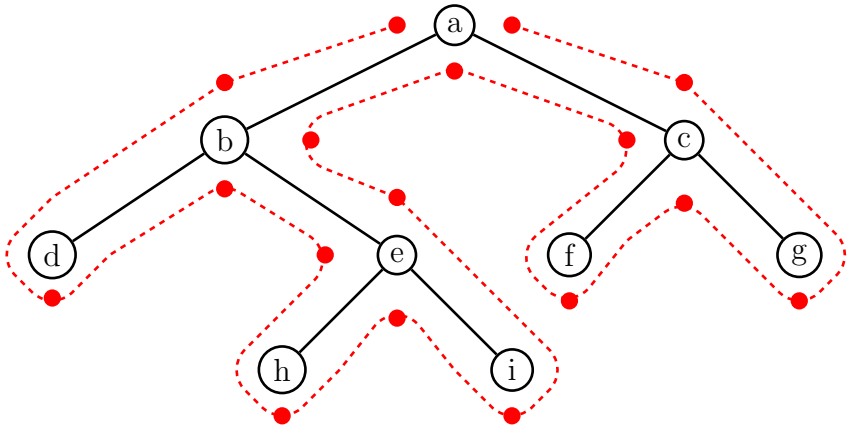
Die Euler-Tour kann sehr leicht mit Hilfe einer Tiefensuche in Zeit $O(n)$ berechnet werden. Der Algorithmus hierfür ist in Abbildung 2.82 angegeben. Man kann sich die Euler-Tour auch bildlich sehr schön als das Abmalen der Bäume anhand ihrer äußeren Kontur vorstellen, wobei bei jedem Antreffen eines Knotens des Baumes dieser in die Liste aufgenommen wird. Die ist in Abbildung 2.83 anhand eines Beispiels illustriert.

Zusammen mit der Euler-Tour, d.h. der Liste der abgelaufenen Knoten, betrachten wir zusätzlich noch die DFS-Nummern des entsprechenden Knotens, die bei der Tiefensuche in der Regel mitberechnet werden (siehe auch das Beispiel in der Abbildung 2.83).

Euler-Tour (tree $T = (V, E)$)

```
begin
    // let r(T) be the root of T
    output r(T);
    // let N(r(T)) be the set of children of the root of T
    forall (v ∈ N(r(T))) do
        // let T(v) be the subtree rooted at node v
        EULER-TOUR(T(v));
        outputr(T);
end
```

Abbildung 2.82: Algorithmus: Konstruktion einer Euler-Tour



Euler-Tour	a	b	d	b	e	h	e	i	e	b	a	c	f	c	g	c	a
DFS-Nummer	1	2	3	2	4	5	4	6	4	2	1	7	8	7	9	7	1

Abbildung 2.83: Beispiel: Euler-Tour

Betrachten wir jetzt die Anfrage an zwei Knoten des Baumes i und j. Zuerst bemerken wir, dass diese Knoten, sofern sie keine Blätter sind, mehrfach vorkommen können. Wir wählen jetzt für i und j willkürlich einen der Knoten, der in der Euler-Tour auftritt, als einen Repräsentanten aus. Zuerst stellen wir fest, dass in der Euler-Tour der niedrigste gemeinsame Vorfahren von i und j in der Teilliste, die durch die beiden Repräsentanten definiert ist, vorkommen muss, was man wie folgt sieht.

Nehmen wir an, dass i in der Euler-Tour vor j auftritt. In der Euler-Tour sind alle Knoten v, die nach einem Repräsentanten von i auftauchen und eine kleinere DFS-Nummer als i besitzen, ein Vorfahre von i. Da die DFS-Nummer von v kleiner als die von i ist und v in der Euler-Tour nach i auftritt, ist die DFS-Prozedur von v noch aktiv, als der Knoten i besucht wird. Das ist genau die Definition dafür, dass i ein Nachfahre von v ist. Analoges gilt für den Knoten j.

Wir betrachten jetzt nur die Knoten der Teilliste zwischen den beiden Repräsentanten von i und j, deren DFS-Nummer kleiner als die von i ist. Nach dem obigen sind dies Vorfahren von i. Nur einer davon, nämlich der mit der kleinsten DFS-Nummer, ist auch ein Vorfahre von j und muss daher der niedrigste gemeinsame Vorfahre von i und j sein. Diesen Knoten haben wir nämlich besucht, bevor wir in den Teilbaum von j eingedrungen sind. Alle Vorfahren davon haben wir mit Betrachten der Teilliste eliminiert, die mit einem Repräsentanten von j endet.

Damit können wir also das so erhaltene Zwischenergebnis im folgenden Lemma festhalten.

Lemma 2.97 *Gibt es eine Lösung für das Range Minimum Query Problem, dass für das Preprocessing Zeit $O(p(n))$ und für eine Anfrage $O(q(n))$ benötigt, so kann das Problem des niedrigsten gemeinsamen Vorfahren mit einen Zeitbedarf für das Preprocessing in Zeit $O(n + p(2n - 1))$ und für eine Anfrage in Zeit $O(q(2n - 1))$ gelöst werden.*

2.8.3.4 Range Minimum Queries

Damit können wir uns jetzt ganz auf das Range Minimum Query Problem konzentrieren. Offensichtlich kann ohne ein Preprocessing eine einzelne Anfrage mit $O(j-i) = O(n)$ Vergleichen beantwortet werden. Das Problem der Range Minimum Queries ist jedoch insbesondere dann interessant, wenn für ein gegebenes Feld eine Vielzahl von Range Minimum Queries durchgeführt werden. In diesem Fall können mit Hilfe einer Vorverarbeitung die Kosten der einzelnen Queries gesenkt werden.

Ein triviale Lösung würde alle möglichen Anfragen vorab berechnen. Dazu könnte eine zweidimensionale Tabelle $Q[i, j]$ angelegt werden. Dazu würde für jedes Paar (i, j) das Minimum der Bereichs $F[i : j]$ mit $j - i$ Vergleichen bestimmt werden. Dies würde zu einer Laufzeit für die Vorverarbeitung von

$$\sum_{i=1}^{n}\sum_{j=i}^{n}(j - i) = \Theta(n^3)$$

führen. In der Abbildung 2.84 ist ein einfacher, auf dynamischer Programmierung basierender Algorithmus angegeben, der diese Tabelle in Zeit $O(n^2)$ berechnen kann. Damit erhalten wir das folgende Resultat für das Range Minimum Query Problem.

RMQ (int $F[]$, int n)

```
begin
    for (i := 1; i ≤ n; i++) do
        T[i,i] := i;

        for (i := 1; i ≤ n; i++) do
            for (j := 1; i + j ≤ n; j++) do
                if (F[T[i, i + (j − 1)]] ≤ F[i + j]) then
                    T[i, i + j] := T[i, i + j − 1];
                else
                    T[i, i + j] := i + j;
end
```

Abbildung 2.84: Algorithmus: Preprocessing für Range Minimum Queries

Theorem 2.98 *Für das Range Minimum Query Problem kann mit Hilfe einer Vorverarbeitung, die mit einem Zeitbedarf von $O(n^2)$ ausgeführt werden kann, jede Anfrage in konstanter Zeit beantwortet werden.*

Es gibt bereits wesentlich effizientere Verfahren für das Range Minimum Query Problem. In unserem Zusammenhang ist dieses leicht zu erzielende Ergebnis jedoch bereits völlig ausreichend und wir verweisen für die anderen Verfahren auf die einschlägige Literatur. Wir halten das für uns wichtige Ergebnis jetzt noch explizit fest.

Theorem 2.99 *Sei T ein gewurzelter Baum mit n Knoten. Nach einer Vorverarbeitung, die in Zeit $O(n^2)$ durchgeführt werden kann, kann jede Anfrage nach einem niedrigsten gemeinsamen Vorfahren zweier Knoten aus T in konstanter Zeit beantwortet werden.*

2.8.3.5 Reale Berechnung der cut-weights

Wie bereits schon angedeutet, berechnet unser effizienter Algorithmus nicht wirklich die cut-weights der Kanten des minimalen Spannbaumes. Dies passiert genau dann, wenn es im minimalen Spannbäume mehrere Kanten desselben Gewichtes gibt. Um dies genauer zu beleuchten, betrachten wir das folgende Beispiel, das in Abbildung 2.85 angegeben ist.

D_ℓ	1	2	3
1	0	1	3
2		0	2
3			0

D_h	1	2	3
1	0	5	6
2		0	5
3			0

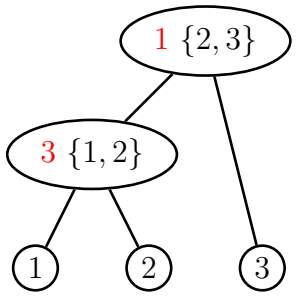


Abbildung 2.85: Beispiel: Falsche Berechnung der cut-weights

Hier stellen wir fest, das alle Kanten des minimalen Spannbaumes ein Kantengewicht von 5 besitzen. Somit sind alle Kanten des Baumes link-edges. Zuerst stellen wir fest, dass die cut-weight der Kante $\{2, 3\}$, die in der Wurzel des kartesischen Spannbaumes, mit 3 korrekt berechnet wird, da ja auch die Kante $\{1, 3\}$ der niedrigste gemeinsame Vorfahre von 1 und 3 im kartesischen Baum ist.

Im Gegensatz dazu wird die cut-weight der Kante $\{1, 2\}$ des minimalen Spannbaumes falsch berechnet. Diese Kante erhält die cut-weight 1, da die Kante $\{1, 2\}$ der niedrigste gemeinsame Vorfahre von Knoten 1 und Knoten 2 ist. Betrachten wir jedoch den Pfad von Knoten 1 über Knoten 2 nach Knoten 3, dann stellen wir fest, dass auch die Kante $\{1, 2\}$ eine link-edge im Pfad von Knoten 1 nach Knoten 3 im minimalen Spannbaum ist und die cut-weight der Kante $\{1, 2\}$ im minimalen Spannbaum daher ebenfalls 3 sein müsste.

Eine einfache Lösung wäre es, die Kantengewicht infinitesimal so zu verändern, dass alle Kantengewichte im minimalen Spannbaum eindeutig wären. Wir können jedoch auch zeigen, dass der Algorithmus weiterhin korrekt ist, obwohl er die cut-weights von manchen Kanten falsch berechnet.

Zuerst überlegen wir uns, welche Kanten eine falsche cut-weight bekommen. Dies kann nur bei solchen Kanten passieren, deren Kantengewicht im minimalen Spannbaum nicht eindeutig sind. Zum anderen müssen diese Kanten bei der Konstruktion des minimalen Spannbaumes vor den anderen Kanten gleichen Gewichtes im minimalen Spannbaum eingefügt worden sein. Dies bedeutet, dass diese Kante im kartesischen Baum ein Nachfahre einer anderen Kante des minimalen Spannbaum gleichen Gewichtes sein muss.

Überlegen wir uns, was in einer solchen Situation im Algorithmus passiert. Wir entfernen ja die Kanten aus dem minimalen Spannbaum nach fallendem cut-weight. Somit wird von zwei Kanten im minimalen Spannbaum, die dasselbe Kantengewicht besitzen und dieselbe cut-weight besitzen sollten, die Kante entfernt, die sich weiter oben im kartesischen Baum befindet. Damit zerfällt der minimale Spannbaum in zwei Teile und die beiden Knoten der untere Schranke, die für die cut-weight der entfernten Kante verantwortlich sind, befinden sich in zwei verschiedenen Zusammenhangskomponenten.

Somit ist die cut-weight der Kante, die ursprünglich falsch berechnet worden, in der neuen Zusammenhangskomponente jetzt nicht mehr ganz so falsch. Entweder ist sie für die konstruierte Zusammenhangskomponente korrekt und wir können mit unserem Algorithmus fortfahrten und er bleibt korrekt. War sie andernfalls falsch, befindet sich in minimalen Spannbaum dieser Zusammenhangskomponente eine weitere Kante mit demselben Gewicht, die im kartesischen Baum ein Vorfahre der betrachteten Kante ist und die ganze Argumentation wiederholt sich.

Also obwohl der Algorithmus nicht die richtigen cut-weights berechnet, ist zumindest immer die cut-weight der Kante mit der schwersten cut-weight korrekt berechnet worden und dies genügt für die Korrektheit des Algorithmus völlig, wie eine kurze Inspektion des zugehörigen Beweises ergibt.

2.8.4 Approximationsprobleme

Wir kommen jetzt noch einmal zu dem Approximationsproblem für Distanzmatrizen zurück.

Ultrametrisches Approximationsproblem

Eingabe: Eine $n \times n$-Distanzmatrizen D.
Gesucht: Eine ultrametrische Distanzmatrix D', die $\|D - D'\|$ minimiert.

Das entsprechende additive Approximationsproblem ist leider $\mathcal{NP}$-hart. Das ultrametrische Approximationsproblem kann für die Maximumsnorm $\|\cdot\|_\infty$ allerdings in Zeit $O(n^2)$ gelöst werden. Für die anderen p-Normen ist das Problem ebenfalls wieder $\mathcal{NP}$-hart.

Theorem 2.100 *Das ultrametrische Approximationsproblem für die Maximumsnorm kann in linearer Zeit (in der Größe der Eingabe) gelöst werden.*

Beweis: Sei D die gegebene Distanzmatrix. Eigentlich müssen wir nur ein minimales $\varepsilon > 0$ bestimmen, so dass es eine ultrametrische Matrix U mit $U \in [D - \varepsilon, D + \varepsilon]$ gibt. Hierbei ist $D + x$ definiert durch $D = (d_{i,j} + x)_{i,j}$.

Wir berechnen also zuerst wieder den minimalen Spannbaum T für $G(D)$. Dann berechnen wir die cut-weights für die Kanten des minimalen Spannbaumes T. Dabei wählen wir für jede Kante e ein minimales ε_e, so dass die Knotenpaare separabel sind, d.h. $\mathrm{CW}(e) - \varepsilon_e \leq D(e) + \varepsilon_e$ für alle Kanten $e \in E(T)$.

Damit dies für alle Kanten des Spannbaumes gilt, wählen ε als das Maximum dieser, d.h.

$$\varepsilon := \max\{\varepsilon_e \ : \ e \in E(T)\} = \frac{1}{2}\max\{\mathrm{CW}(e) - D(e) \ : \ e \in E(T)\}\,.$$

Dann führen wir denselben Algorithmus wie für das ultrametrische Sandwich Problem mit $D_\ell := D - \varepsilon$ und $D_h = D + \varepsilon$ durch. ∎

2.9 Splits und Splits-Graphen

In diesem Abschnitt stellen wir eine andere Möglichkeit vor, wie man zu gegebenen Distanzmatrizen phylogenetische Bäume (hier besser Netzwerke) konstruieren kann, auch wenn die Distanzmatrizen nicht notwendigerweise ultrametrisch oder additiv sind. Dieser Abschnitt orientiert sich im Wesentlichen am Skript von Daniel Huson.

2.9.1 Splits in Bäumen

Zunächst einmal müssen wir noch ein paar Grundlagen formalisieren, die wir auf die eine oder andere Art schon kennen gelernt haben. Definieren wir zuerst, was wir unter einem Split verstehen wollen.

Definition 2.101 *Sei X eine Menge von Taxa. Eine Bipartition $(A, \overline{A})$ von X mit $A \cup \overline{A} = X$ und $A \cap \overline{A} = \emptyset$ wird als* Split *bezeichnet.*

Eine Menge von Bipartitionen von X wird als Menge von Splits über X *bezeichnet.*

Hierbei ist nach Definition $(X, \emptyset)$ bzw. $(\emptyset, X)$ kein Split. Definieren wir nun was wir in diesem Abschnitt unter einem phylogenetischen Baum verstehen wollen.

Definition 2.102 *Sei T ein freier Baum und X eine Menge von Markierungen. T heißt ein* Baum über X*, wenn einige seiner Knoten mit Elementen aus X markiert sind und jedes Element aus X genau einmal als Markierung in T auftritt*

Wir werden im Folgenden allerdings annehmen, dass alle Blätter und nur die Blätter mit Taxa markiert sind.

In einem phylogenetischen Baum über X definiert jede Kante eine Bipartition der Taxa, also einen Split.

Definition 2.103 *Sei T ein Baum über X. Eine Kante $e \in E(T)$ definiert einen* Split $\Sigma(e) = \{A, \overline{A}\}$ *durch die Partitionierung von $X = A \cup \overline{A}$ mit $A \cap \overline{A} = \emptyset$, wobei A und $\overline{A}$ die Mengen der Markierungen in den beiden Bäume von $(V(T), E(T) \setminus \{e\})$ sind.*

Mit $\Sigma(T)$ bezeichnen wir die Menge aller Splits eines Baumes, d.h

$$\Sigma(T) = \{\Sigma(e) \,:\, e \in E(T)\}\,.$$

Ein solcher Split, der durch eine Kante eines phylogenetischen Baumes definiert wird, ist im folgenden Beispiel in Abbildung 2.86 illustriert.

Kommen wir nun zur Definition der Größe eines Splits und der Definition von trivialen Splits.

Definition 2.104 *Sei $S = \{A, \overline{A}\}$ ein Split über X, dann ist* $\min\{|A|, |\overline{A}|\}$ *die* Größe eines Splits.

Ein Split heißt trivialer Split*, wenn seine Größe gleich 1 ist.*

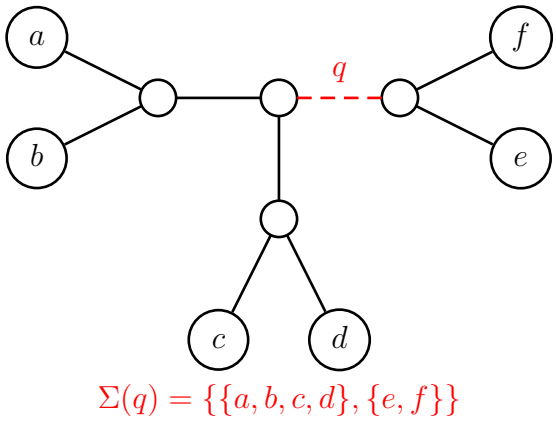


Abbildung 2.86: Beispiel: Ein Split, der durch die Kante q induziert ist

Triviale Splits entsprechen in phylogenetischen Bäumen genau denjenigen Splits, die von einer zu einem Blatt inzidenten Kante erzeugt werden.

Den Split der Größe 0 wollen wir im Folgenden nicht betrachten, da er in Bäumen, deren Blätter alle markiert sind, nicht vorkommen kann. Darüber hinaus wäre dieser Split nach unserer Definition gar kein Split, da ein Split der Größe 0 keine Bipartition darstellt.

Führen wir nun noch eine hilfreiche Notation ein, die uns eine Unterscheidung der beiden Mengen der Bipartition einfacher bezeichnen lässt.

Notation 2.105 *Sei $S = \{A, \overline{A}\}$ ein Split über X, dann bezeichnet $S(x)$ bzw. $\overline{S}(x)$ die Menge der Bipartition des Splits, die $x \in X$ enthält bzw. nicht enthält:*

$$S(x) = \begin{cases} A & \text{für } x \in A, \\ \overline{A} & \text{sonst,} \end{cases} \qquad \text{bzw.} \qquad \overline{S}(x) = \begin{cases} A & \text{für } x \notin A, \\ \overline{A} & \text{sonst.} \end{cases}$$

Für den Split $S = \Sigma(q)$ in Abbildung 2.86 gilt:

$$S(a) = S(b) = S(c) = S(d) = \overline{S}(e) = \overline{S}(f) = \{a, b, c, d\},$$
$$\overline{S}(a) = \overline{S}(b) = \overline{S}(c) = \overline{S}(d) = S(e) = S(f) = \{e, f\}.$$

Kommen wir nun zur ersten zentralen Definition von kompatiblen Splits, die uns eine Charakterisierung von Splits über X erlaubt, die von einem phylogenetischen Baum induziert werden.

Definition 2.106 *Sei X eine Menge von Taxa und Σ eine Menge von Splits über X. Zwei Splits $S_1, S_2 \in \Sigma$ mit $S_i = \{A_i, \overline{A}_i\}$ für $i \in [1:2]$ heißen* kompatibel*, wenn mindestens einer der vier folgenden Durchschnitte leer ist:*

$$A_1 \cap A_2, \quad A_1 \cap \overline{A}_2 \quad \overline{A}_1 \cap A_2, \quad \overline{A}_1 \cap \overline{A}_2.$$

Eine Menge Σ von Splits heißt kompatibel*, wenn jedes Paar von Splits $S, S' \in \Sigma$ kompatibel ist.*

In Abbildung 2.87 sind vier verschiedene Splits über $X = \{a, b, c, d, e\}$ angegeben. Dabei ist $\Sigma = \{S_1, S_2, S_3\}$ kompatibel, während $\Sigma' = \{S_1, S_2, S_3, S_4\}$ nicht kompatibel ist. Der Leser möge dies explizit überprüfen.

$$\begin{aligned} S_1 &= \{\{a, b\}, \{c, d, e\}\} \\ S_2 &= \{\{a, b, c\}, \{d, e\}\} \\ S_3 &= \{\{a, b, c, d\}, \{e\}\} \\ S_4 &= \{\{a, c\}, \{b, d, e\}\} \end{aligned}$$

Abbildung 2.87: Beispiel: $\Sigma = \{S_1, S_2, S_3\}$ ist kompatibel, $\Sigma' = \{S_1, S_2, S_3, S_4\}$ ist nicht kompatibel

Kommen wir nun zu der bereits angekündigten Charakterisierung von Splits über X, die von einem phylogenetischen Baum stammen.

Theorem 2.107 *Eine Menge Σ von Splits über X, die alle trivialen Splits von X enthält, ist genau dann kompatibel, wenn es einen Baum T über X mit $\Sigma(T) = \Sigma$ gibt.*

Beweis: $\Leftarrow$**:** Sei also T ein Baum über X. Betrachten wir zwei Splits aus $\Sigma(T)$, namentlich $\Sigma(e)$ und $\Sigma(f)$ für zwei Kanten $e, f \in E(T)$, wie in der folgenden Abbildung 2.88 illustriert. Offensichtlich gilt

$$\begin{aligned} \Sigma(e) &= \{X(T_1), X(T_2) \cup X(T_3)\}, \\ \Sigma(f) &= \{X(T_1) \cup X(T_2), X(T_3)\}, \end{aligned}$$

wobei $X(T)$ die Menge der Markierungen aus X für einen Baum T bezeichnet. Wie man leicht sieht, sind $\Sigma(e)$ und $\Sigma(f)$ kompatibel, da $X(T_1) \cap X(T_3) = \emptyset$. ■

04.07.24

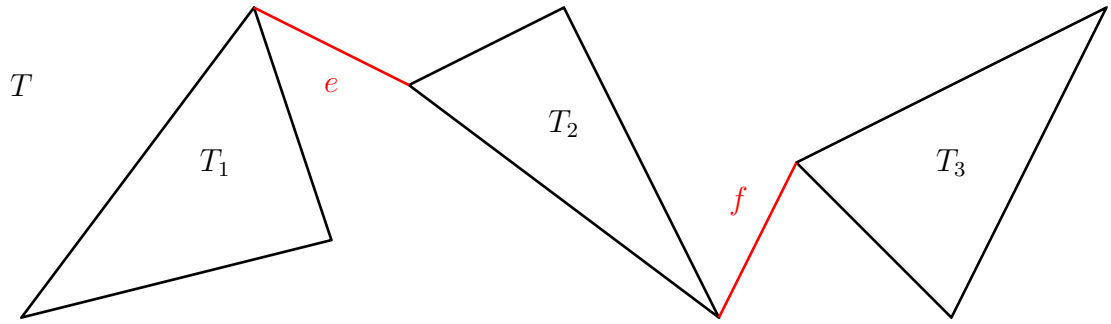


Abbildung 2.88: Skizze: Splits in T

Literaturhinweise A

A.1 Lehrbücher zur Vorlesung

S. Aluru (Ed.): *Handbook of Computational Molecular Biology*, Chapman and Hall/CRC, 2006.

H.-J. Böckenhauer, D. Bongartz: *Algorithmischen Grundlagen der Bioinformatik: Modelle, Methoden und Komplexität*, Teubner, 2003.

P. Clote, R. Backofen: *Introduction to Computational Biology*; John Wiley and Sons, 2000.

R. Deonier, S. Tavaré, M.S. Waterman: *Computational Genome Analysis: An Introduction*; Springer, 2005

A. Dress, K.T. Huber, J. Koolen, V. Moulton, A. Spillner: *Basic Phylogenetic Combinatorics*; Camvbridge University Press, 2012.

J. Felsenstein: *Inferring Phylogenies*; Sinauer Associates, 2004.

M.C. Golumbic: *Algorithmic Graph Theory and Perfect Graphs*; Academic Press, 1980.

D. Gusfield: *Algorithms on Strings, Trees, and Sequences — Computer Science and Computational Biology*; Cambridge University Press, 1997.

D.H. Huson, R. Rapp, C. Scornavacca: *Phylogenetic Networks — Concepts, Algorithms and Applications*; Cambridge University Press, 2010.

V. Mäkinen, F. Cunial, D. Belazzougui, A.I. Tomescu: *Genome-Scale Algorithm Design*, Cambridge University Press, 2015.

M. Nei, S. Kumar: *Molecular Evolution and Phylogenetics*, Oxford University Press, 2000.

C. Semple, M. Steel: *Phylogenetics*, Oxford Lecture Series in Mathematics and its Applications, Vol. 24. Oxford University Press, 2003.

J.C. Setubal, J. Meidanis: *Introduction to Computational Molecular Biology*; PWS Publishing Company, 1997.

W.-K. Sung: *Algorithms in Bioinformatics — A Practical Introduction*, CRC Press, 2009.

A.2 Andere Skripten zur Vorlesung

V. Heun: *Algorithmische Bioinformatik I/II*, Ludwig-Maximilians-Universität München, www.bio.ifi.lmu.de/~heun/lecturenotes

V. Heun: *Algorithmen auf Seqeunzen*, Ludwig-Maximilians-Universität München, www.bio.ifi.lmu.de/~heun/lecturenotes

R. Shamir: *Algorithms in Molecular Biology* Tel Aviv University, www.math.tau.ac.il/~rshamir/algmb.html.

A.3 Originalarbeiten

A.3.1 Genomische Kartierung

A. Bergeron, C. Chauve, F. de Montgolfir, M. Raffinou: Computing Common Intervals of k Permutations. with Applications to Modular Decompositions of Grpahs, *SIAM Journal on Discrete Mathematcis*, Vol. 22(3), 1022–1039, 2008.

K.S. Booth, G.S. Lueker: Testing for the Consecutive Ones Property, Interval Graphs, and Graph Planarity Using PQ-Tree Algorithms, *Journal of Computer and Systems Science*, Vol. 13(3), 335–379, 1976.

C. Chauve, E. Tannier: A Methodological Framework for the Reconstruction of Contiguous Regions of Ancestral Genomes and Its Application to Mammalian Genomes, *PLOS Computationa l Biology*, Vol. 4(11), e1000234, 2008.

W.-L. Hsu: PC-Trees vs. PQ-Trees; *Proceedings of the 7th Annual International Conference on Computing and Combinatorics, COCOON 2001*, Lecture Notes in Computer Science 2108, 207–217, Springer-Verlag, 2001.

W.-L. Hsu: A Simple Test for the Consecutive Ones Property; *Journal of Algorithms*, Vol.43, No.1, 1–16, 2002.

W.-L. Hsu: On Physical Mapping Algorithms — An Error-Tolerant Test for the Consecutive Ones Property, *Proceedings of the Third Annual International Conference on Computing and Combinatorics, COCOON'97*, LNCS Vol. 1276, Springer, 242–250, 1997.

W.-L. Hsu, R.M. McConell: PC-Trees and Circular-Ones Arrangements, *Theoretical Computer Science*, Vol. 296, 99–116, 2003.

H. Jiang, H. Liu, C. Chauve, B. Zhu: Breakpoint Distance and PQ-Trees, *Information and Computation*, Vol. 275, Article No. 104584, 2020.

H. Kaplan, R. Shamir: Bounded Degree Interval Sandwich Problems; *Algorithmica*, Vol. 24, 96–104, 1999.

G.M. Landau, L. Parida, O. Weimann: Gene Proximity Analysis across Whole Genomes via PQ Trees, *Journal of Computational Biology* Vol. 12(10), 1289–1306, 2005.

W.-F. Lu, W.-L.. Hsu: A Test for the Consecutive Ones Property on Noisy Data — Application to Physical Mapping and Sequence Assemby, *Journal of Computational Biology* Vol. 10(05), 709–735, 2003.

J. Meidanis, O. Porto, G.P. Telles: On the Consecutive Ones Property, *Discrete Applied Mathematics*, Vol. 88, 325–354, 1998.

G.P. Telles, J. Meidanis: Building PQR-Trees in Almost-Linear Time, *Technical Report, IC-03-026*, Instituto de Computação, Universidade Estadual de Campinas, 2003.

G.P. Telles, J. Meidanis: Building PQR trees in almost-linear time, *Electronic Notes in Discrete Mathematics*, Vol. 19, 33–39, 2005, dx.doi.org/10.1016/j.endm.2005.05.006

G.R. Zimerman, D. Svetlitsky, M. Zehavi, M. Ziv-Ukelsin: Approximate Search for Known Gene Clusters in New Genoms Using PQ-Trees, *Algorithms for Molecular Biology*, Vol. 16, Article No. 16, 2021.

A.3.2 Evolutionäre Bäume

H.-J. Bandelt, A. Dress: Reconstructing the Shape of a Tree from Observed Dissimliarity Data, *Advances in Applied Mathematics* Vol. 7, 307–343, 1986.

H.-J. Bandelt, A. Dress: A Canonical Decomposition Theory for Metrics on a Finite Set, *Advances in Mathematics*, Vol. 92, 47–105, 1992.

T. Chen, M.-Y. Kao: On the Informational Asymmetry Between Upper and Lower Bounds for Ultrametric Evolutionary Trees, *Proceedings of the 7th Annual European Symposium on Algorithms, ESA'99*, Lecture Notes in Computer Science 1643, 248–256, Springer-Verlag, 1999.

D. Eppstein: Fast Hierarchical Clustering and Other Applications of Dynamic Closest Pairs, *ACM Journal of Experimental Algorithmics*, Vol. 5, Article No. 1, 2000.

M. Farach, S. Kannan, T. Warnow: A Robust Model for Finding Optimal Evolutionary Trees, *Algorithmica*, Vol. 13, 155–179, 1995.

I. Gronau, S. Moran: Optimal implementations of UPGMA and other common clustering algorithms, *Information Processing Letters*, Vol. 104, No. 6, 205–210, 2007.

D. Gusfield: Efficient Algorithms for Inferring Evolutionary Trees, *Networks*, Vol. 21, 19–28, 1991.

V. Heun: Analysis of a Modification of Gusfield's Recursive Algorithm for Reconstructing Ultrametric Trees. *Information Processing Letters*, Vol. 108, No. 4, 222–225, 2008.

K.T. Huber, V. Moulton, M. Steel: Four characters suffice, *Proceedings of Formal Power Series and Algebraic Combinatorics, (FPSAC 2003)*, 133–139, Linköpings Universitet, 2003.

K.T. Huber, V. Moulton, M. Steel: Four Characters Suffice to Convexly Define a Phylogenetic Tree, *SIAM Journal on Discrete Mathemtaics*, Vol. 18(4), 835–843, 2005.

S. Kannan, T. Warnow: Triangulating Three-Colored Graphs, *SIAM Journal on Discrete Mathematics*, Vol. 5, 249–258, 1992.

S. Kannan, T. Warnow: Inferring Evolutionary History from DNA Sequences, *SIAM Journal on Computing*, Vol. 23, 713–737, 1994.

S. Kannan, T. Warnow: A Fast Algorithms dor the Computation and Enumeration of Perfect Phylogenies, *SIAM Journal on Computing*, Vol. 26, 1749–1763, 1997.

F.R. McMorris, T. Warnow, T. Wimer: Triangulating Vertex-Colored Graphs, *SIAM Journal on Discrete Mathematics*, Vol. 7, 296–306, 1994.

F. Murtagh: Complexities of Hierarchic Clustering Algorithms: State of the Art, Computational Statistics Quaterly, Vol. 1, Issue 2, 101–113, 1984.

C. Semple, M. Steel: Tree Reconstruction from Multi-States Characters, *Advances in Applied Mathematics*, Vol. 28, 169–184, 2002.

T. Warnow: Constructing Phylogenetic Trees Eficiently Using Compatiibility Criteria, *New Zealand Journal on Botany*, Vol. 31, 239–248, 1993.

A.3.3 Kombintorische Proteinfaltung

J.M. Kleinberg: Efficient Algorithms for Protein Sequence Design and the Analysis of Certain Evolutionary Fitness Landscapes, *Proceedings of the 3rd ACM International Conference on Computational Molecular Biology, RECOMB'99*, 1999.

W.E. Hart: On the Computational Complexity of Sequence Design Problems, *Proceedings of the 2nd Conference on Computational Molecular Biology, RECOMB 98,* 128–136, 1998.

S. Sun, R. Brem, H.S. Chan, K.A. Dill: Designing Amino Acid Sequences to Fold With Good Hydrophobic Cores, *Protein Engineering*, Vol. 8, No. 12, 1205–1213, 1995.

Index B

D

E

F

G

H

I

K

V

W

Z