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*


**2. Mai 2024**

*Version 8.06*

# 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.

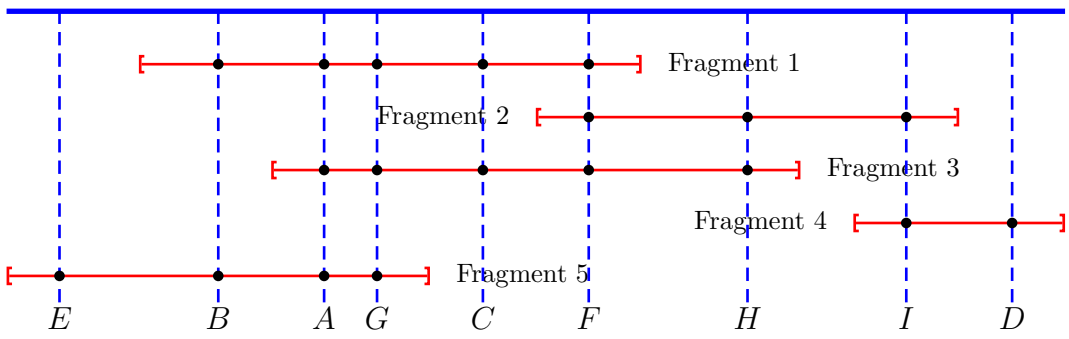



Abbildung 1.1: Skizze: Genomische Kartierung

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 $\Sigma$ ein endliches Alphabet. Dann ist ein* PQ-Baum *über $\Sigma$ induktiv wie folgt definiert:*

- *Jeder einelementige Baum (also ein Blatt), das mit einem Zeichen aus $\Sigma$ markiert ist, ist ein PQ-Baum über $\Sigma$.*
- *Sind $T_1, \ldots, T_k$ PQ-Bäume über $\Sigma$, 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 $\Sigma$.*
- *Sind $T_1, \ldots, T_k$ PQ-Bäume über $\Sigma$, 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 $\Sigma$.*

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

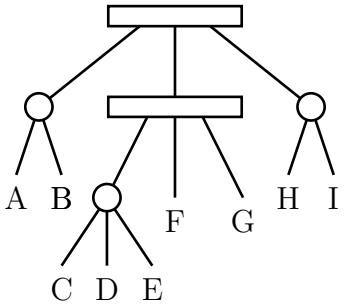



Abbildung 1.4: Beispiel: Ein PQ-Baum

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

**Definition 1.2** *Sei $\Sigma$ ein endliches Alphabet. Ein PQ-Baum über $\Sigma$ 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 $\Sigma$ 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 $\Sigma$. Die* Frontier *von $T$, kurz $f(T)$ ist die Permutation über $\Sigma$, 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 $\Sigma$ 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 $\Sigma$. Dann bezeichnet $\Pi(\Sigma, \mathcal{F})$ die Menge der Permutationen über $\Sigma$, 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.

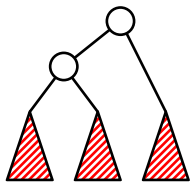


Abbildung 1.6: Skizze: Drei partielle Teilbäume

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.

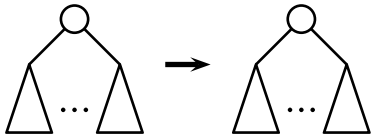


Abbildung 1.7: Skizze: Schablone $P_0$

#### 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.

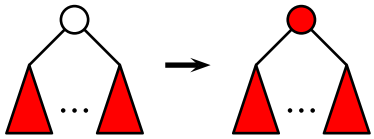


Abbildung 1.8: Skizze: Schablone $P_1$

#### 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.

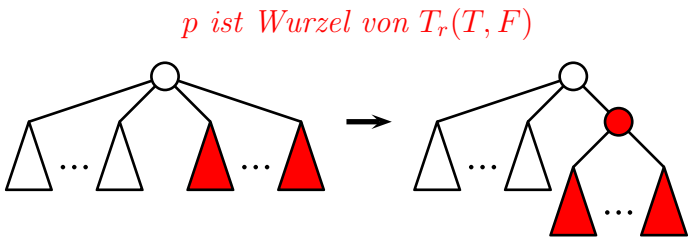



Abbildung 1.9: Skizze: Schablone $P_2$

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.

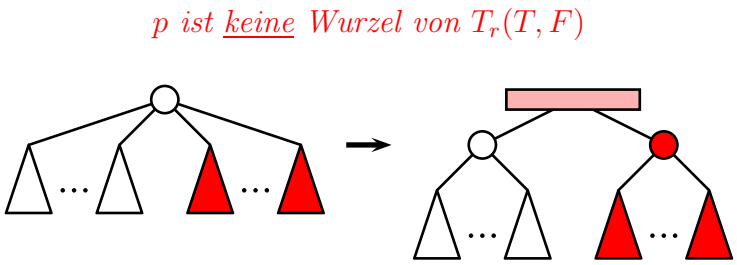



Abbildung 1.10: Skizze: Schablone $P_3$

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

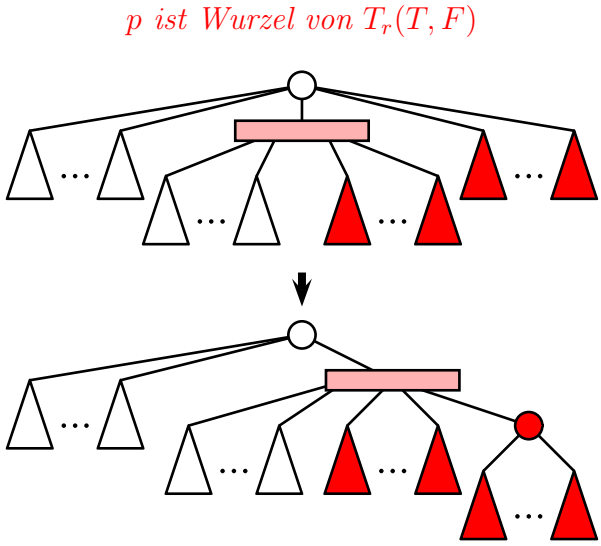



Abbildung 1.11: Skizze: Schablone $P_4$

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

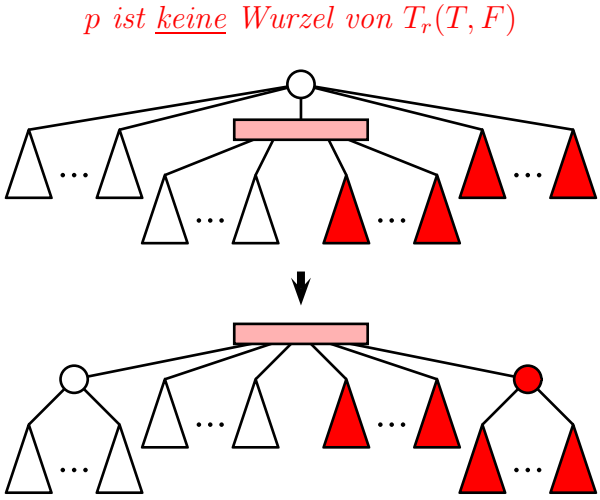



Abbildung 1.12: Skizze: Schablone $P_5$

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-

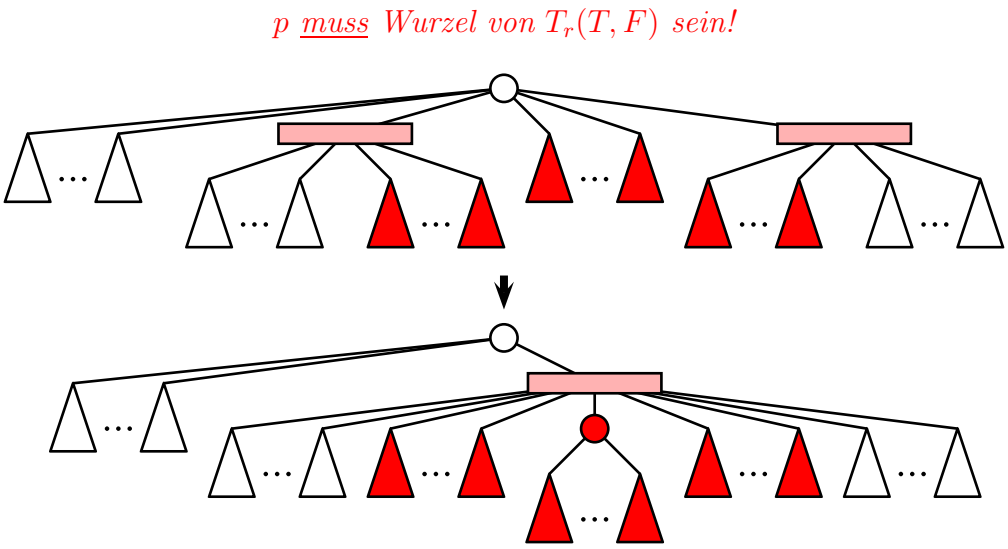



Abbildung 1.13: Skizze: Schablone $P_6$

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.

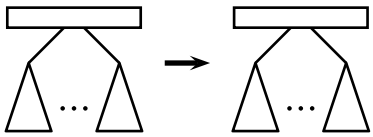


Abbildung 1.14: Skizze: Schablone $Q_0$

#### 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).

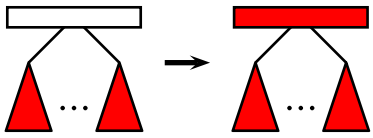


Abbildung 1.15: Skizze: Schablone $Q_1$

#### 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.

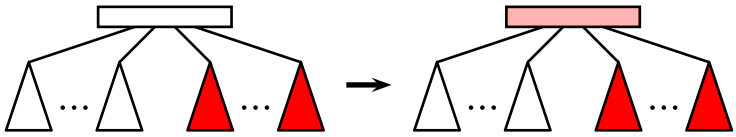


Abbildung 1.16: Skizze: Schablone $Q_2$

#### 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.

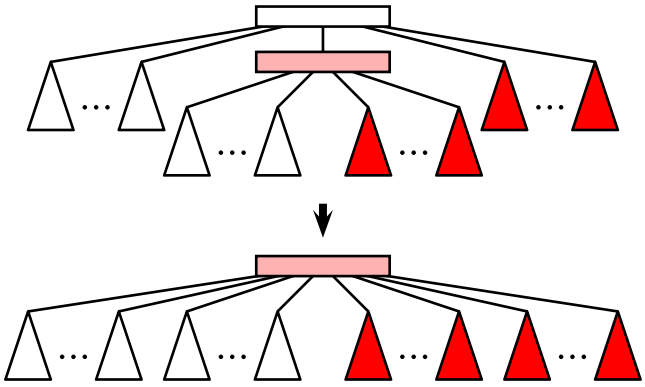


Abbildung 1.17: Skizze: Schablone $Q_3$

#### 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!*

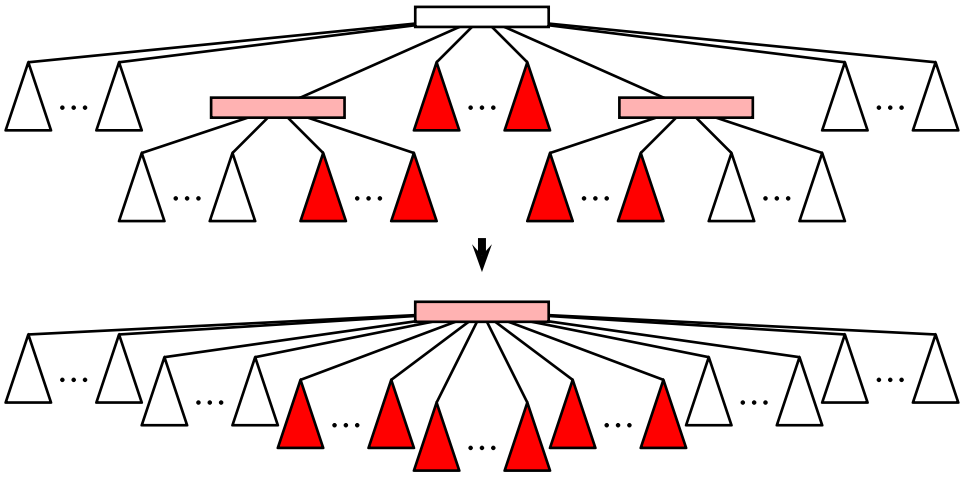


Abbildung 1.18: Skizze: Schablone $Q_4$

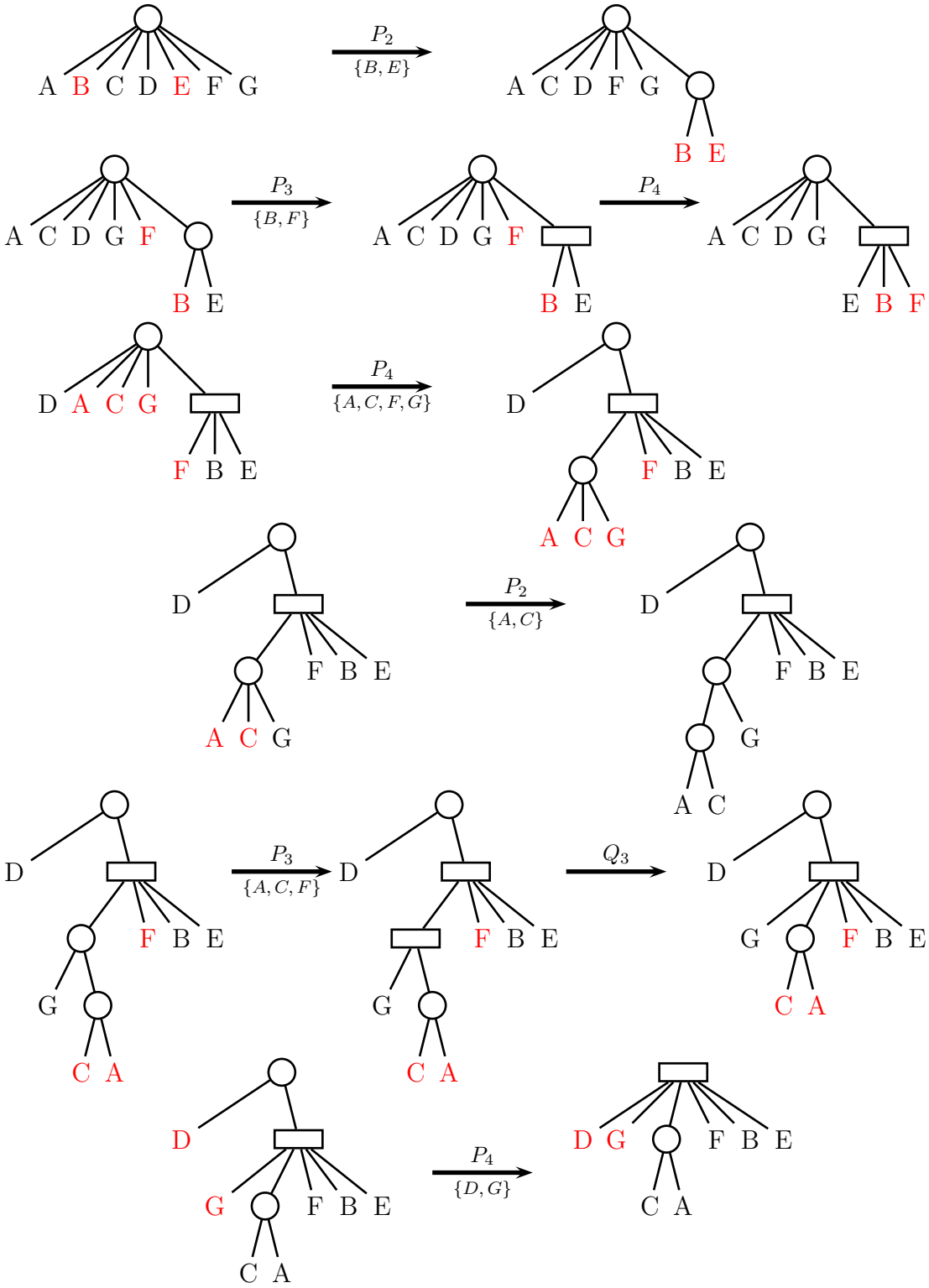



Abbildung 1.19: Beispiel: Konstruktion eines PQ-Baumes

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 $\Sigma$ 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 $\pi$ 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.

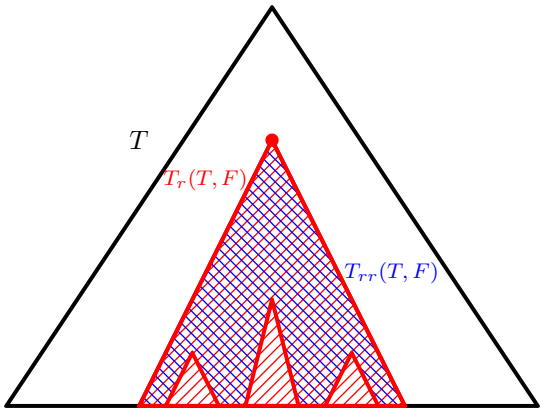



Abbildung 1.20: Skizze: Bearbeitete Teilbäume bei $\text{reduce}(T, F)$

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.

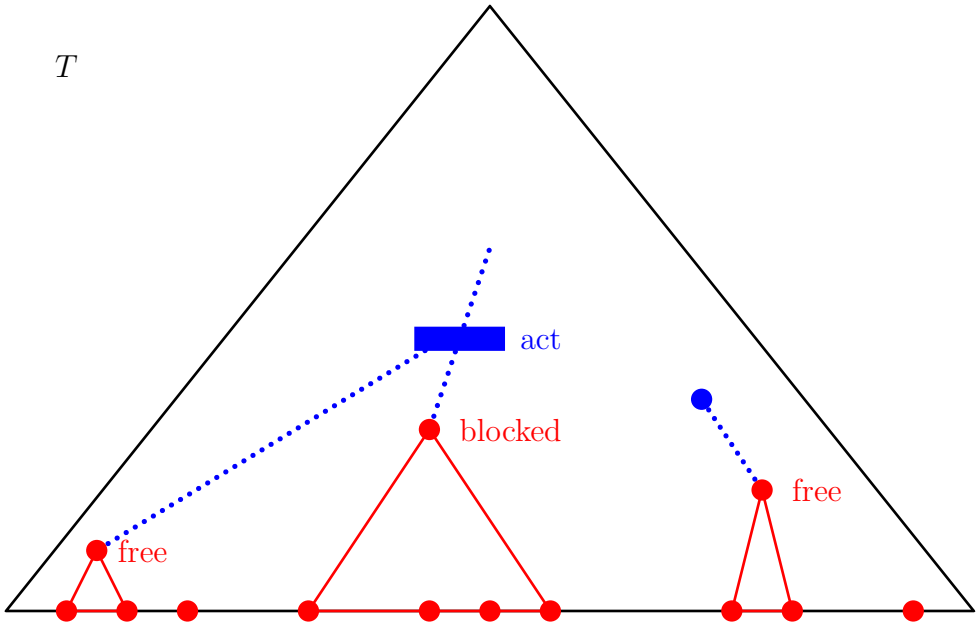



Abbildung 1.22: Skizze: Freie und blockierte Knoten im Teilbaum bei reduce$(T, F)$

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.

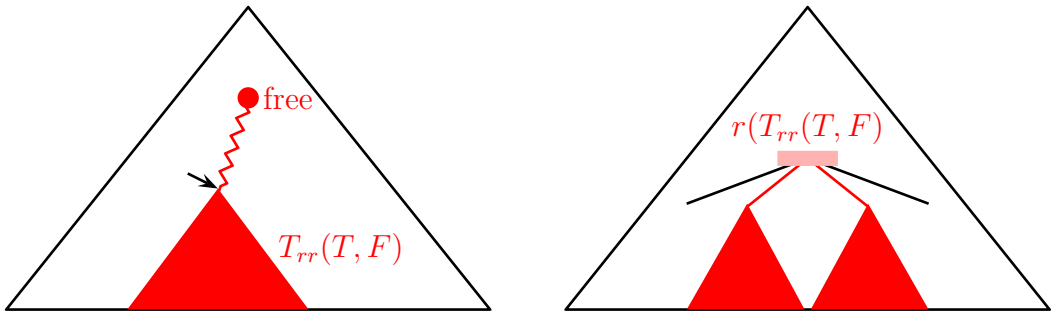



Abbildung 1.23: Skizze: Abbruch bei reduce$(T, F)$

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 $\Sigma$ 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 Abschnitt**??** 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.

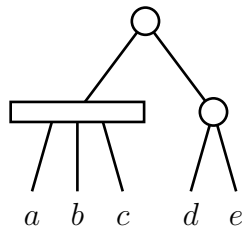
**Definition 1.13** *Die* Linearisierung *eines PQ-Baumes $T$ ist induktiv wie folgt definiert:*

*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 $\pi$ 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 $\pi$ ist ein PQ-Baum $T$ mit $\pi \subseteq \text{cons}(T)$.*

*Ein Konsensus-PQ-Baum $T$ für $\pi$ heißt* minimal*, wenn kein Konsensus-PQ-Baum $T'$ für $\pi$ 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 $\pi_1$ 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 $\pi$, wenn die Elemente der Menge $[r:s]$ konsekutiv in $\pi_i$ für alle $i \in [1:k]$ auftreten.*

*Die Menge aller common intervals von $\pi$ 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 $\Sigma$. 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 $\pi$.*

**Beweis:** Zuerst stellen wir fest, dass $T = \text{reduce}(T(\Sigma), C(\pi))$ ein Konsensus-PQ-Baum für $\pi$ ist. Für einen Widerspruchsbeweis nehmen wir an, dass $\hat{T} \neq T$ ein minimaler Konsensus-PQ-Baum für $\pi$ 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 $\pi$, 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 $\pi$ 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

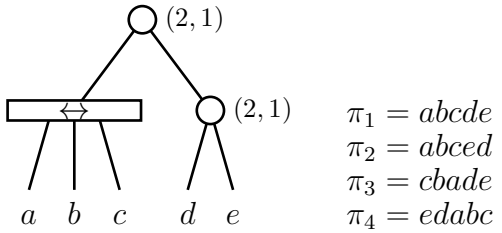



Abbildung 1.25: Beispiel: Spezialisierung eines PQ-Baum

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 $\pi$ kann sehr einfach konstruiert werden. Zuerst wird die Menge $C(\pi)$ der common intervals von $\pi$ 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 $\pi$ 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**

# 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