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

25. April 2024

Version 8.04

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.

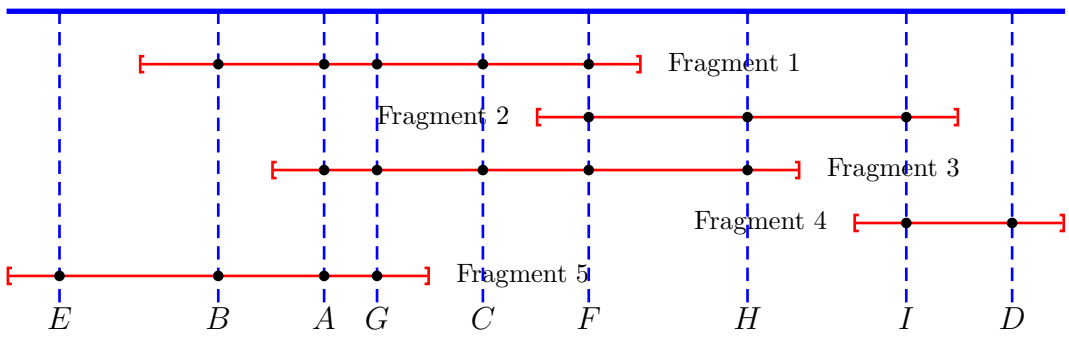

Abbildung 1.1: Skizze: Genomische Kartierung

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

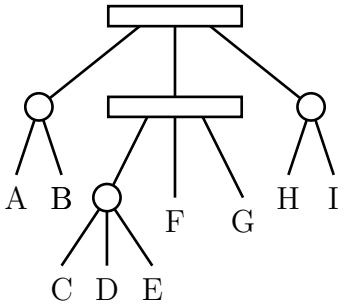

Abbildung 1.4: Beispiel: Ein PQ-Baum

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.

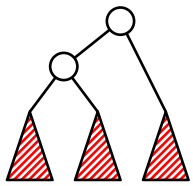

Abbildung 1.6: Skizze: Drei partielle Teilbäume

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.

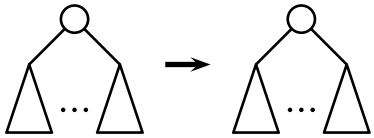

Abbildung 1.7: Skizze: Schablone P_0

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.

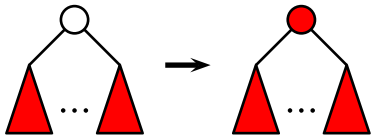

Abbildung 1.8: Skizze: Schablone P_1

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.

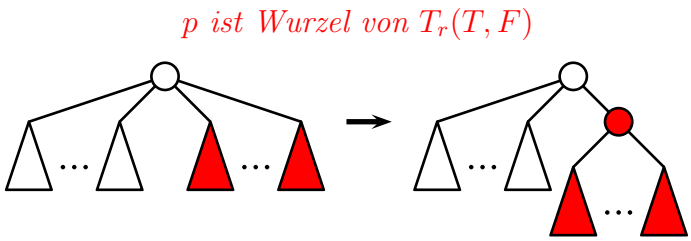

Abbildung 1.9: Skizze: Schablone P_2

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.

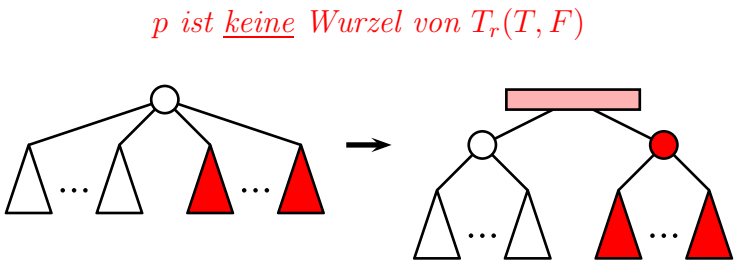

Abbildung 1.10: Skizze: Schablone P_3

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

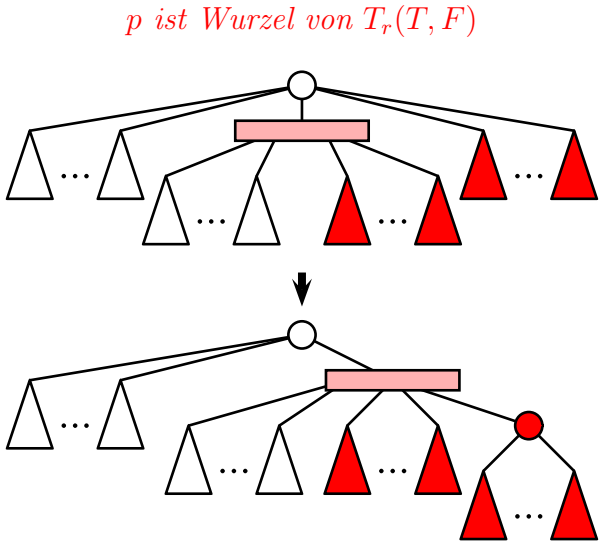

Abbildung 1.11: Skizze: Schablone P_4

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

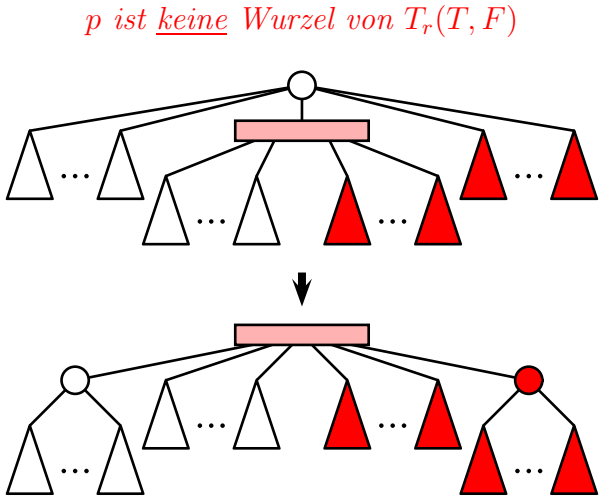

Abbildung 1.12: Skizze: Schablone P_5

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-

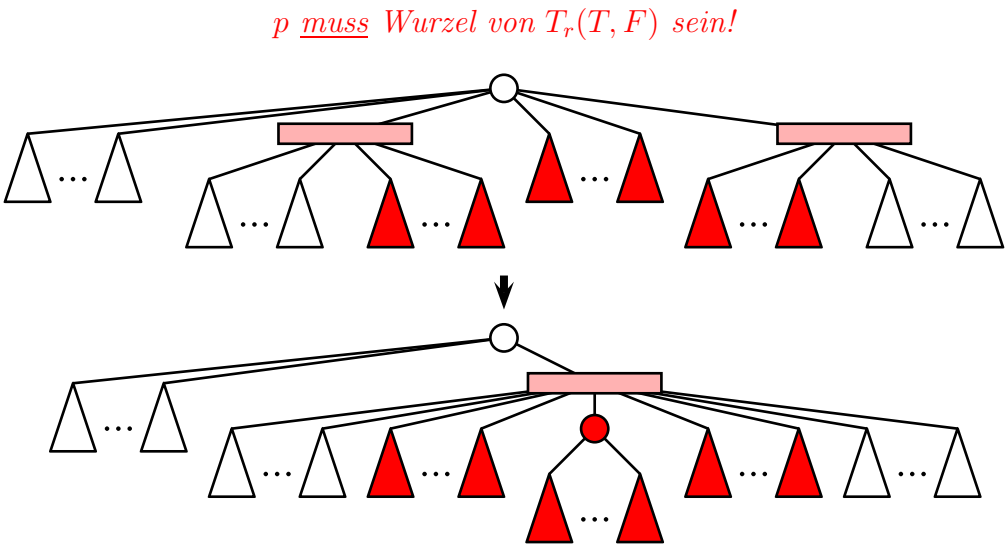

Abbildung 1.13: Skizze: Schablone P_6

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.

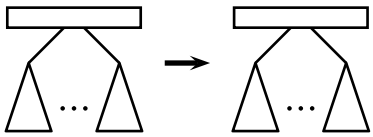

Abbildung 1.14: Skizze: Schablone Q_0

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).

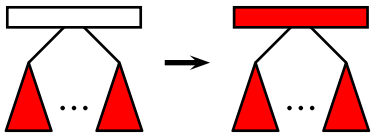

Abbildung 1.15: Skizze: Schablone Q_1

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.

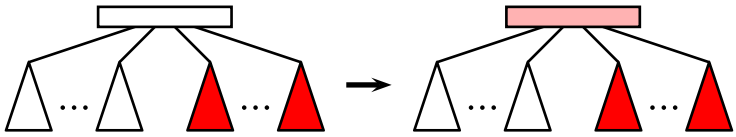

Abbildung 1.16: Skizze: Schablone Q_2

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.

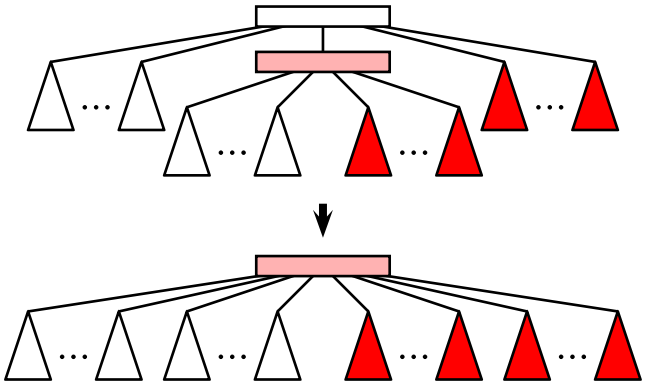

Abbildung 1.17: Skizze: Schablone Q_3

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!

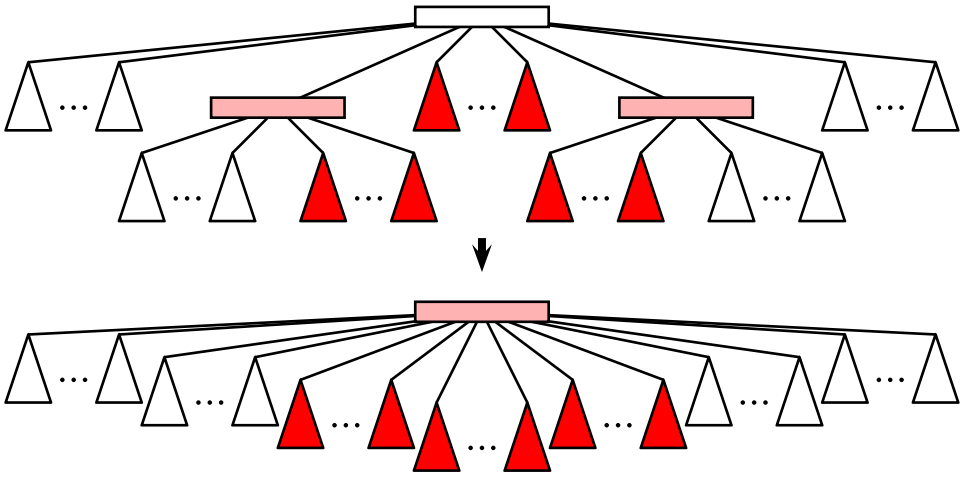

Abbildung 1.18: Skizze: Schablone Q_4

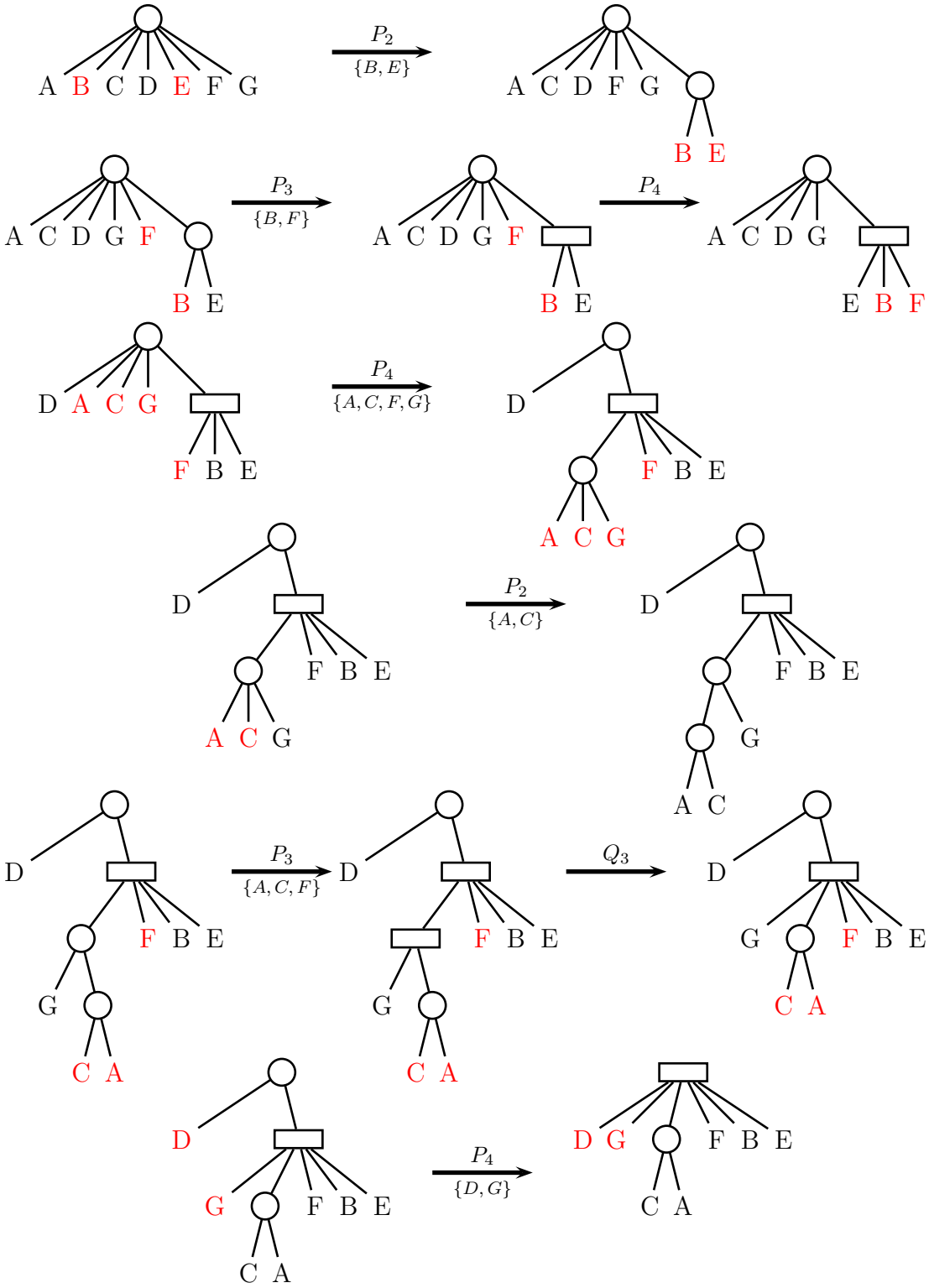

Abbildung 1.19: Beispiel: Konstruktion eines PQ-Baumes

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.

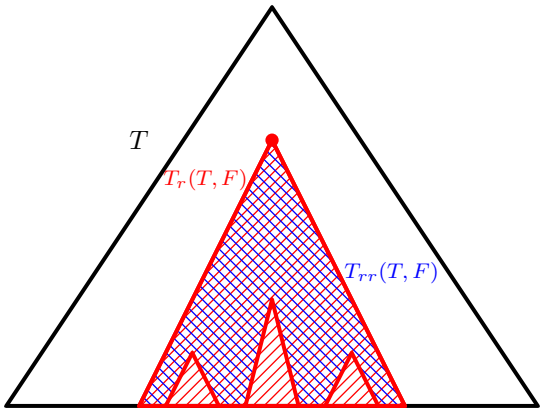

Abbildung 1.20: Skizze: Bearbeitete Teilbäume bei $\text{reduce}(T, F)$

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 sofort einen leeren Baum zurück.

25.04.24

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