INSTITUT FÜR INFORMATIK
DER TECHNISCHEN UNIVERSITÄT MÜNCHEN
LEHRSTUHL FÜR EFFIZIENTE ALGORITHMEN

Skriptum

zur Vorlesung

Algorithmische Bioinformatik I/II

gehalten im Wintersemester 2001/2002

und im Sommersemester 2002 von

Volker Heun

Erstellt unter Mithilfe von:

Peter Lücke – Hamed Behrouzi – Michael Engelhardt

Sabine Spreer – Hanjo Täubig

Jens Ernst – Moritz Maaß

14. Mai 2003

Version 0.96

Vorwort

Dieses Skript entstand parallel zu den Vorlesungen *Algorithmische Bioinformatik I* und *Algorithmische Bioinformatik II*, die im Wintersemester 2001/2002 sowie im Sommersemester 2002 für Studenten der Bioinformatik und Informatik sowie anderer Fachrichtungen an der Technischen Universität München im Rahmen des von der Ludwig-Maximilians-Universität und der Technischen Universität gemeinsam veranstalteten Studiengangs Bioinformatik gehalten wurde. Einige Teile des Skripts basieren auf der bereits im Sommersemester 2000 an der Technischen Universität München gehaltenen Vorlesung *Algorithmen der Bioinformatik* für Studierende der Informatik.

Das Skript selbst umfasst im Wesentlichen die grundlegenden Themen, die man im Bereich Algorithmische Bioinformatik einmal gehört haben sollte. Die vorliegende Version bedarf allerdings noch einer Ergänzung weiterer wichtiger Themen, die leider nicht in den Vorlesungen behandelt werden konnten.

An dieser Stelle möchte ich insbesondere Hamed Behrouzi, Michael Engelhardt und Peter Lücke danken, die an der Erstellung des ersten Teils dieses Skriptes (Kapitel 2 mit 5) maßgeblich beteiligt waren. Bei Sabine Spreer möchte ich mich für die Unterstützung bei Teilen des siebten Kapitels bedanken. Bei meinen Übungsleitern Jens Ernst und Moritz Maaß für deren Unterstützung der Durchführung des Übungsbetriebs, aus der einige Lösungen von Übungsaufgaben in dieses Text eingeflossen sind. Bei Hanjo Täubig möchte ich mich für die Mithilfe zur Fehlerfindung bedanken, insbesondere bei den biologischen Grundlagen.

Falls sich dennoch weitere (Tipp)Fehler unserer Aufmerksamkeit entzogen haben sollten, so bin ich für jeden Hinweis darauf (an heun@in.tum.de) dankbar.

München, im September 2002 Volker Heun

Inhaltsverzeichnis

Physical Mapping

6.1 Biologischer Hintergrund und Modellierung

Bei der *genomischem 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 genau bis auf die Position der Basenpaare ermitteln.

6.1.1 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 Orte von Deutschland bekommen will, dann wäre ein erster Schritt, die Kirchtürme aus ganz Deutschland so 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 die Bilder der Kirchtürme den Orten in Deutschland zugeordnet hat, dann kann man für die meisten Photographien entscheiden, zu welchem Ort sie gehören, sofern ein Kirchturm darauf zu sehen ist. Die äquivalente Aufgabe bei der genomischen Kartierung ist die Zuordnung der Kirchtürme auf die Orte in Deutschland. Ein Genom ist dabei im Gegensatz zu Deutschland ein- und nicht zweidimensional.

Ziel der genomischen Kartierung ist es nun ungefähr alle 10.000 Basenpaare eine charakteristische Sequenz auf dem Genom zu finden und zu lokalisieren. Dies ist wichtig für einen ersten Grob-Eindruck für ein Genom. 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 daher 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 nicht unbedingt benötigt wird, so ist diese zum einen doch hilfreich und zum anderen auch unerlässlich beim Vergleich von ähnlichen Genomen, da auch in absehbarer Zukunft aus Kostengründen nicht jedes beliebige Genom einfach einmal schnell sequenziert werden kann.

6.1.2 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 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 ein STS enthalten oder nicht.

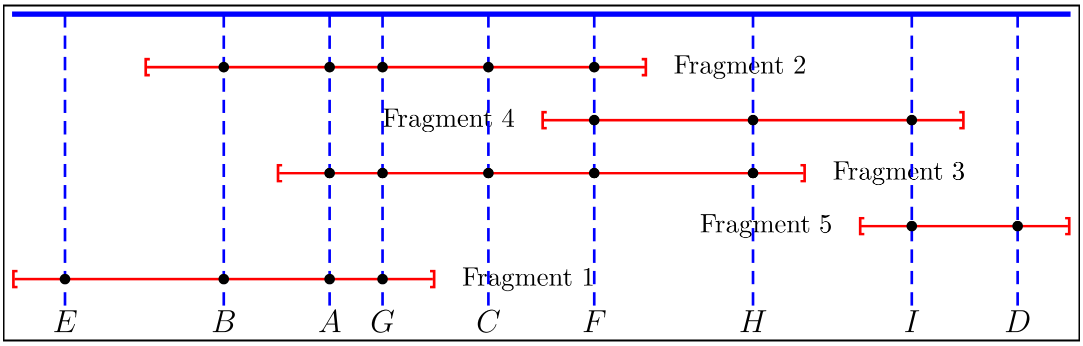

Abbildung 6.1: Skizze: Genomische Kartierung

Dies ist in Abbildung 6.1 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. In dem Beispiel, dass in der Abbildung 6.1 angegeben ist, erhält man als Ergebnis des Experiments nur die folgende Information:

$$\begin{aligned} S_1 &= \{A, B, C, F, G\}, \\ S_2 &= \{F, H, I\}, \\ S_3 &= \{A, C, F, G, H\}, \\ S_4 &= \{D, I\}, \\ S_5 &= \{A, B, E, G\}. \end{aligned}$$

Hierbei gibt die Menge S_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 zu 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.

6.1.3 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 S_i hintereinander in der jeweiligen Permutation auftreten.

Wir merken hier bereits an, dass wir im Prinzip immer mindestens zwei Lösungen erhalten, sofern es 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.

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 6.2 angegeben. Hier ist es nun unser Ziel, die Spalten so 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 6.2 ist rechts eine solche Spaltenpermutation angegeben.

	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 6.2: Beispiel: Matrizen-Darstellung

Daher sagt man auch zu einer 0-1 Matrix, die eine solche Permutation erlaubt, dass sie die *Consecutive Ones Property*, kurz *C1P*, erfüllt.

6.1.4 Fehlerquellen

Im vorigen Abschnitt haben wir gesehen, wir 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.

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.

6.2 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.

6.2.1 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 Blättern.

Definition 6.1 *Sei* $\Sigma = \{a_1, \ldots, a_n\}$ *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.*
- *Sind* $T_1, \ldots, T_k$ *PQ-Bäume, dann ist der Baum, der aus einem so genannten* P-Knoten *als Wurzel entsteht und dessen Kinder die Wurzeln der Bäume* $T_1, \ldots, T_k$ *sind, ebenfalls ein PQ-Baum.*
- *Sind* $T_1, \ldots, T_k$ *PQ-Bäume, dann ist der Baum, der aus einem so genannten* Q-Knoten *als Wurzel entsteht und dessen Kinder die Wurzeln der Bäume* $T_1, \ldots, T_k$ *sind, ebenfalls ein PQ-Baum.*

Abbildung 6.3: Skizze: Darstellung von P- und Q-Knoten

In der Abbildung 6.3 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 6.4 ist das Beispiel eines PQ-Baumes angegeben.

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

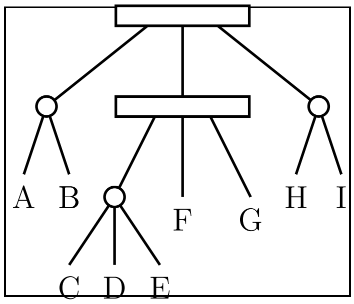

Abbildung 6.4: Beispiel: Ein PQ-Baum

Definition 6.2 *Ein PQ-Baum heißt* echt*, wenn folgende Bedingungen erfüllt sind:*

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

Der in Abbildung 6.4 angegebene PQ-Baum ist also ein echter PQ-Baum.

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

Lemma 6.3 *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 6.4 *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 6.4 ist dann ABCDEFGHI.

Definition 6.5 *Zwei echte 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.*

Definition 6.6 *Sei T ein echter PQ-Baum, dann ist die Menge der konsistenten Frontiers von T, d.h.:*

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

Beispielsweise befinden sich dann in der Menge consistent(T) für den Baum aus der Abbildung 6.4: BADCEFGIH, ABGFCDEHI oder HIDCEFGBA.

Definition 6.7 *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 PQ-Baum T mit

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

konstruieren, sofern dies möglich ist.

6.2.2 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

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

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.

Prinzipiell werden wir zur Realisierung dieser Prozedur den Baum T_{i-1} von den Blättern zur Wurzel hin durchlaufen, um gleichzeitig 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 $T_r(T_{i-1}, S_i)$ 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 6.5).

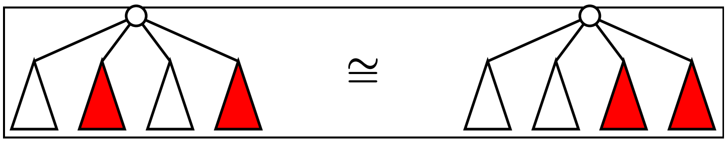

Abbildung 6.5: Skizze: Anordnung leerer und voller Kinder eines P-Knotens

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. Würde ein PQ-Baum drei partielle Knoten besitzen, von den keiner ein Nachfahre eines anderen ist, dann könnten die gewünschten Permutationen aufgrund der gegebenen Restriktionen nicht konstruiert werden. Die Abbildung 6.6 mag dabei helfen, sich dies klar zu machen.

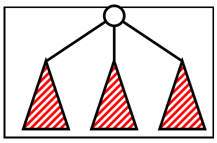

Abbildung 6.6: Skizze: Drei partielle Teilbäume

Im Folgenden werden wir jetzt 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 betrachteten Knoten (oft auch als Wurzel 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.

6.2.2.1 Schablone P_0

Die Schablone P_0 in Abbildung 6.7 ist sehr einfach. Wir betrachten einen P-Knoten, an dem nur leere Teilbäume hängen. Somit ist nichts zu tun.

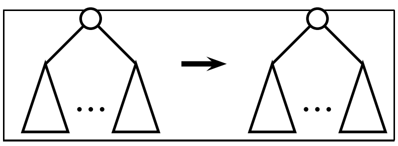

Abbildung 6.7: Skizze: Schablone P_0

6.2.2.2 Schablone P_1

Die Schablone P_1 in Abbildung 6.8 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.

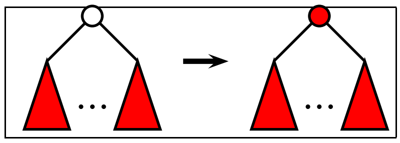

Abbildung 6.8: Skizze: Schablone P_1

6.2.2.3 Schablone P_2

Jetzt betrachten wir einen P-Knoten p, an dem nur volle und leere (also keine partiellen) Teilbäume hängen (siehe Abbildung 6.9). 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 Wurzeln ein und hängen alle volle Teilbäume der ursprünglichen Wurzel an diesen Knoten. Das 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.

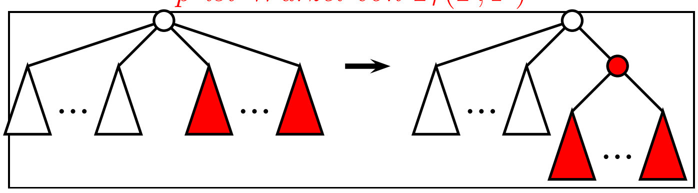

Abbildung 6.9: Skizze: Schablone P_2

Hierbei ist nur zu beachten, dass wir eigentlich nur echte PQ-Bäume konstruieren wollen. Hing also ursprünglich nur ein voller Teilbaum an der Wurzel, so führen wir die oben genannte Transformation nicht aus und belassen alles so wie es war.

In jedem Falle überzeugt man sich leicht, dass alle Frontiers, die nach der Transformation eines äquivalenten PQ-Baumes abgelesen werden könne, 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.

6.2.2.4 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 6.10).

Wir führen als neue Wurzel einen Q-Knoten ein. Alle leeren Kinder der ursprünglichen Wurzel belassen wird 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.

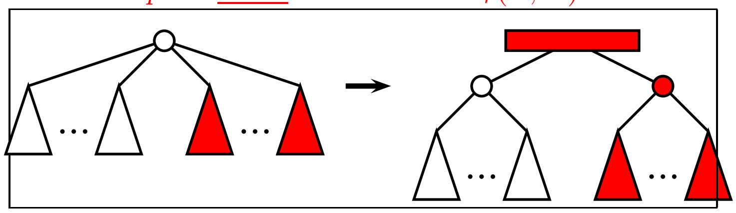

Abbildung 6.10: Skizze: Schablone P_3

Auch hier müssen wir wieder beachten, dass wir einen korrekten PQ-Baum generieren. Gab es vorher nur einen leeren oder 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.

6.2.2.5 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 6.11 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 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 vollen Teilbaum gehangen hat.

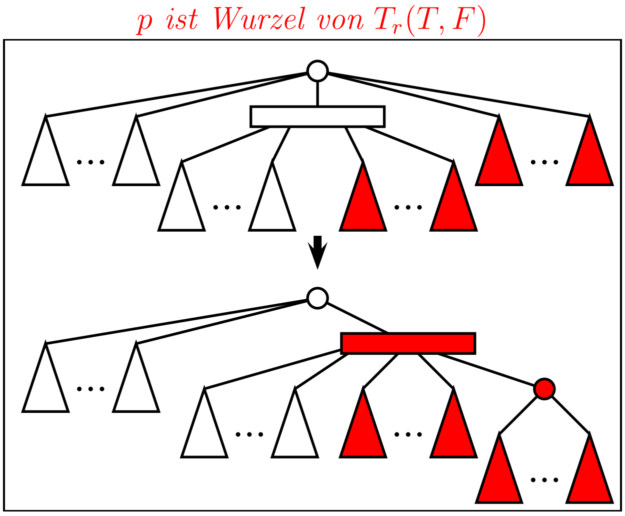

Abbildung 6.11: Skizze: Schablone P_4

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 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 hätten wir ein Problem, da der Q-Knoten dann weiterhin nur zwei Kinder hätte. In diesem Fall ersetzen wir den Q-Knoten durch einen P-Knoten, da ein Q-Knoten mit zwei Kindern dieselben Permutationen beschreibt wie ein P-Knoten.

6.2.2.6 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 6.12 illustriert.

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

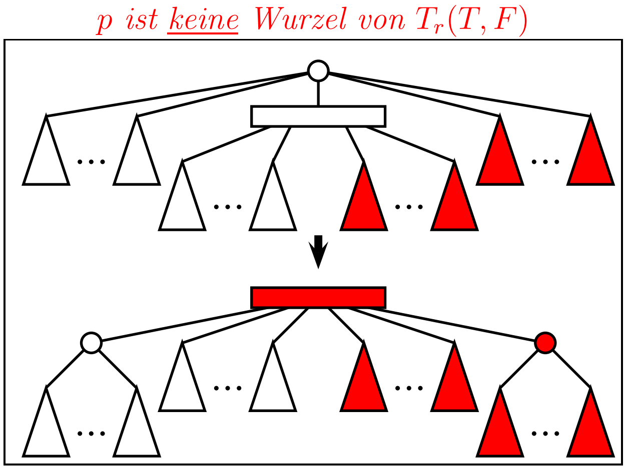

Abbildung 6.12: Skizze: Schablone P_5

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 die benötigten weiteren Kinder, 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.

6.2.2.7 Schablone P_6

Es bleibt noch der letzte Fall zu betrachten, dass an 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 6.13 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 betrachtete Restriktion sich nicht mit den Permutationen des bereits konstruierten PQ-Baumes unter ein Dach bringen lässt.

p muss Wurzel von $T_r(T, F)$ sein!

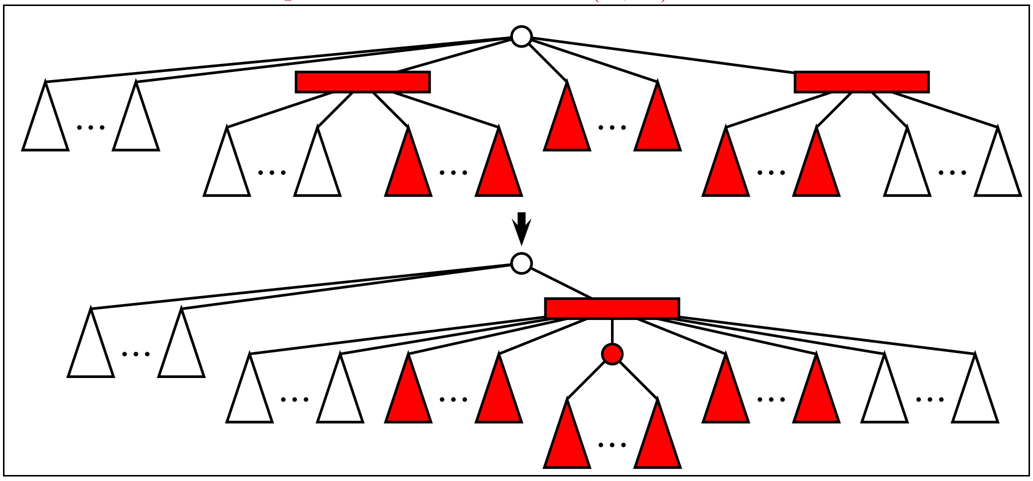

Abbildung 6.13: Skizze: Schablone P_6

Wir vereinen einfach die beiden Q-Knoten zu einem neuen und hängen die vollen Kinder der Wurzel des betrachteten Teilbaumes über eine 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.

6.2.2.8 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 6.14).

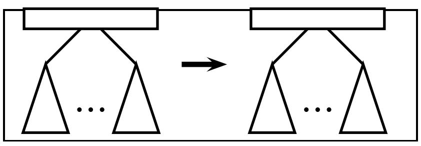

Abbildung 6.14: Skizze: Schablone Q_0

6.2.2.9 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 6.15).

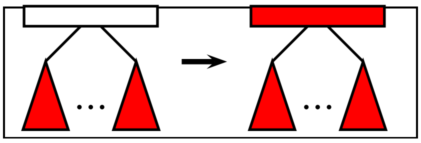

Abbildung 6.15: Skizze: Schablone Q_1

6.2.2.10 Schablone Q_2

Betrachten wir nun den Fall, dass sowohl volle wie leere Teilbäume an einem Q-Knoten hängen. In diesem Fall tun wir gar nichts, denn dann ist die Wurzel ein partieller Q-Knoten. Wir steigen also einfach im Baum weiter auf.

Kommen wir also gleich zu dem Fall, an dem an der Wurzel p des aktuell betrachteten Teilbaumes volle und leere sowie genau ein partieller Q-Knoten hängt. Wir verschmelzen nun einfach den partiellen Q-Knoten mit der Wurzel (die ebenfalls ein Q-Knoten ist), wie in Abbildung 6.16 illustriert. Falls der partielle Q-Knoten aus der Schablone P_3 entstanden ist, erhält er auch her wieder ausreichend viele Kinder.

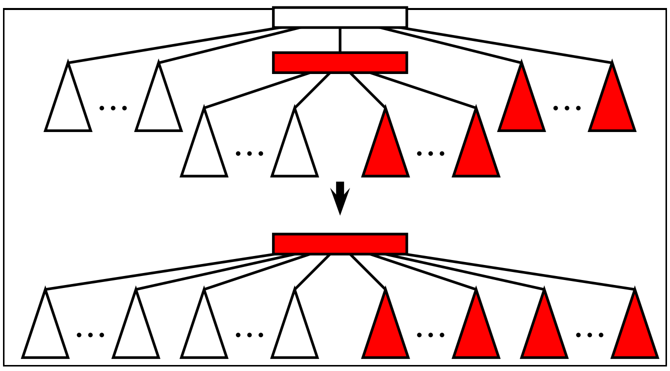

Abbildung 6.16: Skizze: Schablone Q_2

6.2.2.11 Schablone Q_3

Als letzter Fall bleibt der Fall, dass an der Wurzel des aktuell betrachteten Teilbaumes zwei partielle Q-Knoten hängen (sowie volle und leere Teilbäume). Auch hier vereinen wir die drei Q-Knoten zu einem neuen wie in Abbildung 6.17 angegeben. In diesem Fall muss der betrachtete Q-Knoten bereits die Wurzel des reduzierten Teilbaumes und die Prozedur bricht ab.

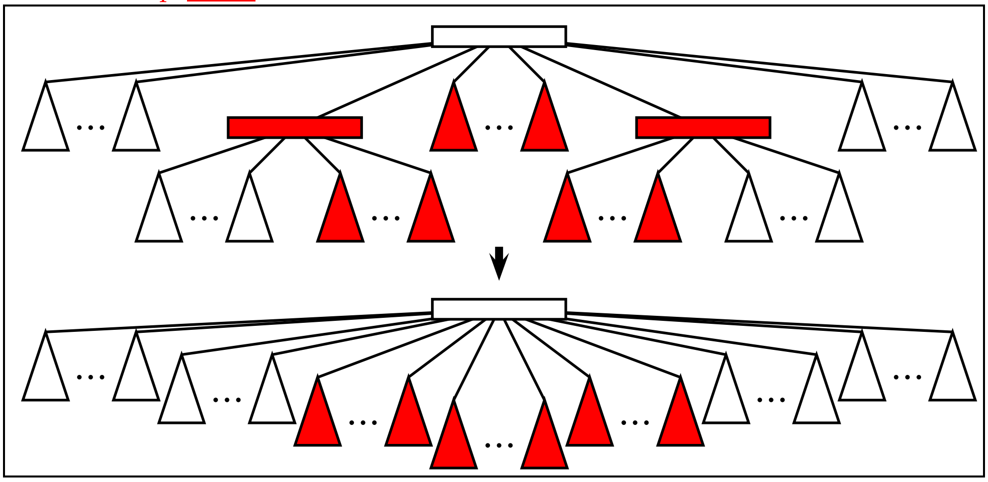

Abbildung 6.17: Skizze: Schablone Q_3

In der Abbildung 6.18 auf Seite 235 ist ein Beispiel zur Konstruktion eines PQ-Baumes für die Restriktionsmenge

$$\{\{B, E\}, \{B, F\}, \{A, C, F, G\}, \{A, C\}, \{A, C, F\}, \{D, G\}\}$$

angegeben.

6.2.3 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.

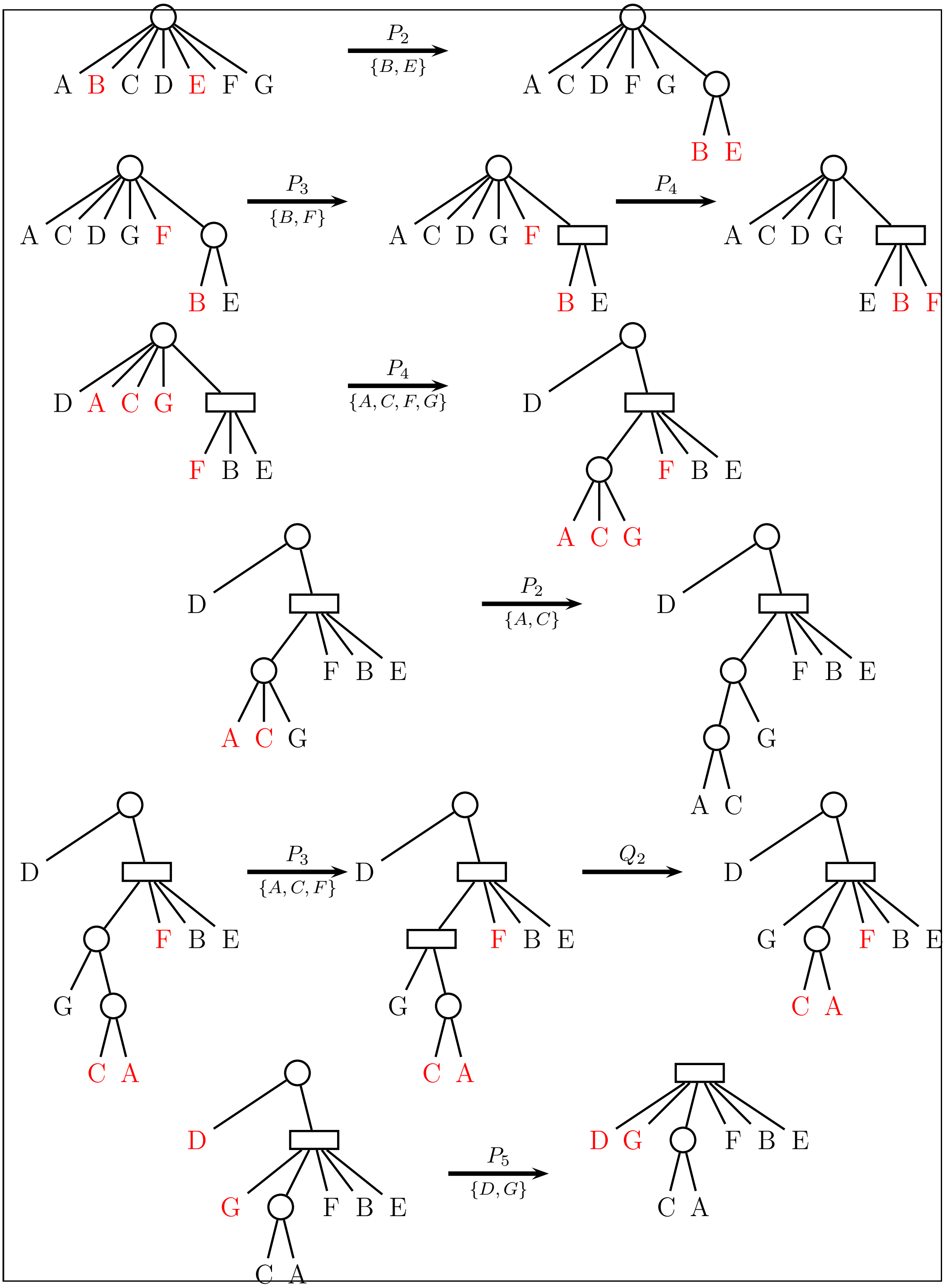

Abbildung 6.18: Beispiel: Konstruktion eines PQ-Baumes

Theorem 6.8 *Sei T eine beliebiger echter PQ-Baum und $F \subseteq \Sigma$. Dann gilt:*

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

Beweis: Zuerst führen wir zwei Abkürzungen ein:

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

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

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

Nach Konstruktion gilt $\pi \in \text{consistent}(T)$. Andererseits gilt nach Konstruktion für jeden erfolgreich abgearbeiteten Knoten x eine der folgenden Aussagen:

- x ist ein Blatt und $x \in F$,
- x ist ein voller P-Knoten,
- x ist ein Q-Knoten, dessen markierte Unterbäume alle konsekutiv vorkommen und die partiellen markierten Unterbäume (sofern vorhanden) am Rand diese konsekutiven Bereichs vorkommen.

Daraus folgt unmittelbar, dass $\pi \in \text{consistent}(T(\Sigma.F))$.

$\boldsymbol{B \subseteq A}$ **:** 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 maximale zwei partielle Kinder. Jeder partielle Knoten wird nach Konstruktion durch einen Q-Knoten ersetzt, dessen Kinnder 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 $\pi \in \text{consistent}(\text{reduce}(T', F))$. Damit ist auch $\pi \in \text{consistent}(\text{reduce}(T, F))$. ■

6.2.4 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 $\Theta(|\Sigma| + \sum_{F \in \mathcal{F}} |F|)$.

Betrachten wir den Baum T auf den wir die Operation 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. Diesee Teilbaum von T sind in Abbildung 6.19 schematisch dargestellt.

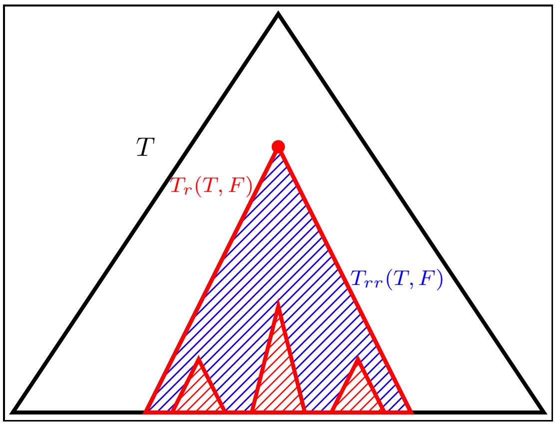

Abbildung 6.19: Skizze: Bearbeitete Teilbäume bei reduce(T, F)

Aber selbst bei nur zwei markierten Blättern, kann dieser Teilbaum sehr groß werden. 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 leer Teilbäume herausschneidet. Diese Teilbäume von T sind in Abbildung 6.19 schematisch 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 $|T_{rr}(T, F)|$ ist. 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

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

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.

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 keine obere Schranke a priori 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. Ein Kind eines P-Knotens erhält jeweils eine Verweis auf seinen Elter. Bei Q-Knoten werden nur die beiden äußersten Kinder einen Verweis auf ihren Elter erhalten. Wir werden im Folgenden sehen, dass dies völlig ausreichend sein wird.

In Abbildung 6.20 ist der Algorithmus zum Ermitteln des relevanten reduzierten Teilbaumes angegeben. Prinzipiell versuchen wir ausgehend von 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.

Um diesen Algorithmus genauer verstehen zu können, müssen wir erst noch ein paar Notationen vereinbaren. Wir halten zwei Listen als FIFO-Queue vor: die Menge free der so genannten *freien Konten* und eine Menge blocked der so genannten *blockierten Knoten*. Dazu müssen wir jedoch zuerst noch *aktive Knoten* definieren.

Ein Knoten heißt aktiv, wenn wir wissen, dass er ein Vorfahr eines markierten Blattes aus F ist. Ein aktiver Knoten heißt frei, wenn die Kante zu seinem Elter noch nicht betrachtet wurde. Ein aktiver Knoten ist 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 oder deren Eltern selbst schon frei sind (wir haben dies nur noch nicht bemerkt). Um die Notation einfacher zu halten, werden wir blockierte Knoten nicht als frei bezeichen (auch wenn diese nach obiger Definition eigentlich der Fall ist).

Wenn wir jetzt versuchen den kleinsten zusammenhängenden Teilbaum, der alle markierten Blätter enthält, konstruieren, 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.

Ein Folge von blockierten Knoten, die aufeinander folgende Kinder desselben Knotens sind (der dann ein Q-Knoten sein muss), nennen wir einen *Sektor*. Beachte,

```
FIND_TREE (tree T, set F)
{
    int sectors = 0;    set free, blocked;    for all (f ∈ F) do free.add(f);
    while (free.size() + sectors > 1)
    {
        if (free.is_empty()) return (∅, ∅);
        else
        {
            v = free.remove_FIFO();
            if (parent(v) ≠ nil)
            {
                if (parent(v) ∉ V(T_rr))
                {
                    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)
                {
                    let y s.t. x ⇌ v ⇌ y;
                    let S be the sector containing v;
                    for all (s ∈ S) do
                    {
                        blocked.remove(s);
                        parent(s) = parent(x);
                        E(T_rr) = E(T_rr) ∪ {{s, parent(s)}};
                    }
                    if (y ∈ blocked) sectors−−;
                }
                elsif (both neighbors of v are blocked) sectors−−;
                elsif (both neighbors of v are not blocked) sectors++;
            }
        }
    }
    return T_rr;
}
```

Abbildung 6.20: Algorithmus: Ermittlung von $T_{rr}(T, F)$

dass ein Sektor nie eines der äußersten Kinder eines Q-Knoten enthalten kann, da diese nach Definition frei sind.

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 Blatt hängt. 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.

Wenn immer wir mindestens zwei Sektoren und keine freie Wurzel 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 Wurzeln zu verschmelzen.

Was tut unser Algorithmus also, wenn es noch freie Wurzeln gibt? Er nimmt eine solche freie Wurzel v her und teste, 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 (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 (wir 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 eine 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, wo 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 blockierte Knoten (besitzt also kein blockiertes Geschwister), so muss die Anzahl der Sektoren um eine erhöht werden. Waren beide Geschwister blockiert, so werden diese Sektoren mithilfe von v zu einem 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 frei 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. Im zweiten Fall ist, wie gesagt, der Elter der blockierten Knoten im gefunden Sektor die gesuchte Wurzel.

6.2.5 Laufzeitanalyse

Wir haben die Lauzeit bereits mit

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

abgeschätzt, wobei $T_0 = T(\Sigma, \emptyset)$ ist und $T_i = \text{reduce}(T_{i-1}, F_i)$. Zuerst wollen wir uns noch wirklich überlegen, dass diese Behauptung stimmt. Das einzige Problem hierbei 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 6.19 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 gleich sehen werden, glücklicherweise nicht der Fall.

6.2.5.1 Die Schablonen P_0, P_1, Q_0 und Q_1

Zuerst bemerken wir, dass die Schablonen P_0 und Q_0 nie angewendet werden, da diese erstens nichts verändern und zweitens nur außerhalb des relevanten reduzierten Teilbaums anwendbar sind. Bei den Schablonen P_1 und Q_1 sind keine Veränderungen des eigentlichen PQ-Baumes durchzuführen.

6.2.5.2 Die Schablone P_2

Bei der Schablone P_2 (siehe Abbildung 6.9 auf Seite 228) 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 neuen Knoten aktualisieren.

6.2.5.3 Die Schablone P_3

Bei der Schablone P_3 (siehe Abbildung 6.10 auf Seite 229) verwenden wir den Trick, dass wir die alte Wurzel als Wurzel der leeren Teilbäume belassen. Somit muss ebenfalls nur an den Wurzeln der vollen Teilbäumen und der neu eingeführten 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.

6.2.5.4 Die Schablone P_4

Bei der Schablone P_4 (siehe Abbildung 6.11 auf Seite 230) ist dies wieder offensichtlich, da wir nur ein paar volle Teilbäume umhängen und einen neuen P-Knoten einführen.

6.2.5.5 Die Schablone P_5

Bei der Schablone P_5 (siehe Abbildung 6.12 auf Seite 231) verwenden wir denselben Trick wie bei Schablone P_3. Die alte Wurzel mitsamt ihrer Kinder wird umgehängt, so dass die eigentliche Arbeit an der vollen und neuen Knoten stattfindet.

6.2.5.6 Die Schablone P_6

Bei der Schablone P_6 (siehe Abbildung 6.13 auf Seite 232) 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 verkettet Liste implementiert haben, ist dies ebenfalls wieder mit konstantem Aufwand realisierbar.

6.2.5.7 Die Schablone Q_2

Bei der Schablone Q_2 (siehe Abbildung 6.16 auf Seite 233) 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.

Einziges Problem ist die Aktualisierung der Kinder des Kinder-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.

6.2.5.8 Die Schablone Q_3

Bei der Schablone Q_3 (siehe Abbildung 6.17 auf Seite 234) gilt die Argumentation von der Schablone Q_2 analog.

6.2.5.9 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 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.

Sind also immer mindestens zwei freie Knoten in der freien Menge. Somit 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.

6.2.6 Anzahlbestimmung angewendeter Schablonen

Da die Anzahl die Knoten im relevanten reduzierten Teilbaum gleich der angewendete 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 $\Pi(\Sigma, \mathcal{F})$.

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

Diese Schablonen werden wie bereits erwähnt nie wirklich angewendet.

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

Man überlegt sich leicht, dass solche Schablonen nur in Teilbäumen angewendet werden kann, in denen alle Blätter markiert sind. Da nach Lemma 6.3 die Anzahl der inneren Knoten durch die Anzahl der markierten Blätter beschränkt sind, gilt:

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

6.2.6.3 Bestimmung von $\sharp P_2$, $\sharp P_4$, $\sharp P_6$ und $\sharp Q_3$

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

$$\sharp P_2 + \sharp P_4 + \sharp P_6 + \sharp Q_3 = O(|\mathcal{F}|).$$

6.2.6.4 Bestimmung von $\sharp P_3$

Diese Schablone generiert einen neuen partiellen Q-Knoten, der vorher noch nicht da war (siehe auch Abbildung 6.10). Da in einem PQ-Baum mit mehr als zwei partiellen 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 angewendet werden. Daher gilt

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

6.2.6.5 Bestimmung von $\sharp P_5 + \sharp Q_2$

Hierfür definieren zunächst einmal recht willkürlich die *Norm eines PQ-Baumes* wie folgt: Die Norm eine PQ-Baumes T, in Zeichen $\|T\|$, ist die Summe aus der Anzahl der Q-Knoten 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_2 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 $S \in \{P_5, Q_2\}$ 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_2 die Norm ganzzahlig erniedrigen und mit jeder anderen Schablone die Norm ganzzahlig um maximal 1 erhöhen, können die Schablonen P_5 und Q_2 nur so oft angewendet werden, wie die anderen. Grob gesagt, es kann nur das weggenommen werden, was schon einmal hingelegt wurde. Es gilt also:

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

Die letzte Gleichung folgt aus der Annahme, dass jedes Alphabetsymbol zumindest in einer Restriktion auftritt. Im Allgemeinen kann man dies zwar nicht annehmen, aber in unserem Kontext der genomischen Kartierung ist dies durchaus sinnvoll, da Landmarks die in keinem Fragment auftreten, erst gar nicht berücksichtigt werden.

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 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 6.3 maximal $|\Sigma|$ innere Knoten besitzt, da er genau $|\Sigma|$ Blätter besitzt, folgt, dass der Aufwand höchsten $O(|\Sigma|)$ ist. Fassen wir das Ergebnis noch einmal zusammen.

Theorem 6.9 *Die Menge* $\Pi(\Sigma, \mathcal{F})$ *kann durch einen PQ-Baum mit*

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

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 voraussetzten, 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.

6.3 Intervall-Graphen

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

6.3.1 Definition von Intervall-Graphen

Zuerst benötigen wir die Definition eines Intervall-Graphen.

Definition 6.10 *Ein Menge $\mathcal{I} = \{[\ell_i, r_i] \subset \mathbb{R} : i \in [1 : n]\}$ von reellen Intervallen $[\ell_i, r_i]$ mit $\ell_i < r_i$ für alle $i \in [1 : n]$ heißt* Intervall-Darstellung.

Der zugehörige Graph $G(\mathcal{I}) = (V, E)$ ist gegeben durch

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

Ein Graph G heißt Intervall-Graph *(engl.* interval graph*), wenn es eine Intervall-Darstellung $\mathcal{I}$ gibt, so dass $G \cong G(\mathcal{I})$.*

In Abbildung 6.21 ist ein Beispiel eines Intervall-Graphen samt seiner zugehörigen Intervall-Darstellung gegeben.

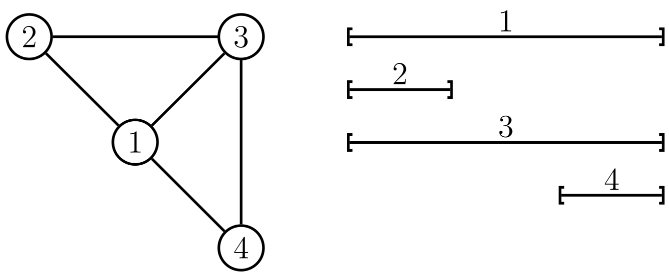

Abbildung 6.21: Beispiel: Ein Intervall-Graph samt zugehöriger Intervall-Darstellung

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

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

Definition 6.11 *Ein Intervall-Graph G heißt* echt *(engl.* proper interval graph*), wenn er eine Intervall-Darstellung $\mathcal{I}$ besitzt (d.h. $G \cong G(\mathcal{I})$), so dass*

$$\forall I \neq I' \in \mathcal{I} : (I \not\subseteq I') \wedge (I \not\supseteq I').$$

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

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

Lemma 6.12 *Ein Graph ist genau dann ein Einheits-Intervall-Graph, wenn er ein echter Intervall-Graph ist.*

Den Beweis dieses Lemmas überlassen wir dem Leser als Übungsaufgabe.

6.3.2 Modellierung

Warum sind Intervall-Graphen für die genomische Kartierung interessant. Schauen wir uns noch einmal unsere Aufgeb der genomischen Kartierung in Abbildung 6.22 an. Offensichtlich entsprechen die Fragmente gerade Intervallen, nämliche den Posi-

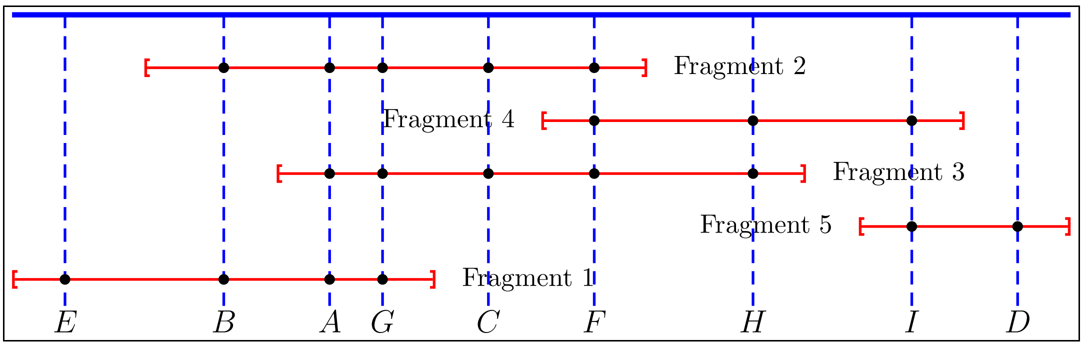

Abbildung 6.22: Skizze: Genomische Kartierung

tionen die sie überdecken. Mit Hilfe unserer Hybridisierungs-Experimente erhalten wir die Information, ob sich zwei Fragmente bzw. Intervalle überlappen, nämlich genau dann, wenn beide Fragmente dasselbe Landmark, also STS, enthalten. Somit bilden die Fragmente mit den Knoten und den Überschneidungen als Kanten einen

Intervall-Graphen. Was in der Aufgabe der genomischen Kartierung gesucht ist, ist die Anordnung der Fragmente auf dem Genom. Dies ist aber nichts anderes als eine Intervall-Darstellung des Graphen, den wir über unsere biologischen Experimente erhalten. Aus diesem Grund sind oft auch Einheits-Intervall-Graphen von Interessen, da in den biologischen Experimenten die Fragmente im Wesentlichen dieselbe Länge besitzen und somit eine Intervall-Darstellung durch gleich lange Intervalle erlauben sollte.

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

PROPER INTERVALL COMPLETION (PIC)

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

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

PROPER INTERVALL SELECTION

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

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

INTERVALL SANDWICH (IS)

Eingabe: Ein Tripel (V, D, F) mit $D, F \subset \binom{V}{2}$.
Ausgabe: Ein Intervall-Graph $G = (V, E)$ mit $D \subset E \subset F$.

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

INTERVALL SANDWICH (IS)

Eingabe: Ein Tripel (V, M, F) mit $D, F \subset \binom{V}{2}$.
Ausgabe: Ein Intervall-Graph $G = (V, E)$ mit $M \subset E$ und $E \cap F = \emptyset$.

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

Bevor wir unser letztes Problem formalisieren, benötigen wir noch die Definition von Färbungen in Graphen.

Definition 6.13 *Sei $G = (V, E)$ ein Graph. Eine Abbildung $c : V \to [1 : k]$ heißt* k-Färbung. *Eine k-Färbung heißt* zulässig, *wenn $c(v) \neq c(w)$ für alle $\{v, w\} \in E$.*

INTERVALIZING COLORED GRAPHS

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

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

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

6.3.3 Komplexitäten

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

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

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

6.4 Intervall Sandwich Problem

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

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

6.4.1 Allgemeines Lösungsprinzip

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

Definition 6.14 *Sei $S = (V, M, F)$ eine Eingabe für IS. Eine Teilmenge $X \subseteq V$ heißt* Kern. *Der* Rand $\beta(X) \subseteq M$ *eines Kerns X ist definiert als*

$$\beta(X) = \{e \in M \ : \ e \cap X \neq \emptyset\}.$$

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

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

Der Hintergrund für diese Definition ist der folgende. Der aktuell betrachtete Kern in unserem Algorithmus wird eine Knotenteilmenge sein, für die wir eine Intervall-Darstellung bereits konstruiert haben. Die aktive Region beschreibt dann die Menge von Knoten des Kerns, für die noch benachbarte Knoten außerhalb des Kerns existieren, die dann über die Kanten aus dem Rand verbunden sind.

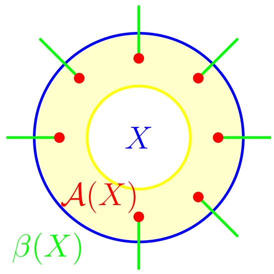

Abbildung 6.23: Skizze: Aktive Region $\mathcal{A}(X)$ und Rand $\beta(X)$ des Kerns X

Kommen wir nun dazu genauer zu formalisieren, was ein Intervall-Darstellung eines Kerns ist.

Definition 6.15 *Sei V eine Knotenmenge und $M, F \subseteq \binom{V}{2}$. Ein* Layout $L(X)$ *eines Kerns $X \subseteq V$ ist eine Funktion $I : X \to \{[a, b] \mid a < b \in \mathbb{R}\}$ mit*

1. $\forall \{v, w\} \in M \cap \binom{X}{2} \ : \ I(v) \cap I(w) \neq \emptyset$,
2. $\forall \{v, w\} \in F \cap \binom{X}{2} \ : \ I(v) \cap I(w) = \emptyset$,
3. $\forall v \in \mathcal{A}(X) : \ r(I(v)) = \max\{r(I(w)) \ : \ w \in X\}$, *wobei* $r([a, b]) = b$ *für alle* $a < b \in \mathbb{R}$.

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

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

Definition 6.16 *Ein zulässiger Kern* $Y = X \cup \{v\}$ erweitert *genau dann einen zulässigen Kern* X*, wenn* $L(Y)$ *aus* $L(X)$ *durch Hinzufügen eines Intervalls* $I(v)$ *entsteht, so dass*

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

In Abbildung 6.24 ist ein Beispiel für eine solche Erweiterung des zulässigen Kerns $\{1, 2, 3, 4\}$ zu einem zulässigen Kern $\{1, 2, 3, 4, 5\}$ dargestellt.

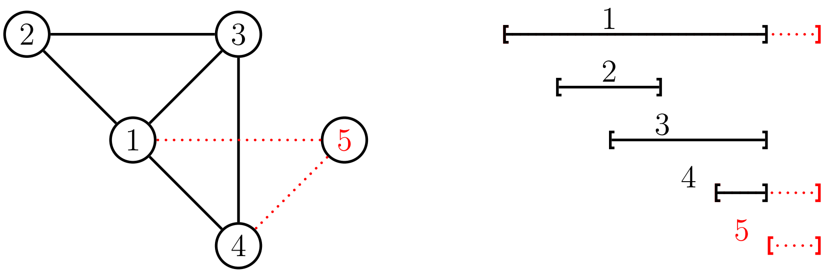

Abbildung 6.24: Skizze: Erweiterung eines Layouts

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

Lemma 6.17 *Sei* X *ein zulässiger Kern.* $Y = X \cup \{v\}$ *ist genau dann ein zulässiger Kern und erweitert* X*, wenn* $(v, w) \notin F$ *für alle* $w \in \mathcal{A}(X)$.

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

⇐: Sei also X ein zulässiger Kern. Wir betrachten das Layout von X in Abbildung 6.25 Da $(v, w) \notin F$ für alle $w \in \mathcal{A}(X)$, können wir nun alle Intervalle der

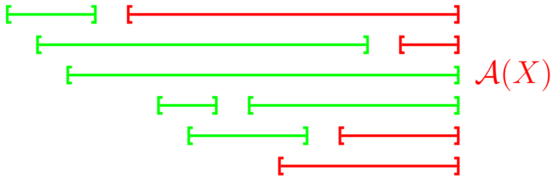

Abbildung 6.25: Skizze:

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

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

Lemma 6.18 *Jeder zulässige Kern Y erweitert mindestens einen zulässigen Kern $X \subsetneq Y$.*

Beweis: Sei $L(Y)$ mit $I : V \to \mathcal{J}(\mathbb{R})$ ein Layout für Y, wobei $\mathcal{J}(\mathbb{R})$ die Menge aller abgeschlossen reellen Intervalle bezeichnet. Wir wählen jetzt $y \in Y$, so dass $\ell(I(y))$ maximal ist. Siehe dazu auch Abbildung 6.26. Beachte, dass y nicht notwendigerweise aus $\mathcal{A}(Y)$ sein muss.

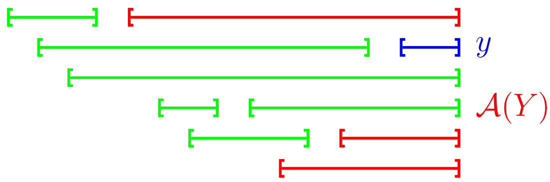

Abbildung 6.26: Skizze: Layout $L(Y)$ für Y

Wir definieren jetzt ein Layout $L(X)$ für X aus $L(Y)$ wie folgt um:

$$\forall (x, y) \in M : r(I(x)) := \max \{r(z) \ : \ z \in \mathcal{A}(Y)\}.$$

Alle anderen Werte von I auf X bleiben unverändert und y wird aus dem Definitionsbereich von I entfernt.

Wir müssen jetzt lediglich die drei Bedingungen aus der Definition eines zulässigen Layouts nachweisen. Offensichtlich gilt weiterhin $I(v) \cap I(w) \neq \emptyset$ für alle $\{v, w\} \in M \cap \binom{X}{2} \subseteq M \cap \binom{Y}{2}$. Außerdem gilt ebenfalls $I(v) \cap I(w) = \emptyset$ für alle

$\{v, w\} \in F \cap \binom{X}{2} \subseteq F \cap \binom{Y}{2}$. Letztendlich gilt nach unserer Konstruktion, dass $r(I(v)) = \max\{r(I(w)) : w \in X\}$ für alle $v \in \mathcal{A}(X)$, da man sich leicht überlegt, dass

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

Damit ist der Beweis abgeschlossen. ■

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

Korollar 6.19 *Für eine Eingabe $S = (V, M, F)$ des Interval Sandwich Problems existiert genau dann eine Lösung, wenn V ein zulässiger Kern ist.*

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

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

mit $|X_{i+1} \setminus X_i| = 1$ gibt. Das bedeutet, dass wir ein Layout iterativ für unsere Problemeingabe konstruieren können. Nach dem Lemma 6.17 wissen wir ferner, dass die Erweiterungen unabhängig vom betrachteten Layout möglich sind. Insbesondere folgt daraus, dass wenn ein Layout eines zulässigen Kerns nicht erweitert werden kann, es auch kein anderes Layout dieses Kerns geben kann, dass sich erweitern lässt.

Somit erhalten wir den in Abbildung 6.27 angegeben Algorithmus zur Konstruktion einer Intervall-Darstellung für eine gegebene Eingabe S des Intervall Sandwich Problems. Hierbei testen wir alle möglichen Erweiterungen der leeren Menge zu einem zulässigen Kern V. Da es leider exponentiell viele Erweiterungspfade gibt, nämlich genau $n!$, wenn $n = |V|$ ist, ist dieser Algorithmus sicherlich nicht praktikabel. Da der Test, ob sich ein Kern erweitern lässt nach Lemma 6.17 in Zeit $O(|V|^2)$ implementieren lässt, erhalten wir das folgende Theorem.

Theorem 6.20 *Für eine Eingabe $S = (V, M, F)$ des Interval Sandwich Problems lässt sich in Zeit $O(|V|! \cdot |V|^2)$ feststellen, ob es eine Lösung gibt, und falls ja, kann diese auch konstruiert werden.*

6.4.2 Lösungsansatz für Bounded Degree Interval Sandwich

Wir wollen nun zwei modifizierte Varianten des Intervall Sandwich Problems vorstellen, die sich in polynomieller Zeit lösen lassen. Dazu erst noch kurz die Definition eine Clique.

```
SANDWICH (S = (V, M, F))
{
    Queue Q;
    Q.enqueue(∅);
    while (not Q.is_empty())
    {
        X = Q.dequeue();
        for each v ∉ X do
            if (Y := X ∪ {v} is feasible and extends X)
                if (Y = V)
                    output Solution found;
                else
                    Q.enqueue(Y);
    }
    output No solutions found;
}
```

Abbildung 6.27: Algorithmus: Allgemeines Intervall Sandwich Problem

Definition 6.21 *Sei $G = (V, E)$ ein Graph. Eine Teilgraph $G' = (V', E')$ heißt* Clique *oder k*-Clique*, wenn folgendes gilt:*

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

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

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

BOUNDED DEGREE AND WIDTH INTERVAL SANDWICH

Eingabe: Ein Tripel (V, M, F) mit $D, F \subset \binom{V}{2}$ sowie zwei natürliche Zahlen $d, k \in \mathbb{N}$ mit $\Delta((V, M)) \leq d$.

Ausgabe: Ein Intervall-Graph $G = (V, E)$ mit $M \subset E$ und $E \cap F = \emptyset$ sowie $\omega(G) \leq k$.

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

Bounded Degree Interval Sandwich (BDIS)

Eingabe: Ein Tripel (V, M, F) und $d \in \mathbb{N}$ mit $D, F \subset \binom{V}{2}$.
Ausgabe: Ein Intervall-Graph $G = (V, E)$ mit $M \subset E$ und $E \cap F = \emptyset$ sowie $\delta(G) \leq d$.

Beim Bounded Degree Interval Sandwich Problem beschränken wir den Suchraum nur dadurch, dass wir für die Lösungen gradbeschränkte Intervall-Graphen zulassen. Wir werden jetzt für dieses Problem einen polynomiellen Algorithmus vorstellen. Für das erstgenannte Problem lässt sich mit ähnlichen Methoden ebenfalls ein polynomieller Algorithmus finden. Die beiden hier erwähnten Probleme sind auch für die genomische Kartierung relevant, da wir bei den biologischen Experimenten davon ausgehen, dass die Überdeckung einer Position im Genom sehr gering ist und aufgrund der kurzen, in etwa gleichlangen Länge der resultierende Intervall-Graph sowohl einen relativ kleinen Grad als auch eine relativ kleine Cliquenzahl besitzt.

Um unseren Algorithmus geeignet modifizieren zu können, müssen wir auch die grundlegende Definition anpassen.

Definition 6.22 *Sei V eine Menge und $M, F \subseteq \binom{V}{2}$. Ein d-*Layout *(oder kurz* Layout*) $L(X)$ eines Kerns $X \subseteq V$ ist eine Funktion $I : X \to \{[a, b] \mid a < b \in \mathbb{R}\}$ mit*

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

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

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

Im Wesentlichen sind die Bedingungen 4 und 5 in der Definition neu hinzugekommen. Die Bedingung 4 ist klar, da wir ja nur Intervall-Graphen mit maximalen Grad kleiner gleich d konstruieren wollen. Daher darf sich jeder fertig konstruierte Knoten mit maximal d anderen Intervalle schneiden. Analog ist es bei Bedingung 5. Hier gibt $|E(v, X)|$ gerade die Anzahl der Nachbarn an, die noch nicht in der aktuelle Intervall-darstellung bzw. Layout realisiert sind. Daher darf ein Intervall der aktiven Region also vorher maximal $d - |E(v, X)|$ andere Intervalle schneiden.

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

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

Lemma 6.23 *Sei X ein d-zulässiger Kern. $Y = X \cup \{v\}$ ist genau dann ein d-zulässiger Kern und erweitert X, wenn $(v, w) \notin F$ für alle $w \in \mathcal{A}(X)$ und X besitzt ein d-Layout L, so dass $I(u)$ höchstens $d - |E(u, X)| - 1$ andere Intervalle schneidet, für alle $u \in \mathcal{A}(X)$ mit $\{u, v\} \notin M$, und $|\mathcal{A}(X)| \leq d - |E(v, Y)|$.*

Beweis: **⇒:** Nach Lemma 6.17 wissen wir, dass $(v, w) \notin F$ für alle $w \in \mathcal{A}(X)$.

Sei $Y = X \cup \{v\}$ ein zulässiger Kern, der X erweitert. Sei $L(Y)$ ein Layout von Y und $L(X)$ das Layout für X, das durch Entfernen des Intervalls für v aus dem Layout $L(Y)$ entsteht. Sei weiter $u \in \mathcal{A}(X)$ mit $\{u, v\} \notin M$. Wir unterscheiden jetzt zwei Fälle, je nachdem, ob u auch in der aktiven Region von Y ist oder nicht.

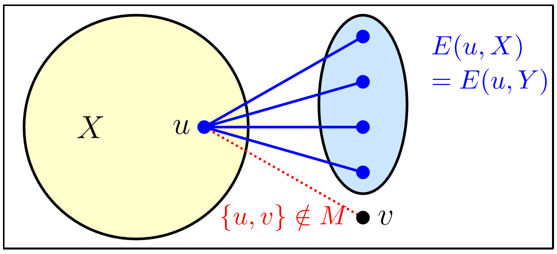

Abbildung 6.28: Skizze: Erweiterung von X um v

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

Fall 2 ($\boldsymbol{u \in \mathcal{A}(Y)}$): Da $L(Y)$ ein Layout von Y ist und sich u in der aktiven Region von Y befindet, schneidet $I(u)$ maximal $d - |E(u, Y)|$ andere Intervalle in $L(Y)$. Durch Hinzunahme von v zu X bleibt die Menge der Nachbarn von u außerhalb von X bzw. Y unverändert, d.h. $E(u, X) = E(u, Y)$ (siehe auch Abbildung 6.28).

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

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

$\Leftarrow$: Nach Lemma 6.17 folgt aus $(v, w) \notin F$ für alle $w \in \mathcal{A}(X)$ bereits, dass die Y eine Erweiterung von X und die Bedingungen 1 mit 3 für das Layout $L(Y)$ für Y gelten. Wir müssen also nur noch die Bedingungen 4 und 5 überprüfen.

Zum Nachweis der Bedingung 4 halten wir zunächst fest, dass Knoten aus X, die in X nicht mehr aktiv sind, sicherlich auch in Y nicht aktiv sind, d.h. es gilt $X \setminus \mathcal{A}(X) \subseteq Y \setminus \mathcal{A}(Y)$. Für alle Knoten aus $X \setminus \mathcal{A}(X)$ gilt also Bedingung 4. Sei jetzt also $y \in Y \setminus \mathcal{A}(Y) \setminus (X \setminus \mathcal{A}(X))$. Daher wird v jetzt inaktiv und es muss daher $\{v, y\} \in M$ gelten. Aufgrund der Bedingung 5 für das Layout $L(X)$ von X schneidet das Intervall von y maximal d andere Intervalle und die Bedingung 4 gilt.

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

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

Fall 1 ($\boldsymbol{y \in \mathcal{A}(X)}$): Ist $\{y, v\} \in M$, dann schneidet das Intervall $I(y)$ im Layout $L(X)$ von X nach Voraussetzung maximal $d - |E(y, X)|$ andere Intervalle. Da $E(y, Y) = E(y, x) \setminus \{v\}$ und somit $|E(y, X)| = |E(y, Y)| + 1$ ist, kann $I(y)$ im erweiterten Layout $L(y)$ maximal

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

andere Intervalle schneiden.

Ist andererseits $\{y, v\} \notin M$, dann schneidet $I(y)$ im Layout $L(X)$ von X nach Voraussetzung maximal $d - |E(y, X)| - 1$ andere Intervalle. Somit kann $I(v)$ im erweiterten Layout $L(Y)$ maximal $d - |E(y, X)| - 1 + 1 = d - |E(y, X)|$ andere Intervalle schneiden.

Fall 2 ($\boldsymbol{y = v}$): Da $\mathcal{A}(X) \leq d - |E(v, Y)|$ ist kann $y = v$ im Layout $L(Y)$ maximal $d - |E(y, Y)|$ andere Intervalle schneiden. ■

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

Lemma 6.24 *Jeder d-zulässige Kern Y erweitert mindestens einen d-zulässigen Kern $X \subsetneq Y$.*

Beweis: Sei Y ein zulässiger Kern und sei $L(Y)$ ein zugehöriges Layout. Sei $y \in Y$ so gewählt, dass $\ell(I(y))$ maximal ist. Weiter sei $L(X)$ das Layout für $X = Y \setminus \{y\}$, das durch Entfernen von $I(y)$ aus $L(Y)$ entsteht. Aus dem Beweis von Lemma 6.18 folgt, dass die Bedingungen 1 mit 3 für das Layout $L(X)$ erfüllt sind. Wir müssen jetzt nur noch zeigen, dass auch die Bedingungen 4 und 5 gelten.

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

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

Fall 1 ($v \notin \mathcal{A}(Y)$): Damit gilt, dass $(v, y) \in M$ sein muss. In $L(Y)$ schneidet $I(v)$ dann maximal d andere Intervalle. In $L(X)$ schneidet $I(v)$ dann maximal $d - 1 = d - |E(v, x)|$ andere Intervalle, da sich $I(v)$ und $I(y)$ in $L(Y)$ schneiden und da $E(v, Y) = \{y\}$.

Fall 2 ($v \in \mathcal{A}(Y)$): Nach Voraussetzung schneidet $I(v)$ maximal $d - |E(v, Y)|$ andere Intervalle im Layout $L(Y)$. Da sich die Intervalle von v und y nach Wahl von y schneiden müssen, schneidet $I(v)$ maximal $d - |E(v, Y)| - 1 = d - |E(v, X)|$ andere Intervalle in $L(X)$, da $E(v, Y) = E(v, Y) \cup \{y\}$. ■

Korollar 6.25 *Für die Eingabe $S = (V, M, F)$ des Bounded Degree Interval Sandwich Problems existiert genau dann eine Lösung, wenn V ein d-zulässiger Kern ist.*

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

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

Definition 6.26 *Für ein d-Layout $L(X)$ von X ist der* Grad *von $v \in \mathcal{A}(X)$ definiert als die Anzahl der Intervalls, die $I(v)$ schneiden.*

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

Das vorherige Lemma impliziert, dass zwei Layouts mit demselben Grad für jeden Knoten $v \in \mathcal{A}(X)$ entweder beide erweiterbar sind oder keines von beiden. Damit können wir unseren generische Algorithmus aus dem vorigen Abschnitt wie folgt für das Bounded Degree Interval Sandwich Problem erweitern. Wenn ein Kern Paar (X, f) betrachtet wird, wird jedes mögliche Kern-Paar (Y, g) hinzugefügt, das ein

Layout für Y mit Gradfolge g besitzt und ein Layout von X mit Gradfolge f erweitert. Aus den vorherigen Lemmata folgt bereits die Korrektheit. Wir wollen uns im nächsten Abschnitt nun noch um die Laufzeit kümmern.

6.4.3 Laufzeitabschätzung

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

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

Lemma 6.27 *Ein zulässiger Kern X ist durch das Paar $(\mathcal{A}(X), \beta(X))$ eindeutig charakterisiert.*

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

Somit können wir jetzt die die Anzahl der zulässigen Kerne abzählen, indem wir die Anzahl der oben beschriebenen charakterisierenden Paare abzählen.

Zuerst einmal stellen wir fest, dass es maximal

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

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

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

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

Lemma 6.28 *Die Anzahl der Kern-Paare, deren aktive Region maximal $d-1$ Knoten besitzt, ist beschränkt durch $O(2^{(1+\varepsilon)d^2} n^{d-1})$ für ein $\varepsilon > 0$.*

Nun müssen wir noch die Anzahl von Kanten im Berechnungsgraphen ermitteln. Statt dessen werden wir jedoch den maximalen Ausgangsgrad der Knoten ermitteln.

Wir betrachten zuerst die Kern-Paare, deren aktive Region maximale Größe, also $d-1$ Knoten, besitzen. Wie viele andere Kernpaare können ein solches Kern-Paar erweitern? Zuerst bemerken wir, dass zu einem solchen Kern nur Knoten hinzugefügt werden können, die zu Knoten der aktiven Region benachbart sind. Andernfalls würde die aktive Region auf d Knoten anwachsen, was nicht zulässig ist.

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

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

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

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

Wie viele Kern-Paare, deren aktive Region maximal $d-2$ Knoten umfasst, gibt es denn überhaupt? Wir haben dies vorhin im Lemma 6.28 für $d-1$ angegeben. Also gibt es $O(2^{(1+\varepsilon)d^2} n^{d-2})$. Da von all diesen jeweils maximal n Kanten in unserem Berechnungsgraphen ausgehen, erhalten wir also insgesamt gibt es also

$O(2^{(1+\varepsilon)d^2} \cdot n^{d-1})$ viele Kanten, die aus Kern-Paaren mit einer aktiven Region von maximal $d-2$ Knoten herausgehen.

Damit erhalten wir zusammenfassend das folgende Theorem.

Theorem 6.29 *Das Bounded Degree Interval Sandwich Problem kann in Zeit $O(2^{(1+\varepsilon)d^2} n^{d-1})$ für ein $\varepsilon > 0$ gelöst werden.*

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

Korollar 6.30 *Das ICG Problem ist in $\mathcal{P}$, wenn der maximale Grad der Lösung beschränkt ist.*

Literaturhinweise

A.1 Lehrbücher zur Vorlesung

Peter Clote, Rolf Backofen: *Introduction to Computational Biology*; John Wiley and Sons, 2000.

Richard Durbin, Sean Eddy, Anders Krogh, Graeme Mitchison: *Biological Sequence Analysis*; Cambridge University Press, 1998.

Dan Gusfield: *Algorithms on Strings, Trees, and Sequences — Computer Science and Computational Biology*; Cambridge University Press, 1997.

David W. Mount: *Bioinformatics — Sequence and Genome Analysis*, Cold Spring Harbor Laboratory Press, 2001.

Pavel A. Pevzner: *Computational Molecular Biology - An Algorithmic Approach*; MIT Press, 2000.

João Carlos Setubal, João Meidanis: *Introduction to Computational Molecular Biology*; PWS Publishing Company, 1997.

Michael S. Waterman: *Introduction to Computational Biology: Maps, Sequences, and Genomes*; Chapman and Hall, 1995.

A.2 Skripten anderer Universitäten

Bonnie Berger: *Introduction to Computational Molecular Biology*, Massachusetts Institute of Technology, http://theory.lcs.mit.edu/~bab/01-18.417-home.html;

Bonnie Berger, *Topics in Computational Molecular Biology*, Massachusetts Institute of Technology, Spring 2001, http://theory.lcs.mit.edu/~bab/01-18.418-home.html;

Paul Fischer: *Einführung in die Bioinformatik* Universität Dortmund, Lehrstuhl II, WS2001/2002, http://ls2-www.cs.uni-dortmund.de/lehre/winter200102/bioinf/

Richard Karp, Larry Ruzzo: *Algorithms in Molecular Biology*; CSE 590BI, University of Washington, Winter 1998. http://www.cs.washington.edu/education/courses/590bi/98wi/

Larry Ruzzo: *Computational Biology*, CSE 527, University of Washington, Fall 2001; http://www.cs.washington.edu/education/courses/527/01au/

Georg Schnittger: *Algorithmen der Bioinformatik*, Johann Wolfgang Goethe-Universität Frankfurt am Main, Theoretische Informatik, WS 2000/2001, http://www.thi.informatik.uni-frankfurt.de/BIO/skript2.ps.

Ron Shamir: *Algorithms in Molecular Biology* Tel Aviv University, http://www.math.tau.ac.il/~rshamir/algmb.html; http://www.math.tau.ac.il/~rshamir/algmb/01/algmb01.html.

Ron Shamir: *Analysis of Gene Expression Data, DNA Chips and Gene Networks*, Tel Aviv University, 2002; http://www.math.tau.ac.il/~rshamir/ge/02/ge02.html;

Martin Tompa: *Computational Biology*, CSE 527, University of Washington, Winter 2000. http://www.cs.washington.edu/education/courses/527/00wi/

A.3 Lehrbücher zu angrenzenden Themen

Teresa K. Attwood, David J. Parry-Smith; *Introduction to Bioinformatics*; Prentice Hall, 1999.

Maxime Crochemore, Wojciech Rytter: *Text Algorithms*; Oxford University Press: New York, Oxford, 1994.

Martin C. Golumbic: *Algorithmic Graph Theory and perfect Graphs*; Academic Press, 1980.

Benjamin Lewin: *Genes*; Oxford University Press, 2000.

Milton B. Ormerod: *Struktur und Eigenschaften chemischer Verbindungen*; Verlag Chemie, 1976.

Hooman H. Rashidi, Lukas K. Bühler: *Grundriss der Bioinformatik — Anwendungen in den Biowissenschaften und der Medizin*,

Klaus Simon: *Effiziente Algorithmen für perfekte Graphen*; Teubner, 1992.

Maxine Singer, Paul Berg: *Gene und Genome*; Spektrum Akademischer Verlag, 2000.

Lubert Stryer: *Biochemie*, Spektrum Akademischer Verlag, 4. Auflage, 1996.

A.4 Originalarbeiten

Kellogg S. Booth, George S. Lueker: Testing for the Consecutive Ones property, Interval Graphs, and Graph Planarity Using PS-Tree Algorithms; *Journal of Computer and System Science*, Vol.13, 335–379, 1976.

Ting Chen, Ming-Yang 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.

Richard Cole: Tight Bounds on the Complexity of the Boyer-Moore String Matching Algorithm; *SIAM Journal on Computing*, Vol. 23, No. 5, 1075–1091, 1994.
s.a. *Technical Report*, Department of Computer Science, Courant Institute for Mathematical Sciences, New York University, TR1990-512, June, 1990, http://csdocs.cs.nyu.edu/Dienst/UI/2.0/Describe/ncstrl.nyu_cs%2fTR1990-512

Martin Farach, Sampath Kannan, Tandy Warnow: A Robust Model for Finding Optimal Evolutionary Trees, *Algorithmica*, Vol. 13, 155–179, 1995.

Wen-Lian 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.

Wen-Lian Hsu: A Simple Test for the Consecutive Ones Property; *Journal of Algorithms*, Vol.43, No.1, 1–16, 2002.

Haim Kaplan, Ron Shamir: Bounded Degree Interval Sandwich Problems; *Algorithmica*, Vol. 24, 96–104, 1999.

Edward M. McCreight: A Space-Economical Suffix Tree Construction Algorithm; *Journal of the ACM*, Vol. 23, 262–272,1976.

Moritz Maaß: *Suffix Trees and Their Applications*, Ausarbeitung von der Ferienakademie '99, Kurs 2, Bäume: Algorithmik und Kombinatorik, 1999. http://www14.in.tum.de/konferenzen/Ferienakademie99/

Esko Ukkonen: On-Line Construction of Suffix Tress, *Algorithmica*, Vol. 14, 149–260, 1995.

Index

P

Q

R