Algorithmen der Bioinformatik II
Teile Genvorhersage
und
Sequenzierung und Assemblierung

Mario Stanke

26. Januar 2006

Inhaltsverzeichnis

Kapitel 1

Genvorhersage

1.1 Eine kurze biologische Einführung

Eine DNA-Sequenz kann als Wort über einem Alphabet mit 4 Buchstaben, a, c, g, t, aufgefasst werden. Wir bezeichnen die einzelnen Buchstaben des Wortes als *Nukleotide* oder *Basen*. In den letzten Jahren wurde die DNA-Sequenz von vielen Organismen bestimmt. Wir unterscheiden zwei Sorten von Organismen. *Eukaryoten* sind Organismen wie Tiere und Pflanzen, deren Zellen durch Membranen getrennte Abteilungen (Zellkern) enthalten. *Prokaryoten* sind Organismen wie Bakterien und Archebakterien, deren Zellen keine solche innere Abtrennung besitzen. Bisher (November 2005) wurden etwa 39 eukaryotische Genome sequenziert und ihre Sequenzen veröffentlicht (siehe http://www.genomesonline.org/). Beispiele sind die Bäckerhefe (1997), der Wurm *Caenorhabditis elegans* (1998), die Fruchtfliege *Drosophila melanogaster* (2000), die Pflanze *Arabidopsis thaliana*, der Mensch (2001), der Malaria-Parasit *Anopheles gambiae* (2002) und die Maus (2002). 547 weitere eukaryotische Sequenzierprojekte sind momentan in Bearbeitung. Für Prokaryoten sind diese Zahlen noch höher. Diese Sequenzierprojekte erzeugen eine große Menge von Rohdaten, da die Sequenz eines Eukaryoten oft länger als hundert Millionen Basenpaare (bp) lang ist und die eines Prokaryoten oft Länger als eine Million bp. Das menschliche Genom hat ungefähr eine Länge von 3 Milliarden bp.

Die meisten der menschlichen Gene kodieren für Proteine. In jüngsten Schätzungen (Artikel vom 21. Oktober 2004 in Nature) ist die Anzahl der proteinkodierenden Gene auf zwischen 20000 und 25000 herabgesetzt worden. Der vereinfachte Prozess, wie in eukaryotischen Zellen ein Protein aus einer DNA-Sequenz-Vorlage gewonnen wird, ist in Abbildung 1.1 gezeigt.

Zuerst wird eine zusammenhängende Region der DNA in eine prä-mRNA-Sequenz Base für Base eindeutig kopiert. Dieser Prozess heißt *Transkription*. Aus dieser Sequenz werden die *Introns* “herausgeschnitten” (splicing) und das Ergebnis ist die Verkettung der Exons und wird messenger-

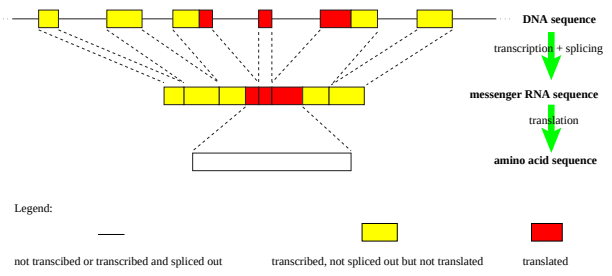


Abbildung 1.1: Ein vereinfachtes Schema der Gen-Expression.

Abbildung 1.2: Beispiel einer eukaryotischen DNA-Sequenz mit einem Gen, das aus drei kodierenden Exons (grün/hell) besteht. Die Zeilen enthalten jeweils 90 Zeichen. Die Exons haben die Längen 76, 115 und 88bp. Die Exongrenzen fallen also hier nicht mit den Kodongrenzen zusammen.

RNA (mRNA) genannt. Die Stelle zwischen einem Exon und dem nächsten Intron stromabwärts wird *Donor Splice Site* (auch 5' Splice Site) genannt, die Stelle zwischen einem Intron und dem nächsten Exon stromabwärts heisst *Acceptor Splice Site* (auch 3' Splice Site). Die Anzahl der Exons pro Gen variiert. Manche Gene enthalten nur ein Exon, also auch keine Introns. Das menschliche Muskelprotein Titin enthält jedoch mit 178 sehr viele Exons. Beim Menschen enthält ein Gen im Durchschnitt etwa 9 Exons. Danach wird ein innerer zusammenhängender Teil der mRNA sequentiell in eine Folge von Aminosäuren übersetzt. Dabei wird jedes Kodon (Tripel von Nukleotiden) nach einem (teilweise spezies-spezifischen) genetischen Code in eine bestimmte Aminosäure übersetzt. Die Translation stoppt direkt nach dem ersten von einem von drei Stopp-Kodons: taa, tag, tga. Die Länge dieses kodierenden DNA-Abschnitts ist also ein Vielfaches von 3. Die Stellen in der mRNA, an denen sich in der prä-mRNA die Introns befanden, können an beliebigen Stellen sein. Insbesondere, kann ein Intron ein Kodon trennen (siehe Abbildung 1.2). Wir bezeichnen in diesem Skript im Folgenden nur die kodierenden Teile der Exons als Exons. Dies steht zwar im Widerspruch zur biologischen Bedeutung des Wortes "Exon", aber ist praktischer, da die hier beschriebenen Methoden sich auf deren Vorhersage konzentrieren. In Abbildung 1.1 sind das also die drei roten (dunklen) Abschnitte in der obersten Zeile. Zuletzt wird die Aminosäuresequenz dreidimensional gefaltet in einer Weise, die (fast immer) durch die Abfolge der Aminisäuren selbst bestimmt ist.

Gene können auf beiden Strängen der doppelsträngigen DNA kodiert sein. Normalerweise sind zwei benachbarte Gene durch eine *intergenische Region* getrennt, überlappen also nicht.

Die Annotation dieser Sequenzen mit experimentellen Methoden kann bei weitem nicht mit dem Tempo der Erzeugung der Sequenzen mithalten. Außerdem basieren experimentelle Methoden auf der Analyse von mRNA und können somit nur Gene finden, die in den untersuchten Zelltypen und unter den gegebenen Bedingungen exprimiert werden. Informatische Methoden, Gene zu finden sind deshalb nötig. Gene zu lokalisieren ist hilfreich und oft sogar notwendig für eine weitere Analyse wie die Charakterisierung der Proteinfunktion, um die Abstammungsverhältnisse zwischen Arten zu bestimmen oder um die Regulation der Gene zu verstehen. Das Problem, Gene in genomischen DNA-Sequenzen zu finden, ist schwer und wurde trotz großer Anstrengungen noch nicht zufriedenstellend gelöst. Die Genauigkeit der jetzigen Genvorhersageprogramme ist (zumindest bei Eukaryoten) nicht hoch genug, um sich auf die Ergebnisse verlassen zu können. Nichtsdestotrotz werden die Ergebnisse solcher Programe für die automatische Annotation in Genomprojekten benutzt, und ausreichend schnelle und möglichst genaue Genfinder werden nachgefragt.

1.2 Verallgemeinerte Hidden-Markow-Modelle (GHMMs)

1.2.1 Einleitung

Ein Hidden-Markow-Modell (HMM) ist ein ein probabilistisches Modell. Unter den Modellannahmen werden beobachtete und unbeobachtete Größen als zufällige Ergebnisse aufgefasst. HMMs sind seit Ende der 60er Jahren bekannt und wurden seither in vielseitigen Anwendungen verwendet. Unter anderem für

- Bioinformatik
  - Genvorhersage
  - Modellierung von Sequenzen von Proteinfamilien
  - Alignments
  - Rekombination von Virengenomen
  - Identifikation von Fremdgenen
- Spracherkennung
- Handschrifterkennung
- Klassifizierung von Musik, Vogelgesang oder Bildern
- Klimaforschung

Eine Zusammenstellung von Referenzen von Arbeiten über HMMs gibt es unter http://www.tsi.enst.fr/ cappe/docs/hmmbib.html.

1.2.2 Notation

Für ein Alphabet Σ (z.B. $\Sigma = \{a, c, g, t\}$) bezeiche Σ^+ die Menge aller endlichen Wörter, die aus Buchstaben aus (Elementen von) Σ gebildet werden können. Z.B. $\{a, c, g, t\}^+ = \{a, c, g, t, aa, ac, ag, at, ca, cc, cg, ct, ga, gc, gg, gt, ta, tc, tg, tt, aaa, aac, \ldots\}$. Es bezeiche ε das leere Wort, also ein Wort, das aus 0 Buchstaben besteht. Und $\Sigma^* := \Sigma^+ \cup \{\varepsilon\}$. Für ein *Wort* $\sigma = \sigma_1\sigma_2\cdots\sigma_n \in \Sigma^*$ bezeichne $|\sigma| = n$ die Länge des Wortes, z.B. $|acg| = 3$. Für $i < j$ bezeichne $\sigma[i..j]$ das Teilwort $\sigma_i \cdots \sigma_j$, $\sigma(i..j]$ das Teilwort $\sigma_{i+1} \cdots \sigma_j$, etc. Für zwei Ereignisse A, B sei $P(A \mid B)$ die bedingte Wahrscheinlichkeit von A gegeben B.

1.2.3 Definition

Definition 1.1 (Markow Kette) Eine Folge von Zufallsvariablen $X_1, X_2, \ldots$ mit Werten in einer diskreten (bei uns immer endlichen) Menge Q heißt eine *Markow-Kette* (erster Ordnung), wenn für alle $i > 1$ und alle $x_1, x_2, \ldots, x_i \in Q$

$$\mathrm{P}(X_i = x_i \mid X_1 = x_1, \ldots, X_{i-1} = x_{i-1}) = \mathrm{P}(X_i = x_i \mid X_{i-1} = x_{i-1}).$$

Die Folge heißt *homogene* Markow-Kette wenn $\mathrm{P}(X_i = s \mid X_{i-1} = r)$ nicht von i abhängt $(r, s \in Q)$, sonst heißt sie *inhomogen*. Bei einer homogenen Markow-Kette wird die Matrix $A = (a_{r,s})_{r,s\in Q}$ mit $a_{r,s} = \mathrm{P}(X_i = s \mid X_{i-1} = r)$ die *Übergangsmatrix* genannt. Die Menge Q heißt *Zustandsraum*. Wenn $X_i = q$, dann sagen wir der Prozess ist zur *Zeit* i in *Zustand* q.

Beispiel: Ein Betrunkener geht von Kreuzung zu Kreuzung und wählt bei jeder Kreuzung eine zufällige Straße zum Weitergehen und zwar unabhängig davon, wo er bisher langgelaufen ist (z.B. wählt er jeweils jede mögliche Straße mit gleicher Wkeit). Dann ist die Folge der Kreuzungen eine Markow-Kette auf dem Zustandsraum aller Kreuzungen.

Um die Verteilung der Markow-Kette vollständig zu definieren muß noch die Verteilung von X_1, die sogenannte *Anfangsverteilung*, mit angegeben werden. Um die Notation zu verteinfachen fügen wir einen speziellen *Startzustand* q_{init} und eine weitere konstante Zufallsvariable $X_0 \equiv q_{\text{init}}$ ein. Dann ist die Verteilung der Markow-Kette $X_0, X_1, \ldots$ vollständig durch die Übergangsmatrix bestimmt. Weil wir uns in der bioinformatischen Praxis für endliche Zustandsfolgen interessieren, fügen wir außerdem einen speziellen *Endzustand* q_{term} ein, der von mindestens einem Zustand in Q erreicht werden kann aber vom Prozess nicht mehr verlassen wird. Wir benutzen die Bezeichnung

$$Q^+ := Q \cup \{q_{\text{init}}, q_{\text{term}}\}. \tag{1.1}$$

Die erweiterte Übergangsmatrix $A = (a_{i,j})_{i,j \in Q^+}$ muß dann

$$\begin{aligned} a_{q,q_{\text{init}}} &= 0 \quad (q \in Q^+) \\ a_{q_{\text{init}},q_{\text{term}}} &= 0 \\ a_{q_{\text{term}},q_{\text{term}}} &= 1 \end{aligned} \tag{1.2}$$

erfüllen. Mit anderen Worten, der Prozess startet im Startzustand, verläßt ihn im ersten Schritt, bleibt für irgendeine Anzahl Schritte in der Menge Q und geht dann in den Endzustand, in dem er dann bleibt. Sei T der letzte Zeitpunkt bevor der Prozess den Endzustand erreicht, also

$$T := \begin{cases} \inf\{t \mid X_t = q_{\text{term}}\} - 1. & \text{, wenn die Menge nicht leer ist} \\ \infty & \text{, sonst.} \end{cases} \tag{1.3}$$

Unter geringen - bei unseren Anwendungen erfüllten - Voraussetzungen an die Übergangsmatrix A ist T fast sicher endlich ($P(T < \infty) = 1$). Der Prozess endet also immer. Dies ist z.B. dann erfüllt, wenn es nur endlich viele Zustände gibt, und man direkt oder indirekt jeden Zustand in Q von jedem anderen mit positiver Wahrscheinlichkeit erreichen kann (sogenannte *irreduzible* Markow-Kette).

Spielzeugbeispiel eines Genmodells:
Bei den sogenannten Niwoniern ist das Erbgut in einer langen Folge von 1,2,3,4,5,6 kodiert. Es besteht zum großen Teil aus unbedeutenden zwischengenischen Regionen (IR: intergenische Region). Und aus zwei Sorten von Genen: Sorte A und Sorte B. Wir nehmen an, dass Folgendes bekannt ist.

- Die zwischengenische Region am Anfang und Ende der Erbgutsequenz und zwischen zwei benachbarten Genen kann beliebig lang sein und hat die durchschnittliche Länge 3. In ihr kommen die 6 Ziffern $1, \ldots, 6$ durchschnittlich gleich häufig vor.
- Die erste Ziffer unmittelbar vor einem Gen ist eine 1 und die erste danach ist eine 2.
- Bei Genen der Sorte A (B) sind durchschnittlich 50% der Ziffern 5 (6), die jeweils anderen Ziffern kommen im Durchschnitt gleich häufig vor.
- Es gibt etwa doppelt soviele Gene der Sorte B wie Gene der Sorte A.
- Gene der Sorte A bestehen aus 1-10 Ziffern, Gene der Sorte B aus 1-5 Ziffern.

Diese Information ist im folgenden Graphen zusammengefasst.

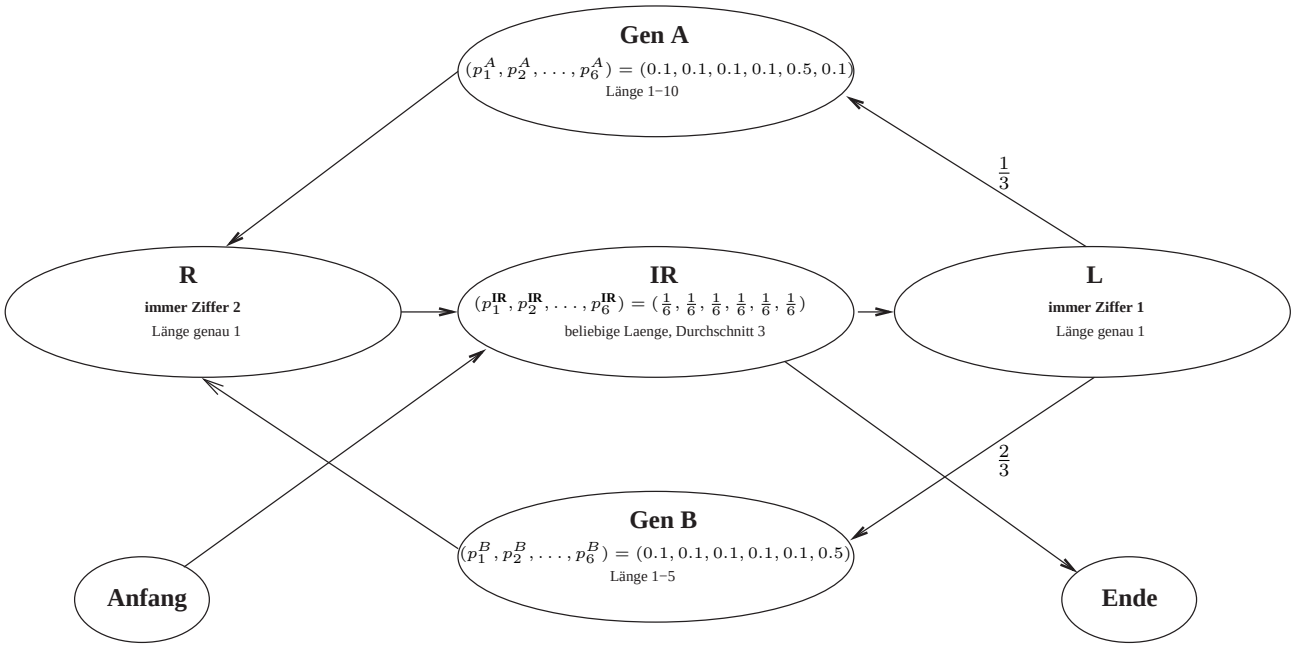


Abbildung 1.3: Spielzeugbeispiel: Erbgutstruktur der Niwonier.

Wir betrachten die Erbgutsequenz als zufällig, weil wir nicht verstehen, warum sie genau so ist wie sie ist. Viele Eigenschaften der beobachteten Regelmäßigkeiten des Zufalls (z.B. ungefähre Länge, durchschnittliche Häufigkeiten der Ziffern) können in einem verallgemeinerten Hidden-Markow-Modell modelliert werden.

Definition 1.2 (GHMM) Sei Q^+ wie in (1.1), A wie in (1.2) und sei Σ eine abzählbare Menge, das *Emissionsalphabet*. Weiter seien die Wahrscheinlichkeiten $e_i(\sigma)$ definiert für $i \in Q^+, \sigma \in \Sigma^*$. Ein *verallgemeinertes Hidden Markow Model (GHMM)* mit Zustandsraum Q^+, Übergangsmatrix A und *Emissionswahrscheinlichkeiten* $e_i(\sigma) \quad (i \in Q^+, \sigma \in \Sigma^*)$ ist eine Folge

$$(X_0, Y_0), (X_1, Y_1), (X_2, Y_2), \ldots$$

bei der $X_0 \equiv q_{\text{init}}$ ist, die Folge $X_0, X_1, X_2, \ldots$ eine homogene Markow-Kette mit Zustandsraum Q^+ und Übergangsmatrix A ist und wobei $Y_0, Y_1, \ldots$ eine Folge von Zufallsvariablen mit Werten in Σ^* ist, so dass $Y_0 \equiv \varepsilon$ und

$$\begin{aligned} e_{x_i}(y_i) &= \mathrm{P}(Y_i = y_i \,|X_i = x_i) \\ &= \mathrm{P}(Y_i = y_i \,|\, X_0 = x_0, \ldots, X_i = x_i, Y_0 = y_0, \ldots, Y_{i-1} = y_{i-1}) \end{aligned}$$

für alle $i > 0$ und $x_0, \ldots, x_i \in Q^+$, $y_0, \ldots y_i \in \Sigma^*$. Die Emissionswahrscheinlichkeiten müssen $e_i(\varepsilon) = 0 \quad (i \in Q)$ und $e_{q_{\text{init}}}(\varepsilon) = 1, e_{q_{\text{term}}}(\varepsilon) = 1$ erfüllen.

Für das Beispiel der Niwonischen Erbgutstruktur könnte man etwa folgendes GHMM benutzen. Wir setzen $\Sigma = \{1, \ldots, 6\}$. Wir wehmen jetzt an, wir hätten die Verteilung der Längen der IR, von A-Genen und B-Genen genau gegeben: Seien $p_{\text{IR}}(\ell), p_{\text{A}}(\ell), p_{\text{B}}(\ell)$ die Wahrscheinlichkeiten dafür dass eine IR-, A-Gen-, bzw. B-Gen-Sequenz die Länge $\ell > 1$ hat. Wir nehmen hier der

Einfachheit halber an, das die Länge der zwischengenischen Region geometrisch verteilt ist und dass die Länge eines Gens gleichverteilt ist auf der Menge aller möglichen Längen:

$$\begin{aligned} p_{\text{IR}}(\ell) &= \frac{2}{3}\left(\frac{1}{3}\right)^{\ell-1} \\ p_{\text{A}}(\ell) &= \begin{cases} 1/10 & \text{falls } 1 \leq \ell \leq 10 \\ 0 & \text{, sonst} \end{cases} \\ p_{\text{B}}(\ell) &= \begin{cases} 1/5 & \text{falls } 1 \leq \ell \leq 5 \\ 0 & \text{, sonst} \end{cases} \end{aligned}$$

Und wir nehmen jetzt an, dass eine Sequenz aus durchschnittlich 100 Genen besteht. Dann wäre etwa folgendes GHMM eine sinnvolles statistisches Näherungsmodell für die niwonischen Genstrukturen.

Zustandsraum: $Q^+ = \{q_{\text{init}}, \text{IR}, \text{L}, \text{R}, \text{A}, \text{B}, q_{\text{term}}\}$

Übergangsmatrix:

A	q_{init}	IR	L	R	A	B	q_{term}
q_{init}	0	1	0	0	0	0	0
IR	0	0	0.99	0	0	0	0.01
L	0	0	0	0	$\frac{1}{3}$	$\frac{2}{3}$	0
R	0	1	0	0	0	0	0
A	0	0	0	1	0	0	0
B	0	0	0	1	0	0	0
q_{term}	0	0	0	0	0	0	1

Emissionswahrscheinlichkeiten: Sei $\sigma = \sigma_1\sigma_2\cdots\sigma_\ell$ ein Wort über Σ der Länge $\ell \geq 1$. Wir setzen dann

$$e_{\text{IR}}(\sigma) = p_{\text{IR}}(\ell) \cdot \prod_{i=1}^{\ell} p_{\sigma_i}^{\text{IR}}, \qquad e_{\text{A}}(\sigma) = p_{\text{A}}(\ell) \cdot \prod_{i=1}^{\ell} p_{\sigma_i}^{\text{A}}, \qquad e_{\text{B}}(\sigma) = p_{\text{B}}(\ell) \cdot \prod_{i=1}^{\ell} p_{\sigma_i}^{\text{B}}$$

$$e_{\text{L}}(\sigma) = \begin{cases} 1 & \text{, wenn } \sigma = \text{'1'} \\ 0 & \text{, sonst.} \end{cases}, \qquad e_{\text{R}}(\sigma) = \begin{cases} 1 & \text{, wenn } \sigma = \text{'2'} \\ 0 & \text{, sonst.} \end{cases}$$

Damit haben wir dieses GHMM vollständig spezifiziert. Beachte etwa, dass die Wahrscheinlichkeit, dass ein im Zustand IR emittiertes Wort W die Länge ℓ hat, tatsächlich $p_{IR}(\ell)$ ist:

$$\text{P(W hat Länge } \ell) = \sum_{\sigma \text{ Wort der Länge } \ell} e_{\text{IR}}(\sigma) = p_{\text{IR}}(\ell) \sum_{\sigma_1,\ldots,\sigma_\ell \in \Sigma} \prod_{i=1}^{\ell} p_{\sigma_i}^{\text{IR}} = p_{\text{IR}}(\ell)$$

Und die Ziffer $i \in \{1, \ldots, 6\}$ kommt tatsächlich an jeder Stelle in W mit Wahrscheinlichkeit p_i^{IR} vor. Wir konnten also in diesem Fall alle uns zur Verfügung stehenden Erfahrungswerte über typische niwonische Gene im Modell berücksichtigen. Jetzt stellt sich die Frage, wie benutzt man ein GHMM, das man bereits konstruiert hat, um die unbekannte Struktur einer Sequenz vorherzusagen.

Direkt aus der Definition von GHMMs folgt durch wiederholte Anwendung der Definition der bedingten Wahrscheinlichkeit, dass für alle $t > 0, x_1, \ldots x_t \in Q^+, y_1, \ldots, y_t \in \Sigma^*, x_0 := q_{\text{init}}$.

$$\begin{aligned} & P(((X_1, Y_1), \ldots, (X_t, Y_t)) = ((x_1, y_1), \ldots, (x_t, y_t))) \\ = \; & P((X_1, Y_1) = (x_1, y_1)) \cdot \\ & P((X_2, Y_2) = (x_2, y_2) \,|\, (X_1, Y_1) = (x_1, y_1)) \cdot \\ & \ldots \\ & P((X_t, Y_t) = (x_t, y_t) \,|\, (X_1, Y_1) = (x_1, y_1), \cdots, (X_{t-1}, Y_{t-1}) = (x_{t-1}, y_{t-1})) \\ = \; & \prod_{i=1}^{t} a_{x_{i-1}, x_i} \cdot e_{x_i}(y_i). \end{aligned} \tag{1.4}$$

Wir bezeichnen mit $\mathbf{X}$ die Folge der Zustände $X_0, X_1, \ldots$ und mit $\mathbf{Y}$ die Folge der *Beobachtungen* $Y_0, Y_1, \ldots$. Mit Formel 1.4 können wir gemeinsame Wahrscheinlichkeiten von Zustands- und Beobachtungsfolgen ausrechnen und damit (theoretisch) die Wahrscheinlichkeit jedes von $\mathbf{X}$ oder $\mathbf{Y}$ abhängigen Ereignisses. Zunächst noch eine Definition.

Definition 1.3 Sei $x_1, \ldots, x_n \in Q$ und $d_1, \ldots, d_n \geq 1$. Der Vektor

$$((x_1, d_1), \ldots, (x_n, d_n)) \tag{1.5}$$

wird ein *Parse* der Länge ℓ genannt, wenn $d_1 + \cdots + d_n = \ell$. Er *endet* in x_n. Der von $(\mathbf{X}, \mathbf{Y})$ induzierte Parse Φ wird definiert als

$$\Phi := ((X_1, |Y_1|), \ldots (X_T, |Y_T|)) \tag{1.6}$$

Für $\ell \geq 1$ wird der *ℓ-gestutzte* durch $(\mathbf{X}, \mathbf{Y})$ induzierte Parse definiert durch

$$\Phi_\ell := ((X_1, |Y_1|), \ldots (X_r, |Y_r|)) \text{ mit } r := \max\{n \,|\, |Y_1| + \cdots + |Y_n| \leq \ell, Y_n \neq \varepsilon\} \tag{1.7}$$

Der ℓ-gestutzte Parse kann interpretiert werden als der längste anfängliche Teilparse, dessen Emissionslänge ℓ nicht überschreitet. Beachte, dass Φ_ℓ ein Parse der Länge ℓ ist, wenn $|Y_1| + \cdots + |Y_n| = \ell$ ist für irgendein n.

Sei S das Wort, das man durch Aneinanderhängen der Wörter $Y_0, Y_1, \ldots$ erhält. Die Y_i's nennen wir *Emissionen*. Die praktische Absicht hinter dem formalen Modell ist die Folgende. In den Anwendungen von GHMMs ist S beobachtbar. Aber in welchem Zustand die Zeichen in S emittiert wurden ist unbekannt und soll sinnvoll geraten werden. Mit anderen Worten, der Parse Φ ist "versteckt" und muß unter Benutzung der Beobachtung S aufgedeckt werden. Das Wort verallgemeinert (generalized) in GHMM bezieht sich auf die Tatsache, dass – im Gegensatz zu normalen HMMs – die Zustände in einem GHMM ein ganzes Wort emittieren anstatt nur ein einzelnes Zeichen. In einem normalen HMM haben die Wörter Y_i's alle Länge 1.

1.2.4 Viterbi-Algorithmus

Der wahrscheinlichste Parse gegeben eine Beobachtung $\sigma \in \Sigma^+$ der Länge t ist eine intuitive Wahl als Mutmaßung für den unbekannten wahren Parse. So ein Parse ψ_{vit} wird *Viterbi-Parse* genannt:

$$\psi_{\text{vit}} \in \underset{\psi \text{ Parse der Länge } t}{\operatorname{argmax}} \mathrm{P}(\Phi = \psi \,|\, S = \sigma). \tag{1.8}$$

Wir nennen die bedingte Verteilung des Prozesses $(\mathbf{X}, \mathbf{Y})$, gegeben, dass $S = \sigma$, die *a-posteriori*-Verteilung von Zuständen und Emissionen. In diesem Sinne ist der Viterbi-Parse ein Parse mit

maximaler a-posteriori-Wahrscheinlichkeit. Anders formuliert: Wenn wir unter allen möglichen Ausgängen $(\mathbf{x}, \mathbf{y})$ des zufälligen Prozesses, die mit der tatsächlichen Beobachtung σ konsistent sind (also wo die aneinandergehängten emittierten Wörter gleich σ sind), den (einen) wahrscheinlichsten Ausgang nehmen, dann ist der induzierte Parse der (ein) Viterbi-Parse.

Formel 1.4 kann zwar theoretisch benutzt werden um direkt einen Viterbi-Parse zu finden, z.B. indem von jedem möglichen Paar von Zustand- und Emissionsfolge $(\mathbf{x}, \mathbf{y})$ die zur tatsächlich beobachteten Gesamtemission σ führt die Wahrscheinlichkeit berechnet wird. Eine solche Methode ist aber wegen der zu großen Zahl möglicher Parses praktisch unmöglich.

Ein Viterbi-Parse kann mittels dynamischer Programmierung effizient berechnet werden. Wir beschreiben jetzt eine Variante des sogenannten *Viterbi-Algorithmus* [Vit67].

Sei eine Eingabesequenz σ der Länge t gegeben. Wir definieren die sogenannten *Viterbi-Variablen*

$$\gamma_{q,\ell} := \max_{\psi \text{ Parse der Länge } \ell \text{ endend in } q} \mathrm{P}(\Phi_\ell = \psi, S[1..\ell] = \sigma[1..\ell]) \tag{1.9}$$

für alle $q \in Q$ and $1 \leq \ell \leq t$. Aus Notationsgründen setzen wir außerdem $\gamma_{q_{\text{init}},0} = 1$ und $\gamma_{q,0} = 0$ für alle $q \neq q_{\text{init}}$. Die Viterbi-Variablen können mit einer einfachen Rekursion berechnet werden. Diese Rekursion leiten wir her indem wir erst darauf bedingen, ob ψ aus mehr als einem Schritt besteht, und dann in dem Fall, dass ψ aus mehr als einem Schritt besteht, auf die möglichen Ausgänge des letzten Schritts bedingen.

$$\begin{aligned}
\gamma_{q,\ell} &= \max\Bigg\{ \max_{\psi=((q,\ell))} \mathrm{P}(\Phi_\ell = \psi, S[1..\ell] = \sigma[1..\ell]), \\
&\qquad \max_{\substack{\psi=(\psi',(q,d)) \\ \text{Parse der Länge} \\ \ell \text{ endend in } q}} \mathrm{P}(\Phi_\ell = \psi, S[1..\ell] = \sigma[1..\ell]) \Bigg\} \\
&= \max\Bigg\{ a_{q_{\text{init}},q} \cdot e_q(\sigma[1..\ell])), \\
&\qquad \max_{\substack{q' \in Q,\, \ell' = \ell - d \\ \psi' \text{ Parse der Länge} \\ \ell' \text{ endend in } q'}} \mathrm{P}(\Phi_{\ell'} = \psi', S[1,\ell'] = \sigma[1..\ell']) \cdot a_{q',q} \cdot e_q(\sigma(\ell'..\ell]) \Bigg\} \\
&= \max\Bigg\{ a_{q_{\text{init}},q} \cdot e_q(\sigma[1..\ell])), \\
&\qquad \max_{1 \leq \ell' < \ell, q' \in Q} a_{q',q} \cdot e_q(\sigma(\ell'..\ell]) \cdot \max_{\substack{\psi' \text{ Parse der Länge} \\ \ell' \text{ endend in } q'}} \mathrm{P}(\Phi_{\ell'} = \psi', S[1..\ell'] = \sigma[1..\ell']) \Bigg\} \\
&= \max\Big\{ a_{q_{\text{init}},q} \cdot e_q(\sigma[1..\ell])), \max_{1 \leq \ell' < \ell, q' \in Q} \gamma_{q',\ell'} \cdot a_{q',q} \cdot e_q(\sigma(\ell'..\ell]) \Big\} \\
&= \max_{\substack{1 \leq \ell' < \ell, q' \in Q \\ \text{oder } q' = q_{\text{init}}, \ell' = 0}} \gamma_{q',\ell'} \cdot a_{q',q} \cdot e_q(\sigma(\ell'..\ell])
\end{aligned} \tag{1.10}$$

Der folgende Satz zeigt, was diese Viterbi-Variablen mit einem Viterbi-Parse zu tun haben.

Satz 1.4 Sei σ eine Emission der Länge t. Sei $\psi = ((x_1, d_1), \ldots, (x_n, d_n))$ mit $x_1, \ldots, x_n \in Q$ ein Parse der Länge t. Definiere $D_0 := 0, D_i := d_1 + \cdots + d_i \quad (i = 1, \ldots, n)$. Wenn ψ

$$\gamma_{x_n,t} \cdot a_{x_n,q_{\text{term}}} = \max_{q \in Q} \gamma_{q,t} \cdot a_{q,q_{\text{term}}} \tag{1.11}$$

erfüllt und

$$\gamma_{x_i,D_i} = \gamma_{x_{i-1},D_{i-1}} \cdot a_{x_{i-1},x_i} \cdot e_{x_i}(\sigma(D_{i-1}, D_i]) \tag{1.12}$$

ist für alle $i = 1, \ldots, n$ (mit $x_0 := q_{\text{init}}$), dann ist ψ ein Viterbi-Parse.

Beweis: Da $\mathrm{P}(\Phi = \psi, S = \sigma) = \mathrm{P}(S = \sigma) \cdot P(\Phi = \psi \,|\, S = \sigma)$, ist jeder Parse der Länge t, der $P(\Phi = \psi, S = \sigma)$ maximiert, ein Viterbi-Parse. Sei p_{vit} diese maximale Wahrscheinlichkeit:

$$p_{\text{vit}} := \max_{\psi \text{ Parse der Länge } t} \mathrm{P}(\Phi = \psi\,,\, S = \sigma).$$

Und sei ψ ein Parse der Länge t der (1.11) und (1.12) erfüllt. Dann gilt

$$\begin{aligned}
P(\Phi = \psi, S = \sigma) &= \left(\prod_{i=1}^{n} a_{x_{i-1},x_i} \cdot e_{x_i}(\sigma(D_{i-1}, D_i]) \right) \cdot a_{x_n,q_{\text{term}}} \\
&= \gamma_{x_1,D_1} \cdot \left(\prod_{i=2}^{n} a_{x_{i-1},x_i} \cdot e_{x_i}(\sigma(D_{i-1}, D_i]) \right) \cdot a_{x_n,q_{\text{term}}} \\
&= \gamma_{x_2,D_2} \cdot \left(\prod_{i=3}^{n} a_{x_{i-1},x_i} \cdot e_{x_i}(\sigma(D_{i-1}, D_i]) \right) \cdot a_{x_n,q_{\text{term}}} \\
&= \vdots \\
&= \gamma_{x_n,t} \cdot a_{x_n,q_{\text{term}}} \\
&= \max_{q \in Q} \gamma_{q,t} \cdot a_{q,q_{\text{term}}} \\
&= \max_{q \in Q} \max_{\substack{\psi \text{ Parse der Länge } t \\ \text{endend in } q}} \mathrm{P}(\Phi_t = \psi\,,\, S = \sigma) \cdot a_{q,q_{\text{term}}} \\
&= \max_{\psi \text{ Parse derLänge } t} \mathrm{P}(\Phi = \psi\,,\, S = \sigma) \\
&= p_{\text{vit}}
\end{aligned}$$

Dabei folgt die erste Zeile aus (1.4), und die zweite bis fünfte Zeile unter Benutzung von (1.12). In der Zeile mit den zwei Maxima wurde die Definition von $\gamma_{q,t}$ eingesetzt. Im Schritt zur vorletzten Zeile wurde berücksichtigt, dass der vollständige Parse gleich dem t-gestutzten Parse ist, wenn er in nächsten Schritt in den Endzustand übergeht. Die letzte Zeile folgt aus der vorhergehenden nach dem Satz von der totalen Wahrscheinlichkeit.

Satz 1.4 legt den sogenannten Viterbi-Algorithmus nahe:

Algorithmus 1 (Viterbi-Algorithmus)
1. Berechne iterativ die Viterbi-Variablen $\gamma_{q,l}$ nach aufsteigenden ℓ's mittels Rekursion (1.10) und speichere sie in einer Tabelle (dynamische Programmierung).
2. Mache ein Backtracing durch die Tabelle und benutze Satz 1.4 um einen Viterbi-Parse zu konstruieren. D.h. benutze Formel (1.11) um x_n zu bestimmen (n ist unbekannt) und danach für $i = n, n-1, \ldots, 1$ die Formeln (1.12) und (1.10) um x_{i-1} und d_i zu bestimmen. Das Backtracing terminiert, wenn $\sum d_i = t$ erreicht ist.

	$\ell = 1$	2	3	4	5	6	7	8
	3	1	4	5	1	5	2	6
$q =$IR	0.111	$6.17 \cdot 10^{-3}$	$3.43 \cdot 10^{-4}$	$1.91 \cdot 10^{-5}$	$1.06 \cdot 10^{-6}$	$5.88 \cdot 10^{-8}$	$3.27 \cdot 10^{-9}$	$1.02 \cdot 10^{-6}$
L	0	0.11	0	0	$1.89 \cdot 10^{-5}$	0	0	0
A	0	0	$3.67 \cdot 10^{-4}$	$1.83 \cdot 10^{-4}$	$1.83 \cdot 10^{-5}$	$9.17 \cdot 10^{-6}$	$9.17 \cdot 10^{-7}$	$9.17 \cdot 10^{-8}$
B	0	0	$1.47 \cdot 10^{-3}$	$1.47 \cdot 10^{-4}$	$1.47 \cdot 10^{-5}$	$1.47 \cdot 10^{-6}$	$1.47 \cdot 10^{-7}$	$1.26 \cdot 10^{-8}$
R	0	0	0	0	0	0	$9.17 \cdot 10^{-6}$	0

	$\ell = 9$	10	11	12	13	14	15	16
	6	1	2	3	6	5	2	4
$q =$IR	$5.66 \cdot 10^{-8}$	$3.14 \cdot 10^{-9}$	$1.75 \cdot 10^{-10}$	$1.02 \cdot 10^{-10}$	$5.66 \cdot 10^{-12}$	$3.14 \cdot 10^{-13}$	$1.74 \cdot 10^{-14}$	$4.14 \cdot 10^{-13}$
L	0	$5.60 \cdot 10^{-8}$	0	0	0	0	0	0
A	$9.17 \cdot 10^{-9}$	$9.17 \cdot 10^{-10}$	$9.17 \cdot 10^{-11}$	$1.86 \cdot 10^{-11}$	$1.86 \cdot 10^{-12}$	$9.33 \cdot 10^{-13}$	$9.33 \cdot 10^{-14}$	$9.33 \cdot 10^{-15}$
B	$6.3 \cdot 10^{-9}$	$6.3 \cdot 10^{-10}$	$7.46 \cdot 10^{-10}$	$7.46 \cdot 10^{-11}$	$3.73 \cdot 10^{-11}$	$3.73 \cdot 10^{-12}$	$3.73 \cdot 10^{-13}$	0
R	0	0	$9.17 \cdot 10^{-10}$	0	0	0	$3.73 \cdot 10^{-12}$	0

Abbildung 1.4: Beispiel für den Verlauf des Viterbi-Algorithmus bei dem Spielzeug-GHMM für die Eingabesequenz $\sigma = 3145152661236524$ der Länge $t = 16$. Obige *V*iterbi-Tabelle mit den Einträgen $\gamma_{q,\ell}$ wird zunächst von links nach rechts spaltenweise gemäß der Viterbi-Rekursion (1.10) ausgefüllt. Dann wird der Zustand q bestimmt, für den $\gamma_{q,t} \cdot a_{q,q_{\text{term}}}$ maximal ist. Hier ist das $q = IR$, weil dies der einzige Zustand ist, von dem aus man den Endzustand erreichen kann. Man setzt $\ell = t$, ermittelt dann das Paar (q', ℓ') das den Ausdruck in (1.10) maximiert, merkt sich dieses Paar und setzt dann $(q, \ell) = (q', \ell')$. Dies wiederholt man bis man bei $q = q_{\text{init}}, \ell = 0$ angekommen ist. Die hierbei durchlaufenen Paare (q, ℓ) sind rot markiert. Sie ergeben von links nach rechts gelesen die Abfolge der Zustände und ihre Endpositionen. Der Viterbi-Parse ist $((\text{IR}, 1), (\text{L}, 1), (\text{A}, 4), (\text{R}, 1), (\text{IR}, 2), (\text{L}, 1), (\text{B}, 4), (\text{R}, 1), (\text{IR}, 1))$. *Bemerkung:* Man kann das nochmalige Berechnen der Viterbi-Rekursion beim Backtracing einsparen, wenn man sich gleich beim Ausfüllen der Tabelle zu jedem Paar (q, ℓ) jeweils auch merkt, bei welchem ℓ' und q' das Maximum angenommen wurde. Bei mancher Anwendung möchte man allerdings den dafür zusätzlich benötigten Speicher sparen.

Die Speicherplatz-Komplexität dieser direkten Implementation ist $O(|Q| \cdot t)$. Die Zeit-Komplexität ist $O(|Q|^2 \cdot t^2)$, wenn die Emissionswahrscheinlichkeiten in konstanter Zeit berechnet werden können. In einzelnen Anwendungen hängt die Zeit-Komplexität stark von der Zeit ab, die benötigt wird um die Emissionswahrscheinlichkeiten zu berechnen. In typischen Anwendungen muß zur Berechnung von $\gamma_{q,\ell}$ mittels 1.10 auch nicht das Maximum über alle $q' \in Q$ und $\ell' < \ell$ gebildet werden, sondern nur über solche, bei denen die Übergangswahrscheinlichkeiten und Emissionswahrscheinlichkeiten nicht verschwinden.

Der Grund, warum die sehr komplex erscheinende Aufgabe, unter allen (exponentiell vielen!) Parses den Wahrscheinlichsten zu finden, praktisch überhaupt effizient möglich ist, ist die relativ einfache Abhängigkeitsstruktur der Zustands- und Emissionsfolgen. Die Verteilung von X_i hängt im GHMM nur von X_{i-1} ab, die von Y_i nur von X_i. Dies rechtfertigt nachträglich die Verwendung einer Markow-Kette für die Folge $X_1, X_2, \ldots$, die zunächst willkürlich erschienen sein mag.

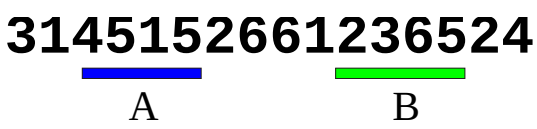


Abbildung 1.5: Eine niwonische Erbgutsequenz σ und die dem Viterbi-Parse entsprechende Genstruktur.

1.2.5 GHMMs für die Genvorhersage

Prokaryoten

Das erste Genvorhersageprogramm, dass auf einem Hidden-Markow-Modell beruhte, war ECOPARSE [KMH94]. Es war ein HMM, das Gene in dem Prokaryoten *Escherichia coli* vorhersagte. Es beruhte auf einem einfachen Modell für die intergenische Region und berücksichtigte, die typische Häufigkeit der 61 kodierenden Kodons in *E.coli*. Abbildung 1.6 zeigt ein **G**HMM, dass dem HMM in ECOPARSE sehr nahe kommt. Es gibt 4 Zustände $Q = \{\text{IR}, \text{Startkodon}, \text{Kodon}, \text{Stoppkodon}\}$. Die Übergänge mit einer Übergangswahrscheinlichkeit größer als Null sind als Pfeile eingezeichnet und mit der Übergangswahrscheinlichkeit beschriftet. Jeder Zustand emittiert eine feste Anzahl an Zeichen. 'IR' emittiert jeweils ein Nukleotid. Die Wahrscheinlichkeit für A,C,G,T ist jeweils die beobachtete relative Häufigkeit dieser Basen in intergenischen Regionen. 'Startkodon' emittiert drei Nukleotide: Mit großer Wahrscheinlichkeit das Wort atg mit kleinerer Wahrscheinlichkeit die beiden anderen möglichen Startkodons gtg oder ttg, alle anderen Wörter der Länge 3 haben Wahrscheinlichkeit 0. Der Zustand 'Kodon' emittiert auch jeweils 3 Nukleotide, nämlich jedes der 61 Kodons, die kein Stoppkodon sind, mit der Wahrscheinlichkeit gemäß einer Tabelle der Kodonhäufigkeiten in *E.coli*. Der Zustand 'Stoppkodon' emittiert eines von drei Stoppkodons ebenfalls mit vorher trainierten Häufigkeiten. Die Übergangswahrscheinlicheiten p und q sind klein. Sie bestimmen implizit die Wahrscheinlichkeit für die Länge eines Gens oder einer intergenischen Region (Aufgabe).

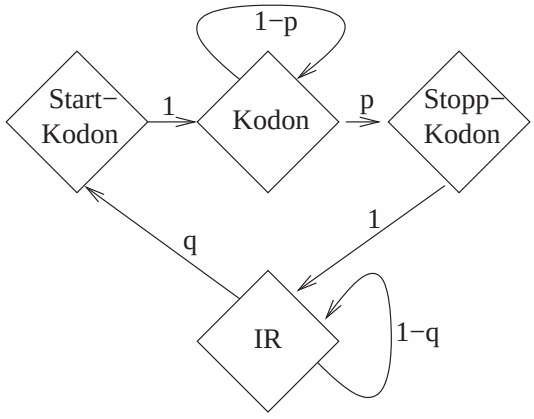


Abbildung 1.6: Ein einfaches GHMM, das Gene in Prokaryoten findet. Zustand 'IR' emittiert jeweils 1 Nukleotid, die anderen Zustände emittieren jeweils ein ganzes Kodon.

Obiges Modell hat den Nachteil, dass es nur Gene auf dem Vorwärtsstrang findet. Ausserdem neigt es dazu, Gene auf dem Vorwärtsstrang in Regionen zu finden, die auf dem Rückwärtsstrang ein Gen enthalten und umgekehrt. Das liegt daran, dass kodierende Regionen normalerweise einen höheren Anteil and g und c haben (GC-Gehalt) und dies gilt dann auch für den gegenüberliegenden

Strang wegen der Komplementarität von g und c. Diese fälschlicherweise auf dem falschen Strang vorhergesagten Gene heißen 'Schattengene'. Das Modell kann so abgeändert, dass es gleichzeitig Gene auf beiden Strängen vorhersagt und das Vorhersagen von Schattengenen weitgehend vermeidet. Siehe Abbildung 1.7. Die drei unteren Zustände, deren Namen mit RC (reverse complement) beginnen, entsprechen jeweils einem Zustand oben. Ein Zustand unten emittiert mit derselben Wahrscheinlichkeit das *reverse Komplement* eines Wortes, wie der entsprechende Zustand oben das Wort emittiert.

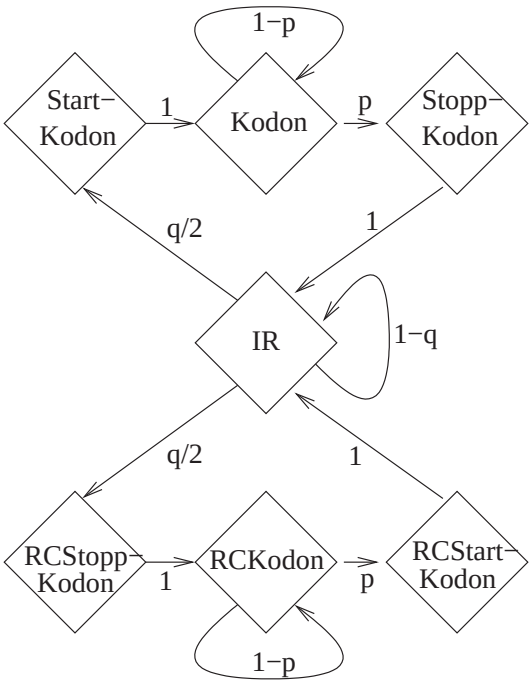


Abbildung 1.7: Ein GHMM für Prokaryoten, das Gene auf beiden Strängen vorhersagt. Die obere Hälfte modelliert Gene auf dem Vorwärtsstrang, die untere Gene auf dem Rückwärtsstrang.

Eukaryoten

Die Genvohersage bei Prokaryoten ist aus zwei Gründen vergleichsweise einfach im Vergleich zu der bei Eukaryoten. Zum einen ist wegen der fehlenden Introns die Länge der kodierenden Abschnitte der DNA meistens so lang, dass der sie enthaltende offene Leserahmen (ORF), statistisch auffällig lang ist. Solche langen ORFs kommen normalerweise nicht in nicht-kodierenden Regionen vor. Das Fehlen eines Stoppkodons in einem bestimmten Leserahmen in einem langen Abschnitt lässt also mit grosser Sicherheit bereits auf das Ende eines Gens schliessen: Das erste Stoppkodon nach dem ORF. Zum anderen sind die intergenischen Regionen bei Prokaryoten normalerweise viel kürzer als bei Eukaryoten.

Es gibt also viel weniger "Gelegenheit" falsche Gene vorherzusagen, wo keine sind. Abbildung 1.8 zeigt die Zustände und möglichen Übergänge eines einfachen GHMMs für Eukaryoten. Die Karoförmigen Zustände emittieren jeweils nur ein Nukleotid haben aber einen möglichen Übergang zu sich selbst (Selbstschleife), so dass ein beliebig langes zusammenhängendes Stück Sequenz Base für Base emittiert werden kann. Zustand IR steht wieder für die intergenische Region. I0, I1, I2 sind Intronzustände. Hierbei ist es sinnvoll, sich bei Genen mit mehreren Exons den Leserahmen des vorhergehenden Exons zu "merken" indem man für jede der 3 möglichen

Leserahmen einen Intronzustand einführt. In diesem Sinne ist I0 ein Intron, dessen vorhergehendes Exon mit einem vollständigen Kodon endet, I1 ein Intron, dessen Vorgängerexon mit einem unvollständigen Kodon mit nur 1 Nukleotid endet und I2 ein Intron, dessen Vorgängerexon mit einem unvollständigen Kodon mit 2 Nukleotiden endet. Die runden Zustände entsprechen Exons und emittieren die gesamte Exonsequenz zufälliger Länge in einem Stück. Dies erlaubt, die Verteilung der Exonlängen genau zu modellieren. Die Ziffer am Ende des Zustandsnamen der initialen (INI) und internen (INT) Exons gibt wie oben beschrieben die Position im Leserahmen an, in dem die Exons enden. In dem hier beschriebenen Modell, das eine Vereinfachung des Modells in AUGUSTUS [SW03] ist, hängen entgegen obigem theoretischen Modell die Emissionslängen der Exons nicht nur vom gerade aktuellen Zustand, sondern auch vom Vorgängerzustand ab: Die kodierende Sequenz, die in einem INTEXONi Zustand emittiert wird, muss, wenn der Vorgängerzustand der Intronzustand Ij war, die Länge $i - j$ modulo 3 haben. Auf diese Weise liefert jeder Pfad eine biologisch konsistente Abfolge von Exons inklusive Leserahmeninformation. Die Definition des GHMMS und der Viterbi-Algorithmus können problemlos um die Möglichkeit erweitert werden, dass die Emissionen von den letzten beiden Zuständen abhängen ($e_{x_{i-1},x_i}(\sigma)$ anstatt $e_{x_i}(\sigma)$). Der Übersichtlichkeit halber habe ich dies in den vorigen Abschnitten nicht zugelassen. Man kann diese Erweiterung der Definition von GHMMs jedoch auch vermeiden, indem man zusätzliche Exonzustände einführt (Übung).

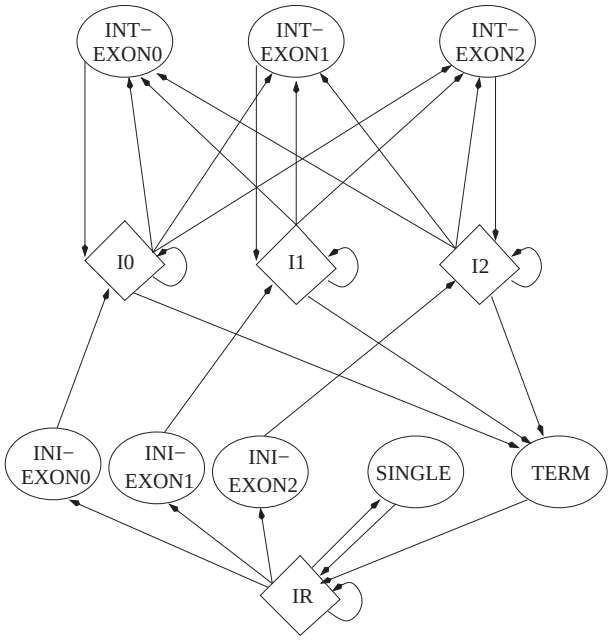


Abbildung 1.8: Ein einfaches GHMM für Eukaryoten. Ein Gen besteht entweder nur aus einem Exon (Single) oder aus einem initialen Exon (INIEXON), einem Intron (I) dann aus einer beliebigen Anzahl von abwechselnden internen Exons (INTEXON) und Introns und schliesslich aus einem terminalen Exon (TERM).

1.2.6 Vorwärts-Algorithmus

Ein Algorithmus, der dem Viterbi-Algorithmus sehr ähnlich ist, ist der sogenannte *Vorwärts-Algorithmus*. Der Viterbi-Algorithmus erlaubt zwar die Berechung eines Parses ψ_{vit} mit maximaler a-posteriori-Wahrscheinlichkeit, aber wir können mit ihm *nicht* auch die a-posteriori-Wahrscheinlichkeit dieses Parses ψ_{vit} ausrechnen. Zu diesem Zweck kann man den Vorwärts-Algorithmus verwenden:

Er erlaubt insbesondere, die Wahrscheinlichkeit $\mathrm{P}(S = \sigma)$ der Emission σ zu berechnen. Somit kann man also die a-posteriori-Wahrscheinlichkeit von ψ_{vit}

$$\mathrm{P}(\Phi = \psi_{\text{vit}} \,|\, S = \sigma) = \frac{\mathrm{P}(\Phi = \psi_{\text{vit}}\,,\, S = \sigma)}{\mathrm{P}(S = \sigma)}$$

bestimmen, denn den Zähler (oben p_{vit} genannt) ist Nebenprodukt des Viterbi-Algorithmus. Der Vorwärts-Algorithmus ist außerdem ein notwendiger Vorverarbeitungsschritt des sogenannten *Sampling-Algorithmus*, der weiter unten besprochen wird.

Analog zu den Viterbi-Variablen definieren wir die *Vorwärts-Variablen*

$$\begin{aligned} \alpha_{q,\ell} &:= \mathrm{P}(\Phi_\ell \text{ ist ein Parse der Länge } \ell \text{ endend in } q,\ S[1..\ell] = \sigma[1..\ell]) \\ &= \sum_{\substack{\psi \text{ Parse der Länge } \ell \\ \text{endend in } q}} \mathrm{P}(\Phi_\ell = \psi,\ S[1..\ell] = \sigma[1..\ell]) \end{aligned} \tag{1.13}$$

für alle $q \in Q$ und $1 \leq \ell \leq t$. Wiederum definieren wir ergänzend $\alpha_{q_{\text{init}},0} = 1$ und $\alpha_{q,0} = 0$ für alle $q \neq q_{\text{init}}$. Wir können die folgende **Vorwärts-Rekursion** für die Vorwärts-Variable herleiten, analog zur Herleitung der Viterbi-Rekursion.

$$\boxed{\alpha_{q,\ell} = \sum_{\substack{1 \leq \ell' < \ell, q' \in Q \\ \text{oder } q' = q_{\text{init}}, \ell' = 0}} \alpha_{q',\ell'} \cdot a_{q',q} \cdot e_{q'}(\sigma(\ell', \ell])} \tag{1.14}$$

Der *Vorwärts-Algorithmus* besteht einfach nur aus der Berechnung der Vorwärts-Variablen.

Vorwärts-Algorithmus:
1. Berechne mittels (1.14) iterativ die Vorwärts-Variablen $\alpha_{q,\ell}$ in einer Reihenfolge aufsteigender ℓs und speichere sie in einer Tabelle.

Die Vorwärts-Variablen können benutzt werden, um die Wahrscheinlichkeit $\mathrm{P}(S = \sigma)$ einer Emission σ zu berechnen.

Satz 1.5 Sei $\sigma \in \Sigma^*$ eine Emission der Länge t. Dann ist

$$\mathrm{P}(S = \sigma) = \sum_{q \in Q} \alpha_{q,t} \cdot a_{q,q_{\text{term}}} \tag{1.15}$$

Beweis: Für alle $q \in Q$ gilt

$$\alpha_{q,t} \cdot a_{q,q_{\text{term}}} = \mathrm{P}(\Phi \text{ ist ein Parse der Länge } t \text{ endend in } q,\ S[1..t] = \sigma[1..t])$$

Also folgt (1.15) durch Anwendung des Satzes von der totalen Wahrscheinlichkeit.

□

Die Kenntnis der Wahrscheinlichkeit der Emission σ erlaubt uns die Berechnung der a-posteriori-Wahrscheinlichkeit eines Parses ψ mittels folgenden Satzes.

Satz 1.6 (A-posteriori Wahrscheinlichkeit eines Parses) Sei $\sigma \in \Sigma^+$ eine Emission der Länge t und sei $\psi = ((x_1, d_1), \ldots, (x_n, d_n))$ ein Parse der Länge t. Definiere $y_1, y_2, \ldots y_n \in \Sigma^*$ so, dass

diese Wörter aneinandergehängt σ ergeben, $y_1 y_2 \cdots y_n = \sigma$, und so dass $|y_i| = d_i$ für $i = 1, \ldots n$ und sei $x_0 := q_{\text{init}}, y_0 := \varepsilon$. Dann gilt

$$P(\Phi = \psi \mid S = \sigma) = \frac{\left(\prod_{i=1}^{n} a_{x_{i-1},x_i} \cdot e_{x_{i-1}}(y_i)\right) \cdot a_{x_n,q_{\text{term}}}}{\sum_{q \in Q} \alpha_{q,t} \cdot a_{q,q_{\text{term}}}}$$

Beweis:

$$\begin{aligned} & P(\Phi = \psi \mid S = \sigma) \\ = \; & \frac{P(\Phi = \psi, S = \sigma)}{P(S = \sigma)} \\ = \; & \frac{P(((X_1, Y_1), \ldots, (X_n, Y_n)) = ((x_1, y_1), \ldots, (x_n, y_n))) \cdot a_{x_n,q_{\text{term}}}}{S = \sigma)} \\ = \; & \frac{\left(\prod_{i=1}^{n} a_{x_{i-1},x_i} \cdot e_{x_{i-1}}(y_i)\right) \cdot a_{x_n,q_{\text{term}}}}{\sum_{q \in Q} \alpha_{q,t} \cdot a_{q,q_{\text{term}}}} \end{aligned}$$

Hierbei folgt die erste Zeile aus der Definition der bedingten Wahrscheinlichkeit, die zweite folgt, da der Parse zusammen mit der Gesamtemission das Ergebnis $(\mathbf{X}, \mathbf{Y})$ bestimmt, und die dritte Zeile folgt aus (1.4) und Satz 1.5.

1.3 Paar-Hidden-Markow-Modelle

1.3.1 Vergleiche genomischer DNA zwischen Arten

Die oben beschriebenen GHMMs finden Gene in einer DNA-Sequenz u, wobei sie als Eingabe auch nur diese Sequenz u haben. Solch eine Methode der Genvorhersage nennt man eine *ab-initio*-Methode, manchmal auch *intrinsische* Methode. Im Gegensatz dazu nennt man Methoden der Genvorhersage, bei denen außer u noch weitere Eingaben verwendet werden, *extrinsische* Methoden. Eine der extrinsichen Methoden ist die sogenannte *vergleichende Genvorhersage*. Hierbei verwendet man eine zweite DNA-Sequenz v (manchmal auch mehrere) als zusätzliche Eingabe. Dies kann im Vergleich zu ab-initio-Methoden insbesondere von Vorteil sein, wenn u und v ein *orthologes* Gen-Paar enthalten. Ein orthologes Gen-Paar ist ein Paar von Genen in verschiedenen Spezies, die in der Evolution ein gemeinsames Vorfahr-Gen haben. Normalerweise behalten orthologe Gene die gleiche Funktion während der Evolution.

Im Falle der Spezies *Homo sapiens* und *Mus musculus* (Maus) sind die Ähnlichkeiten von Sequenzpaaren, die ein orthologes Gen-Paar enthalten, untersucht worden [BPM$^+$00]. Für ein Beispiel siehe Abbildung 1.9. In den untersuchten Gen-Paaren waren die Anzahl der Exons des menschlichen Gens und die Anzahl der Exons des Maus-Gens in 95% der Fälle identisch. An Alignments, die Exons zusammen alignieren, die sich entsprechen, kann man erkennen, dass die kodierenden Sequenzen eine viel größere Ähnlichkeit aufweisen als die nicht-kodierenden Sequenzen. Die Sequenzidentität innerhalb der kodierenden Sequenzen wird in [BPM$^+$00] mit etwa 85% angegeben und die der nicht-kodierenden mit nur etwa 35%.

Bemerkung: Der Begriff Sequenzidentität (percent sequence identity) wird sehr oft ohne klare Definition verwendet, wie zum Beispiel auch in [BPM$^+$00]. Er bezieht sich auf ein Alignment von zwei Sequenzen. Darin werden die Spalten gezählt, in denen beide Sequenzen identisch sind. Der Begriff hängt aber (unter Umständen sogar stark) davon ab, womit man normiert. Teilt man durch die Gesamtzahl der Spalten des Alignments? Oder teilt man durch die Länge der kürzeren oder die der längeren Sequenz? Es gibt hier keine einheitliche Vorgehensweise in der Literatur. Außerdem hängt die Sequenzidentität noch vom verwendeten Alignment ab. Wenn der Begriff auftaucht, muss er oft als grobe und leider schwammige Angabe interpretiert werden.

Die Längen der sich entsprechenden Exons sind in etwa 3 von 4 Fällen identisch. Falls die Längen verschieden sind, unterscheiden sie sich nur wenig und die Längendifferenz ist fast immer ein Vielfaches von 3. Dies kann wie folgt erklärt werden. Ein Längenunterschied, der nicht Vielfaches von 3 ist, würde einen zweien Längenunterschied in folgenden Exons nötig machen, der den ersten kompensiert so dass der gleiche Leserahmen in sich entsprechenden Exons für den Rest des Gens wiederhergestellt wird. Dies ist sehr selten. Die entsprechenden Intronlängen variieren stärker wie auch in Abbildung 1.9 zu sehen ist. Das mittlere Verhältnis vom jeweils längeren zum jeweils kürzeren Intron ist etwa 1.5. Bei Intronpaaren gibt es keine Tendenz zu Längendifferenzen, die ein Vielfaches von 3 sind.

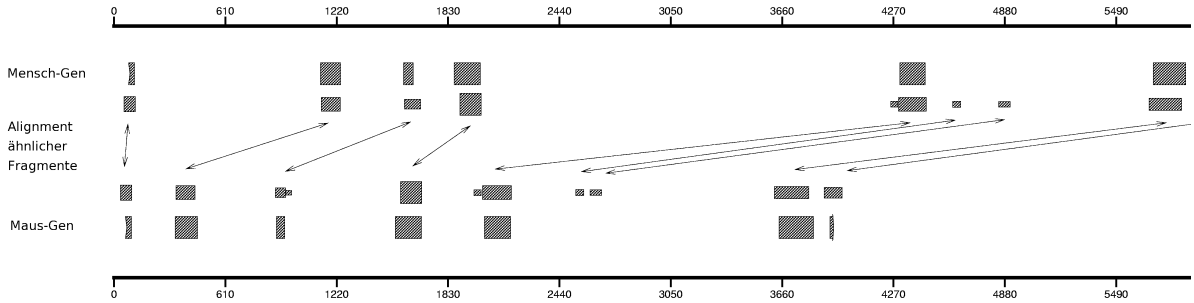


Abbildung 1.9: Ein paar orthologer Sequenzen. Die oberste bzw. unterste Zeile zeigt ein Gen vom Menschen bzw. von der Maus. Die Exons sind als Balken dargestellt. Diese Sequenzen wurden mit dem Programm DIALIGN aligniert. Die Mitte des Schaubilds zeigt die Segmentpaare der beiden Sequenzen, die sich besonders ähnlich sind.

Diese typische Stuktur der Konserviertheit bei Sequenzen u und v, die orthologe Gen-Paaren enthalten, ist hilfreich für die Genvorhersage. Im folgenden wird ein verallgemeinertest Paar-Hidden-Markow-Modell (GPHMM) beschrieben, das diese zusätzliche Information ausbeuten kann, die Gene in beiden Sequenzen vorhersagen kann und gleichzeitig die Sequenzen teilweise aligniert. Neben diesem GPHMM gibt es noch viele andere Ansätze der vergleichenden Genvorhersage.

1.3.2 Definition GPHMM

Ein verallgemeinertes Paar-Hidden-Markow-Modell (GPHMM) ist wiederum eine Verallgemeinerung des GHMM aus dem vorigen Abschnitt. Der Unterschied ist, dass in jedem Zustand *zwei* Sequenzen Y und Z emittiert werden anstatt nur einer Sequenz Y. Diese Sequenzen können wieder beliebige und voneinander verschiedene Längen haben. Deshalb heißt es 'verallgemeinert' (**g**eneralized), bei einem Paar-Hidden-Markow-Modell (PHMM) bestehen Y und Z jeweils aus höchstens einem Zeichen.

Definition 1.7 (GPHMM) Sei Q^+ wie in (1.1), A wie in (1.2) und sei Σ eine abzählbare Menge, das *Emissionsalphabet*. Weiter seien die Wahrscheinlichkeiten $e_q(\sigma, \tau)$ definiert für $q \in Q^+, \sigma, \tau \in \Sigma^*$. Ein *verallgemeinertes Paar-Hidden Markow Model (GPHMM)* mit Zustandsraum Q^+, Übergangsmatrix A und *Emissionswahrscheinlichkeiten* $e_q(\sigma, \tau) \quad (q \in Q^+, \sigma, \tau \in \Sigma^*)$ ist eine Folge

$$(X_0, Y_0, Z_0), (X_1, Y_1, Z_1), (X_2, Y_2, Z_2), \ldots$$

bei der $X_0 \equiv q_{\text{init}}$ ist, die Folge $X_0, X_1, X_2, \ldots$ eine homogene Markow-Kette mit Zustandsraum Q^+ und Übergangsmatrix A ist und wobei $Y_0, Y_1, \ldots$ und $Z_0, Z_1, \ldots$ Folgen von Zufallsvariablen

mit Werten in Σ^* sind, so dass $Y_0, Z_0 \equiv \varepsilon$ und

$$\begin{aligned} e_{x_i}(y_i, z_i) &= \mathrm{P}(Y_i = y_i, Z_i = z_i \,|\, X_i = x_i) \\ &= \mathrm{P}(Y_i = y_i, Z_i = z_i \,|\, X_0 = x_0, \ldots, X_i = x_i, \\ &\qquad\qquad Y_0 = y_0, \ldots, Y_{i-1} = y_{i-1}, \\ &\qquad\qquad Z_0 = z_0, \ldots, Z_{i-1} = z_{i-1}) \end{aligned}$$

für alle $i > 0$ und $x_0, \ldots, x_i \in Q^+$, $y_0, \ldots y_i, z_0, \ldots z_i \in \Sigma^*$. Die Emissionswahrscheinlichkeiten müssen $e_q(\varepsilon, \varepsilon) = 0 \quad (q \in Q)$ und $e_{q_{\text{init}}}(\varepsilon, \varepsilon), e_{q_{\text{term}}}(\varepsilon, \varepsilon) = 1$ erfüllen.

Zur Erinnerung: T ist der zufällige Endzeitpunkt der Markow-Kette. In den Anwendungen von einem GPHMM ist die versteckte Struktur die Abfolge der Zustände X_i und die Abfolge der Emissionslängen *beider* Emissionen Y_i und Z_i.

Definition 1.8 (Biparse, induzierter Biparse) Seien $x_1, \ldots, x_n \in Q$ und $d_1^y, \ldots, d_n^y, d_1^z, \ldots, d_n^z \geq 0$ ganze Zahlen. Der Vektor

$$((x_1, d_1^y, d_1^z), \ldots, (x_n, d_n^y, d_n^z)) \tag{1.16}$$

wird ein *Biparse* der Längen ℓ und r genannt, wenn $d_1^y + \cdots + d_n^y = \ell$ und $d_1^z + \cdots + d_n^z = r$. Er *endet* in x_n.

Der von $X_0, X_1, \cdots, Y_0, Y_1, \cdots, Z_0, Z_1, \cdots$ induzierte Biparse Φ wird definiert als

$$\Phi := ((X_1, |Y_1|, |Z_1|), \ldots (X_T, |Y_T|, |Z_T|)) \tag{1.17}$$

Für $\ell \geq 1, r \geq 1$ wird der (ℓ, r)-*gestutzte* durch induzierte Biparse definiert durch

$$\begin{aligned} \Phi_{\ell,r} &:= ((X_1, |Y_1|, |Z_1|), \ldots (X_r, |Y_r|, |Z_r|)) \\ &\quad \text{mit } r := \max\{n \,|\, |Y_1| + \cdots + |Y_n| \leq \ell, |Z_1| + \cdots + |Z_n| \leq r, x_n \neq q_{\text{term}}\} \end{aligned}$$

Analog zum GHMM ist in der Anwendung des GPHMM der Biparse unbekannt. Sei der String U die Verkettung der Strings $Y_1, Y_2, \ldots, Y_T$ und sei der String V die Verkettung der Strings $Z_1, Z_2, \ldots, Z_T$. Beobachtet werden können nur U und V, der Biparse Φ ist unbekannt, d.h. man weiss nicht, an welchen Stellen der Strings U und V Übergänge stattfinden. U, V und Φ zusammen bestimmen wiederrum eindeutig den Ausgang $(X_0, Y_0, Z_0), (X_1, Y_1, Z_1), \ldots$ des Zufallsexperiments.

Beispiel:

$X_0 = q_{\text{init}}$	X_1	X_2	X_3	$X_4 = q_{\text{term}}$	
$Y_0 = \varepsilon$	$Y_1 = \text{aatg}$	$Y_2 = \varepsilon$	$Y_3 = \text{cctc}$	$Y_4 = \varepsilon$	$U = \text{aatgcctc}$
$Z_0 = \varepsilon$	$Z_1 = \text{acg}$	$Z_2 = \text{tt}$	$Z_3 = \text{ccgc}$	$Z_4 = \varepsilon$	$V = \text{acgttccgc}$

$\Phi = ((X_1, 4, 3), (X_2, 0, 2), (X_3, 4, 4))$, $\Phi_{4,5} = ((X_1, 4, 3), (X_2, 0, 2))$, $\Phi_{8,4} = ((X_1, 4, 3))$.

1.3.3 Viterbi-Algorithmus

Es sind zwei Sequenzen $u, v \in \Sigma^*$ gegeben. Bei der Genvorhersage sind dies die nicht-alignierten Sequenzen zweier Spezies, die orthologe Gene enthalten. Wir suchen den unbekannten Biparse. Dieser wird dann eine Genstruktur auf beiden Sequenzen definieren. Der wahrscheinlichste Biparse, gegeben die beobachteten Sequenzen u und v lässt sich wieder mit einer Erweiterung des

Viterbi-Algorithmus berechnen. Wir nennen einen wahrscheinlichsten Biparse ψ_{vit} wieder einen Viterbi-Biparse:

$$\psi_{\text{vit}} \in \underset{\substack{\psi \text{ Biparse der Längen} \\ |u| \text{ und } |v|}}{\operatorname{argmax}} \mathrm{P}(\Phi = \psi \,|\, U = u, V = v). \tag{1.18}$$

Die Berechnung kann analog zum Viterbi-Algorithmus für GHMMs geschehen. In diesem Fall braucht man jedoch eine Viterbi-Variable für jeden Zustand q jede Position ℓ in der Sequenz u *und* jede Position r in der Sequenz v.

Die *Viterbi-Variablen* sind hier

$$\gamma_{q,\ell,r} := \max_{\substack{\psi \text{ Biparse der Längen} \\ \ell \text{ und } r \\ \text{endend in } q}} \mathrm{P}(\Phi_{\ell,r} = \psi, U[1..\ell] = u[1..\ell], V[1..r] = v[1..r]). \tag{1.19}$$

für $q \in Q$, $1 \leq \ell \leq |u|$ und $1 \leq\ r \leq |v|$. Zusätzlich setzen wir wieder $\gamma_{q_{\text{init}},0,0} = 1$ damit es unten übersichtlicher wird. Die Rekursion für die Viterbi-Variablen kann analog hergeleitet werden. Es gilt

$$\gamma_{q,\ell,r} = \max_{\substack{1 \leq \ell' \leq \ell \\ 1 \leq r' \leq r \\ (\ell',r') \neq (\ell,r) \\ q' \in Q \\ \text{oder } q' = q_{\text{init}}, \ell' = r' = 0}} \gamma_{q',\ell',r'} \cdot a_{q',q} \cdot e_q(u(\ell'..\ell], v(r'..r]). \tag{1.20}$$

Die Fälle $\ell' = \ell$ und $r' = r$ sind erlaubt weil beim GPHMM emittierte Strings die Länge 0 haben dürfen (der leere String ε). Es ist allerdings nicht erlaubt (wegen $e_q(\varepsilon, \varepsilon) = 0$ für $q \in Q$), dass in einem Zustand beide Strings die Länge 0 haben. Deshalb ist der Fall $(\ell', r') = (\ell, r)$ ausgeschlossen. Der Viterbi-Algorithmus besteht wieder daraus, die Tabelle der Viterbi-Variablen zu berechnen und durch Backtracing durch die Tabelle einen Viterbi-Pfad zu bestimmen. Der Speicherbedarf ist $O(|Q||u||v|)$ und im allgemeinen Fall ist der Zeitbedarf $O(|Q||u|^2|v|^2)$, wenn die Emissionswkeiten in (1.20) in konstanter Zeit bestimmt werden können. Kann die Länge der Emissionen durch eine Konstante d nach oben beschränkt werden, ist die Laufzeit $O(|Q||u||v|d^2)$. Dies ist etwa beim PHMM der Fall; dort ist $d = 1$. Allerdings verbietet sich für viele praktische Anwendungen eine Laufzeit, die proportional zum Produkt der beiden Sequenzlängen ist. Aus diesem Grund beschränken sich einige Implementationen auf die Berechnung von $\gamma_{q,\ell,r}$ für bestimmte Bereiche von Paaren (ℓ, r), die vorher ermittelt wurden (ungefähres Alignment, approximate alignment).

1.3.4 Beispiel: GPHMM für vergleichende Genvorhersage

Das Konzept eines GPHMMs für die Genvorhersage wurde in jüngster Vergangenheit mehrfach implementiert: DOUBLESCAN (2002), SLAM (2003), TWAIN. Abbildung 1.10 zeigt den Zustandsgraphen eines einfachen, theoretischen GPHMMs für eukaryotische Gene. Die Emissionswahrscheinlichkeiten $e_q(\sigma, \tau)$ müssen so definiert werden, dass sie relativ groß nur für solche Sequenzpaare σ, τ sind, die bei orthologen Sequenzpaaren häufig einander entsprechen. Etwa für q =Kodons könnte man $e_q(\sigma, \tau)$ so wählen, dass $e_q(\sigma, \tau) = 0$ ist, wenn σ und τ eine zu stark voneinander abweichende Länge haben. Weiter sollte berücksichtigt werden, dass im Zustand q =Kodons die Sequenzen σ und τ sich typischerweise ähnlicher sind als in einem Zustand $q \in \{\text{Intron0}, \text{Intron1}, \text{Intron2}, \text{IR}\}$. In q =Kodons kommt in keiner der beiden emittierten Sequenzen ein Stoppkodon im Leserahmen vor. Das in Figur 1.10 beschriebene GPHMM dient nur

zur Erläuterung des Konzepts. In praktischen GPHMMs modelliert man nicht-kodierende Zustände aus Effizienzgründen mit Zuständen, die Seqenzpaare emittieren, die nur aus 1 oder 0 Zeichen bestehen, und in die eine Selbstschleife führt.

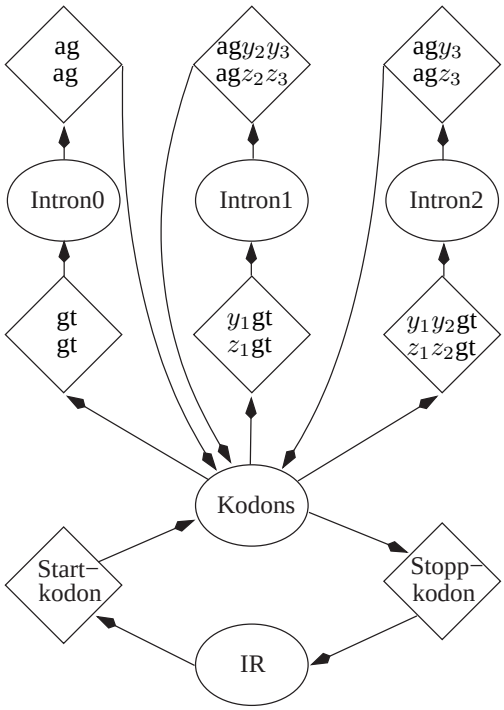


Abbildung 1.10: GPHMM für Eukaryoten. In jedem Zustand werden zwei Sequenzen emittiert. In IR (intergenic region) werden zwei zwischengenische Regionen variabler Länge emittiert. Im Zustand Startkodon/Stoppkodon werden zwei Startkodons bzw. Stoppkodons emittiert. Im Zustand Kodons werden zwei Sequenzen mit jeweils einer ganzen Anzahl Kodons emittiert. Die beiden Längen sind hier also Vielfaches von 3, brauchen aber nicht gleich sein. In den drei Intronzuständen Intron0, Intron1 und Intron2 werden jeweils zwei Intronsequenzen emittiert, exklusive zwei kleiner Stücke an den Rändern des Introns, die in den Splice-Site-Zuständen emittiert werden. Die Splice-Site-Zustände emittieren jeweils den Donor-Splice-Site-Konsensus gt oder den Acceptor-Splice-Site-Konsensus ag und weitere 0-2 Basen, je nachdem an welcher Stelle im Leserahmen das Intron das Exon unterbricht.

1.4 Spliced Alignment

1.4.1 Netzwerke und Sequenzen ausrichten - Network Alignment

Ähnlickeit zweier Wörter

Zur Erinnerung und zur Einführung der Notation beschreibe ich hier das Problem der globalen Ähnlichkeit von zwei Wörtern. Es ist das sogenannte Needleman-Wunsch-Problem. Sei Σ ein Alphabet und seien $S, T \in \Sigma^+$ zwei Wörter. Sei $\Sigma' = \Sigma \cup \{-\}$ das um das Gap-Zeichen '$-$' erweiterte Alphabet. Und sei für jeweils zwei Buchstaben $x, y \in \Sigma'$ ein reeller oder ganzzahliger score $s(x, y)$ definiert. Ein Alignment $\mathcal{A}$ von S mit T ist ein Paar von Wörtern S' und T', so dass S' aus S bzw T' aus T durch Einfügen einer beliebigen Anzahl von Gap-Zeichen an beliebigen Stellen entstanden ist und so dass S' und T' die gleiche Länge ℓ haben. Dann ist der Wert (Score)

des Alignments $\mathcal{A}$ definiert als $s(\mathcal{A}) = \sum_{i=1}^{\ell} s(S'[i], T'[i])$. Die Ähnlichkeit von S und T ist definiert als

$$s(S,T) = \max_{\mathcal{A} \text{ Alignment von } S \text{ mit } T} s(\mathcal{A}).$$

Damit das Maximum überhaupt definiert ist, wollen wir verlangen, dass $s(x,-), s(-,x) \leq 0$ für alle $x \in \Sigma'$.

Beispiel: $\Sigma = \{A, C, G, T\}$, $s(x,x) = 1$ für $x \in \Sigma$, $s(x,y) = 0$ für $x, y \in \Sigma$ und $x \neq y$. Einfügungskosten (insert costs) $s(-,x) = -1$, Löschkosten (delete costs) $s(x,-) = -1$ für $x \in \Sigma$. $S =$ AGCTC, $T =$ ACGC, $\mathcal{A} = \begin{matrix} \text{A} & \text{G} & \text{C} & \text{T} & \text{C} \\ \text{A} & \text{-} & \text{C} & \text{G} & \text{C} \end{matrix}$ ist ein Alignment mit maximalem Score. Also $s(S,T) = s(\mathcal{A}) = 1 + (-1) + 1 + 0 + 1 = 2$

In obigem Scoring-Schema werden die Gapkosten als *linear* bezeichnet, weil die Gesamtkosten für ein Gap proportional zu seiner Länge sind. Ein optimales Aligment kann mittels dynamischen Programmierens berechnet werden. Der Algorithmus ist bekannt und wird hier nicht besprochen. Das Problem ist aber ein ein Spezialfall des sogenannten 'Network Aligment'-Problems, das unten gelöst wird.

Das 'Network Aligment'-Problem

Ein *Netzwerk* ist ein gerichteter azyklischer Graph (häufig DAG=directed acyclic graph) G mit einem ausgezeichneten Startknoten Start und einem ausgezeichneten Endknoten Ende, bei dem jede Kante mit einem Buchstaben beschriftet ist. Wir nehmen an, dass jeder Knoten durch einen Pfad in G von Start aus erreichbar ist und dass der Endknoten von jedem anderen Knoten durch einen Pfad in G erreichbar ist.

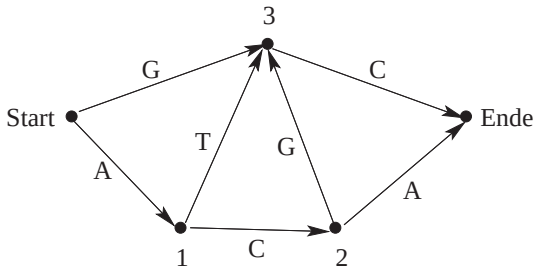


Abbildung 1.11: Beispiel eines Netzwerks.

Jeder Pfad von Start nach Ende in G buchstabiert ein Wort: Das Wort der Beschriftungen der Kanten entlang des Pfades. Wir sagen, das Wort *ist in* G. In obigem Beispiel sind die Wörter GC,ACA,ATC und ACGC in G. Wir betrachten im Folgenden nur Pfade, die in Start anfangen. Das **'Network-Alignment'-Problem** ist:

Für ein Wort S der Länge n und ein gegebenes Netzwerk G, finde ein Wort S' in G mit maximaler Ähnlichkeit zu S unter allen Wörtern in G. Finde ausserdem diese maximale Ähnlichkeit und einen Pfad von Start nach Ende in G, der S' buchstabiert.

Das Problem kann wie folgt gelöst werden. Sei $G = (V, E)$. Also V ist die Menge der Knoten in G und E die Menge der Kanten. Für eine Kante $e \in E$ sei $\ell(e)$ der Buchstabe, mit dem e beschriftet (gelabeled) ist. Sei jetzt ein Wort S der Länge n gegeben. Um das Problem mit dynamischem Programmieren zu lösen, definieren wir für $0 \leq i \leq n$ und $v \in V$

$$A(v,i) := \text{maximale Ähnlichkeit vom Präfix } S[1..i] \text{ zu einem Wort, das von einem Pfad in } G \text{ buchstabiert wird, der in Knoten } v \text{ endet.}$$

Die gesuchte maximale Ähnlichkeit von S zu einem Wort in G ist dann $A(n, \text{Ende})$. Für die $A(v,i)$ gilt für $0 < i \leq n, v \in V$ die folgende Rekursion:

$$\begin{aligned} A(v,i) \quad = \max \big\{ \quad & \max_{e=(u,v)\in E} A(u,i-1) + s(S[i], \ell(e)), \\ & \max_{e=(u,v)\in E} A(u,i) + s(-, \ell(e)), \\ & A(v,i-1) + s(S[i], -) \big\} \end{aligned} \tag{1.21}$$

Beweis: Seien $i > 0$ und $v \in V$. Sei $\phi = (v_1 \ldots, v_n)$ mit $u := v_{n-1}, v = v_n$ und $e := (u,v)$ ein Pfad in G von Start nach v, der ein Wort T größter Ähnlichkeit zu $S[1..i]$ buchstabiert. Es gibt drei Fälle für die letzte Spalte im optimalen Alignment von $S[1..i]$ mit T. (Wenn ein solcher Pfad nicht existiert, ist v der Startknoten und nur der letzte der 3 Fälle ist möglich.)
Erster Fall - Match. $S[i]$ wird zusammen mit dem letzten Buchstaben von T, also mit $\ell(e)$, ausgerichtet. Der Score für diese letzte Spalte ist $s(S[i], \ell(e))$. Dann war der um einen Schritt verkürzte, in u endende Pfad $\phi' = (v_1, \ldots, v_{n-1})$ ein Pfad, der das Wort größter Ähnlichkeit zu $S[1..i-1]$ buchstabierte unter allen Pfaden, die in u enden. Diese größte Ähnlichkeit ist also nach Definition $A(u, i-1)$. Es gilt also $A(v,i) = A(u,i-1) + s(S[i], l(e))$. Weil ϕ optimal war, sind alle anderen Pfade höchstens so gut und es gilt $A(v,i) = \max_{e=(u,v)\in E} A(u,i-1) + s(S[i], l(e))$.
Zweiter Fall - Insert. '-' wird zusammen mit dem letzten Buchstaben von T, also mit $\ell(e)$, ausgerichtet. Der um einen Schritt verkürzte, in u endende Pfad $\phi' = (v_1, \ldots, v_{n-1})$ ist ein Pfad, der das Wort größter Ähnlichkeit zu $S[1..i]$ buchstabierte unter allen Pfaden, die in u enden. Es gilt also $A(v,i) = A(u,i-1) + s(-, l(e))$. Weil ϕ optimal war, sind alle anderen Pfade höchstens so gut und es gilt $A(v,i) = \max_{e=(u,v)\in E} A(u,i-1) + s(-, l(e))$.
Dritter Fall - Delete. $S[i]$ wird zusammen mit $-$ ausgerichtet. Dann war der Pfad ϕ bereits der optimale Pfad, der ein Wort maximaler Ähnlichkeit zu $S[1..i-1]$ buchstabiert und in v endet. Diese maximale Ähnlichkeit ist $A(v, i-1)$. Also ist $A(v,i) = A(v,i-1) + s(S[i], -)$. Da das Alignment von $S[1..i]$ mit T optimal war, muß $A(v,i)$ gleich dem Maximum der sich aus den drei Möglichkeiten ergebenden $A(v,i)$'s sein.
Wenn die Knoten in V topologisch sortiert (wenn von u nach v ein Pfad führt, kommt u vor v in der Sortierung) sind in Richtung von Start nach Ende, kann die Tabelle der $A(v,i)$ mit wachsendem i und der Sortierung der Knoten v gemäß schrittweise gefüllt werden. Für die Tabelle, die dem obigen Beispielgraphen und dem Wort $S =$ AGCTC entspricht, siehe 1.1.

Die für die dynamische Programmierung nötigen Anfangsfälle sind: $A(\text{Start}, 0) = 0$, $A(\text{Start}, i)$ ist für $i > 0$ die Ähnlichkeit von $S[1..i]$ mit dem leeren Wort, also die Kosten für das Löschen der ersten i Zeichen von S. $A(v,0)$ kann mit der Rekursion $A(v,0) = \max_{e=(u,v)\in E} A(u,0) + s(-, \ell(e))$ bestimmt werden. Ein Algorithmus der das Problem löst, besteht also aus den Schritten. Sortiere zuerst die Knoten topologisch (geht in Zeit $O(|E|)$), fülle dann die erste Zeile und Spalte in der Matrix $(A(v,i))$ und berechne dann die Rekursion 1.21 in geeigneter Reihenfolge.

Satz: Das Netzwerk-Alignment Problem für eine Eingabesequenz S und ein Netzwerk mit Kantenmenge E kann in $O(|S| \cdot |E|)$ gelöst werden.

Bemerkung: In einer späteren Anwendung wird es sinnvoll sein, Netzwerke zu definieren, bei denen manche Kanten *nicht* mit einem Buchstaben beschriftet sind. Wir nennen diese Kanten *stille Kanten*. Das von einem Pfad buchstabierte Wort ist dann die Verkettung der Buchstaben entlang der nicht-stillen Kanten des Pfades. Obiger Algorithmus funktioniert auch in diesem Fall, wenn man die stillen Kanten mit dem Gap-Zeichen '-' beschriftet und $s(-,-) = 0$ definiert.

		i					
	$A(v,i)$	0	1	2	3	4	5
	Start	0	-1	-2	-3	-4	-5
	1	-1	1	0	-1	-2	-3
v	2	-2	0	1	1	0	-1
	3	-1	0	1	1	1	0
	Ende	-2	-1	0	2	1	2

$$\mathcal{A} = \begin{array}{cccccc} \text{A} & \text{G} & \text{C} & \text{T} & \text{C} \\ \text{A} & \text{-} & \text{C} & \text{G} & \text{C} \end{array}$$

Tabelle 1.1: Matrix des dynamischen Programmierens mittels Rekursion 1.21. Die Ähnlichkeit wurde gemessen mit dem Scoring-Schema aus dem Beispiel im Abschnitt über die Ähnlichkeit zweier Wörter: Match: +1, Mitmatch: 0, Insert/Delete: -1. Die Einträge entlang des Backtracing-Pfades der Pfeile haben jeweils das Maximum bestimmt. Das Wort $S' = ACGC$ ist das Wort in G mit der größten Ähnlichkeit zu $S = AGCTC$. S' und das optimale Alignment $\mathcal{A}$ rechts ergeben sich aus dem Backtracing.

1.4.2 Spliced Alignment mittels Network Alignment

Das Spliced-Alignment-Problem entsteht typischerweise in den beiden folgenden Anwendungen, in denen man in einer DNA-Sequenz ein Gen mit möglicherweise mehreren Exons vorhersagen will. In der einen Anwendung haben wir bereits die Aminosäuresequenz T (wir sagen auch 'Proteinssequenz' für 'Aminosäuresequenz') eines verwandten Proteins zur Verfügung. Wir gehen davon aus, dass sich die unbekannte Proteinssequenz und die gegebene Sequenz T ähnlich sind. Aber Einfügungen, Löschungen und Ersetzungen können vorkommen. Diese Problemstellung entsteht etwa, wenn man ein Protein T von einer verwandten Spezies gefunden hat, das dem Protein entspricht, das vom unbekannten Gen kodiert wird. Dieses Zielprotein T zur Eingabesequenz kann man z.B. mit einer schnellen Datenbanksuche (BLAST) in einer Proteindatenbank finden, wenn ein solches überhaupt existiert. In der anderen Anwendung hat man eine Datenbank mit sogenannten "expressed sequence tags" (ESTs). Diese sind einfach abgelesene, im Durchschnitt ca. 500 bp lange zufällige Stücke von cDNAs (cDNA: Kopie von mRNA). Die meisten Genomprojekte beinhalten auch das Sequenzieren einer großen Anzahl von ESTs, die allerdings von relativ geringer Qualität sind. Ein Spliced Alignment das ESTs verwendet, kann helfen, einen Teil des Gens zu rekonstruieren.

Zunächst wird das Problem des *Spliced Alignments* formal eingeführt, dann beschrieben, wie obiges Genvorhersage-Problem als Spliced-Alignment-Problem formuliert werden kann. Und zuletzt gezeigt, dass das Problem als Network-Alignment-Problem formuliert werden kann.

Sei $S = s_1 \cdots s_n$ ein Wort und seien $B = s_i \cdots s_j$ und $B' = s_{i'} \cdots s_{j'}$ Teilwörter von S. Wir schreiben $B \prec B'$, wenn $j < i'$, also wenn B endet bevor B' anfängt. Eine Folge $\Gamma = (B_1, \ldots, B_p)$ von Teilwörtern von S ist eine *Kette*, wenn $B_1 \prec B_2 \prec \cdots \prec B_p$. Wir bezeichnen mit Γ^* die Verkettung (*engl.* concatenation) $B_1 B_2 \cdots B_p$ der Teilwörter von Γ. Für zwei Wörter S und T sei $s(S,T)$ der Score des optimalen Alignments von S mit T.

Spliced-Alignment-Problem: Sei $S = s_1 \cdots s_n$ ein Wort (genomische Sequenz) und $T = t_1 \cdots t_m$ ein Wort (Zielsequenz) und $\mathcal{B} = \{B_1, \ldots, B_b\}$ eine Menge von Teilwörtern von S, *Blöcke* genannt. Das Spliced-Alignment-Problem ist das Problem, eine Kette Γ von Blöcken zu finden, so dass der Score $s(\Gamma^*, T)$ des optimalen Alignments zwischen der Verkettung dieser Blöcke

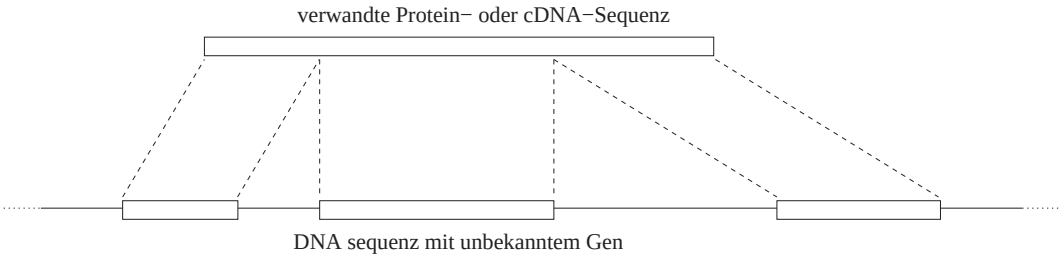


Abbildung 1.12: Beispiel für Spliced Alignment. Wenn die die oben angegebene Zielsequenz T eine cDNA-Sequenz ist, ist die Verkettung der unbekannten Exonsequenzen ähnlich zu T. Im Falle einer Protein-Zielsequenz ist die Verkettung der in Proteinsequenzen übersetzten Exons ähnlich zu T.

und der Zielsequenz maximal unter allen Ketten aus $\mathcal{B}$ ist.

In der konkreten Anwendung, in der eine EST-Zielsequenz gegeben ist, ist $\mathcal{B}$ eine Menge von Exonkandidaten, die möglichst alle wahren Exons enthält. Im einfachsten Fall ist dies die Menge aller Teilwörter, die gewisse minimale Zusatzvoraussetzungen erfüllen. Am linken Ende des Teilwortes muß eine Acceptor Splice Site oder der Translationsstart und am rechten Ende des Teilworts muß eine Donor Splice Site oder das Translationsende möglich sein können. In mindestens einem der möglichen Leserahmen darf kein Stopp-Kodon enthalten sein. Wenn man etwa speziell fordert, dass der Block mit atg beginnt, oder direkt auf das Dinukleotid ag (Minimalkonsensus einer Acceptor SS) folgt und mit einem Stopp-Kodon endet oder direkt vom Dinukleotid gt (Minimalkonsensus einer Donor SS) gefolgt wird, ist die durchschnittliche Anzahl der Exon-Kandidaten, also $|\mathcal{B}|$, etwa so groß wie die Anzahl von Basen in S: $|B| \approx |S|$. (Das ist zufällig so.). Wenn die Zielsequenz eine Proteinsequenz ist, können für die Blöcke die in allen möglichen Leserahmen in Aminosäuresequenzen übersetzten Exonkandidaten genommen werden. Dann sind die Blöcke zwar formal nicht mehr Teilsequenzen einer einzelnen Sequenz S aber die unten beschriebene Methode funktioniert trotzdem, da auch in diesem Fall eine Halbordnung '$\prec$' auf $\mathcal{B}$ gegeben ist. Für die Score-Funktion s wird dann sinnvollerweise eine Protein-Scoringmatrix benutzt. Die Gap-Kosten sind wie oben beschrieben linear und symmetrisch bezüglich Inserts and Deletes. Die hier beschriebene Methode ist aus [GMP96] und implementiert im Programm PROCRUSTES. Dort wird um die Laufzeit zu reduzieren, die Menge aller Exonkandidaten zusätzlich gefiltert. $\mathcal{B}$ enthält dann nur solche Exonkandidaten, die eine relativ typische Splice-Site Umgebung und Exonsequenz-Zusammensetzung haben.

Das Spliced-Alignment-Problem kann wie folgt als Network-Alignment-Problem aufgefasst werden. Die Menge $\mathcal{B} = \{B_1, \ldots, B_b\}$ von Blöcken definiert das Netzwerk $G = (V, E)$ wie folgt. (Vergleiche Abbildung 1.13). Für jeden Block $B = b_1 b_2 \cdots b_k$ aus $\mathcal{B}$ werden den Buchstaben entsprechende Knoten $v_1 v_2 \ldots v_k$ eingeführt, Kanten gibt es jeweils zwischen v_i und v_{i+1} $(i = 1, \ldots, k-1)$. Vom letzten Knoten v_k eines Blockes führen Kanten zum Knoten Ende und jeweils zum *ersten* Knoten aller derjenigen Blöcke B', die komplett nach B kommen: $B \prec B'$. Vom Knoten Start führen Kanten zum ersten Knoten jedes Blockes. Die Kanten werden jeweils beschriftet mit dem Buchstaben, der dem Knoten entspricht, aus dem die Kante kommt. Die Kanten aus dem Start-Knoten sind stille Kanten. Dann buchstabieren die Pfade von Start zu Ende alle möglichen Wörter Γ^*, die aus Ketten Γ von Blöcken gebildet werden können. Die Lösung des Netzwerk-Alignment-Problems von G und T liefert also die Lösung zum Spliced-Alignment-Problem.

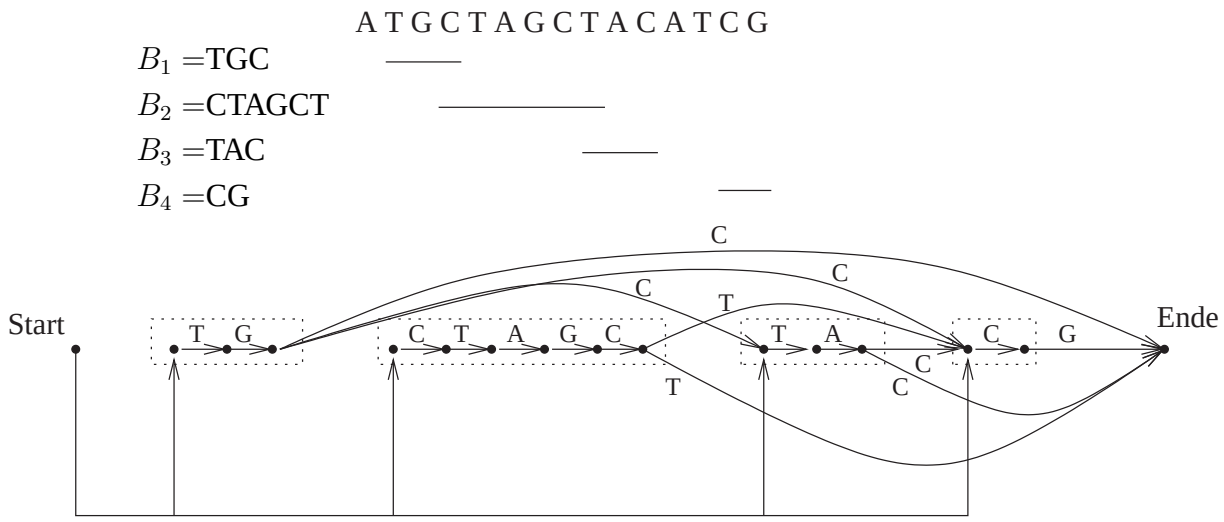


Abbildung 1.13: Beispiel eines Netzwerks G für Spliced Alignment. $\mathcal{B} = \{B_1, B_2, B_3, B_4\}$ ist die Menge der Blöcke.

Sei Länge(B_i) die Länge von Block i, also $c = \sum_{i=1}^{b}$ Länge(B_i) die Gesamtlänge aller Blöcke. Die benötigte Zeit für das Netzwerk-Alignment Problem mit Netzwerk G ist $O(|T| \cdot |E|) = O(|T| \cdot (c + b^2))$, da die Anzahl der Kanten innerhalb der Blöcke höchstens c ist und die Anzahl der Kanten zwischen den Blöcken höchstens b^2. Bei langen Sequenzen S verbietet sich diese Vorgehensweise, weil b typischerweise proportional zur Sequenzlänge von S wächst. Folgender Trick ermöglicht es, eine Laufzeit linear in b zu erreichen. Der ehrgeizige Leser sollte zunächst selber darüber nachdenken bevor er weiterliest.

Wir fügen für jeden Block zusätzlich höchstens zwei Knoten ein, aus denen nur stille Kanten kommen. Wir machen dies mit dem Ziel, die Anzahl der Kanten zu verringern. Hierbei nutzen wir aus, dass in G die Übergange zwischen den Blöcken eine spezielle Struktur haben. Nämlich, wenn Block $B_1 \prec B_2$ und $B_2 \prec B_3$, dann ist auch $B_1 \prec B_3$. Für einen Block $B = s_i \cdots s_j$ seien vor$(B) = i - 1$ und end$(B) = j$ die Vorgänger- und Endpositionen des Blocks. Sei $L = \{\text{vor}(B) \mid B \in \mathcal{B}\} \cup \{\text{end}(B) \mid B \in \mathcal{B}\}$ die Menge aller Vorgänger- und Endpositionen. Wir fügen für jede solcher Positionen $a \in L$ einen "Hilfsknoten" v_a zu V hinzu. Ist a die Vorgängerposition von Block B, dann fügen wir eine stille Kante von v_a zum ersten Knoten in B hinzu. Ist a die Endposition von Block B, dann fügen wir anstelle aller derjenigen Kanten, die den letzten Knoten aus B verlassen, *eine* Kante mit derselben Beschriftung zum Knoten v_a hinzu. Aus jedem Knoten v_a gibt es noch eine stille Kante in den nächsten Hilfsknoten, also in v_b mit $b = \min\{c \in L \mid c > a\}$. Gibt es keinen nächsten Hilfsknoten fügen wir eine Kante in den Knoten Ende hinzu. Vom Knoten Start geht eine Kante in den ersten Hilfsknoten v_a mit $a = \min L$. Das daraus entstehende Netzwerk heisse $G' = (V', E')$. Für das Beispiel siehe Abbildung 1.14.

Die Anzahl der Kanten $|E'|$ im Netzwerk G' ist jetzt $O(c)$, da für jeden Block höchstens 3 neue Kanten hinzukommen und $3b = O(c)$ ist.

Satz: Das Spliced-Alignment-Problem für eine Zielsequenz T und Blöcken $B_1, \ldots, B_b$ kann in Zeit $O(|T| \cdot c)$ gelöst werden. Dabei ist $c = \sum_{i=1}^{b}$ Länge(B_i).

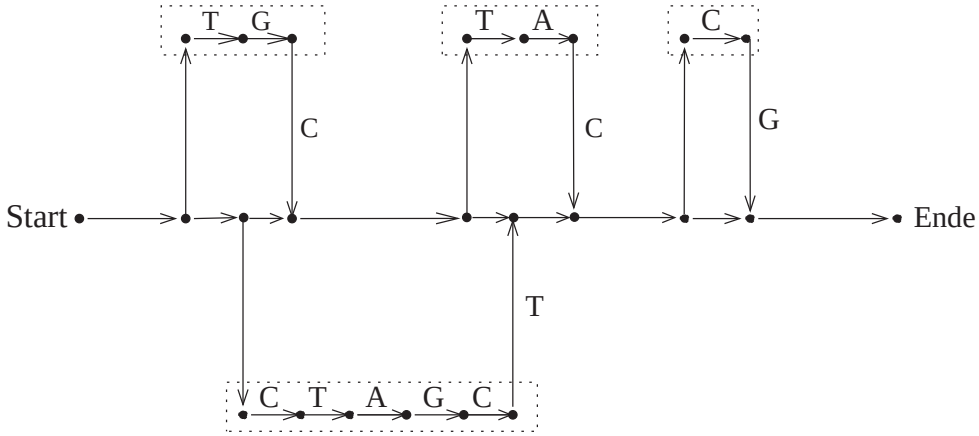


Abbildung 1.14: Optimiertes Netzwerk G' für Spliced Alignment. Die asymptotische Anzahl von Kanten ist reduziert.

1.5 Genvorhersage durch Finden bester Exonketten

1.5.1 Das eindimensionale Chaining-Problem

Sei $S = s_1 s_2 \cdots s_n$ ein Wort und sei $\mathcal{B} = \{B_1, \ldots, B_r\}$ eine Menge von Teilwörtern (Blöcken) von S, denen jeweils eine Zahl, ihr *Wert* zugeordnet ist. Sei $v(j)$ der Wert von Block B_j. Eine Kette $\Gamma = (B_1, \ldots, B_n)$ ist wie im vorigen Kapitel eine sortierte Folge von nicht-überlappenden Blöcken aus $\mathcal{B}$. Der Wert einer Kette ist die *Summe* der Werte der Blöcke der Kette. Das eindimensionale **Chaining-Problem** ist das Problem eine Kette mit maximalem Wert zu finden. Es ist sehr verwandt mit einem Problem, aus einer Menge von bewerteten Exonkandidaten, eine möglichst gute Genstruktur zusammenzusetzten. Auf dieses Problem geht der nächste Abschnitt ein.

Ein Algorithmus zur Lösung des Chaining-Problems:
Für einen Block $B_j = s_a \cdots s_b$ sei $\ell(j) := a$ die Anfangsposition von Block B_j und sei $r(j) := b$ die Endposition. Sei $P = (P[1], P[2], \ldots, P[m])$ ein aufsteigend sortiertes Array mit allen Anfangs- und Endpositionen der Blöcke in $\mathcal{B}$. P hat also $m \leq 2r$ Elemente. Der Algorithmus berechnet für jeden Block B_j einen Wert V_j. Dieser ist der Wert der besten Kette, die mit Block B_j endet.

Algorithmus 2 (Chaining-Algorithmus)

0 Berechne das Array P.
1 M $\leftarrow$ 0
2 Für $i = 1..m$
3 Für jedes j mit $\ell(j) = P[i]$ setze $V(j) \leftarrow v(j) + M$
4 Setze $M \leftarrow \max\{M, \max_{\{j|r(j)=P[i]\}} V(j)\}$
5 Gebe M als Wert der besten Kette aus.

Abbildung 1.15 zeigt ein Beispiel für das Chaining Problem. Die Kette selbst, kann gefunden werden, wenn für jeden Block j, zusätzlich zu $V(j)$ der Vorgängerblock in der besten Kette die mit j endet, gespeichert wird.

Beweis der Korrektheit: Wir machen eine Induktion über die Schleifendurchläufe in Zeile 2. Die Induktionsbehauptungen sind:

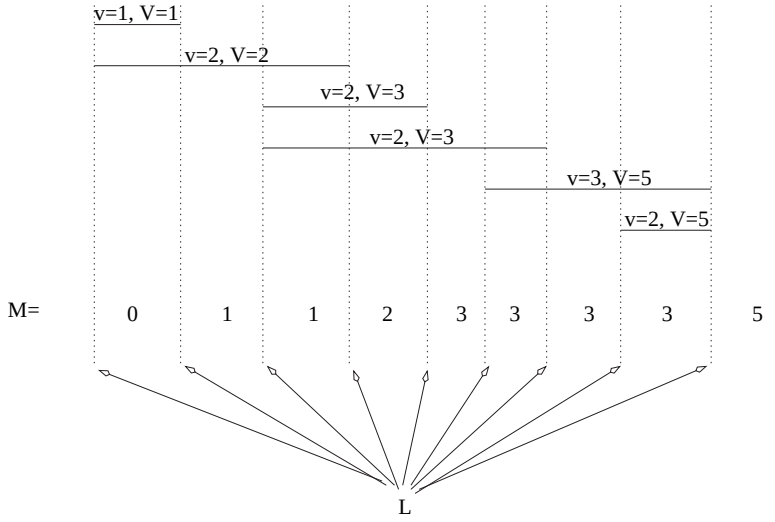


Abbildung 1.15: Beispiel für den Verlauf des Chaining-Algorithmus. Die Blöcke haben den Wert v. V ist der Wert der besten Kette, die mit dem Block endet. Die Werte von M werden jeweils an den Endpositionen von Blöcken geändert.

1. Nach jedem Schleifendurchlauf ist $V(j)$ der Wert der besten Kette die mit Block j endet, für alle j mit $\ell(j) \leq P[i]$.

2. Nach jedem Schleifendurchlauf ist M der Wert der besten Kette, die bis höchstens zur Position $P[i]$ geht.

Induktionsanfang: trivial.
Induktionsschritt: $i-1 \rightarrow i$. Wir zeigen zunächst 1. Nach Induktionsvoraussetzung ist M, wenn Zeile 3 für ein i ausgeführt wird, der Wert der besten Kette, die bis höchstens zur Position $P[i-1] < P[i]$ geht. Der Wert der besten Kette, die mit einem Block j endet, der in Position $P[i]$ anfängt ist also $M + v(j)$.
Jetzt zeigen wir 2. Sei $\Gamma = B_{j_1} \ldots B_{j_d}$ die beste Kette, die bis höchstens zur Position $P[i]$ geht und sei V ihr Wert. Wenn $r(j_d) = P[i]$ ist, endet die beste Kette mit einem Block $j = j_d$ in Position $P[i]$. Da der Block mindestens Länge 1 hat, ist $\ell(j) < P[i]$ und nach Induktionsvoraussetzung ist $V(j)$ der Wert der besten Kette die mit Block j endet, die gleichzeitig sogar die beste Kette ist, die bis höchstens zur Position $P[i]$ geht. Also $V(j) = V$. In Zeile 4 wird M also auf $V(j)) = V$ gesetzt, weil kein Element, aus dem das Maximum gebildet wird, größer sein kann als V. Wenn $r(j_d) < P[i]$ ist, war Γ bereits die beste Kette, die bis höchstens zur Position $P[i-1]$ geht. Nach Induktionsvoraussetzung ist dieser Wert der Wert von M vor dem i-ten Schleifendurchlauf. M wird in Zeile 3 nicht geändert, weil nach Annahme keines der $V(j)$ aus der Maximumsbildung größer als das M vor dem Schleifendurchlauf sein kann.

Laufzeit: Effizienterweise speichert man gleich beim Anlegen der Liste L für jeden Eintrag i auch die Listen der Blöcke, die dort beginnen oder enden. Da diese Listen zusammen alle Blöcke genau zweimal enthalten, ist der Gesamtaufwand für die Zeilen 0 bis 4 $O(r)$. Die Laufzeit des Algorithmus wird dominiert von der Laufzeit, die für das Sortieren von L benötigt wird: $O(r \log r)$.

Satz: Das eindimensionale Chaining-Problem kann mit obigem Algorithmus in Zeit $O(r \log r)$ gelöst werden.

1.5.2 Ein Modell für kodierende Sequenzen

Bei verschiedenen Genvorhersage-Methoden stellt sich häufig als Teilproblem die Frage, ob ein gegebenes Stück DNA-Sequenz σ kodierend ist oder nicht. Kodierend sind die Teile der Sequenz, die später übersetzt werden, also die Exons. Oder allgemeiner stellt sich eine Frage wie: Wie wahrscheinlich ist es, dass σ kodierend ist? Dies wird in der Praxis häufig dadurch gelöst, dass man der Sequenz σ eine Zahl (eine Wahrscheinlichkeit oder einen Score) zuordnet, die umso größer ist, je 'wahrscheinlicher' es ist, dass σ kodierend ist. Dabei basiert die Zuordnung der Zahl zur Sequenz auf einem Model für kodierende Sequenzen, das versucht, das zu modellieren, was für kodierende Sequenzen typisch ist. Die ersten drei Spalten folgender Tabelle zeigen, wie häufig jedes der 4 Nukleotide an jeder der drei möglichen Leserahmenpositionen in kodierenden Sequenzen vorkommt.

	kodierende Sequenzen				nichtkod. Seq.
	$f=0$	$f=1$	$f=2$	alle f	
a	0.248	0.291	0.146	0.229	0.26
c	0.264	0.243	0.351	0.286	0.24
g	0.321	0.201	0.312	0.278	0.24
t	0.166	0.265	0.190	0.207	0.26

Tabelle 1.2: Nukleotidhäufigkeiten. Erste drei Spalten: Die Leserahmenpositionen $f=0, f=1, f=2$: erste, zweite oder dritte Position im Kodon. Vierte Spalte: Häufigkeit in kodierenden Sequenzen ohne Berücksichtigung des Leserahmens. Fünfte Spalte: Häufigkeit in Introns und der intergenischen Region. (Daten von menschlichen Genen.)

Es fällt auf, dass der Unterschied in der Basenzusammensetzung zwischen kodierenden und nicht-kodierenden Sequenzen geringer ist als der Unterschied in den verschiedenen Leserahmenpositionen innerhalb der kodierenden Sequenzen. Ein gutes Modell sollte dies also auch berücksichtigen. Zu den erfolgreichsten Modellen für kodierende Sequenzen gehören sogenannte 3-periodische Markow-Ketten höherer (4. oder 5.) Ordnung k. Sie berücksichtigen sowohl, dass die Häufigkeit der Nukleotide vom Leserahmen abhängt als auch die Häufigkeit von Mustern der Länge höchstens $k+1$ (z.B. bei $k=2$: Wie häufig kommt das Dinukleotid 'at' vor? wie häufig kommt das Kodon 'cga' vor?) In einer 3-periodischen Markow-Kette der Ordnung k wird der Sequenz $\sigma = \sigma_1 \dots \sigma_n$ folgende Wahrscheinlichkeit zugeordnet, wenn man annimmt, dass die erste Position von σ in Leserahmenposition $f \in \{0,1,2\}$ ist.

$$P_f(\sigma_1 \dots \sigma_n) = p_f(\sigma_1, \dots, \sigma_k) \cdot \prod_{i=k+1}^{n} p_{f(i)}(\sigma_i | \sigma_{i-k}, \sigma_{i-k+1}, \dots, \sigma_{i-1}).$$

Dabei ist

- $p_f(\sigma_1, \dots, \sigma_k)$ irgendeine Startwahrscheinlichkeit für die ersten k Zeichen von σ.
- $f(i) \in \{0,1,2\}$ so, dass $f - 1 + i \equiv f(i) \mod 3$ ist die Leserahmenposition an der i-ten Stelle
- $p_r(x|y_1, \dots, y_k)$ ist die Wahrscheinlichkeit, das Nukleotid x an der Leserahmenposition $r \in \{0,1,2\}$ zu beobachten, wenn der Nukleotidstring $y_1, \dots, y_k$ in Leserichtung unmittelbar vorangegangen ist.

Beispiel: Wir suchen die Wahrscheinlichkeit des Strings 'attctgc', wenn folgender Leserahmen unterstellt ist: 'a|ttc|tgc'. Also $f = 2$. Es ist

$$P_2(\text{attctgc}) = p_2(\text{at}) \cdot p_1(\text{t|at}) \cdot p_2(\text{c|tt}) \cdot p_0(\text{t|tc}) \cdot p_1(\text{g|ct}) \cdot p_2(\text{c|tg}).$$

Die Wahrscheinlichkeiten $p_r(x|y_1, \ldots, y_k)$ und $p_f(\sigma_1, \ldots, \sigma_k)$ müssen dafür vorher geeignet geschätzt werden unter Zuhilfenahme von bekannten kodierenden Sequenzen.

1.5.3 Exons verketten

Ein hierarchischer Ansatz zur eukaryotischen Genvorhersage, wie er etwa im Programm GENEID [PEG00] Anwendung findet, ist folgender.

Im ersten Schritt werden *Signale* gesucht. Mögliche Startpositionen sind die Acceptor Splice Site und der Translationsstart, mögliche Endpositionen sind die Donor Splice Site und das Translationsende. Alle möglichen Start- und Endpositionen von Exons werden bewertet. Ihr Score ist eine reelle Zahl, die umso größer ist, je wahrscheinlicher (in einem informellen Sinne) es ist, dass die Stelle tatsächlich ein Translationsstart bzw. eine Acceptor Splice Site, ... ist.

Im zweiten Schritt werden mit Hilfe dieser Signalpositionen und ihrer Bewertungen Exonkandidaten gefunden und bewertet. Ein Exonkandidat ist ein Paar Start-/Endposition zusammen mit der Information, in welcher Leserahmenposition das Exon beginnt. Hierbei werden nur solche Exonkandidaten betrachtet, die kein Stopkodon in ihrem Leserahmen enthalten. Auch hier wird die Exonsequenz selbst bewertet, danach wie wahrscheinlich es ist, dass sie tatsächlich kodierend ist (mit obiger 3-periodischer Markowkette). Der Score eines Exons ist die Summe der Scores der beiden Signale am Anfang und Ende und dem Score der Exonsequenz. Auf Seite 31 ist so eine Menge von Exonkandidaten für eine Beispielsequenz.

Im dritten Schritt werden aus den Exonkandidaten Gene zusammengesetzt. Dieser Schritt wird *exon assembly* genannt. Dabei wird berücksichtigt, dass jedes Gen (auf dem Vorwärtsstrang) mit einem Translationsstart beginnt, mit einem Translationsende aufhört, und bei jedem Intron das Exon rechts davon mit der Leserahmenposition anfängt, die im Exon links vom Intron als nächstes gekommen wäre, wenn es nicht durch das Intron unterbrochen würde. Es wird eine in diesem Sinne konsistente Kette von Exons gesucht, deren Werte maximale Summe haben.

Das charakteristische dieses Ansatzes ist, dass jeder Schritt aus dem vorigen Schritt nur die Bewertungen braucht. Wenn etwa die die Signalpositionen oder Exonkandidaten zusätzlich gefiltert werden, beschleunigt das die nachfolgenden Schritte. In diesem Ansatz bleibt jedoch unberücksichtigt, ob die vorhergesagten *Introns* typisch sind. Für sie gibt es kein Inhaltsmodell und ihre Länge (wie die Länge der Exons) wird nicht berücksichtigt.

Algorithmisch interessant ist nur der dritte Schritt. Wir formulieren das Exon-Assembly-Problem zunächst wieder abstrakt.

Sei $\mathcal{B} = \{B_1, \ldots, B_r\}$ eine Menge von Blöcken in Wort S. Sei T eine endliche Menge, deren Elemente wir *Typen* nennen. Und seien zu jedem Block B_j neben dem Wert $v(j)$ auch der *Vorgänger- und Nachfolgertyp* $\text{vor}(j) \in T$ bzw. $\text{nach}(j) \in T$ des j-ten Blocks definiert. Der Wert einer Kette Γ ist wieder die Summe der Werte der Blöcke. Das **Exon-Assembly-Problem** ist das Problem, eine Kette maximalen Werts zu finden unter den Ketten $\Gamma = (B_{j_1}, \ldots, B_{j_k})$, die folgende Nebenbedingung erfüllen:

$$\text{nach}(j_i) = \text{vor}(j_{i+1}) \qquad (i = 1 \ldots, k-1)$$

In Worten: Für jeden Block muß der Vorgängertyp gleich dem Nachfolgertyp des Vorgängerblocks sein. Wir nennen Ketten, die diese Bedingung erfüllen *konsistent*. Dieses Problem kann mit einer Variante des oben angegebenen Chaining-Algorithmus gelöst werden. Anstatt nur eine Variable

M zu halten, halten wir für jeden Typ $t \in T$ eine Variable M_t. Im Verlauf des Algorithmus ist dann M_t der Wert der besten Kette, die bis höchstens zur gerade aktuellen Position in S geht, *und* die mit einem Block des Nachfolgertyps t endet. Der Algorithmus, der das Exon-Assembly-Problem löst, ist der folgende: Sei P wie oben die sortierte Liste der Anfangs- und Endpositionen der Blöcke und sei m die Anzahl der Elemente von P.

Algorithmus 3 (Gene-Assembly-Algorithmus)

0 Berechne die Liste P.
1 Setze $M_t \leftarrow 0$ für alle $t \in T$
2 Für $i = 1..m$
3 Für jedes j mit $\ell(j) = P[i]$ setze $V(j) \leftarrow v(j) + M_{\text{vor}(j)}$
4 Setze $M_t \leftarrow \max\{M_t, \max_{\{j|r(j)=P[i],\text{nach}(j)=t\}} V(j)\}$ für alle $t \in T$.
5 Gebe $\max_{t\in T} M_t$ als Wert der besten Kette aus.

Der Korrektheitsbeweis verläuft ganz analog. Die Laufzeit ist auch dieselbe wie beim Chaining-Algorithmus, $O(r \log r)$, wenn man annimmt, dass die Zahl der Typen eine Konstante ist. Im Problem, aus einer Menge von Exonkandidaten, eine Folge von biologisch konsistenten, nicht-überlappenden Genen auf beiden Strängen zu machen, muß die Menge der Typen T geeignet gewählt werden damit dieser Algorithmus angewendet werden kann (Übung). Im oben beschriebenen, hierarchischen Vorgehen, ist das Sortieren der Anfangs- und Endpositionen möglich durch speichern der Anfangs- und Endpositionen in einem Array der Länge $|S|$. Die Laufzeit ist dann $O(|S|+r)$, der Anteil $|S|$ fällt sowieso beim Einlesen der Eingabe an. Beachte, dass es bei diesem Problem – im Gegensatz zum im vorigen Abschnitt beschriebenen Chaining-Problem – sinnvoll sein kann (und in der Praxis in GENEID auch passiert), dass der Wert von Exons negativ ist.

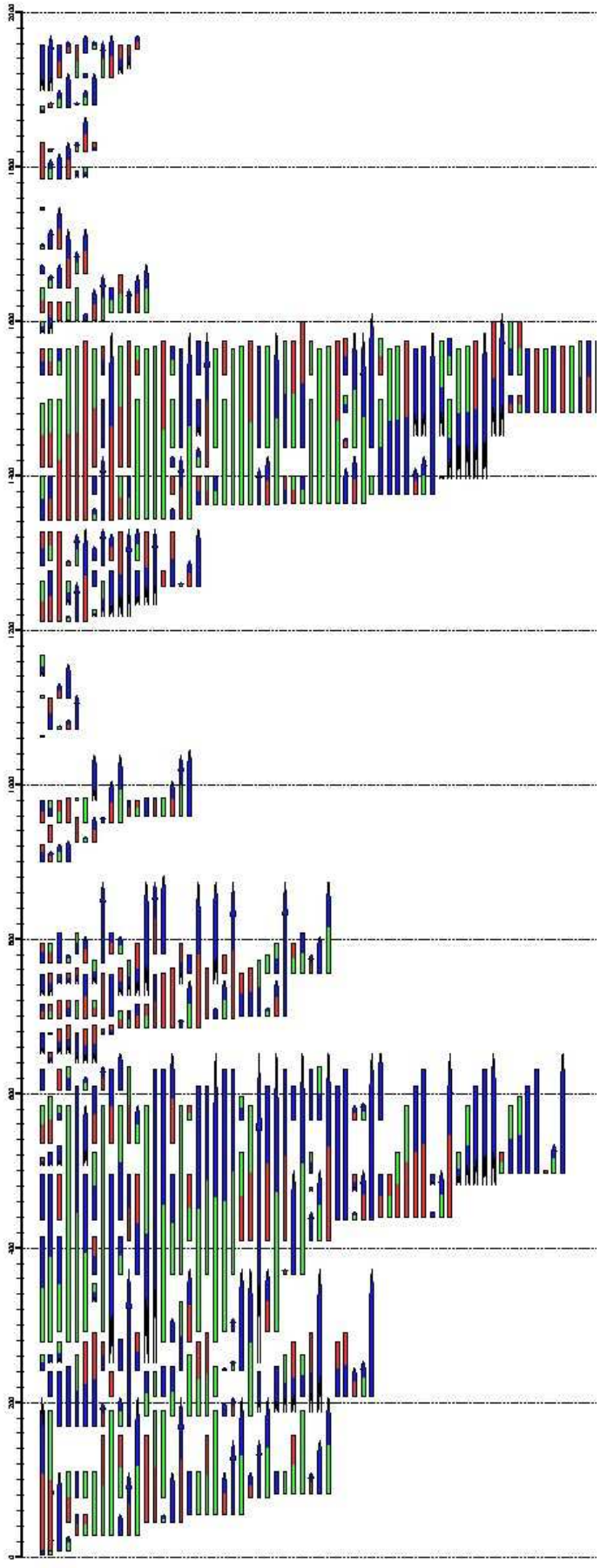


Abbildung 1.16: Menge von Exonkandidaten auf dem Vorwärtsstrang zu einer DNA-Eingabesequenz S der Länge 2000. Die Farben (grün, blau, rot) des linken und rechten Teils eines Exons spezifizieren die Leserahmenposition, mit dem das Exon anfängt bzw. aufhört. Eine Pfeilspitze bedeutet das Translationsende (Stopkodon), ein Pfeilende den Translationsstart (Startkodon). Diese Abbildung stammt von der Seite http://monstre1.imim.es/courses/genefinding/T5/.

Kapitel 4

Sequenzierung, Assemblierung und Mapping

4.1 Eine kurze biologische Einführung in DNA-Sequenzierung und Assemblierung

Bei der *Chain-Termination*-Methode von Sanger und anderen (1977) wird die Sequenz eines Stückes einzelsträngiger DNA durch Bestimmen der relativen Längen von komplementären Polynukleotid-Ketten, die mit bestimmten Nukleotiden enden, ermittelt. Dies heißt *Base-Calling*. Die Technik dafür – Polyacrylamid-Gelelektrophorese – erlaubt, je nach Konzentration des Gels, die Bestimmung von Sequenzen einer Länge zwischen 10 und 1500 bp. Dabei nimmt gegen Ende der Sequenz die Fehlerrate zu und typischerweise beschränkt man sich darauf, Sequenzen der Länge 500-750 bp zu lesen. Diese in einem Schritt gelesenen Sequenzen heißen *Reads*. Die *Zielsequenz* eines Genomprojekts ist wesentlich Länger als ein Read, so dass sie aus vielen Reads zusammengesetzt, *assembliert*, werden muß. Bei der Sequenzierung und Assemblierung gibt es verschiedene Fehlerquellen und Schwierigkeiten: Es treten Fehler beim Base-Calling auf (Substitutionen, Insertionen und Deletionen, etwa in der Größenordnung von einem Fehler pro 200 bp). Ein *Repeat* ist eine Sequenz, die an zwei oder mehr Stellen als Teilstring in der Zielsequenz vorkommt. Die Intervalle in der Zielsequenz, an denen ein Repeat auftritt, können sich auch überlappen. Repeats, die so lang sind, dass sie nicht zumindest von zwei Reads ganz überdeckt werden können, können verhindern, dass die Assemblierung eindeutig möglich ist (siehe Abbildungen 4.1, 4.2 und 4.3). Beim Bakterium *Streptococcus pneumoniae* etwa gibt es eine 24bp-Sequenz, die auf einer Länge von etwa 14000 bp wiederholt wird (Tandem Repeat). Eine dritte Fehlerquelle ist das notwendige Klonieren, also das Einbringen einer fremden DNA-Sequenz in eine Gastgeber-Zelle (host cell, häufig *E. coli*), um sie in großer Anzahl zu kopieren. Dabei kann es passieren, dass die fremde DNA durch DNA der Gastgeber-Zelle verunreinigt wird, oder dass Teile der fremden DNA-Sequenz fehlen.

Shotgun-Sequenzierung

Eine wichtige Methode zur Sequenzierung ist die sogenannte *Shotgut*-Methode (shotgun = Schrotflinte). Hierbei werden viele Kopien einer langen Sequenz s (mehrere Megabasen: z.B. ein ganzes Chromosom, oder das ganze Genom eines Prokaryoten) zuerst durch hochfrequente Schallwellen oder einen Hochdruck-Luftstrom in zufällige Fragmente zerbrochen. Von den Fragmenten zufälliger Länge und Position wird mittels Elektrophorese ein Teil der Fragmente ausgesucht, die in einem bestimmten Längenbereich liegen (z.B. zwischen 1.6 kb und 2 kb beim Genomprojekts des Bakteriums *Haemophilius influenzae*, beim Celera-Sequenzierprojekt des Menschen etwa zwi-

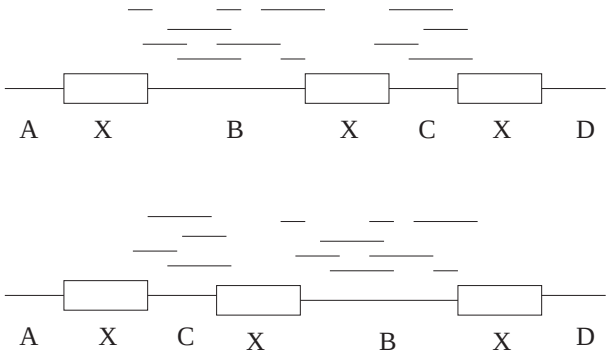


Abbildung 4.1: Die Boxen bezeichnen ein Repeat in der DNA-Sequenz. Die Striche darüber entsprechen den Reads. Nur die Reads über den kritischen Abschnitten sind eingezeichnet. Ein Repeat der Form XXX führt zu mehrdeutigen Assemblies: Abschnitte B und C sind vertauscht.(aus [Set97])

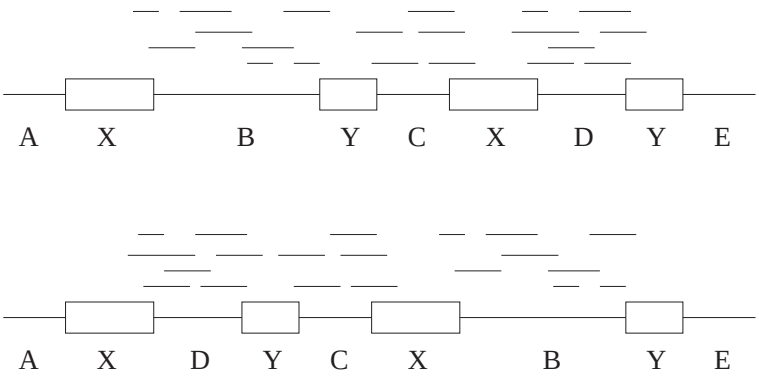


Abbildung 4.2: Ein Repeat der Form $XYXY$ führt zu mehrdeutigen Assemblies. Abschnitte B und D sind vertauscht.

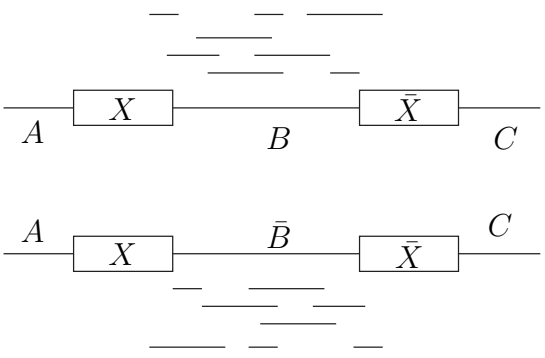


Abbildung 4.3: $\bar{X}$ bezeichne das reverse Komplement von X. Ein Repeat der Form $X\bar{X}$ führt zu mehrdeutigen Assemblies mit entgegengesetzten Leserichtungen der Sequenz B zwischen den Repeats.

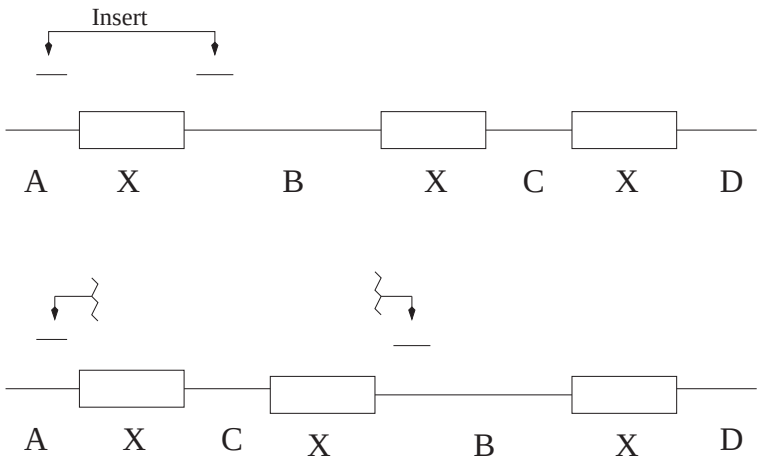


Abbildung 4.4: Das Paired-End-Sequencing kann manchmal falsche Assemblies verhindern. Oben haben die beiden Enden des Inserts den richtigen Abstand, unten nicht. Das Assembly unten muß falsch sein.

schen 2 kb und 50kb). Diese Sequenzen heißen *Inserts*, weil sie sie zwecks Vervielfältigung in einen sogenannten Klonierungsvektor eingefügt werden. Das entstandene Zwischenresultat ist eine *Bibliothek* von Inserts, von denen man jeweils die ungefähre Länge kennt. Beim sogenannten *Paired-End-Sequencing* wird von jedem der beiden Enden des Inserts ein Read gelesen (500-750 bp). Dann erhält man eine Menge von Reads. Die Leserichtung eines Reads r ist unbekannt, so dass entweder r oder das reverse Komplement von r ein Teilstring von s ist. Zusätzlich weiss man zu manchen Paaren von Reads ihren ungefähren Abstand, weil sie die Endstücke desselben Inserts sind. Von diesen Paaren weiss man auch, dass sie entgegengesetzte Leserichtungen haben. Der Vorteil davon, dass die Inserts länger sind als die Reads, ist, dass es damit möglich wird, Repeats zu "überbrücken", die länger sind als ein Read. Z.B. haben die meisten Repeats der Fruchtfliege *Drosophila melanogaster* eine Länge von höchstens 8 kb, weswegen man bei ihrer Shotgun-Sequenzierung unter anderem eine Bibliothek mit Inserts der mittleren Länge 10 kb gewählt hat (siehe Abbildung 4.4).

Für eine bestimmte Position in s ist die *Coverage* an dieser Position die Anzahl der Reads, die diese Position enthielten. Die *mittlere Coverage* ist also die Summe der Längen aller Reads geteilt durch die Länge von s. Damit der größte Teil von s durch Reads überdeckt wird und damit wenige Lücken ganz ohne Coverage entstehen, muss die mittlere Coverage mindestens zwischen 6.5 und 8 sein (siehe Abschnitt 4.2).

Ein *Contig* (contiguous = zusammenhängend) ist eine aus überlappenden Sequenzen lückenlos zusammengesetze Sequenz. Das erste Ergebnis des Assemblierens ist eine Menge von *Scaffolds* (Gerüste, siehe Abbildung 4.5). Ein Scaffold ist eine Menge von Contigs, die durch überbrückbare Sequenzlücken getrennt sind und zwar solche Lücken, die zwischen den beiden Reads an den Enden eines Inserts liegen. Diese können geschlossen werden durch Sequenzieren dieser Inserts. Die Contigs eines Scaffolds sind geordnet und die relative Orientierung (Leserichtung) zueinander ist bekannt. Die Scaffolds ihrerseits sind getrennt durch *physical Gaps*, die nicht ohne weiteres geschlossen werden können, weil die Sequenzen dieser Lücken nicht in der Bibliothek enthalten sind. Dennoch können die ungefähren Positionen eines Scaffolds auf einer sogenannten *Genkarte* mit Markern bestimmt werden. *Sequence Tagged Sites* (STS) sind eindeutige Teilsequenzen des Genoms. STSs, deren Position auf einer Genkarte bekannt sind, dienen als Marker. Beim Menschen gibt es im Durchschnitt etwa alle 100kb eine STS, deren Position bekannt ist.

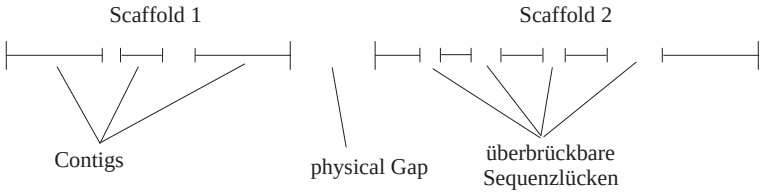


Abbildung 4.5: Scaffolds

Das Human-Genomprojekt

Das öffentlich finanzierte International Human Genome Sequencing Consortium (IHGSC) wählte zur Sequenzierung der etwa 3 Milliarden Basen des menschlichen Genoms den sogenannten *Clone-Contig*-Zugang. Dabei wird das Genom in Fragmente einer Länge bis zu 1.5 Mb zerbrochen und diese werden in Kloniervektoren geklont. Diese Fragmente heissen *Clones*. Beim Human-Genomprojekt waren es 300000 Clones. Überlappende Clones wurden gefunden, z.B. mittels STSs. Diese Clones wurden dann *einzeln* mit der Shotgun-Methode sequenziert.

Beim Celera-Sequenzierprojekt wurde der sogenannte *Whole-Genome-Shotgun*-Zugang gewählt [V$^+$01]. Dabei wird die oben beschriebene Shotgun-Methode direkt auf das ganze Genom angewandt und nicht wie beim IHGSC auf kürzere Clones. 14,8 Milliarden Basen aus 27271853 Reads wurden gelesen. Das entspricht einer mittleren Coverage von 5,1. Es wurde hierzu die DNA von 5 Menschen (3 Frauen, 2 Männer; 1 Afrikaner, 1 Asiat, 1 Mexikaner, 2 Kaukasier; 1 Craig Venter, 4 anonyme Personen) verwendet. Bei jeweils 2 Menschen hat die DNA mindestens 99,9% Sequenzübereinstimmung. Zusätzlich wurden bei Celera Daten aus dem öffentlich finanzierten Projekt benutzt.

Es ist allgemein anerkannt, dass mit dem Clone-Contig-Zugang eine höhere Genauigkeit erreicht werden kann als mit dem Whole-Genome-Shotgut-Zugang. Die Ergebnisse der beiden Sequenzierprojekte wurden im Jahr 2001 veröffentlicht. Es sind sogenannte *draft sequences* (draft = Skizze). Sie enthalten Lücken und Fehler. *Finishing* (to finish = fertig stellen) heißt es, wenn aus einer draft sequence eine komplette Sequenz eines Genoms, Chromosoms oder Klons erstellt wird, mit einer Genauigkeit von mindestens 99.99%. Aber man spricht auch manchmal von 'finished', wenn die Sequenz noch wenige Lücken enthält. Finishing ist aufwendig und komplexe Bereiche müssen von Menschen fertig gestellt werden.

Bisher (Dezember 2004) wurden zwölf menschliche Chromosomen, also die Hälfte der Chromosomen, fertig gestellt: 22, 21, 20, 14, Y, 7, 6, 13, 19, 10, 9, 5. Das Chromosom 5 wurde im September 2004 fertig gestellt. Es ist mit etwa 181 Megabasen das längste fertiggestellte Chromosom, wobei aber noch 4 Lücken nicht geschlossen werden konnten.

4.2 Benötigte mittlere Coverage

Vor einem Shotgun-Sequenzierprojekt muß abgeschätzt werden, wieviele Reads nötig sind, um die Sequenz mit einer beschränken Anzahl von Lücken assemblieren zu können. Hierzu bedienen wir uns folgenden Modells. Sei g die Länge der Zielsequenz, r die Anzahl der Reads. Wir gehen vereinfachend davon aus, dass alle Reads dieselbe Länge ℓ haben. Sei

$$c := \frac{r\ell}{g}$$

die mittlere Coverage. Die entsprechenden Zahlen aus dem Celera-Projekt des Humangenoms sind $g \approx 2.9 \cdot 10^9, r \approx 27 \cdot 10^6, \ell \approx 550, c \approx 5.1$. Dabei kann r und damit c gewählt werden, g und

ℓ können als gegeben betrachtet werden. Auf jeden Fall gilt $\ell << g$, was unten vereinfachende Näherungen erlaubt. Seien $X_1, X_2, \ldots, X_\ell$ die Anfangspositionen der Reads auf der Zielsequenz, also $1 \leq X_i \leq g - \ell + 1$ für alle $1 \leq i \leq \ell$. Wir gehen davon aus, dass die X_i unabhängig sind und gleichverteilt auf der Menge $\{1, \ldots, g - \ell + 1\}$ der möglichen Anfangspositionen.

Welcher Anteil der Zielsequenz wird im Mittel durch die Reads überdeckt?

Wir fragen uns, was der Erwartungswert der Anzahl U der Positionen in der Zielsequenz, die eine Coverage von 0 haben, ist. Diese Positionen mit Coverage 0 wurden nicht gesampled.

Sei für $j = 1, \ldots, g$, C_j die Coverage an Position j, also

$$C_j = |\{i \,|\, j - \ell + 1 \leq X_i \leq j\}|$$

Wenn $\ell \leq j \leq g - \ell + 1$, also j nicht nahe am Rand der Zielsequenz liegt, dann ist C_j binomialverteilt mit Parametern r und $\ell/(g - \ell + 1)$, weil jedes der r Reads an ℓ Anfangspositionen von $g - \ell + 1$ möglichen Anfangspositionen Position j abdeckt. Hier benutzen wir die geforderte Unabhängigkeit der X_i. Wie erwähnt ist ℓ in der Praxis viel kleiner als g, so dass wir 1. die ersten ℓ und letzten ℓ Positionen von g in der Rechnung vernachlässigen und 2. $\ell/(g - \ell + 1) \approx \ell/g$ annähern. Die Binomialverteilung kann sehr gut durch die Poissonverteilung angenähert werden, wenn die Anzahl der Versuche (hier r) groß ist und die Erfolgswahrscheinlichkeit (hier ℓ/g) klein. Der Erwartungswert von C_j ist $r\ell/g = c$, was klar ist, weil c ja die mittlere Coverage ist. Also,

$$P(C_j = k) \approx e^{-c}\frac{c^k}{k!}.$$

(Für den Fehler, den man durch die Annäherung der Binomialverteilung mit der Poissonverteilung macht gilt hier: $\sum_{k=0}^{\infty} |P(C_j = k) - e^{-c}\frac{c^k}{k!}| \leq 2r(\ell/g)^2 = 2c\ell/g$; er ist also hier sehr klein.) Die Wahrscheinlichkeit, dass eine Position *nicht* gesampled wird, also Coverage $k = 0$ hat, ist also e^{-c}. Die Anzahl U der Positionen mit Coverage 0 hat also den Erwartungswert

$$\mathrm{E}[U] = \mathrm{E}[\sum_{j=1}^{g} \mathbb{1}_{\{C_j=0\}}] \approx ge^{-c}.$$

Der erwartete Anteil, der durch die Reads überdeckt wird, ist also $1 - e^{-c}$, was mit obigen Zahlen 99.4% ergibt. Eine mittlere Coverage von 8 bedeutet z.B. unter obigen Annahmen, dass im Mittel $1 - e^{-8} \approx 99.97\%$ der Positionen überdeckt werden.

Wieviele Contigs entstehen bei der Assemblierung?

Wir wollen die Anzahl der Contigs schätzen. Diese ist eins größer als die Anzahl der Lücken die selbst ohne oben beschriebene Probleme der Sequenzierung (Base-Calling Fehler, Repeats, Verunreinigungen) bei der Assemblierung entstehen. Für typische Anwendungen ist die Anzahl der Contigs groß und es kommt uns im Sinne einer vereinfachten Darstellung nicht auf wenige Contigs, bzw. Lücken an. Sei t eine Mindestschranke für die Länge des Überlapps zweier Reads, so dass man annehmen kann, dass die Reads tatsächlich in der Zielsequenz benachbart sind und der Überlapp nicht nur zufällig ist. Sei $\theta := t/\ell$. Bei dem Celera-Projekt war $t = 40$, also $\theta \approx 7\%$ (allerdings waren Fehler im Überlapp zugelassen). Wir sagen, wir haben eine Assembly-Lücke zwischen zwei benachbarten Reads, wenn sie nicht mindestens um t Basen überlappen, also, wenn entweder eine Lücke im normalen Wortsinn zwischen ihnen ist oder, wenn sie zwar überlappen, aber zu wenig. Wir bezeichnen Read i, der an Position X_i beginnt, als *Lückenread*,

wenn er unmittelbar vor einer Assembly-Lücke ist. Das ist, abgesehen vom letzten Read überhaupt, gleichbedeutend damit, dass kein Read zwischen Positionen $X_i + 1$ und $X_i + \ell - t - 1$ beginnt. Also

$$
\begin{aligned}
P(\text{Read } i \text{ Lückenread}) &= \prod_{\substack{j=1 \\ j \neq i}}^{r} P(X_j \notin [X_i + 1, X_i + \ell - t]) && (4.1)\\
&\approx \left(\frac{g - (\ell - t)}{g}\right)^r && (4.2)\\
&= \left(\left(1 + \frac{-(1-\theta)}{\frac{g}{\ell}}\right)^{\frac{g}{\ell}}\right)^c && (4.3)\\
&\approx e^{-(1-\theta)\cdot c} && (4.4)
\end{aligned}
$$

In Zeile (4.1) haben wir benutzt, dass die Positionen der Reads unabhängig sind. In Zeile (4.2) haben wir die Näherung gemacht, dass es für jeden Read ungefähr g mögliche Anfangspositionen gibt, von denen alle, bis auf $\ell - t$ günstig sind für das Ereignis aus (4.1). Außerdem haben im Exponenten die Näherung $r \approx r - 1$ gemacht. Zeile (4.3) beruht auf elementaren Umformungen und den Definitionen von θ und c. Zeile (4.4) benutzt den Grenzwert $\lim_{x\to\infty}(1+z/x)^x = e^z$, was eine Näherung erlaubt, weil g/ℓ groß ist. Wenn wir wieder die (seltenen) Fälle vernachlässigen, wo vor einer Assembly-Lücke mehrere Lückenreads auf dieselbe Position fallen, erhalten wir eine erwartete Anzahl von

$$r \cdot e^{-(1-\theta)\cdot c}$$

Assembly-Lücken. Wie Abbildung 4.6 zeigt, steigt die Anzahl der Contigs zunächst mit der mittleren Coverage, weil bei kleiner mittleren Coverage Überlapps selten sind, und fällt dann schnell ab.

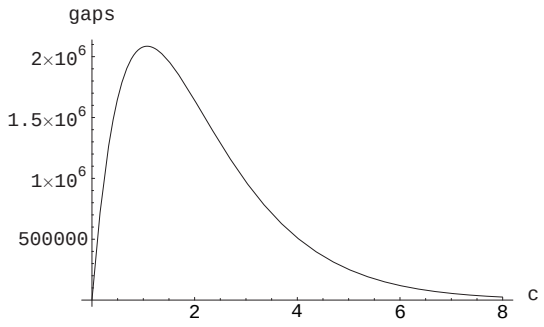


Abbildung 4.6: Anzahl der geschätzten Assembly-Lücken (gaps) in Abhängigkeit von der mittleren Coverage c bei einer Genomlänge von $g = 2,9 \cdot 10^9$ einer Readlänge von $\ell = 550$ und einem Mindestüberlapp von $t = 40$.

Die Formeln für die Anzahl der überdeckten Basen und die Anzahl der Assembly-Lücken/Contigs heißen Lander-Waterman-Gleichungen.

4.3 Kürzeste gemeinsame Oberstrings (KGO)

Definition 4.1 Ein String S ist ein *Oberstring* von String T, wenn T ein Teilstring von S ist.

Das Problem des kürzesten, gemeinsamen Oberstrings (KGO, shortest common superstring problem) ist, zu einer gegebenen Menge $\mathcal{F}$ von Strings unter allen Strings, die ein Oberstring von jedem String in $\mathcal{F}$ sind, einen String S zu finden, der minimale Länge hat. Beispiel: $\mathcal{F} = \{cagt, aggtca, gtagg\}$. Der String $aggtcagtagg$ ist ein Oberstring der Strings in $\mathcal{F}$ und der String $S = gtaggtcagt$ ist ein (der) KGO der Strings in $\mathcal{F}$. Dieses Problem ist NP-vollständig, also ist kein effizierter Algorithmus bekannt, der das allgemeine Problem löst. In Abschnitt 4.3.1 wird gezeigt, dass sich dieses Problem unter gewissen stark idealisierten Voraussetzungen bei der Assemblierung stellt und in Abschnitt 4.3.2 eine Sequenziermethode vorgestellt, bei der sich ein Spezialfall des KGO-Problems stellt.

4.3.1 Fragment Assemblierung

Wir gehen in diesem Abschnitt von einer idealisierten Problemstellung aus: Wir nehmen an, dass beim Sequenzieren keine Fehler entstanden sind und dass die Orientierung der Reads bereits bekannt ist. Das sind für die Praxis unzulässige Annahmen, aber die Ideen hinter den Algorithmen sind nützlich für Algorithmen, die tatsächliche Instanzen des Problems behandeln. Die Darstellung der in diesem Abschnitt vorgestellten Ergebnisse folgt dem Buch von Setubal und Meidanis [Set97]. Sei $\mathcal{F}$ eine Menge von Strings. Es sind in der Anwendung die Sequenzen der Reads. Wir nennen die Elemente von $\mathcal{F}$ *Fragmente*. Wir nehmen an, dass $\mathcal{F}$ Teilstring-frei ist: Kein String in $\mathcal{F}$ ist Teilstring eines anderen Strings in $\mathcal{F}$. Wenn eine gegebene Menge $\mathcal{F}'$ von Fragmenten nicht Teilstring-frei ist, kann man alle Fragmente löschen, die Teilstring eines anderen sind. Dies kann man mit Hilfe von Suffixbäumen effizient machen. Die so erhaltene Teilstring-freie Menge $\mathcal{F}$ hat dann genau dieselben Oberstrings wie $\mathcal{F}'$.

KGOs als Pfade

Wir führen jetzt einen auf $\mathcal{F}$ basierenden Graphen ein, in dem Pfade gemeinsamen Oberstrings von Fragmenten entsprechen.

Definition 4.2 Der Überlapp-Multigraph $MG(\mathcal{F})$ einer Menge $\mathcal{F}$ von Strings ist ein gerichteter, gewichteter Multigraph. Die Menge der Knoten von $MG(\mathcal{F})$ ist gleich $\mathcal{F}$. Von Knoten $u \in \mathcal{F}$ zu Knoten $v \in \mathcal{F}$, $v \neq u$, ist genau dann eine Kante mit Gewicht $t \geq 0$, wenn $|u|, |v| \geq t$ und das Suffix von u der Länge t ein Präfix von v ist.

In $MG(\mathcal{F})$ kann es zwischen zwei Knoten mehrere Kanten mit verschiedenen Gewichten geben (deshalb "multi"). Zwischen zwei beliebigen Knoten gibt es immer eine Kante mit Gewicht 0. Sei P in $MG(\mathcal{F})$ ein gerichteter Pfad entlang der Knoten $f_1, f_2, \ldots, f_d$. (Bei einem Pfad wird jeder Knoten höchstens einmal besucht). Sei $F = \{f_1, f_2, \ldots, f_d\}$ die Menge der besuchten Knoten. Sei $t_i \geq 0$ das Gewicht der Kante in P von f_i zu f_{i+1} $(i = 1, \ldots, d-1)$. Das Gewicht des Pfades $w(P)$ sei definiert als die Summe der Gewichte der Kanten von P, also $w(P) = t_1 + \cdots + t_{d-1}$. Dem Pfad P in $MG(\mathcal{F})$ entspricht auf folgende Weise ein Oberstring $S(P)$ von F. $S(P)$ ist die Konsensus-Sequenz eines Alignments der Fragmente $f_1, f_2, \ldots, f_d$, wobei das Ende von f_i mit dem Anfang von f_{i+1} um t_i Positionen überlappt. Also

$$S(P) = f_1[1..|f_1| - t_1]\, f_2[1..|f_2| - t_2] \cdots f_{d-1}[1..|f_{d-1}| - t_{d-1}]\, f_d.$$

Abbildung 4.7 zeigt ein Beispiel für $MG(\mathcal{F}), P$ und $S(P)$.

Sei $||F|| := |f_1| + \cdots + |f_d|$ die Summe der Längen der Fragmente des Pfades P. Es gilt

$$\begin{aligned} |S(P)| &= (|f_1| - t_1) + (|f_2| - t_2) + \cdots + (|f_{d-1}| - t_{d-1}) + |f_d| \\ &= ||F|| - w(P), \end{aligned}$$

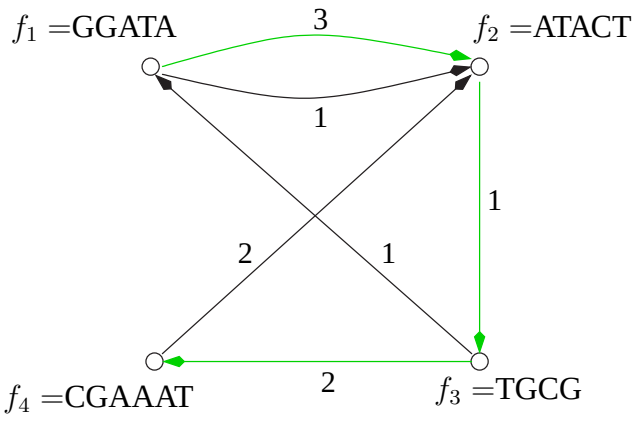


Abbildung 4.7: Überlapp-Multigraph $MG(\mathcal{F})$ zur Menge $\mathcal{F} = \{\text{GGATA}, \text{CGAAAT}, \text{ATACT}, \text{TGCG}\}$. Die Kanten mit Gewicht 0 sind weggelassen. Der Pfad P, der die Knoten f_1, f_2, f_3, f_4 entlang der grünen (hellen) Kanten besucht, definiert einen Oberstring $S(P) = GGATACTGCGAAAT$ mit Gewicht $w(P) = 3 + 1 + 2 = 6$.

also

$$|S(P)| + w(P) = ||F||. \tag{4.5}$$

Ein Pfad, der jeden Knoten des Graphen besucht, heißt *Hamiltonscher Pfad*. Für einen Hamiltonschen Pfad P ist der String $S(P)$ also ein Oberstring von $\mathcal{F}$. Wegen (4.5) gilt also für Hamiltonsche Pfade P

$$|S(P)| + w(P) = ||\mathcal{F}||. \tag{4.6}$$

Das bedeutet, da $||\mathcal{F}||$ bei der Assembly gegeben ist, dass das Problem $|S(P)|$ zu minimieren, äquivalent ist dazu, $w(P)$ zu maximieren.

Jedem Hamiltonschen Pfad in $MG(\mathcal{F})$ entspricht ein Oberstring von $\mathcal{F}$, aber nicht zu jedem Oberstring von $\mathcal{F}$ gibt es einen Pfad, der ihm entspricht. Ein beliebiger Oberstring kann ja unnötige Zeichenketten enthalten, z.B. an den Enden. Wie der folgende Satz zeigt, gibt es aber zu jedem KGO S einen Hamiltonschen Pfad, der ihm entspricht.

Satz 4.3 Wenn S ein kürzester gemeinsamer Oberstring der Teilstring-freien Menge $\mathcal{F}$ von Fragmenten ist, dann gibt es einen Hamiltonschen Pfad P, so dass $S = S(P)$.

Beweis: Da S ein gemeinsamer Oberstring von $\mathcal{F}$ ist, gibt es für jedes Fragment $f \in \mathcal{F}$ mindestens ein Interval $[l(f)..r(f)]$ von S (, d.h. $1 \leq l(f) \leq r(f) \leq |S|$), so dass $S[l(f)..r(f)] = f$ ist. Wenn es mehrere Vorkommen von f in S gibt, fixiere ein beliebiges solches Invervall. Da kein Fragment in $\mathcal{F}$ Teilstring eines anderen ist, lassen sich die Intervalle der Fragmente so sortieren, dass sowohl die linken als auch die rechten Endpunkte der Invervalle eine streng aufsteigende Folge bilden (Warum?). Seien $f_1, f_2, \ldots, f_d$ die Fragmente in $\mathcal{F}$ in dieser Reihenfolge. Es gilt dann $l(f_1) = 1$ und $r(f_d) = |S|$, weil S sonst an den Rändern verkürzt werden könnte und immer noch ein gemeinsamer Oberstring wäre. Ferner gilt $r(f_i) \geq l(f_{i+1}) - 1$ für $i = 1, 2, \ldots d - 1$, sprich entweder überlappen die Intervalle von f_i und f_{i+1} oder das Intervall von f_{i+1} schließt direkt an das Intervall von f_i an, denn, wenn es dazwischen eine Lücke gäbe, könnte man S wieder verkürzen und hätte einen Widerspruch dazu, dass S ein KGO ist. Es gibt also in $MG(\mathcal{F})$ eine Kante von f_i zu f_{i+1} mit Gewicht $r(f_i) - l(f_{i+1}) + 1$. Der Pfad P besuche die Knoten von $\mathcal{F}$ in der Reihenfolge $f_1, f_2, \ldots, f_d$ und nehme zwischen f_i und f_{i+1} diese Kante mit Gewicht

$t_i := r(f_i) - l(f_{i+1}) + 1 \geq 0$. Dann ist

$$\begin{aligned} S(P) &= f_1[1..|f_1| - t_1]\, f_2[1..|f_2| - t_2] \cdots f_{d-1}[1..|f_{d-1}| - t_{d-1}]\, f_d \\ &= S[l(f_1)..l(f_2) - 1]\, S[l(f_2)..l(f_3) - 1] \cdots S[l(f_{d-1})..l(f_d) - 1]\, S[l(f_d)..r(f_d)] \\ &= S \end{aligned}$$

□

Da jeder Hamiltonsche Pfad in $MG(\mathcal{F})$ einen gemeinsamen Oberstring von $\mathcal{F}$ definiert und der KGO von $\mathcal{F}$ nach obigem Satz auch durch einen Hamiltonschen Pfad definiert wird, kann man einen KGO finden, indem man in $MG(\mathcal{F})$ einen Hamiltonschen Pfad P findet, der einen String $S(P)$ minimaler Länge definiert, also nach Formel (4.6) einen Hamiltonschen Pfad mit maximalem Gewicht $w(P)$. Das Problem, einen Hamiltonschen Pfad (maximalen Gewichts) zu finden, ist wieder NP-vollständig aber für die Instanzen aus der Anwendung in der Assemblierung kann man hierfür Heuristiken wie die Greedy-Methode verwenden.

Assembly von Sequenzen ohne lange Repeats

In diesem Abschnitt wollen wir eine Forderung an das Sampling der Reads stellen. Ähnlich wie wir das schon bei der Analyse der benötigten mittleren Coverage gemacht haben, wollen wir fordern, dass die Intervalle der Reads genügend überlappen. Was hier "genügend" bedeutet, hängt von der Länge der Repeats in der Zielsequenz ab. Nach wie vor gilt hier leider wegen der unrealistischen Annahmen, dass die Ergebnisse nicht direkt auf praktische Probleme angewendet werden können.

Um in diesem Abschnitt begrifflich besser zwischen der Sequenz eines Fragments und seiner Lage auf der Zielsequenz S unterscheiden zu können führen wir einige Bezeichnungen ein. Ein Intervall von S ist ein ganzzahliges Intervall $[i..j]$, so dass $i \geq 1, j \leq |S|$ und $i \leq j + 1$. Wenn $i = j + 1$ ist, ist das Intervall leer. Ein *Sampling* von S ist eine Menge $\mathcal{A}$ von Intervallen von S. Das Sampling *überdeckt* S, wenn die Vereinigung seiner Intervalle gleich $[1..|S|]$ ist, also wenn es für jedes i mit $1 \leq i \leq |S|$ mindestens ein Intervall $\alpha \in \mathcal{A}$ gibt mit $i \in \alpha$. Zwei Intervalle α und β *überlappen um mindestens* t, wenn $|\alpha \cap \beta| \geq t$. Ein Teilintervall-freies Sampling $\mathcal{A}$ ist *verbunden auf Niveau* t, wenn es für je zwei Intervalle α und β aus $\mathcal{A}$ mit $l(\alpha) < l(\beta)$ Intervalle $\gamma_1, \ldots, \gamma_d$ gibt, so dass $\gamma_1 = \alpha, \gamma_d = \beta$, $l(\gamma_i) < l(\gamma_{i+1})$ und γ_i mit γ_{i+1} um mindestens t überlappt für jedes $i = 1, \ldots, d - 1$. Die Intervalle sind dann also indirekt verbunden und der Parameter t mißt die Stärke der Verbindung. Die Menge von Fragmenten, die durch das Sampling $\mathcal{A}$ *erzeugt* wird, ist $\mathcal{F}[\mathcal{A}] := \{S[\alpha] \,|\, \alpha \in \mathcal{A}\}$. Wir wollen Mengen von Fragmenten untersuchen, die von Samplings erzeugt werden, die auf Niveau t verbunden sind. Wir nennen ein Sampling $\mathcal{A}$ *Teilintervall-frei*, wenn es keine zwei verschiedenen Intervalle in $\mathcal{A}$ gibt, von denen das eine im anderen enthalten ist. Für eine nichtnegative, ganze Zahl t und eine Menge $\mathcal{F}$ von Fragmenten sei $MG(\mathcal{F}, t)$ der Multigraph, den man aus dem Überlapp-Multigraphen $MG(\mathcal{F})$ erhält, wenn man alle Kanten streicht, die ein kleineres Gewicht als t haben. Es sei $G(\mathcal{F}, t)$ der gerichtete, gewichtete Graph, den man aus $MG(\mathcal{F}, t)$ erhält, wenn man von allen Kanten, die von einem Knoten u zu einem Knoten v gehen, nur die Kante mit *maximalem* Gewicht behält und die anderen (sofern es andere gibt) streicht. Die Schreibweise $\alpha \xrightarrow{w} \beta$ ist eine Abkürzung dafür, daß das rechte Ende vom Intervall α und das linke Ende vom Intervall β um genau w Positionen überlappen, also daß $|\alpha|, |\beta| \geq w$ und $l(\beta)+w-1 = r(\alpha)$. Entsprechend bedeute für Fragemente f und g die Schreibweise $f \xrightarrow{w} g$, daß es eine Kante mit Gewicht w von f nach g in $MG(\mathcal{F})$ gibt, also dass das Suffix von f der Länge w ein Präfix von g ist. Das folgende Lemma zeigt, bei einem "guten" Sampling der Oberstring S der Fragmente sogar von einem Hamiltonschen Pfad im verkleinerten Graphen definiert wird.

Lemma 4.4 Sei S ein String, sei $t \geq 0$ und $\mathcal{A}$ ein Teilintervall-freies, auf dem Niveau t verbundenes Sampling von S, das S überdeckt. Dann enthält der Multigraph $MG(\mathcal{F}[\mathcal{A}], t)$ einen Hamiltonschen Pfad P mit $S(P) = S$.

Beweis: Seien $\alpha_1, \alpha_2, \ldots, \alpha_n$ die Intervalle von $\mathcal{A}$ nach dem linken Endpunkt aufsteigend sortiert. Da $\mathcal{A}$ Teilintervall-frei ist, sind von keinen zwei verschiedenen Intervallen die linken Endpunkte gleich. Ebenso sind von keinen zwei verschiedenen Intervallen die rechten Endpunkte gleich. Weiter folgt aus der Teilintervall-Freiheit auch, daß in dieser Sortierung die rechten Randpunkte eine streng aufsteigende Folge bilden. Wir behaupten, daß $\alpha_i \xrightarrow{w_i} \alpha_{i+1}$ für ein $w_i \geq t$ für alle $i < n$. Um dies einzusehen beachte, daß es, da $\mathcal{A}$ auf dem Niveau t verbunden ist, eine Folge von $d > 1$ Intervallen $\gamma_1, \ldots \gamma_d$ aus $\mathcal{A}$ gibt mit $\gamma_1 = \alpha_i$ und $\gamma_d = \alpha_n$, so daß γ_j mit γ_{j+1} um mindestens t überlappt und $l(\gamma_j) < l(\gamma_{j+1})$ $(j = 1, \ldots, d-1)$. Das Intervall $\beta := \gamma_2$ erfüllt also $r(\alpha_i) - l(\beta) + 1 \geq t$ und $l(\alpha_i) < l(\beta)$; deshalb auch $l(\alpha_{i+1}) \leq l(\beta)$, da die Intervalle aufsteigend sortiert sind. Also ist

$$w_i = |\alpha_i \cap \alpha_{i+1}| = r(\alpha_i) - l(\alpha_{i+1}) + 1 \geq r(\alpha_i) - l(\beta) + 1 \geq t.$$

Da

$$\alpha_1 \xrightarrow{w_1} \alpha_2 \xrightarrow{w_2} \cdots \xrightarrow{w_{n-1}} \alpha_n$$

impliziert, daß

$$f_1 \xrightarrow{w_1} f_2 \xrightarrow{w_2} \cdots \xrightarrow{w_{n-1}} f_n,$$

gibt es in $MG(\mathcal{F}, t)$ für jedes $i < n$ eine Kante des Gewichts $w_i \geq t$ von f_i nach f_{i+1}. Die Folge dieser Kanten definiert also einen Hamiltonschen Pfad P. Da $\alpha_1 \cup \cdots \cup \alpha_n = S$, ist $S(P) = S$. □

Es gibt einen Zusammenhang zwischen dem Vorhandensein von Repeats in der Zielsequenz und der Existenz von Kreisen im Graphen $G(\mathcal{F}, t)$. Dieser wird im Folgenden untersucht. Das Vorhandensein eines Repeats in S bedeutet, dass es zwei verschiedene Intervalle $\alpha \neq \beta$ gibt mit $S[\alpha] = S[\beta]$. Die *Größe* des Repeats ist die Länge der beiden gleichlangen Strings. Ein Paar von Intervallen α, β ist ein *falsch-positives* Paar auf Niveau t, wenn es ein $w \geq t$ gibt, so dass $S[\alpha] \xrightarrow{w} S[\beta]$ aber nicht $\alpha \xrightarrow{w} \beta$. Falsch-positiv heißt es deshalb, weil man einen Fehler macht, wenn man annimmt, dass die sich überlappenden Fragmente $S[\alpha]$ und $S[\beta]$ auch von sich entsprechend überlappenden Intervallen in der Zielsequenz stammen.

Wie unten gezeigt wird, ist die Existenz von Kreisen im Überlapp-Graphen $G(\mathcal{F}, t)$ immer auf Repeats zurückzuführen. Die Umkehrung gilt aber nicht, d.h. es kann Repeats geben aber der Überlapp-Graph ist trotzdem azyklisch. Zunächst zwei Lemmata.

Lemma 4.5 Wenn $\alpha \xrightarrow{w} \beta$, dann gilt $l(\alpha) \leq l(\beta)$ und $r(\alpha) \leq r(\beta)$.

Beweis: Sei $\alpha \xrightarrow{w} \beta$. Deswegen und, da $|\alpha| \geq w$, gilt $l(\alpha) \leq r(\alpha) - w + 1 = l(\beta)$. Entsprechend schließen wir $r(\beta) \geq l(\beta) + w - 1 = r(\alpha)$.

□

Lemma 4.6 Wenn in einem Sampling von S ein falsch-positives Paar auf Niveau t existiert, dann gibt es in S ein Repeat mindestens der Größe t.

Beweis: Seien α und β zwei Intervalle mit $S[\alpha] \xrightarrow{w} S[\beta]$ aber nicht $\alpha \xrightarrow{w} \beta$. Wegen ersterem gilt $|\alpha|, |\beta| \geq w \geq t$, wegen letzterem gilt $l(\beta) \neq r(\alpha) - w + 1$. Sei α' das Intervall, dass aus den rechtesten w Elementen von α besteht und sei β' das Intervall, dass aus den linkesten w Elementen von β besteht. Wegen $S[\alpha] \xrightarrow{w} S[\beta]$ gilt dann $S[\alpha'] = S[\beta']$. Aber α' und β' können nicht gleich sein, weil die linken Randpunkte verschieden sind: $l(\beta') = l(\beta) \neq r(\alpha) - w + 1 = r(\alpha') - w + 1 = l(\alpha')$. Also ist der String $S[\alpha']$ ein Repeat von S der Länge $w \geq t$.

□

Satz 4.7 Sei $\mathcal{A}$ ein Sampling von String S. Wenn $G(\mathcal{F}[\mathcal{A}], t)$ einen gerichteten Kreis enthält, dann gibt es in S ein Repeat mindestens der Größe t.

Beweis: Sei $f_1 \to f_2 \to \cdots \to f_d \to f_1$ mit $d \geq 2$ ein gerichteter Kreis in $G(\mathcal{F}[\mathcal{A}], t)$. Es gibt Intervalle $\alpha_1, \ldots, \alpha_d$, so dass $S[\alpha_i] = f_i$ für $i = 1, \ldots, d$. Wir zeigen, dass mindestens eines der Paare (α_i, α_{i+1}), einschließlich (α_d, α_1), ein falsch-positives auf Niveau t ist. Dann gibt es nach Lemma 4.6 ein Repeat der Länge mindestens t.

Angenommen keines dieser Paare wäre falsch-positiv. Dann wäre also $\alpha_1 \longrightarrow \cdots \longrightarrow \alpha_d \longrightarrow \alpha_1$ (, jeweils mit irgendwelchen Gewichten). Wegen Lemma 4.5 gälte dann auch $l(\alpha_1) \leq \ldots \leq l(\alpha_d) \leq l(\alpha_1)$. Also wären alle $l(\alpha_i)$ gleich. Entsprechend kann man schliessen, dass alle $r(\alpha_i)$ gleich wären. Dann wären alle α_i gleich und somit auch alle f_i. Das kann nicht sein, denn dann wäre obiger Pfad gar kein Pfad.

□

Folgender Satz liefert einen effizienten Algorithmus für die Assemblierung, wenn die Repeats in der Zielsequenz kürzer sind als das Niveau der Verbindung der Intervalle des Samplings.

Satz 4.8 Sei $\mathcal{A}$ ein Teilintervall-freies, auf dem Niveau t verbundenes Sampling von String S, das S überdeckt. S habe keine Repeats der Größe t oder größer. Dann ist der Graph $G(\mathcal{F}[\mathcal{A}], t)$ azyklisch und hat genau einen Hamiltonschen Pfad P. Für diesen gilt $S(P) = S$.

Beweis: Unter den genannten Voraussetzungen sind die Graphen $G(\mathcal{F}[\mathcal{A}], t)$ und $MG(\mathcal{F}[\mathcal{A}], t)$ gleich, weil es in $MG(\mathcal{F}[\mathcal{A}], t)$ keine Mehrfachkanten geben kann. Um dies einzusehen, nehmen wir an, es gäbe zwei Intervalle α, β in $\mathcal{A}$ und zwei verschiedene Gewichte $w_1, w_2 \geq t$ mit $S[\alpha] \xrightarrow{w_1} S[\beta]$ und $S[\alpha] \xrightarrow{w_2} S[\beta]$. Da es keine falsch-positiven Paare gibt, müsste dann auch $\alpha \xrightarrow{w_1} \beta$ und $\alpha \xrightarrow{w_2} \beta$ gelten. Zwei Intervalle können aber nicht um verschiedene Längen überlappen. Die Annahme führt also zum Widerspruch. Nach Satz 4.7 ist $G(\mathcal{F}[\mathcal{A}], t)$ azyklisch und nach Lemma 4.4 existiert in ihm ein Hamiltonscher Pfad P mit $S(P) = S$, da obige Graphen gleich sind. Es gilt allgemein: Wenn in einem azyklischen, gerichteten Graphen ein Hamiltonscher Pfad existiert, dann ist er eindeutig (Übungsaufgabe).

□

Das Problem, in einem azyklischen Graphen, in dem ein Hamiltonscher Pfad existiert, diesen zu finden, läßt sich effizient lösen (Übungsaufgabe). *Unter den genannten Voraussetzungen* ist die Assemblierung also möglich. Man konstruiert zunächst zur gegebenen Menge von Fragmenten $\mathcal{F}$ und einem geeigneten t den Überlappgraphen $MG(\mathcal{F}, t)$. Dies ist effizient unter Verwendung von Suffixbäumen möglich (siehe Anwendung 7.10 im Gusfield [Gus99]: All-pairs suffix-prefix matching) und benötigt $O(|E| + m)$, wobei $|E|$ die Anzahl der Kanten von $MG(\mathcal{F}, t)$ ist und m die Gesamtlänge aller Fragmente. Dann kann man den Hamiltonschen Pfad P ermitteln und erhält so die Zielsequenz $S(P)$.

Praktisch muss man jedoch sowohl mit Fehlern rechnen als auch mit längeren Repeats. Dennoch gibt es erfolgreiche Assemblier-Programme die darauf basieren, in einem Überlappgraphen (bei dem bei den Überlapps Fehler erlaubt sind) Mengen von disjunkten Pfaden zu finden. Jeder dieser Pfade entspricht dann einem Contig.

4.3.2 Sequenzierung durch Hybridisierung

DNA-Chips

Die sogenannte 'Sequenzierung durch Hybridisierung" (Sequencing by Hybridization, SBH) wurde in den späten 80er Jahren vorgeschlagen als eine Alternative zu Gel-basierten Sequenzierme-

thoden. Bei dieser Technik wird eine große Anzahl verschiedener, kurzer Oligonukleotide (Proben genannt) auf einer festen Oberfläche (Glas oder Silizium) in einem zweidimensionalen Feld (ähnlich wie die Felder eines Schachbretts) angeordnet; einem sogenannten DNA-Chip oder Microarray. Bei der Verwendung eines solchen Chips wird zunächst eine Lösung mit vielen Kopien einer Ziel-DNA-Sequenz hergestellt. Diese Sequenzen werden radioaktiv oder fluoreszierend markiert. Dann wird die Lösung auf dem Chip aufgebracht und die Oligonucleotide auf dem Chip hybridisieren mit der Ziel-DNA-Sequenz, wenn ihr Watson-Crick-Komplement als Teilstring in der Ziel-DNA-Sequenz vorkommt. Die Nicht-Hybridisierten Sequenzen werden abgewaschen und die Felder des Chips werden anschließend automatisch auf die Markierung untersucht. Das Resultat ist das sogenannte *Spektrum*. Es besteht aus der Information, von welcher der Proben das Komplement mindestens einmal als Teilstring in der Ziel-DNA-Sequenz vorkommt.
Beispiel: Eine Probe *accgttgc* hybridisiert mit der Ziel-DNA-Sequenz *actggcaacgct*, weil das Komplement der Probe, *tggcaacg*, ein Teilstring der Zielsequenz ist.
Bei traditionellen Chips werden als Proben alle k-mere benutzt. Die Biotechnologiefirma Affimetrix hat 1994 den ersten DNA-Chip hergestellt, der alle 8-mere als Proben enthielt, das sind also $4^8 = 256 \times 256 = 65536$ Proben auf einem Chip. Mittlerweile sind mindestens Chips mit etwa einer Million Proben möglich.

Idealisierte Problemstellung

Wir gehen von einem Chip aus, der als Proben alle k-mere enthält. Sei T die Ziel-DNA-Sequenz und sei das Spektrum S die Menge aller Teilwörter von T der Länge k.

Definition 4.9 (Spektrum) Für einen DNA-Chip mit Oligomeren der Länge k ist das Spektrum S einer Sequenz T ist die Menge

$$S = \{w \mid w \text{ ist ein Teilwort von } T \text{ der Länge } k\}.$$

SBH-Problem: Rekonstruiere mittels ihres Spektrums S die Sequenz T.

Definition 4.10 Ein String U heißt kompatibel mit dem Spektrum S, wenn S das Spektrum von U ist, also wenn U jede Sequenz aus S als Teilstring enthält aber keine k-mere als Teilstring enthält, die nicht in S sind.

Wir wissen von den Elementen in S nur, dass sie mindestens einmal in T vorkommen, aber nicht, wie oft sie vorkommen. Wenn ein k-mer mehr als einmal in T vorkommt, läßt sich T nicht mehr eindeutig aus S rekonstruieren. Z.B., wenn $T = agtgtgtc$ ist und $k = 3$, ist das Spektrum $S = \{agt, gtg, tgt, gtc\}$. Aber die Sequenzen $agtgtc$, $agtgtgtgtc$, $agtgtgtgtgtc$, usw. besitzen dasselbe Spektrum. Wir nehmen zunächst vereinfachend an, dass in T kein k-mer mehr als einmal vorkommt. Dann ist T der kürzeste gemeinsame Oberstrings der Sequenzen in S. Die Wahrscheinlichkeit, dass tatsächlich kein k-mer doppelt vorkommt, hängt von k und der Länge der Zielsequenz ab (siehe unten). Ausserdem nehmen wir an, dass das Spektrum fehlerfrei ist. Also, dass weder eine Probe fälschlicherweise hybridisiert, obwohl die Basenpaarung nicht vollständig stimmt (falsch positiv), noch die Markierung bei einer hybridisierten Probe nicht erkannt wird (falsch negativ). In tatsächlichen Experimenten kommen jedoch beide Arten Fehler vor.

Reduktion zu Euler-Pfaden

Unter der idealisierten Problemstellung ist das SBH-Problem zwar das schwierige Problem, einen kürzesten gemeinsamen Oberstring der Sequenzen in S zu finden, aber in diesem Problem haben die Wörter in S noch eine zusätzliche Struktur, die das Problem vereinfachen: Zu jedem Wort s

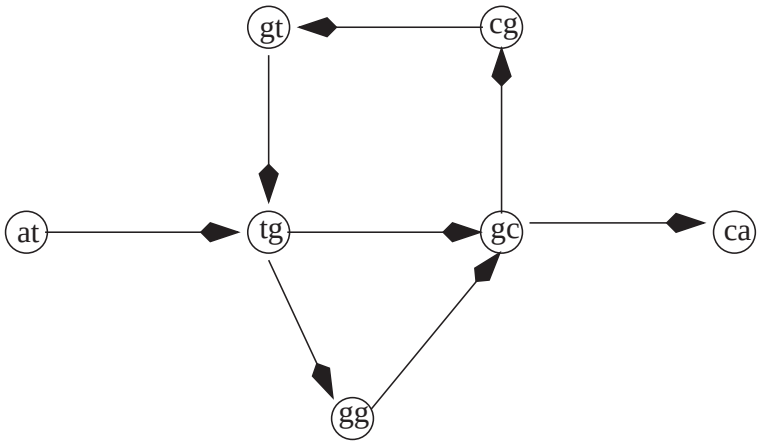


Abbildung 4.8: Der Graph G zum Spektrum $S = \{atg, tgg, tgc, gtg, ggc, gca, gcg, cgt\}$

in S außer dem Wort, das am Ende in T auftritt, gibt es ein anderes Wort in S dessen Präfix der Länge $k-1$ gleich dem Suffix der Länge $k-1$ von s ist. Die Sequenzen in S überlappen sich besonders viel. Dies kann man hier ausnutzen. Es erlaubt, einen Graphen zu definieren, bei dem die *Kanten* Teilstrings in S entsprechen, anders als im Überlapp-Multigraphen aus dem vorigen Abschnitt 4.3.1, wo die Knoten den Teilstrings entsprechen.

Der gerichtete Graph $G = (V, E)$ für S sei wie folgt definiert. V ist die Menge aller $(k-1)$-mere, also $V = \{a, c, g, t\}^{k-1}$. E enthält die Kante (v, w) von Knoten v zu Knoten w genau dann, wenn es im Spektrum ein k-mer s gibt, so dass die ersten $k-1$ Zeichen von s gleich v und die letzten $k-1$ Zeichen von s gleich w sind. Figur 4.8 zeigt den Graphen für das Spektrum $S = \{atg, tgg, tgc, gtg, ggc, gca, gcg, cgt\}$. Dabei sind die isolierten Knoten (z.B. ag) weggelassen.

Ein Pfad in G spezifiziert auf folgende Weise einen String U. U beginnt mit dem String des ersten Knotens des Pfades. Dann folgen die aneinander gehängten jeweils letzten Buchstaben der Knoten entlang des Pfades. Zum Beispiel der Pfad entlang der Knoten at,tg,gc,ca spezifiziert den String $atgca$.

Definition 4.11 In einem gerichteten Graphen G heißt ein Pfad, der jede Kante von G genau einmal besucht, ein *Euler-Pfad*. Endet ein Euler-Pfad in dem Knoten, in dem er beginnt, heißt er auch *Euler-Kreis*.

Ein Euler-Pfad in dem Graphen für S spezifiziert einen String, der kompatibel mit dem Spektrum ist, denn jeder Kante $e = (u, v)$ des Pfades entspricht ein Element des Spektrums, nämlich das k-mer, das aus der Aneinanderhängung der $k-1$ Zeichen des Knotens u und dem ersten Zeichen des Knotens v besteht. Der String enthält kein k-mer mehr als einmal, weil der Euler-Pfad keine Kante zweimal benutzt. Umgekehrt wird auch jeder String U, der kompatibel mit dem Spektrum ist und, der kein k-mer mehr als einmal enthält, von einem Euler-Pfad ϕ im Graphen für S spezifiziert: Sei n die Länge von U. Es ist nämlich ϕ die Abfolge der Knoten $U[1..k-1], U[2..k], U[3..k+1], \ldots, U[n-k+2, n]$ und enthält jede Kante genau einmal. Wir haben also folgenden

Satz 4.12 Ein String U, der kein k-mer mehr als einmal enthält, ist genau dann kompatibel mit dem Spektrum S, wenn er durch einen Euler-Pfad spezifiziert wird.

Wenn in einem Graphen $G = (V, E)$ ein Eulerpfad existiert, so kann er in Zeit $O(|E|)$ gefunden werden (Wie?). In dieser idealisierten Problemstellung gibt es immer einen Eulerpfad, nämlich

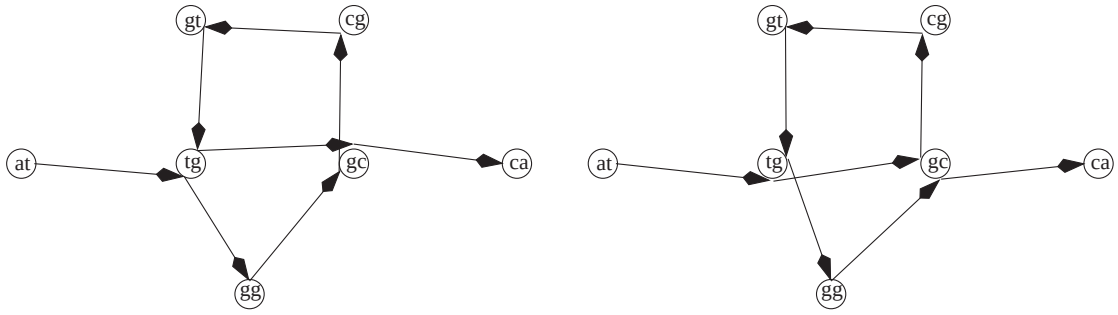


Abbildung 4.9: Zwei Eulerpfade. Die von ihnen spezifizieren Strings sind atggcgtgca (links) und atgcgtggca (rechts).

den, der die Zielsequenz spezifiziert.

Figur 4.9 zeigt zu obigem Beispielgraph G zwei verschiedene Eulerpfade und die von ihnen spezifizierten mit dem Spektrum kompatiblen Strings. Sie zeigt, dass selbst unter der Annahme, dass kein k-mer mehrfach vorkommt, die Rekonstruktion nicht immer eindeutig möglich ist. Allerdings kann in einem solchen Fall möglicherweise ein Teil des unbekannten Strings rekonstruiert werden, indem man die Gemeinsamkeiten aller möglichen Eulerpfade herausfindet.

Grenzen von traditionellem SBH

Eine Ziel-DNA-Sequenz ist eindeutig rekonstruierbar, wenn sie die Einzige mit ihrem Spektrum ist. Theoretische Studien haben gezeigt, dass die erwartete Länge eindeutig rekonstruierbarer Ziel-DNA-Sequenzen mit der traditionallen SBH Methode $O(2^k)$ ist, also höchstens proportional zur Wurzel der verwendeten Proben (4^k) ist. Dabei wurde angenommen, dass die Basen der Sequenzen unabhängig sind und gleich wahrscheinlich. Für einen zufälligen Ziel-DNA-String der Länge ℓ und eine Länge k der Oligomere der Proben sei $p(\ell, k)$ die Wahrscheinlichkeit, dass der String eindeutig aus seinem Spektrum rekonstruierbar ist. Es ist etwa $p(200, 8) \approx 0.9$, d.h. ein zufälliger String der Länge 200 kann mit einem traditionellen Chip mit $4^8 = 65536$ Proben mit Wahrscheinlichkeit 90% aus seinem Spektrum rekonstruiert werden. Die Länge der mit der traditionellen Methode sequenzierbaren Sequenzen ist viel zu kurz dafür, dass die Methode mit Gel-basierten Sequenziermethoden konkurrieren könnte.

Proben mit universellen Basen

Es hat viele Vorschläge gegeben um mit der gleichen Anzahl von Proben, also der gleichen Chipgröße, längere Strings zu sequenzieren, indem man andere Chips benutzt als die traditionellen, die jedes k-mer enthalten. Eine relativ erfolgreiche Methode ist die Verwendung einer sogenannten *universellen Base* (eine solche ist Inosin, engl.: inosine), die mit jeder der vier Standard-Basen paaren kann. Wenn man die universelle Base mit dem Zeichen '*' bezeichnet, würde etwa der String ac*t*g eine Probe bezeichnen, die mit allen Ziel-DNA-Sequenzen hybridisiert, die einen der 16 Strings, acatag, acatcg, ..., actttg, als Teilstring enthalten. Mit Hilfe solcher Proben wurde in [FPU99] ein Chip und ein Algorithmus für die Rekonstruktion konstruiert, der asymptotisch folgende theoretische Untergrenze für die benötigte Chipgröße erreicht.

Satz 4.13 Die Anzahl der Proben, die benötigt wird um einen beliebigen String der Länge m eindeutig zu rekonstruieren, ist mindestens $2m$.

Beweis: Ein Spektrum, das auf t Proben basiert, kann als binärer Vektor mit t Komponenten dargestellt werden. Es gibt 2^t solcher Vektoren, und jeder kann höchstens eine Sequenz eindeutig definieren. Die Anzahl möglicher Sequenzen ist 4^m. Also ist $4^m \leq 2^t$, also $t \geq 2m$. □

Diese Untergrenze für die Chipgröße wurde allerdings nur asymptotisch erreicht. Andere Methoden können die Konstante der Asymptotik verbessern oder die Sequenz mit größerer Verlässlichkeit bei fehlerhaften Daten rekonstuieren. Aktuelle SBH-Methoden mit universellen Basen können je nach Fehlerrate mit einem Chip mit $2^{20} \approx 10^6$ Proben Sequenzen bis zu einer Länge von 8000 Basen rekonstruieren ([HHHS03]).

4.4 Die Anzahl exakter Matches eines Worts

Die sogenannte Burrows-Wheeler-Transformation ist eine 1994 erstmals von Burrows und Wheeler veröffentlichte Methode. Dabei wird ein String S auf bestimmte Weise in einen String T permutiert, also nur die Reihenfolge der Buchstaben geändert. Diese Permutation kann auf effiziente Weise rückgängig gemacht werden, d.h. man kann S aus T zurückgewinnen. Da der permutierte String T sich im Allgemeinen wesentlich besser komprimieren lässt, findet die Burrows-Wheeler-Transformation Anwendung bei der verlustfreien Datenkompression, wo auf ihr basierende Methoden sehr hohe Kompressionsraten erzielen. In diesem Abschnitt stelle ich eine Methode vor, wie mit Hilfe der Burrows-Wheeler-Transformation schnell die Anzahl der exakten Matches eines Musters innerhalb eines Genoms gefunden werden kann. Die Methode benötigt weniger Speicherplatz als eine, die auf Suffixbäumen oder Suffixarrays basiert. Diese Datenstruktur hat vielfältige Anwendungsmöglichkeiten, wie etwa das Design von eindeutigen Proben für DNA-Chips. Healy und andere [HTSW03] benutzen sie auch für den Vergleich verschiedener Sequenz-Assemblies, weshalb sie in dieses Kapitel über Assemblierung eingeordnet ist.

Die Problemstellung, die wir hier besprechen ist Folgende. Wir haben einen sehr langen String S der Länge n gegeben. In der wichtigsten Anwendung ist S die DNA-Sequenz des Menschen, also etwa 3 Milliarden Zeichen lang und besteht aus Zeichen aus dem Alphabet $\Sigma = \{A, C, G, T, N\}$. Wir wollen eine Datenstruktur konstruieren, die es erlaubt, die Anfragen

"Kommt Muster p in S als Teilstring vor, und wenn ja, wie häufig?"

für viele p zu stellen. Diese Anfragen sollen schnell beantwortet werden. Zusätzlich soll der Speicherbedarf für diese Datenstruktur beim menschlichen Genom so klein sein, dass sie in den Hauptspeicher eines modernen PCs passt (sagen wir wenige Gigabyte RAM), weil Zugriffe auf die Festplatte wesentlich langsamer sind als Zugriffe auf RAM. Die Zeit, die für die Konstruktion der Datenstruktur benötigt wird, ist nicht kritisch.

Der Speicherbedarf für einen Suffixbaum wäre zu groß, denn um eine Position im menschlichen Genom zu speichern benötigt man 4 Bytes ($2^{32} \approx 4.3 \cdot 10^9$). Allein für die $3 \cdot 10^9$ Blätter des Suffixbaums benötigt man also mindestens 12 GB Speicher. Dazu kommt der Speicher, der für das Speichern von S benötigt wird.

Wenn die Muster nicht zu lang sind, kann man dieses Problem einfach lösen indem man eine Tabelle mit allen Mustern bis zu einer Höchstlänge und ihren Häufigkeiten in S anlegt. Aber ab etwa einer Musterlänge von 16 verbietet sich das Anlegen einer solchen Tabelle, weil der Speicherbedarf zu groß wird.

4.4.1 Anzahl der Vorkommen eines einzelnen Buchstabens.

Beim unten vorgestellten Algorithmus für das Problem, die Häufigkeit von Mustern in S zu finden, wird sich das Teilproblem stellen, die Häufigkeit eines *einzelnen Zeichens* x in einem Intervall von

String T zu bestimmen. Für ein ganzzahliges Intervall $[\ell..r] \subset [1..|T|]$ und ein Zeichen $x \in \Sigma$ sei

$$k(x, [\ell..r]) := \#\{i \in [\ell..r] \,|\, T[i] = x\},$$

also die Häufigkeit des Zeichens x im Teilstring $T[\ell..r]$.

Eine einfache Datenstruktur für dieses Problem ist die Folgende. Sie stellt einen Kompromiss zwischen schneller Beantwortung der Anfragen und geringem Speicherbedarf dar.

Sei K eine positive, ganze Zahl, die Blockgröße (z.B. etwa 100). Die Datenstruktur besteht aus einer Tabelle, die die Anzahl der Vorkommen jedes Zeichens x in den Präfixen von T, die ein Vielfaches von K lang sind, enthält. Für ein Zeichen x und ein i mit $1 \leq i \leq |T|$, das ein ganzzahliges Vielfaches von K ist sei

$$b(x, i) := k(x, [1..i]).$$

Mithilfe dieser b's kann mann dann ein $k(x, [\ell..r])$ relativ schnell bestimmen:

Seien $\ell' := \lfloor \ell/K \rfloor K$ und $r' := \lfloor r/K \rfloor K$ die Vielfachen von K, die am nächsten links von ℓ bzw. r liegen. Dann ist

$$k(x, [\ell..r]) = b(x, r') + \#\{i \in (r'..r] \,|\, T[i] = x\} - b(x, \ell') - \#\{i \in (\ell'..\ell] \,|\, T[i] = x\}.$$

Der Speicherbedarf dieser Methode ist $\Theta(n/K \cdot |\Sigma| \log n)$ Bits, weil es $n/K \cdot |\Sigma|$ Variablen $b(x, i)$ gibt und jede eine Zahl der Größe höchstens n enthält. Um Zahlen bis zu einer Größe n zu speichern benötigt man $\lceil \log n \rceil$ Bits. Die Laufzeit für die Beantwortung einer Anfrage ist eine Konstante plus die Zeit, die benötigt wird, die Anzahl der Vorkommen von x in T zwischen ℓ' und ℓ und zwischen r' und r zu zählen. Dies sind höchstens $2K$ Positionen und im Mittel K Positionen. Die Laufzeit für eine Anfrage ist also $O(K)$. (Man kann die Methode so verbessern, dass man im Durchschnitt nur $K/2$ Positionen vergleichen muß: Man wählt für ℓ' das nächste Vielfache von K zu ℓ, egal ob links oder rechts von ℓ; und entsprechend wählt man r'.)

4.4.2 Die Burrows-Wheeler-Transformation

Bevor wir die Burrows-Wheeler-Transformation (BWT) auf einen String S' anwenden, erweitern wir diesen (ähnlich wie bei den Suffixbäumen) am Ende um das Zeichen \$, das nicht in Σ enthalten ist. Sei $S = S'\$$, dieser erweiterte String und sei $n = |S|$ die Länge von S. Wir sortieren die Menge $S[1..n], S[2..n], \ldots, S[n..n]$ aller Suffixe von S lexikographisch. Dabei komme das Zeichen \$ vor allen anderen Zeichen (A,C,G,T,N) im Alphabet. Die so sortierte Liste aller Suffixe nennen wir das *Genom-Wörterbuch*. Der String der Zeichen, die in S den jeweiligen Suffixen in dieser Reihenfolge vorangehen, ist die Burrows-Wheeler-Transformation von S. Dabei sei das vorangehende Zeichen \$, wenn das Suffix gleich dem ganzen String S ist. Wir nennen die BWT T. Beispiel: $S =$ ABRAKADABRA\$. Dann erhalten wir folgende BWT und folgendes Genom-Wörterbuch:

T	Genom-Wörterbuch
A	$
R	A$
D	ABRA$
$	ABRAKADABRA$
K	ADABRA$
R	AKADABRA$
A	BRA$
A	BRAKADABRA$
A	DABRA$
A	KADABRA$
B	RA$
B	RAKADABRA$

Die BWT von S ist $T =$ ARD$KRAAAABB. Das Sortieren der Suffixe kann mit Hilfe eines Suffixbaums gemacht werden, der (auch für spätere Verwendung) auf der Festplatte gespeichert wird. Das Genom-Wörterbuch braucht nicht berechnet oder gespeichert zu werden, es dient aber der Erklärung des Algorithmus im nächsten Abschnitt.

Für Kompressionsalgorithmen wichtig, aber für das hier besprochene Problem unwichtig, ist folgende Tatsache: Der String S kann allein aus dem String T (auf wundersame Weise) schnell rekonstruiert werden.

4.4.3 Der Wortzähl-Algorithmus

Wir nehmen jetzt an, dass wir in dem transformierten String T die Anzahl der Vorkommen von einzelnen Buchstaben in Intervallen von T ermitteln können wie im Abschnitt 4.4.1 beschrieben.

Obwohl wir das Genom-Wörterbuch nicht tatsächlich zur Verfügung haben, benutzen wir es gedanklich. Zunächst berechnen wir bestimmte Indices in dieses Wörterbuch, nämlich die Nummern der Zeilen, an denen ein neuer Anfangsbuchstabe beginnt. Für ein Wort w seien F_w und L_w der Index vom ersten bzw. letzten Vorkommen von w als Anfang eines Wortes im Genom-Wörterbuch. k_w sei die Häufigkeit des Wortes w in S. Für das Beispiel aus dem vorigen Abschnitt 4.4.2 hätten wir z.B. $F_{\mathrm{A}} = 2$, $L_{\mathrm{A}} = 6$, $k_{\mathrm{A}} = 5$, $F_{\mathrm{BRA}} = 7$, $L_{\mathrm{BRA}} = 8$, $k_{\mathrm{BRA}} = 2$. Für den Spezialfall, in dem das Wort w ein einzelnes Zeichen x ist, speichern wir die F_x und L_x in einer (kleinen) Hilfsdatenstruktur, die wir *Alphagrenzen* nennen. (Wenn das Genom-Wörterbuch der Duden wäre, dann würde wir uns für jeden Buchstaben A..Z durch ein Lesezeichen merken, wo er anfängt.) k_x ist einfach die Häufigkeit von x im ganzen String T. F_x ist 1 plus die Summe der Häufigkeiten in T der Zeichen, die lexikographisch vor x kommen. Und L_x ist $F_x + k_x - 1$. Die Alphagrenzen können so also vorab bestimmt werden.

Der Algorithmus, der die Häufigkeit eines Musters p der Länge m bestimmt, wird schrittweise die abnehmenden Häufigkeiten der Muster $p[m], p[m-1..m], \ldots, p[1..m]$ bestimmen.

Sei x ein Zeichen und w ein Suffix von p. Angenommen, w kommt im Genom vor, wir kennen die Indices F_w und L_w im Genom-Wörterbuch und wir wollen die Häufigkeit k_{xw} vom Wort xw bestimmen. Weil die Häufigkeit von xw gleich die Anzahl der Vorkommen vom String w ist, denen ein x vorangeht, ist k_{xw} gleich der Anzahl der x in T zwischen Positionen F_w und L_w. Z.B. ist die Häufigkeit vom Wort RA in ABRAKADABRA$ gleich der Häufigkeit vom Zeichen R in T zwischen Positionen $F_{\mathrm{A}} = 2$ und $L_{\mathrm{A}} = 6$.

Wir können jetzt also iterativ Zeichen für Zeichen links an w anhängen, wenn wir erneut die Indices F_{xw} und L_{xw} berechnen können. Dazu sei b_{xw} die Anzahl der Wörter im Genom-Wörterbuch, die mit x beginnen und lexikographisch *vor* xw kommen. Wir können b_{xw} ermitteln, indem wir die Anzahl der x'e zählen, die in T vor Position F_w kommen. Damit können wir die Indices von xw berechnen, nämlich

$$\begin{aligned} F_{xw} &= F_x + b_{xw} \\ L_{xw} &= F_{xw} + k_{xw} - 1 \end{aligned}$$

Damit sind die Voraussetzungen für den nächsten Schritt der Iteration gegeben. Das Wort w kann links um ein weiteres Zeichen erweitert werden. Die Schritte enden, wenn entweder w nicht im Genom vorkommt oder, wenn $w = p$ ist.

Wortzähl-Algorithmus:

Parameter: Muster p der Länge m
Burrows-Wheeler-transformierter String T

Voraussetzung: Für jedes Zeichen x sind die Alphagrenzen F_x und L_x schon berechnet.

1) $w \leftarrow p[m]$
2) $k_w \leftarrow L_w - F_w + 1$
3) Wenn $m = 1$, liefere k_w zurück.
4) Für $i = m-1, m-2, \ldots, 1$
   Begin
5) $x \leftarrow p[i]$
6) $k_{xw} \leftarrow$ Anzahl der x'e in T zwischen F_w und L_w.
7) Wenn $k_{xw} = 0$, liefere 0 zurück.
8) $b_{xw} \leftarrow$ Anzahl der x'e in T vor F_w.
9) $F_{xw} \leftarrow F_x + b_{xw}$
10) $L_{xw} \leftarrow F_{xw} + k_{xw} - 1$
11) $w \leftarrow xw$
    End
12) Liefere k_w zurück.

Alle Schritte, bis auf Schritt 6) und 8), bei denen eine Anzahl von Zeichen in einem Intervall von T gezählt wird, brauchen konstante Zeit. Die Anzahl der Schleifendurchläufe ist höchstens m. Die Laufzeit, die dieser Algorithmus für eine Anfrage nach dem Muster p braucht, ist also $O(mK)$. Der Speicherbedarf ergibt sich aus dem Speicherbedarf obiger Hilfsdatenstruktur für die Häufigkeit einzelner Zeichen und dem Speicherbedarf dafür, den String T zu speichern, also $O(n/K \cdot |\Sigma| \log n + n \log |\Sigma|)$ Bits. Die Autoren des genannten Artikels komprimieren den String T mit einer einfachen Methode. Sie kodieren jedes aufeinanderfolgende Tripel der 125 möglichen Tripel von Zeichen aus $\{A, C, G, T, N\}$ mit einem Byte. So können sie die etwa drei Milliarden lange BWT T des menschlichen Genoms mit etwa 1 GB Speicherbedarf speichern und trotzdem schnell adressieren. Der von ihnen insgesamt benötgte Speicherplatz ist bei $K = 300$ etwas über 1 GB, also deutlich unter dem Speicherbedarf von Suffixbäumen.

4.5 Karten

4.5.1 Einleitung

Eine Karte (map) beschreibt die Anordnung von gewissen Merkmalen auf einer DNA-Sequenz zueinander. Solche Merkmale können Marker wie STSs oder ESTs sein oder andere Loci, die von Interesse sind, wie Gene. Eine genetische Karte (genetic map) verzeichnet die Abstände zwischen den Merkmalen gemessen in der Rekombinationshäufigkeit der Merkmale. Tendenziell gilt: Je weiter zwei Merkmale, zum Beispiel zwei Gene, auf der DNA entfernt sind, desto häufiger kann es passieren, dass bei der Vererbung die Allele der beiden Eltern neu kombiniert werden. Die übliche Einheit des Abstands ist hier *Centimorgan*. Eine physikalischen Karte (physical map) enthält die Reihenfolge der Merkmale und deren Abstände gemessen in Basenpaaren. Eine physikalische Karte von STSs auf einem Chromosom kann helfen, bei der Assemblierung ein neu sequenziertes Contig oder Scaffold, dessen Position innerhalb des Genoms zunächst unbekannt ist, auf dem Chromosom zu lokalisieren, wenn es STSs enthält, die in der Karte verzeichnet sind.

Die beiden nächsten Abschnitte beschäftigen sich mit algorithmischen Problemen, die bei der Erstellung einer physikalischen Karte auftreten.

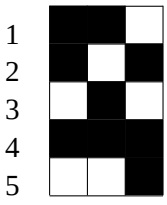
4.5.2 Hybridization Mapping

Wenn die STSs gefunden werden, sind meistens weder ihre Lage im Genom noch ihre relative Ordnung zueinander bekannt. Ebenso ist bei einer Klon-Bibliothek, wie sie etwa für die Sequenzierung verwendet wird, nicht die Reihenfolge dieser Klone auf der DNA bekannt.
Hybridization Mapping ist eine Methode, mit der im Idealfall gleichzeitig die Reihenfolge der STSs und der Klone bestimmt werden kann, wenn ermittelt wird (mittels PCR), welches STS sich auf welchem Klon befindet (siehe Abbildung 4.10).

Das Problem der aufeinanderfolgenden Einsen

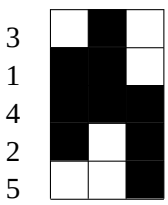
Gegeben sei eine binäre $n \times m$-Matrix M. Wir sagen diese Matrix hat die Eigenschaft der aufeinanderfolgenden Einsen (**C1P**, Consecutive Ones Property), wenn es eine Permutation der Zeilen gibt, so dass nach Anwendung der Permutation die Einsen in jeder Spalte aufeinanderfolgend sind.

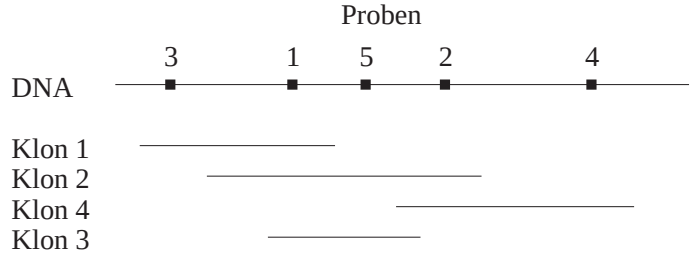
Beispielmatrix:



(= 1, = 0)

Folgende Permutation der Zeilen der Beispielmatrix zeigt, dass sie die C1P hat:



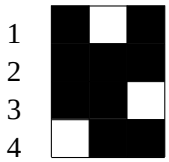


		Klon			
		1	2	3	4
	1	1	1	1	0
	2	0	1	0	1
Probe	3	1	0	0	0
	4	0	0	0	1
	5	0	1	1	0

		Klon			
		1	2	3	4
	3	1	0	0	0
	1	1	1	1	0
Probe	5	0	1	1	0
	2	0	1	0	1
	4	0	0	0	1

Abbildung 4.10: Oben: Klone und Proben. Die Proben sind z.B. STSs. Weder ihre Reihenfolge noch die der Klone sind bekannt. Unten: Die Matrizen haben bei Probe i und Klon j genau dann eine 1, wenn Probe i in Klon j vorkommt. Wenn die Zeilen der Matrix in der Reihenfolge der Proben in der DNA sortiert werden (rechts), sind die Einsen in jeder Spalte aufeinanderfolgend. Dann ist auch die relative Anordnung der Klone bekannt.

Beispielmatrix, die C1P nicht hat:



Wir setzen im Folgenden voraus, dass in jeder Spalte von M mindestens eine 1 steht und dass keine zwei Spalten gleich sind.

Für jede Spalte $j = 1, 2, \ldots, m$ sei

$$S_j := \{i \in \{1, \ldots, n\} \mid M_{i,j} = 1\}$$

die Menge der Zeilen, die in der j-ten Spalte eine 1 haben. S_j ist also eine Menge von Zeilen, die untereinander stehen müssen damit in der j-ten Spalte alle Einsen aufeinander folgen. Sei π eine Permutation von $\{1, \ldots, n\}$. Wir sagen *S_j ist ein Intervall bzgl. π*, wenn es $\ell, r \in \{1, \ldots, n\}$ gibt, so dass

$$S_j = \{\pi(\ell), \pi(\ell + 1), \ldots, \pi(r)\} =: \pi[\ell, r]$$

ist.

C1P-Problem: Sei M eine binäre Matrix wie oben. Finde heraus, ob M die C1P hat. Falls ja, finde eine Permutation π der Menge der Zeilen $\{1, \ldots, n\}$, so dass für jedes $j \in \{1, \ldots, m\}$ die Menge S_j ein Intervall bzgl. π ist.

In obigem Beispiel ist die Permutation $\pi = (3, 1, 5, 2, 4)$ eine Lösung.

Das C1P-Problem kann in folgenden 3 Schritten gelöst werden.

1. Zerlege die Menge der Spalten auf bestimmte Weise in disjunkte Teilmengen.

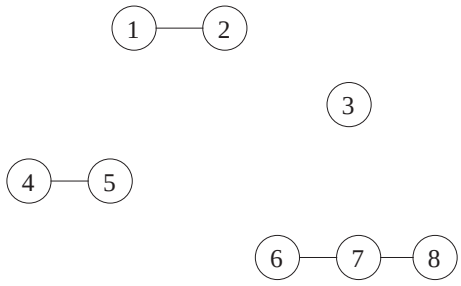
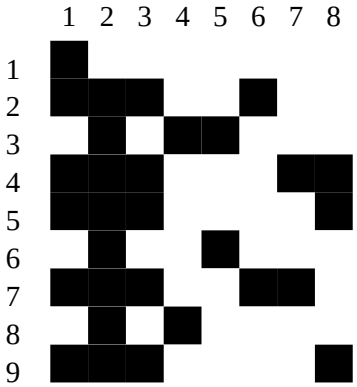


Abbildung 4.11: Überschneidungsgraph G der Matrix M. Die Zusammenhangskomponenten sind $\{1,2\},\{3\},\{4,5\},\{6,7,8\}$.

2. Löse dann für jede dieser Teilmengen, das Problem, falls es eine Lösung gibt.
3. Kombiniere die Lösungen für die Teilmengen der Spalten zu einer Lösung, die das C1P-Problem für alle Spalten löst.

Wir werden den Algorithmus bei folgender 9×8-Matrix M verfolgen.



Zerlegung der Spaltenmenge

Definition 4.14 Sei M eine binäre $n \times m$-Matrix. Der *Überschneidungsgraph* von M ist der ungerichtete Graph $G = (V, E)$ mit Knotenmenge

$$V = \{1, \ldots, m\}$$

und Kantenmenge

$$E = \{\{j,k\} \mid S_j \cap S_k \neq \emptyset,\ S_j \not\subset S_k,\ S_k \not\subset S_j\}.$$

Wenn $\{j,k\} \in E$ ist, sagen wir, dass sich Spalten j und k *überschneiden*.

Die Menge der Zusammenhangskomponenten von G ist die oben beschriebene Zerlegung der Spaltenmenge $\{1,\ldots,m\}$. Unten werden wir zeigen, dass M die C1P genau dann hat, wenn für jede Zusammenhangskomponente C von G die Teilmatrix die C1P hat, die man aus M erhält, wenn man alle Spalten $j \notin C$ streicht. Das bedeutet, dass man zunächst jeweils für Teile der Spalten das Problem unabhängig lösen kann.

Zusammenfügen der Zusammenhangskomponenten

Angenommen, man könnte für jede Zusammenhangskomponente von G das Problem lösen. Wir werden in diesem Abschnitt sehen, dass dann M die C1P hat.

Definition 4.15 Der *Inklusionsgraph* zur Matrix M ist der gerichtete Graph $G_I = (V_I, E_I)$ dessen Knotenmenge V_I die Menge der Zusammenhangskomponenten von G ist und dessen Kantenmenge gleich

$$E_I = \{(\alpha, \beta) \,|\, \alpha \neq \beta \in V_I, \text{ es existiert } a \in \alpha, b \in \beta, \text{ mit } S_b \subset S_a\}$$

ist.

Folgende Abbildung zeigt den Inklusionsgraphen für die Beispielmatrix M.

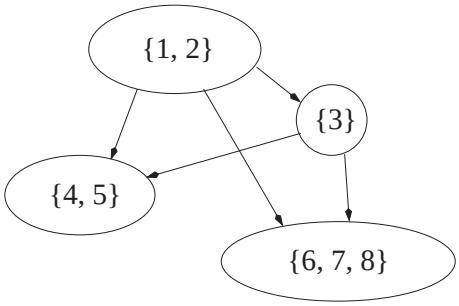


Abbildung 4.12: Inklusionsgraph G_I der Matrix M.

Lemma 4.16 Seien $\alpha \neq \beta \in V_I$ und sei $S_b \subset S_a$ für ein $a \in \alpha$ und ein $b \in \beta$. Dann gilt $S_{b'} \subset S_a$ für jedes $b' \in \beta$.

Beweis: Sei zunächst b', so dass $\{b, b'\} \in E$. b' ist also in G ein Nachbar von b. Weil b' und a in verschiedenen Zusammenhangskomponenten liegen, also $\{b', a\} \notin E$ gilt

$$S_{b'} \cap S_a = \emptyset \quad \text{oder } S_{b'} \subset S_a \quad \text{oder } S_a \subset S_{b'}$$

Die erste dieser drei Alternativen ist ausgeschlossen, weil $S_b \cap S_{b'} \neq \emptyset$ und $S_b \subset S_a$. Die dritte der drei Alternativen ist ausgeschlossen, weil dann $S_b \subset S_a \subset S_{b'}$ gelten würde, was ein Widerspruch zu $\{b, b'\} \in E$ wäre. Also muß die zweite Alternative gelten: $S_{b'} \subset S_a$. Also gilt für alle Nachbarn b' von b die Inklusion. Also auch für die Nachbarn der Nachbarn, usw.. ; also für die ganze Zusammenhangskomponente β.

□

Lemma 4.17 Jeder Inklusionsgraph ist azyklisch.

Beweis: Angenommen, er wäre es nicht und es gäbe also Zusammenhangskomponenten $\beta_1, \ldots, \beta_d$ ($d > 1$) von G mit

$$(\beta_1, \beta_2), (\beta_2, \beta_3), \ldots, (\beta_{d-1}, \beta_d), (\beta_d, \beta_1) \in E_I$$

Zur Vereinfachung der Schreibweise setze $\beta_{d+1} = \beta_1$. Dann gäbe es nach der Definition von G_I und nach Lemma 4.16 $a_i \in \beta_i$ $(i = 1, 2, \ldots, d)$ mit

$$\bigcup_{b \in \beta_{i+1}} S_b \subset S_{a_i} \tag{4.7}$$

Induktiv gälte dann also $\bigcup_{b\in\beta_1} S_b \subset S_{a_1}$. Dann müßte $\beta_1 = \{a_1\}$ sein, also nur aus diesem einen Element bestehen. Genauso könnten wir $\beta_i = \{a_i\}$ für alle $i = 1, 2, \ldots, d$ schließen. Dann würde die Inklusionskette (4.7) $S_{a_1} = S_{a_2} = \cdots = S_{a_d}$ liefern, was ein Widerspruch ist, weil wir angenommen haben, dass keine zwei Spalten von M gleich sind.

□

Korollar 4.18 Sei $(\alpha, \beta) \in E_I$. Dann gilt für alle $a \in \alpha$

$$\text{entweder } \bigcup_{b\in\beta} S_b \subset S_a \qquad \text{oder } (\bigcup_{b\in\beta} S_b) \cap S_a = \emptyset.$$

Beweis: Falls $a \in \alpha$ so ist, dass ein $b \in \beta$ existiert mit $S_b \subset S_a$, dann gilt nach Lemma 4.16

$$\bigcup_{b\in\beta} S_b \subset S_a.$$

Sei $a \in \alpha$ nun so, dass für jedes $b \in \beta$ S_b nicht Teilmenge von S_a ist. Sei $b \in \beta$ jetzt beliebig. Weil a und b in verschiedenen Zusammenhangskomponenten liegen, gilt

$$S_b \cap S_a = \emptyset \quad \text{oder } S_a \subset S_b \quad \text{oder } S_b \subset S_a.$$

Letzteres haben wir schon ausgeschlossen. Wenn der zweite Fall eintreten würde, gäbe es in G_I auch eine Kante (β, α) und G_I wäre nicht azyklisch. Es muß also $S_b \cap S_a$ gelten. Da b beliebig war, gilt

$$(\bigcup_{b\in\beta} S_b) \cap S_a = \emptyset.$$

□

Lemma 4.19 Wenn $\alpha \neq \beta$ und weder (α, β) noch (β, α) eine Kante im Inklusionsgraphen ist, gilt

$$\left(\bigcup_{a\in\alpha} S_a\right) \bigcap \left(\bigcup_{b\in\beta} S_b\right) = \emptyset.$$

Beweis: Übung

Satz 4.20 Die binäre Matrix M hat die C1P genau dann, wenn für jede Zusammenhangskomponente $\beta \in V_I$ von G die Teilmatrix, die man erhält, wenn man aus M alle Spalten $j \notin \beta$ streicht, die C1P hat.

Beweis: Wenn M die C1P hat, hat auch jede Matrix, die durch Streichen von Spalten aus M entsteht, die C1P. Zum Beweis der anderen Richtung nehmen wir nun an, dass $\beta_1, \beta_2, \ldots, \beta_d$ die Zusammenhangskomponenten von G in topologischer Ordnung bzgl. E_I sind. Diese Sortierung ist möglich, da G_I nach Lemma 4.17 azyklisch ist. Wir gehen induktiv vor. Angenommen, wir haben bereits eine Permutation π der Zeilen von M, so dass die Spalten in $\alpha := \beta_1 \cup \cdots \cup \beta_k$ aufeinanderfolgende Einsen haben. Jetzt möchten wir diese Permutation abändern, so dass dann die Spalten in β_{k+1} aufeinanderfolgend sind, ohne dass die Eigenschaft der aufeinanderfolgenden Einsen bei den Spalten in α zerstört wird. Sei

$$B := \bigcup_{b\in\beta_{k+1}} S_b$$

die Menge der Zeilen in denen in einer Spalte in β_{k+1} eine Eins steht. Bzgl. π ist jedes S_a mit $a \in \alpha$ bereits ein Intervall

$$S_a = \pi[\ell_a, r_a]$$

Weil $\beta_1, \ldots, \beta_k$ in der topologischen Ordnung vor β_{k+1} kommen, geht in G_I keine Kante von β_{k+1} zu einer dieser vorangehenden Zusammenhangskomponenten. Nach Korrolar 4.18 und Lemma 4.19 gilt dann für jedes $a \in \alpha$ entweder $B \subset S_a$ oder $B \subset \overline{S_a}$. Dabei bezeichne $\overline{S}$ für eine Spaltenmenge S das Komplement: $\overline{S} := \{1, \ldots, m\} \setminus S$. Die Menge der $a \in \alpha$, für die $B \subset S_a$ gilt, sei α_1, die Menge der a's, für die $B \subset \overline{S_a}$ gilt, sei α_2. Es ist dann

$$B \subset \bigcap_{a \in \alpha_1} \pi[\ell_a, r_a] \cap \bigcap_{a \in \alpha_2} \overline{\pi[\ell_a, r_a]} =: L$$

Seien ℓ und r, so dass $\pi[\ell, r]$ das kleinste Intervall ist, das L enthält (siehe Abbildung 4.13).

Die Schlüsselidee ist folgende. Weil für jedes $a \in \alpha_1$ das Intervall $\pi[\ell, r]$ in $\pi[\ell_a, r_a]$ enthalten ist, kann man die Zeilen zwischen ℓ und r in der bereits nach π permutierten Matrix beliebig weiterpermutieren, ohne dass die Spalten in α_1 die C1P verlieren, denn in diesen Spalten stehen in diesem Intervall nur Einsen. Für die Spalten in $a \in \alpha_2$ gilt, dass die Intervalle der Einsen $\pi[\ell_a, r_a]$ entweder auch ganz in $\pi[\ell, r]$ enthalten sind oder ganz im Komplement $\overline{\pi[\ell, r]}$. Sei α_2' die Menge der Spalten, die Ersteres erfüllen:

$$\alpha_2' = \{a \in \alpha \mid \pi[\ell_a, r_a] \subset \pi[\ell, r]\}$$

Für die Spalten in $\alpha_2 \setminus \alpha_2'$ gilt dasselbe wie für die Spalten in α_2. Eine Permutation der Zeilen innerhalb der Grenzen ℓ und r zerstört für sie nicht die C1P, weil in diesen Spalten zwischen ℓ und r nur Nullen stehen. Bleiben die Spalten in α_2'. L ist die Menge der Zeilen zwischen ℓ und r, in denen in jeder Spalte von α_2' eine Null steht. Wir ändern (bzgl. π) die Reihenfolge der Zeilen innerhalb des Intervalls $[\ell, r]$ so, dass die Zeilen in L benachbart sind. Z.B. legen wir diese Zeilen ganz nach oben (siehe Abbildung 4.13). Dann ist L bzgl. diser neu erhaltenen Permutation der Zeilen ein zusammenhängendes Intervall. L ist jetzt ein Intervall von Zeilen, das alle Zeilen enthält, in denen in einer Spalte von β_{k+1} eine Eins kommt und das wir beliebig permutieren können ohne die C1P für eine Spalte in α zu zerstören. Jetzt machen wir von der vorausgesetzten Permutation gebrauch, die die Einsen in den Spalten in β_{k+1} aufeinanderfolgend macht, und erhalten so durch Hintereinanderausführen von π und der oben implizit genannten Permutationen eine neue Permutation, bei der alle Spalten in $\beta_1 \cup \cdots \cup \beta_{k+1}$ die C1P haben.

□

Obiger Beweis ist konstruktiv, denn er beschreibt unter der Voraussetzung, dass die Zusammenhangskomponenten von G die C1P haben, einen Algorithmus für das Finden einer Permutationen der Zeilen, so dass die Einsen in den Spalten aufeinanderfolgend sind.

C1P-Problem für eine Zusammenhangskomponente vom Überschneidungsgraphen

In diesem Abschnitt wollen wir das Problem lösen, für die Spalten einer Zusammenhangskomponente von G eine Permutation zu finden, so dass in jeder dieser Spalten die Einsen zusammenhängend sind. Dieses Teilproblem läßt sich direkt schrittweise Spalte für Spalte lösen, weil bei einer geeigneten Auswahl der Reihenfolge der Spalten in der Zusammenhangskomponente, bis auf Symmetrie jeweils kein Spielraum für die Auswahl einer solchen Permutation bleibt.

Betrachten wir als Beispiel folgende Matrix A mit nur einer Zusammenhangskomponente.

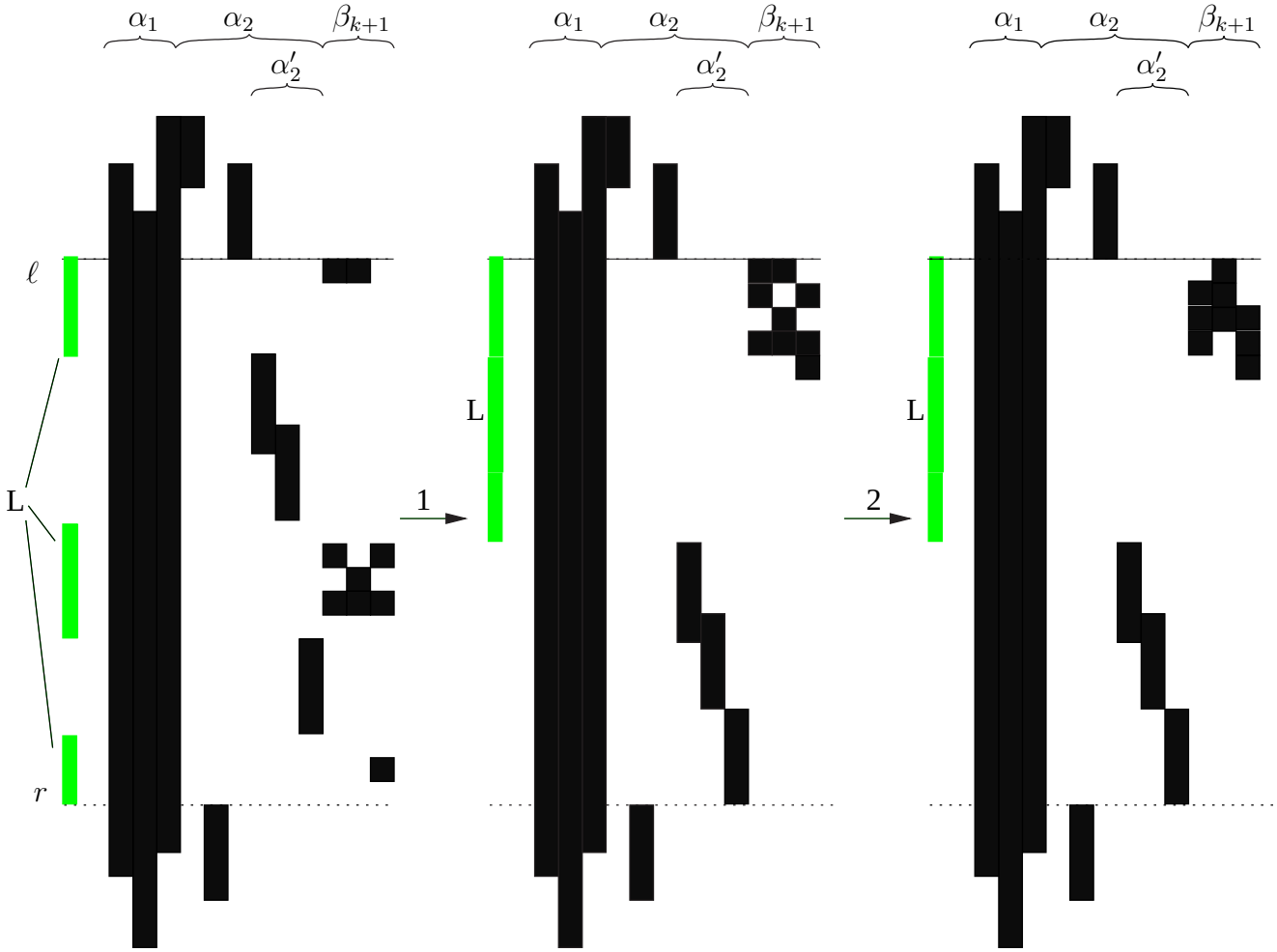
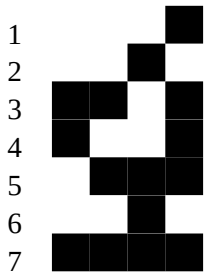
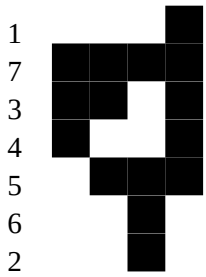


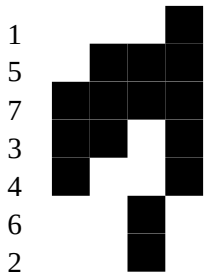
Abbildung 4.13: Das Bild zeigt dreimal die Matrix M mit den Spalten der ersten $k+1$ Zusammenhangskomponenten. Die Spalten in $\alpha = \alpha_1 \cup \alpha_2$ haben in der links gezeigten Permutation der Zeilen bereits zusammenhängende Einsen. Die Zusammenhangskomponente β_{k+1} soll zusammenhängende Einsen bekommen. Im ersten Schritt werden die Zeilen zwischen ℓ und r so umgeordnet, dass die Zeilen in L zusammenhängend sind. Im zweiten Schritt werden die Zeilen in B, also die Zeilen in denen in einer Spalte in β_{k+1} eine Eins steht, so umsortiert, dass sie aufeinanderfolgenden Einsen haben.



Wir fangen mit einer beliebigen Spalte an, z.B. mit Spalte $j = 1$. Wir permutieren die Zeilen so, dass die Einsen in Spalte j untereinander stehen. Hier vertauschen wir Zeilen 2 und 7.



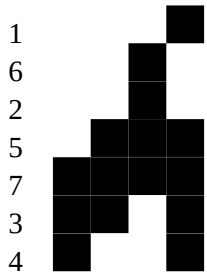
Falls die Zusammenhangskomponente weitere Spalten enthält, nehmen wir eine Spalte k, so dass die Kante $\{j, k\}$ im Überschneidungsgraphen ist. Hier z.B. Spalte $k = 2$. Spalten j und k überschneiden sich also: Es gibt Zeilen, die in beiden Spalten eine Eins haben, Zeilen, die in Spalte j eine Eins haben und nicht in Spalte k und Zeilen, die in Spalte k eine Eins haben und nicht in Spalte j. Das Intervall von Spalte k muss entweder über oder unter dem von Spalte j liegen. Da mit jeder Reihenfolge der Zeilen auch die umgekehrte Reihenfolge der Zeilen eine Lösung ist, kann man sich hier ohne Einschränkung für eine der Möglichkeiten entscheiden. Wir legen das Intervall der Einsen von Spalte 2 über das der Spalte 1:



Jetzt gehen wir induktiv vor. Wenn bereits Spalten j und k aufeinanderfolgende Einsen haben und $\{j, k\} \in E$ ist, nehmen wir eine Spalte ℓ, so dass die Kante $\{k, \ell\}$ im Überschneidungsgraphen ist. Im Beispiel ist $\ell = 3$ so eine Spalte. Eine Schlüsselbeobachtung ist nun, dass das Intervall der Einsen in der dritten Spalte und jeder weitere Spalte relativ zu den beiden vorangehenden Intervallen von Einsen nur eine Lage haben kann.

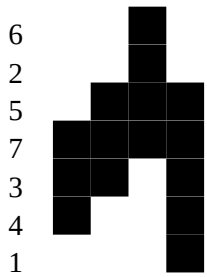
Wenn die Bedingung B: $|S_\ell \cap S_j| < \min\{|S_j \cap S_k|, |S_k \cap S_l|\}$ erfüllt ist, kann das Intervall der Einsen in Spalte ℓ nur oberhalb (bzw. in der gleichen Richtung wie k relativ zu j) von dem Intervall der Einsen von k liegen. Zur Begründung siehe Abbildung 4.14. Wenn die Bedingung B nicht erfüllt ist kann das Intervall der Einsen in Spalte ℓ – wenn die Positionierung überhaupt möglich ist – nur unterhalb (bzw. in der anderen Richtung wie k relativ zu j) von dem Intervall

der Einsen von k liegen. In diesem Fall kann es allerdings sein, dass es gar keine Lösung gibt. Im Beispiel ist $|S_\ell \cap S_j| = 1$, $|S_j \cap S_k| = 2$, $|S_k \cap S_l| = 2$, also ist die Bedingung B erfüllt und Spalte 3 muss oberhalb von Spalte 2 liegen.



Wir wählen jetzt wieder eine neue Spalte ℓ und bereits abgehandelte Spalten j und k mit $\{j,k\}, \{k,\ell\} \in E$. Im Beispiel ist jetzt $j = 2, k = 3, \ell = 4$.

Die Bedingung B ist diesmal nicht erfüllt. Deshalb muss das Intervall der Einsen von Spalte ℓ in der anderen Richtung relativ zu k liegen wie k zu j liegt, also unterhalb.



C1P-Algorithmus:

Parameter: binäre $n \times m$-Matrix M

1) Konstruiere den Überschneidungsgraphen G von M
2) Konstruiere den Inklusionsgraphen G_I von M
3) Sortiere die Knoten in G_I topologisch.
4) Sortiere die Spalten nach obiger topologischen Ordnung. Sortiere die Spalten innerhalb jeder Zusammenhangskomponente in der Reihenfolge einer Tiefensuche in G. Sei $\sigma_1, \ldots, \sigma_m$ die erhaltene Reihenfolge der Spalten.
5) Für $i = 1, 2, \ldots, m$

   Begin

6) Versuche, wie oben beschrieben die Zeilen von M zu sortieren, so dass Spalte i die C1P bekommt und Spalten 1 bis $i-1$ die C1P behalten.
7) Wenn Schritt 6) nicht möglich ist, breche ab: M hat nicht die C1P.

   End

Die Laufzeit für Schritt 1), für die naive Kontruktion des Überschneidungsgraphen, ist $O(m^2 n)$, weil es $O(m^2)$ Paare von Spalten gibt, und man für jedes Paar von Spalten n Zeilen überprüfen muß um zu entscheiden, ob dieses Paar eine Kante in G definiert. Schritte 2) bis 4) sind ebenfalls in $O(m^2 n)$ möglich. Die Schleife in 5) wird höchstens m mal ausgeführt. Jeder Schleifendurchlauf kann in $O(mn)$ gemacht werden. Die Gesamtlaufzeit des Algorithmus ist also $O(m^2 n)$.

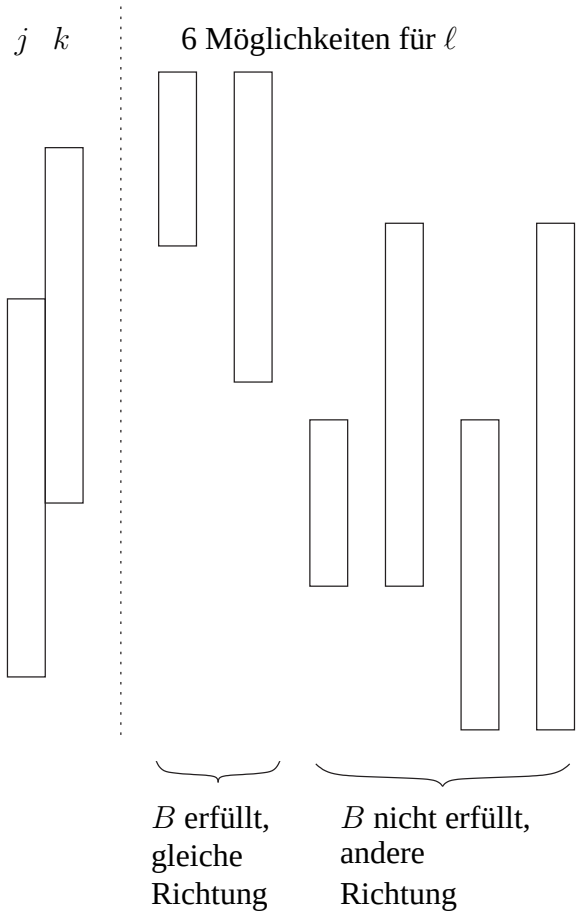


Abbildung 4.14: Die Möglichkeiten eine dritte Spalte ℓ relativ zu zwei anderen Spalten j und k zu Positionieren. Bei den beiden ersten Möglichkeiten ist die Bedingung $B : |S_\ell \cap S_j| < \min\{|S_j \cap S_k|, |S_k \cap S_\ell|\}$ erfüllt. Bei den vier letzten Möglichkeiten ist die Bedingung B verletzt. B ist genau dann erfüllt, wenn das Intervall der Einsen von Spalte ℓ, relativ zu dem von Spalte k in derselben Richtung liegt, wie das von Spalte k zu dem von Spalte j.

In Abbildung 4.15 ist der Ablauf dieses Algorithmus am Beispiel der Matrix M (Seite 52) dargestellt. Die Spalten können in diesem Fall in der Reihenfolge von links nach rechts behandelt werden, weil die Spalten von M bereits entsprechend geordnet waren.

4.5.3 Radiation-Hybrid Mapping

Siehe Abschnitt 16.6 in “Algorithms on Strings, Trees, and Sequences”, Dan Gusfield.

4.5.4 Map Alignment

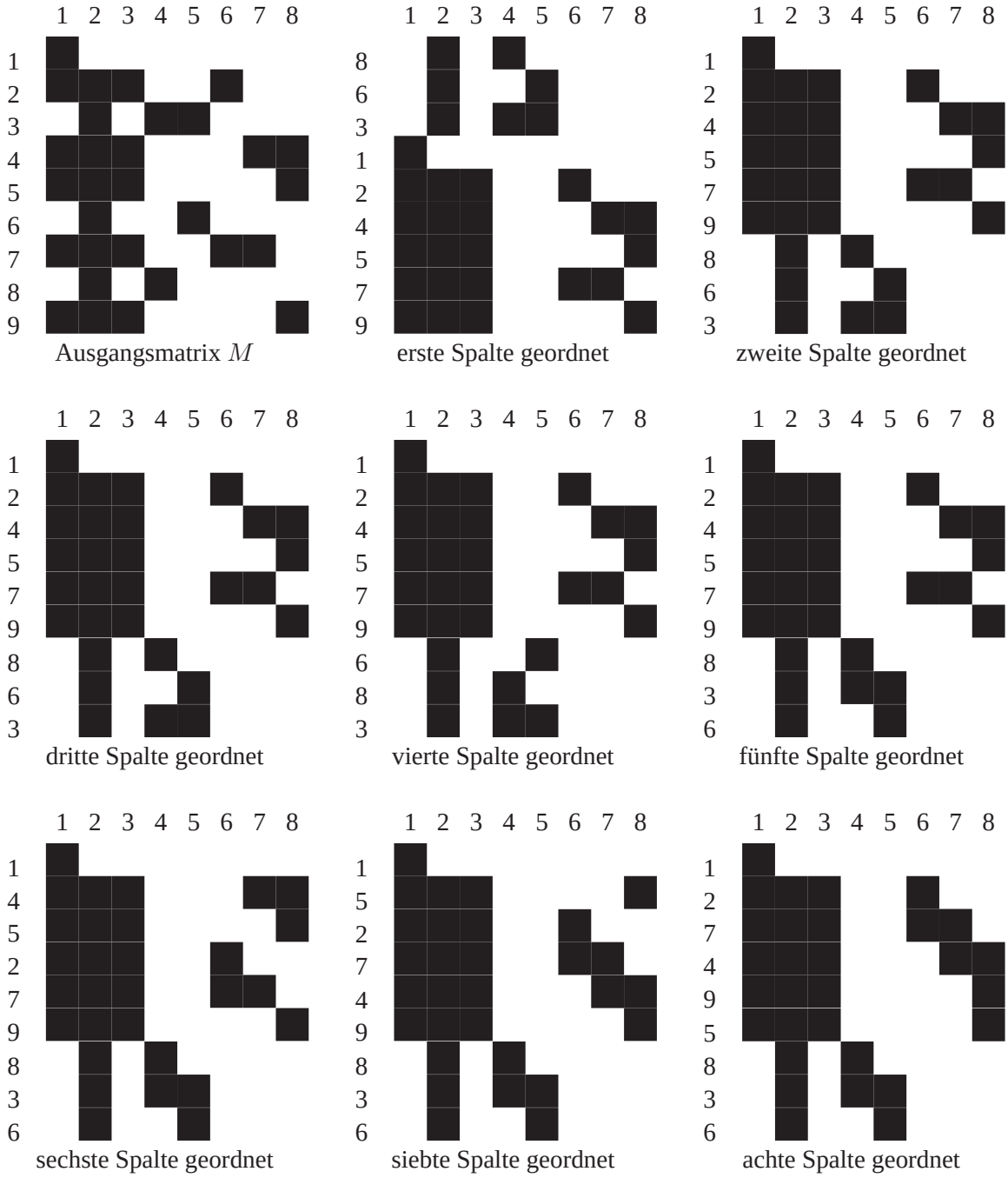


Abbildung 4.15: Ablauf des C1P-Algorithmus bei der Beispielmatrix.

Literaturverzeichnis

[BPM+00] Serafim Batzoglou, Lior Pachter, Jill P. Mesirov, Bonnie Berger, and Eric S. Lander. Human and Mouse Gene Sstructure: Comparative Analysis and Application to Exon Prediction. *Genome Research*, 10:950–958, 2000.

[FPU99] M. Frieze, Alan, P. Preparata, Franco, and Eli Upfal. Optimal reconstruction of a sequnce from its probes. *Journal of Computational Biology*, 6(3/4):361–368, 1999.

[GMP96] Mikhail S. Gelfand, Andrey A. Mironov, and Pavel A. Pevzner. Gene recognition via spliced sequence alignment. *Proc. Natl. Acad. Sci. USA*, 93:9061–9066, 1996.

[Gus99] Dan Gusfield. *Algorithms on Strings, Trees and Sequences.* University of Cambridge, 1999.

[HHHS03] Eran Halperin, Shay Halperin, Tzvika Hartman, and Ron Shamir. Hadling long targets and errors in sequencing by hybridization. *Journal of Computational Biology*, 10(3-4):483–497, 2003.

[HTSW03] John Healy, Elizabeth E. Thomas, Jacob T. Schwartz, and Michael Wigler. Annotating large genomes with exact word matches. *Genome Research*, 13:2306–2315, 2003.

[KMH94] A. Krogh, I.S. Mian, and D. Haussler. A hidden Markov model that finds genes in E. coli DNA. *Nucleic Acids Research*, 22:4768–4778, 1994.

[MW02] Rainer Merkl and Stephan Waack. *Bioinformatik Interaktiv*. Wiley-VCH, 2002.

[PEG00] Genís Parra, Blanco Enrique, and Roderic Guigó. GeneID in Drosophila. *Genome Research*, 10:511–515, 2000.

[Pev00] *Computational Molecular Biology: An Algorithmic Approach.* Bradford Book, 2000.

[PSS02] Mihai Pop, Steven L. Salzberg, and Martin Shumway. Genome sequence assembly: Algorithms and issues. *IEEE*, 2002.

[Set97] *Introduction to Computational Molecular Biology*. Brooks/Cole Publishing Company, 1997.

[SW03] Mario Stanke and Stephan Waack. Gene prediction with a hidden markov model and new intron submodel. *Bioinformatics*, 19 Suppl. 2:ii215–ii225, 2003.

[V+01] J. Craig Venter et al. The sequence of the human genome. *Science*, 291:1304–1351, 2001.

[Vit67] A. Viterbi. Error bounds for convolutional codes and an asymptotically optimal decoding algorithm. *IEEE Trans. Informat. Theor.*, IT-13:260–269, 1967.