Seminar Bioinformatik SS98, Bettina Lademann

Detecting Patterns in unaligned Sequences: Pratt1,2

Nachdem im gestrigen Vortrag über die theoretischen Grundlagen verschiedener Algorithmen zur Patternerkennung berichtet wurde, werde ich heute näheres zu einer Implementation eines solchen Verfahrens erklären. Zuerst werde ich Pratt1 vorstellen, die Nachteile erläutern und anschließend zu Pratt2 übergehen**.**

Pratt ist ein Werkzeug zur Mustererkennung in unalignierten Sequenzen. Und zwar werden konservierende Patterns gesucht.
Das heißt also, als Input erhält das Verfahren nur eine Menge positiver Beispiele von Sequenzen, deren Gemeinsamkeiten herausgefunden werden sollen. Der Output besteht dann aus einer Menge von Mustern, die nicht nur konservierend sind sondern, auch einen gewisses Maß an Informationsgehalt besitzen. Eine Stärke von Pratt, im Vergleich zu vorher entwickelten Verfahren, liegt darin, daß recht allgemeine Muster erkannt werden können.

Die Begriffe „Sequenz“, „Pattern“, „Matching“, „Segment“, ...
setze ich als vom gestrigen Vortrag her bekannt voraus. Im Falle von Gedächtnislücken bitte ich um Rückfragen.

Die gestern gegebene Funktion eines Patterns hat ausgereicht, die Muster zu beschreiben, die durch den Algorithmus von Neuwald und Green identifiziert werden können. Im folgenden werden wir den Begriff Pattern verallgemeinern, um anschließend Methoden vorstellen zu können, die auf dem derart erweiterten Suchraum arbeiten.

Wie beschreibt man ein Muster/Pattern?

Sei P ein Muster. Dann läßt es sich folgendermaßen darstellen:
$P = A_1 - x(i_{1,j1}) - A_2 - x(i_2 j_2) - \ldots - x(i_{p-1},,j_{p-1}) - A_p$
wobei die A_i nichtleere Teilmengen eines Alphabets Σ sind, und $i_1 \le j_{1,} i_2 \le j_{2,} \ldots, i_{p-1} \le j_{p-1}$ ganze Zahlen.
Die A_i bezeichnen wir als **Komponenten**. Eine Komponente A_i ist singulär, falls #(Ai) = 1, sonst ist sie ambig/uneindeutig.
Die x(i, j) stehen für **Wildcardregionen**,d.h., anstelle von x(i,,j) stehen mindestens i und höchstens j beliebige Zeichen aus dem Alphabet. Falls i = j, ist die Wildcardregion **fix**, ansonsten ist sie **flexibel**. Fixe Wildcardregionen schreibt man als x(j). Gilt i = j = 0, so ist die Wildcardregion leer. Leere Wildcardregionen werden nicht aufgeschrieben.
Die **Flexibilität** einer Wildcardregion x(i,j) ergibt sich aus der Differenz i-j,

als **Produkt der Flexibilitäten** bezeichnet man den Term $\prod_{k=1}^{p-1}(j_k - i_k + 1)$.

L(P) gibt die **maximale Länge** eines Segments an, das Muster P matcht: $L(P) = p + \sum_{k=1}^{p-1} j_k$.

Die geänderte Definition des Patterns zieht natürlich auch eine Veränderung des Begriffes „Matching" nach sich: eine Komponente A_i matcht ein Symbol eines Strings *a* genau dann, wenn gilt: $a \in A_i$, eine Wildcardregion x(i,j) matcht einen Teilstring genau dann, wenn dieser aus k beliebigen Symbolen mit $i \le k \le j$ besteht. Ein Pattern P matcht ein Segment S genau dann, wenn aufeinanderfolgende Elemente des Patterns aufeinanderfolgende Elemente des Segments matchen. Ein Pattern matcht eine Sequenz, wenn es ein Segment von dieser matcht.

Beispiel: Muster und Bezeichnungen
Sei Muster P = A-x(1,3)-C.
Flexibilität: 2 , Anzahl Komponenten: 2, maximale Länge L(P): 5;
mögliche Segmente, die P matcht: AAC, ABC, ACCCC,...

Pratt kann Muster mit uneindeutigen/ambigen Komponenten und flexiblen Wildcard-Regionen finden. Das vergrößert den Suchraum natürlich erheblich. Obwohl bei einer Suche von Pratt nicht alle möglichen Muster generiert und durchgetestet werden, kann das Verfahren sich übermäßig ressourcenfeindlich verhalten. Dem soll der Anwender entgegenwirken, indem er den Suchraum geeignet restriktiert:
er wählt die maximale zulässige Länge L(P) eines Musters P, die maximale Anzahl von Komponenten pro Muster, die maximale Länge einer Wildcard-Region sowie maximale Flexibilität und Häufigkeit einer flexiblen Wildcard-Region und das maximale Produkt der Flexibilitäten.
Außerdem bestimmt er die Minimalzahl N_{min} von Sequenzen, die zu einem Muster passen, damit Pratt es als Matching bewertet und gibt die Menge der Komponenten an, die auftreten dürfen.
Allgemein also wird der Suchraum C durch folgendes 7-Tupel beschrieben:

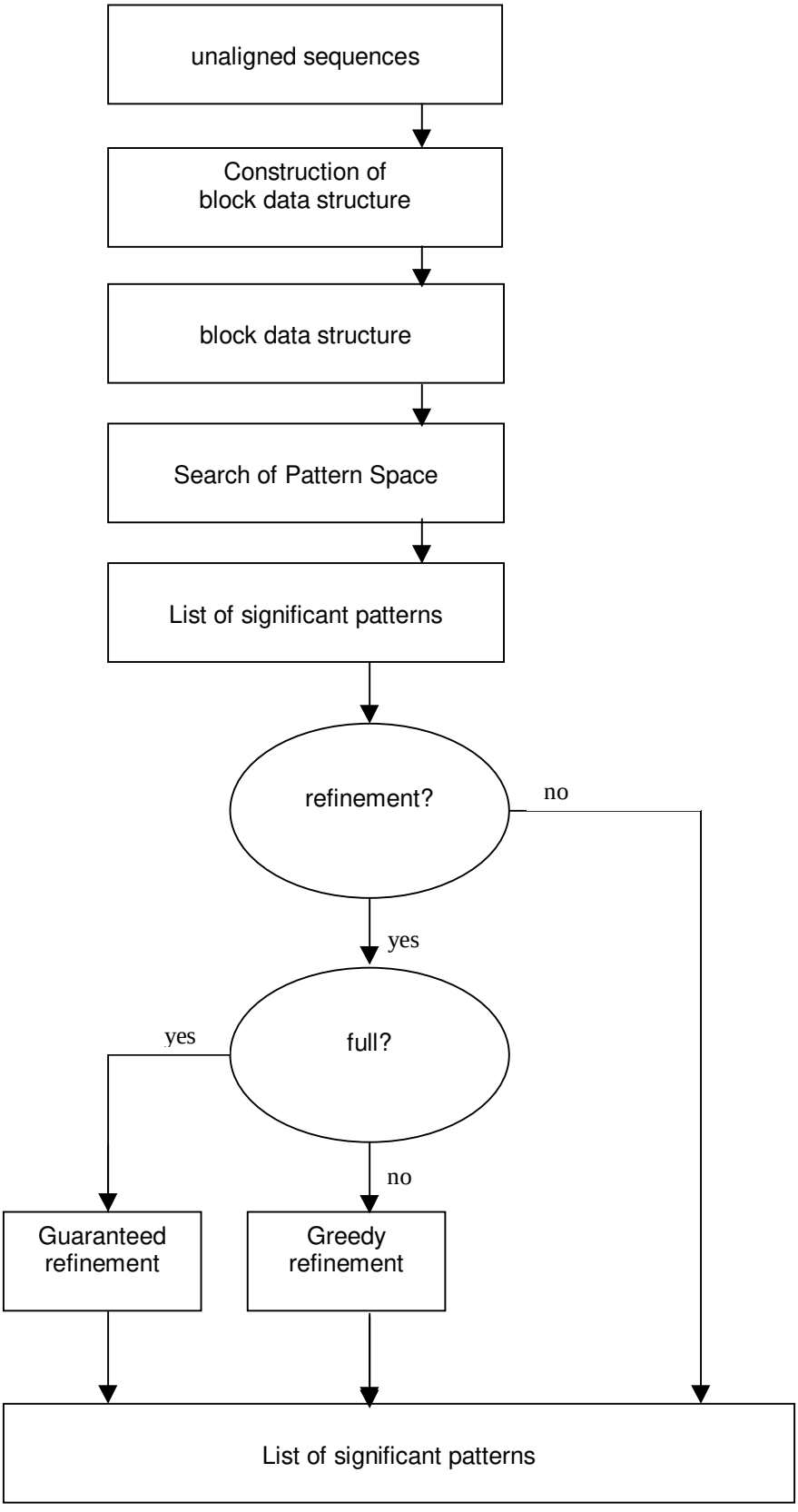
$$B = (A, P, L, W, F, N, FP)$$

$$\begin{aligned} C = \{ \quad & A_1 - x(i_1, j_1) - A_2 - x(i_2, j_2) - \ldots - x(i_{p-1}, j_{p-1}) - A_p \quad | \\ & (p \le P) \wedge \left(p + \sum_{k=1}^{p} j_k \le L \right) \qquad \wedge \\ & \forall k : 1 \le k \le p : (A_k \in A \wedge i_k \le j_k \le W \wedge j_k - i_k \le F) \quad \wedge \\ & \#(\{k | (1 \le k \le p-1) \wedge (j_k > i_k) \}) \quad \le N \qquad \wedge \\ & \prod_{k=1}^{p-1} (j_k - i_k + 1) \le FP \qquad \} \end{aligned}$$

Dabei muß beachtet werden, daß jede Restriktion des Suchraums zwangsläufig die Menge der erkennbaren Muster begrenzt. Daher ist Pratt stark in mehreren Durchläufen: man startet ihn zuerst sehr restriktiv, und je nach Mächtigkeit der Ausgabe werden die Restriktionen nach und nach gelockert.
Pratt gestaltet seine Suche anhand einer geschickten Tiefensuche: er gewinnt Effizienz, indem er bei der Durchforstung des Suchbaums möglichst viele Teilbäume abholzt.
Seine Methode umfaßt eine Weiterentwicklung der von Neuwald & Green vorgeschlagenen Methode zur Identifikation von Mustern in unalignierten verwandten Proteinsequenzen. So benutzt auch Pratt die Idee der Blockstruktur.

Die Ausgabe wird in zwei Phasen ermittelt: zuerst werden recht grobe Muster ausgewählt, um den Suchraum auszudünnen. Im zweiten Schritt werden die so gefundenen Muster dann weiter verfeinert. Das kann entweder- je nach Anwenderwunsch- mit einem teuren Algorithmus, der ausschöpfend sucht, geschehen, oder aber mit einem billigen, der aber nicht mit unbedingt alle konservierten Muster findet.

Programmstruktur von Pratt1



Die Methode

Schritt 1: grobe Suche

Der erste Teil des Algorithmus‘ ist lediglich eine Variante der Methode von Neuwald und Green insofern, daß auch Muster mit flexiblen Wildcard-Regionen generiert werden.
Das Prinzip ist folgendes: Pratt generiert ein Muster. Entweder wird es erweitert oder verworfen. Die Suche erfolgt rekursiv: sei P das momentan betrachtete Muster, N_{min} die Mindestzahl der zu matchenden Sequenzen. Falls P matcht, werden alle möglichen einfachen Erweiterungen P´ des Musters betrachtet. Ein Muster, dessen Erweiterungen alle verworfen worden sind, wird ausgegeben, falls seine Signifikanz einen gewissen Schwellwert übersteigt..
Was ist eine einfache Erweiterung und wie berechnet man die Kardinalität der Menge, die P matcht? Eine einfache Erweiterung hat die Form P‘ = P- x(i,j) - a, wobei “x(i,j)“ eine flexible Wildcard-Region bezeichnet und “a“ eine Aminosäure. Erweiterungen werden immer auf der rechten Seite des Musters angehängt. Das geschieht aufgrund der asymmetrischen Blockstruktur.
Die Menge der k-Segmente, die P matchen, war M_P, für die Menge $M_{P'}$ der k-Segmente, die P‘ matchen, gilt $M_{P'} \subseteq M_P$. Im Falle von nichtflexiblen Wildcard-Regionen, also P‘ = P - x(i) - a, ließe sich $M_{P'}$ folgendermaßen beschreiben: $M_{P'} = M_P \cap b_{L(P)+i+1,a}$, wobei $b_{j,a}$ ein Segment bezeichnet, an dessen j.ter Position ein ’a‘ steht. Dieser Schnitt kann effizient unter Verwendung der Blockdatenstruktur berechnet werden.
Für flexible Wildcard-Regionen sieht die Konstruktion etwas komplexer aus. Hier ist ja die Position von der Aminosäure ‘a‘ im neuen Muster P‘ nicht festgelegt. Auch die Länge von P ist variabel, da bereits P flexible Regionen enthalten kann. Daher müssen wir M_P mit einer Vereinigung von mehreren Segmenten der Form $b_{j,a}$ schneiden.
Um ein flexibles Muster zu beschreiben, nimmt man sich die zugehörige Menge von fixen Mustern zu Hilfe. Diese Menge wird im folgenden F(P) genannt. D.h. wenn ein Muster P beispielsweise die Wildcard-Region x(2,3) enthielte, bestände F(P) aus Mustern mit den fixen Wildcard-Regionen x(2) oder x(3) anstelle von x(2,3). (zum Beispiel: sei P= D-x(2,3)-E-x-A-x(4,5)-G. Dann ist F(P)= { D-x(2)-E-x-A-x(4)-G; D-x(2)-E-x-A-x(5)-G; D-x(3)-E-x-A-x(4)-G; D-x(3)-E-x-A-x(5) }.)
Formal schreibt sich das dann so:

$$F(P) = \bigcup_{i_1 \le k_1 \le j_{1..}, i_{p-1} \le k_{p-1} \le j_{p-1}} \left\{ \mathrm{A}_1 \text{ - } \mathrm{x}(\mathrm{k}_1) \text{ - } \mathrm{A}_2 \text{ - } \mathrm{x}(\mathrm{k}_2) \text{ - ... } \mathrm{A}_{\mathrm{p-1}} \text{ - } \mathrm{x}(\mathrm{k}_{\mathrm{p-1}}) \text{ - } \mathrm{A}_{\mathrm{p}} \right\}$$

bzw. rekursiv:

$$F(P') = \bigcup_{i \le k \prec j, Q \in F(P)} \{ Q - x(k) - a \}$$

$$M_P = \bigcup_{Q \in F(P)} M_Q$$

Und M_P kann jetzt so beschrieben werden:

Das ist dann wieder effizient berechenbar, da die Q ja fixe Muster sind.

Und für $M_{P'}$ gilt dann:

$$M_{P'} = \bigcup_{i \le k \le j, Q \in F(P)} \{ M_Q \cap b_{L(Q)+k+1,a} \}$$

Zu Beginn des Algorithmus wird folgendermaßen initialisiert:
P ist das leere Muster, und M_P dieGesamtheit aller k-Segmente.

$$P = \varnothing ;\ M_P = B_U^k ;$$

Schritt 2: Verfeinerung der Muster

Hier werden die in Schritt 1 gefundenen Muster weiter verfeinert, indem man als Komponenten auch Mengen zuläßt. Pratt stellt für Schritt 2 zwei unterschiedliche Methoden zur Verfügung: a ist ein greedy-Algorithmus, ressourcensparend, aber ohne Garantie, alle erkennbaren konservierten Muster zu finden, b ist der sichere Verfeinerungsalgorithmus, der im Vergleich aber sehr kostspielig ist.
Für den Anwender heißt das wieder: erst mal schauen, was 2a bringt, und falls dann noch Bedarf an Verfeinerung besteht, sich Zeit und Speicher nehmen für einen Lauf von 2b.

Welche Muster können gefunden werden?
2a findet Verfeinerung von Mustern, die aus Ersetzung von fixen Wildcardpositionen durch ambige Komponenten hervorgehen, 2b kann darüberhinaus auch flexible Positionen ersetzen.

Beispiel:
Wir haben eine Menge von Sequenzen, zu denen folgendes Muster paßt:
P = D-x(1,2)-[ILV]-E-x(2)-[DE]-F.
Die Parameter sind mit Defaultwerten besetzt, in der Liste der zulässigen Komponenten finden sich sowohl die Menge {D, E} als auch {I, L,V}.
Der erste Schritt findet: P1 = D-x(2,3)-E-x(3)-F.
Einfache Verfeinerung ergibt: P2a = D-x(2,3)-E-x(2)-[DE]-F.
Die sichere Verfeinerung bestimmt: P2b = D-x(1,2)-[ILV]-E-x(2)-[DE]-F.
Verfahren 2a konnte also die dritte Position der zweiten Wildcardregion ersetzen. 2b findet eine weitere Ersetzung in der flexiblen Wildcardregion.

Verfahren 2a ist nicht nur nicht fähig, Ersetzungen flexibler Wildcardpositionen vorzunehmen, es bietet –wie schon erwähnt- nicht einmal die Sicherheit, alle möglichen Verfeinerungen fixer Wildcardregionen zu finden.

Schritt 2a: Greedy-Algorithmus

Dieser Algorithmus kann Positionen in fixen Wildcard-Regionen durch ambige Komponenten ersetzen, sowie an die rechte Seite eines bereits gefundenen Musters ambige Komponenten und fixe Widcard-Regionen anhängen.

Auch hier ist die Vorgehensweise rekursiv:
Man betrachte Pattern P. Die rechte Seite wird mit Wildcards aufgefüllt, d.h. bis zur Maximallänge, die ein Muster haben darf (wurde in der Parameterliste gesetzt).
Das Ziel ist, eine fixe Wildcard-Position durch eine Menge R von Buchstaben so zu ersetzen, daß das modifizierte Muster immer noch mindestens N_{min} Sequenzen matcht und die ersetzende Menge natürlich in der Parameterliste, im folgenden L genannt, aufgeführt war oder aber Teilmenge einer Menge aus L war.
Dazu werden von links nach rechts alle fixen Wildcard-Positionen durchgecheckt:
sei j die betrachtete Position im Muster. Jetzt wird für jeden Buchstaben r_i im Zähler $n(r_i)$ notiert, wieviele Sequenzen aus M_P an ihrer j-ten Position ein r_i aufweisen. Die $n(r_i)$ sollen dann in geordneter Form vorliegen, z.B in einer Liste N (mit den zugehörigen Funktionen *pop( ), empty( ),..)*
Jetzt wird eine Menge R von Buchstaben zusammengesucht nach folgender Strategie:

```
R := leere Menge;
match := 0;
while((match < N_min) & !(empty(N))                    // N ist nicht leer
{
        r = pop(N);
        falls es ein M aus L gibt mit: (R ∪ {r}) ⊆ M,
        dann
        {
                R:= R ∪ {r};
                match := match + n(r);                 // n(r) Häufigkeit des Vorkommens von r
        };
}
```

Falls am Ende match ≥ N_{min} ist, hat man ein Muster P‘ gefunden, daß aus P hervorgeht durch Ersetzung der i.ten Position mit R. Das Verfahren fährt jetzt fort, indem es die Wildcard-Positionen von P und P‘ links von Position i weiter durchsucht.

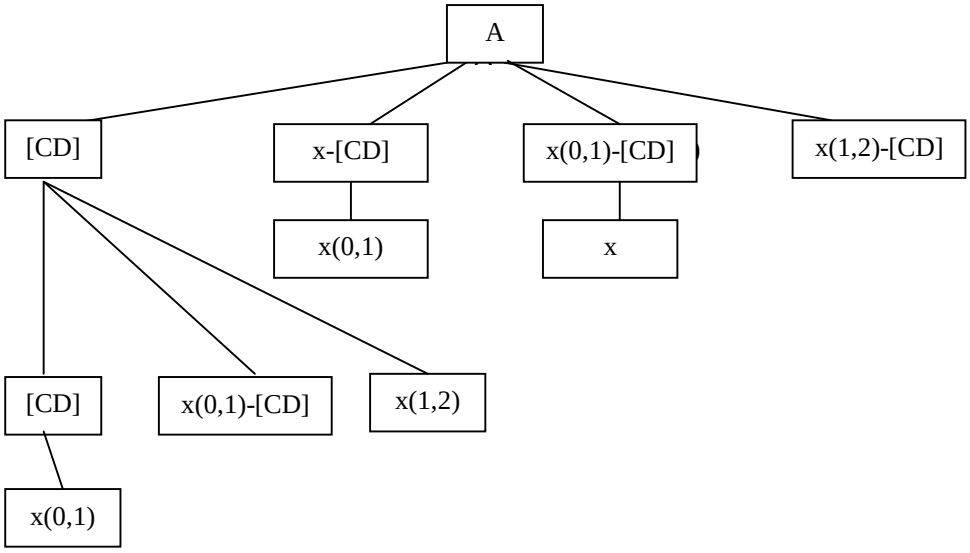
Beispiel:
im ersten Schritt wurde Muster P gefunden:
P = A-x-C
Häufigkeiten verschiedener Buchstaben an zweiter Position:
A: 50
I: 33
L: 27
V: 30
D: 4
N_{min} = 70.
Mengen in L:
{A,D}; {I,L,V}

Der Algorithmus wählt R = A, nimmt D hinzu, verwirft diese Wahl aber wieder, da die Kardinalität von R kleiner als 70 ist. {I,L,V} wäre zulässige Verfeinerung, wird aber nicht gefunden.

Schritt 2b: sichere Verfeinerung

dieser kostenintensive Algorithmus kann nicht nur Stellen in fixen Wildcard-Regionen sondern auch in Regionen variabler Länge durch Mengen ersetzen, sowie Muster nach rechts (bis zu Maximallänge) erweitern. Auch hier wird als Suchraum die Menge der in Schritt 1 gefundenen Muster gesetzt.
Das eigentliche Verfahren verhält sich wie Schritt 1, nur daß als Komponenten auch alle Mengen, die in der Parameterliste stehen, zugelassen werden. Aus der hierfür erforderlichen erweiterten Blockstruktur folgt die hohe Komplexität des Verfahrens.
Das bedeutet: jedes in Schritt 1 gefundene Pattern dient als Ausgangspunkt für einen weiteren Lauf des Algorithmus‘ wie in Schritt1, wobei statt $B_k^U \; M_P$ als Suchraum gesetzt wird und die Blockdatenstruktur für zulässige uneindeutige/ambige Komponenten gebildet wird.
Jede Wildcardregion wird durch alle möglichen einfachen Erweiterungen so ersetzt, daß die

Beispiel:
P = A – x(2,3) – C wurde in Schritt 1 gefunden. {C,D} sei einzige zulässige ambige Komponente.
Jetzt wird die Wildcardregion x(2,3) anhand einfacher Erweiterungen ersetzt:



Signifikanz:

zur Bewertung der gefundenen Muster wird –anders als im Algorithmus von Neumann und Greene- ein Maß verwendet, das den Informationsgehalt widerspiegeln soll, sodaß sich die Signifikanz des Musters aus der Summe seiner Komponenten angeben läßt.Die Komponenten gehen in diese Rechnung positiv ein, während für die Flexibilitäten Strafabzüge verbucht werden: wenn also I der

Informationsgehalt ist, c eine Konstante, die durch erfolgreiches Ausprobieren gewählt wurde, gilt:

$$I(P) = \sum_{i=1}^{p} I(A_i) - c\sum_{k=1}^{p-1}(j_k - i_k)$$

Für den Informationsgehalt einer Komponente gilt:

$$I(A_i) = -\sum_{a \in S}[p_a \log_2(p_a)] + \sum_{a \in A_i}\left[\frac{p_a}{p_{A_i}} \log_2\left(\frac{p_a}{p_{A_i}}\right)\right]$$

wobei S die Menge bzw. das Alphabet der Aminosäuren ist, p_a die Häufigkeit desVorkommens einer Aminosäure(das ist ein aus der Datenbank SWISS-PROT gewonnener Erfahrungswert),
und $p_{A_i} = \sum_{a \in A_i} p_a$ angibt, wie oft ein Element der Menge A_i vorkommt.

Komplexität:

Die Laufzeit des Verfahrens ist exponentiell zur maximalen Tiefe des Suchbaums, das ist die maximale Patternlänge. Zwar kann der Suchraum durch restriktive Wahl der Parameter eingegrenzt werden, das verbessert aber nicht die theoretisch ableitbare Ordnung des Verfahrens, bringt lediglich Zeitvorteile in der Anwendung.

Probleme:

Die Strategie, die Pratt verfolgt, heißt also: stelle einen Suchbaum auf, wobei die Wurzel aus dem leeren Muster besteht, die Nachfolger eines Knoten aus allen einfachen Erweiterungen des Musters des Knotens. Durchlaufe den Suchbaum nun in Tiefensuche, wobei Teilbäume, die an einem Knoten hängen, dessen Muster nicht genügend viele Sequenzen matcht, abgeschnitten werden.
Trotz dieser Einsparungen gibt es Fälle, in denen Pratt recht ineffizient arbeitet:
z.B wenn er auf einer Menge von Sequenzen arbeitet, die durch ein sehr starkes Muster konserviert werden: angenommen, er findet das Muster L-S-L-V-L-S-V-C-V-L. Dann berechnet er zusätzlich viele schwächere Muster, wie zum Beispiel L-x-L-V-L-S-V-C-V-L, L-x(2)-V-L-S-V-C-V-L, L-x(3)-L-S-V-C-V-L, oder L-S-(x)-V-L-S-V-C-V-L , usw., deren Erkennung keine wirkliche Informationsbereicherung darstellt.
Desweiteren verhält Pratt sich ineffizient bei der Identifikation von Mustern, die nur zu einer kleinen Untermenge von Sequenzen passen sollen, und findet keine Muster, die nur ein oder zwei Komponenten besitzen. Auch ist das Signifikanzmaß, das Pratt verwendet, theoretisch nicht untermauert.

Pratt2

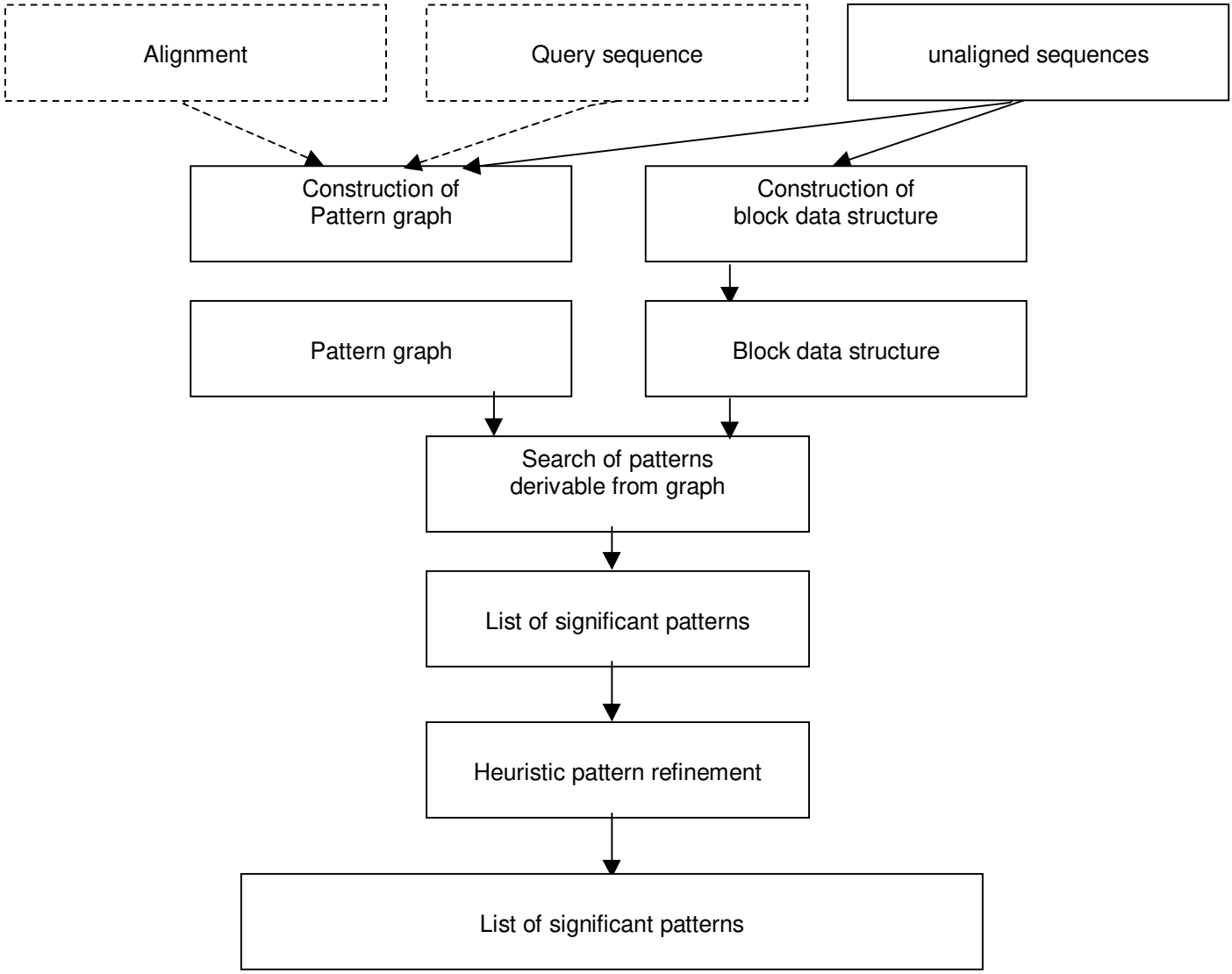
Abhilfe in einigen der oben genannten Probleme schafft Pratt2, der im folgenden nur skizziert werden soll:

Vorherrschende Idee ist dieEinführung eines „Pattern-Graphs“ und die Bestimmung erlaubter Wege darin, so daß die Wege Mustern entsprechen und die Abarbeitung aller Wege das sichere Erkennen aller konservierten Muster verspricht.

Diese Vorgehen verrschafft mehrere Vorteile: der Graph kann aus verschiedenen Voraussetzungen konstruiert werden, was auch Mustererkennung in alignments erlaubt. Außerdem kann der Suchbaum noch effizienter beschnitten werden: ein wichtiges Werkzeug hierzu ist die Definition von Generalisationen oder Verallgemeinerungen eines Musters.

Falls ein Nachfolger genauso oft matcht wie sein Vorfahr, werden die weiteren Nachfolger weggeschnitten, wobei die Nachfolger in einer Ordnung betrachtet werden, die garantiert, daß kein schwächerer Sohn auf Kosten eines stärkeren überlebt.

Programmstruktur von Pratt2

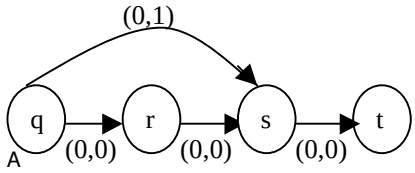


Ich werde definieren, was man unter einem Pattern-Graphen versteht, dann die Konstruktion erklären und anschließend zeigen, wie dann alle Muster aus solch einem Graphen abgeleitet werden können.

Pattern-Graph:

ein Pattern-Graph wird angegeben durch einen gerichteten G = (V,E). Die Knotenmenge V repräsentiert Musterkomponenten, die Kantenmenge E steht für Wildcardregionen.
Jeder Knoten v aus V ist beschriftet mit einer Komponente $\alpha(v)$, jede Kante e = (u -> v) aus E hat zwei Beschriftungen: $\delta(e)$ gibt die minimale Länge der zugehörigen wildcardregion an, $\delta'(e)$ die maximale Länge. Ein Pfad $p = u_1,u_2,...u_n$ in G definiert ein Muster durch die Zuordnung $\Pi(p) = \alpha(u_1)$-$x(\delta(u_1 \rightarrow u_2), \delta'(u_1 \rightarrow u_2))$-$\alpha(u_2)$-...-$\alpha(u_n)$.
Die Länge eines Pfades p soll definiert sein durch die Anzahl der Knoten aus p.

Beispiel:



Durch den Pfad p = q,s,t wird dann das Muster $\Pi(p)$ = A – x(0,1) – C – D dargestellt.

Die Konstruktion des Patterngraphs:

Pratt2 erlaubt drei Möglichkeiten zur Konstruktion des Graphen:
-multiple alignments
-Suchsequenz
-kürzeste Sequenzen.
Mit der ersten Möglichkeit gefundene Patterns werden auf jeden Fall das der Konstruktion zugrundeliegende Alignment matchen, bei der zweiten Möglichkeit werden nur Patterns generiert, die die Suchsequenz matchen. Mit der dritten Möglichkeit werden die Sequenzen gefunden, die mindestens Nmin Sequenzen matchen, wie man das von Pratt1 gewohnt war. Im folgenden werde ich nur über diese dritte Möglichkeit berichten.

Sei $k = N_{min}$ und n die Anzahl der Sequenzen, die uns vorliegen.
Wir konstruieren den Graphen aus den (n-k+1) kürzesten Sequenzen. Ein Muster, das k Sequenzen matcht, paßt zu mindestens einer Sequenz aus dieser Menge, ein Muster, das aus dem Graphen abgeleitet wurde, paßt auf jeden Fall zu einer der dem Graphen zugrunde liegenden Sequenzen.
Die Konstruktion an sich sieht so aus:

-Konkateniere die (n-k+1) kürzesten Sequenzen zu einem String $S=s_1...s_l$.

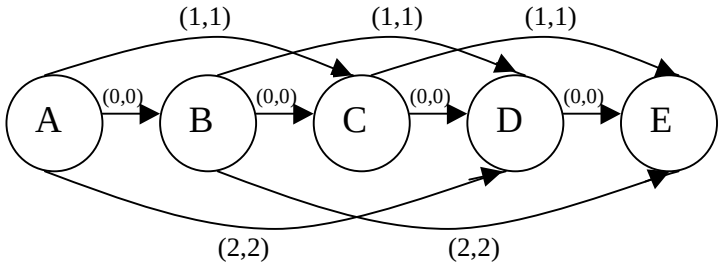
-erzeuge für jedes si einen Knoten ui mit Label $\alpha(u_i)=s_i$.

-erstelle von jedem Knoten u_i Kanten zu allen Knoten u_j (sofern u_i und u_j in derselben Ausgangsfrequenz lagen), so daß die Bedingung erfüllt ist:

$i < j \leq \min(i+W+1,l)$ und beschrifte die Kante (u_i ->uj) mit $(\delta_{ij}, \delta'_{ij}) =(j-i-1,j-i+1)$.

Beispiel:

aus der Frequenz ABCDE wird dann folgender Graph konstruiert:



einfache Tiefensuche:

Wie auch schon bei Pratt1 ist die Vorgehensweise rekursiv: das jeweilige Muster wird mit einer einfachen Erweiterung versehen. Anschließend wird das Muster generalisiert.

Eine einfache Erweiterung entspricht einer Verlängerung eines Pfades im Graphen:

Falls der alte Pfad $p = v_1,...v_k$, betrachtet man alle Kanten, die von v_k ausgehen. Seien $w_1,...,w_L$ die Endknoten dieser Kanten. Dann entstehen die einfachen Erweiterungen $P_l = P\text{-}x(i_l,,j_l)\text{-}A_l$ durch Hinzunahme einer Kante zum Pfad: $P_l = P\text{-}x(\Delta(v_k\text{->}w_l),\delta'(v_k \text{->}w_l)\text{-}\alpha(w_l)$.

Jetzt werden die P_l generalisiert:

Die Wildcardregion $x(i_l,,j_l)$ wird bis zum erlaubten Maß an Flexibilität verallgemeinert, und A_l wird durch eine Menge höherer Kardinalität ersetzt.

Die neu generierten Muster werden dann getestet in bezug auf ihre Konservierungseigenschaften, um dann mit anhand ihrer Signifikanz in eine Prioritätenschlange eingeordnet zu werden.

Beispiel:

Gegeben sei der Graph aus dem vorherigen Beispiel, außerdem maximale Flexibilität F = 1, und die Menge aller zulässigen Komponenten sei {{A}, {B}, {C}, {D}, {E}, {F}, {G}}.

Ferner sei das Muster P = A-x-C-D bereits abgeleitet worden vom Pfad A,C,D.

Durch die Kante D->E kann erweitert werden zu A-x-C-D-E, A-x-C-D-x(0,1)-E.

Kante D->F erweitert zu A-x-C-D-x-F, A-x-C-D-x(0,1)-F, A-x-C-D-x(1,2)-F,

und Kante D->G ergibt: A-x-C-D-x(2)-G, A-x-C-D-x(1,2)-G und A-x-C-D-x(2,3).

„Pruning“

es gibt einige Möglichkeiten, dem Suchbaum Äste abzuhacken: zum einen nutzt man besondere Eigenschaften des Graphen, die entstehen, wenn bestimmte Restriktionen gesetzt werden, in allgemeineren Fällen nimmt man Heuristiken, die allerdings die Optimalitätsgarantie zunichte machen.

Eine weitere Strategie ist das Branch-and-Bound, das Redundanzen verhindert, also Muster mit niedrigem Informationsgehalt ausmerzt.
Voraussetzung für die Pruning-Strategien ist eine (totale) Ordnung auf den Knoten.
Wir werden sehen, daß der Graph nach Konstruktion kreisfrei ist. Es existiert daher eine partielle Ordnung der Knoten, induziert durch u < v, falls u,v aus V und (u->v) aus E. Wir können diese leicht in eine totale Ordnungsrelation (antisymmetrisch, transitiv, reflexiv) überführen (siehe Konstruktion), so daß ui < uj, falls i<j.
Die Ordnung auf den Knoten wird jetzt benutzt, um eine Ordnung der einfachen Erweiterungen einzuführen: Eine Erweiterung E1 ist spezieller als Erweiterung E2, falls der Endknoten der die Erweiterung E1 definierenden Kante geringeren Index hat als Der zu E2 gehörige Knoten.
Sind die Endknoten für E1 und E2 identisch, entscheidet die Kardinalität der Komponente, mit der erweitert wurde: das Muster mit der Komponente geringerer Kardinalität ist das speziellere.
Falls Knotenindizes und Kardinalitäten jeweils gleich sind, ist die Erweiterung mit der Wildcardregion höherer Flexibilität die generellere.

Verbot von flexiblen Wildcardregionen und Ambiguitäten:
in diesem Fall gilt:
es müssen keine Verallgemeinerungen auf die durch Pfade definierten Muster angewandt werden.
Folgende Strategie bietet sich an: Falls es mehrere Möglichkeiten gibt, einen Pfad zu erweitern, erweitere durch den Knoten mit geringstem Index. Falls ein Muster Pgenauso viele Segmente matcht wie sein Vorfahrmuster, brauchen die Geschwister von P nicht mehr betrachtet werden, da keines eine größere Menge von Segmenten matchen kann.

Branch and Bound:
Eine Methode, eine obere Grenze für die Signifikanz eines Patterns zu finden, ist diese:
der Algorithmus verarbeitet die Knoten entgegen ihrer durch den Index gegebenen Reihenfolge und merkt sich den Wert des signifikantesten Patterns mit Ursprung im jeweiligen Knoten.. Wenn dann ein Pfad für Pattern P –ausgehend von Knoten u- durch den Knoten v verlängert werden soll, haben wir uns vorher den Wert H des signifikantesten Musters von v gemerkt.
Falls I(P)+H(v) kleiner ist als der derzeitige (vorläufige) Highscore von u, wird das Muster P-v auch nicht weiter betrachtet.

Soll ich die Generalisierungsrelationen wirklich explizit mit reinbringen, weil das zwar formal ist,aber kein bisschen beweist, dass damit wirklich genügend Muster abgegrast werden?
Komplexität fand ich blöd, da das worst-case-Verhalten ja eh nicht verbessert wird.

Fragen:
k-Segmente: wie wird k gezählt?
flexible patterns: auch links/1.Position? (Schritt 1)
was denn beweisen? 2b findet alle?
kann man 2b auf von 2a erkannten Mustern laufen lassen (mit garantie?)

asymmetrische Blockstruktur

guckt2a bei x(1,2)erste Position an?