Script generated by TTT

Title: Seidl: GAD (05.07.2016)

Date: Tue Jul 05 14:22:36 CEST 2016

Duration: 91:56 min

Pages: 59

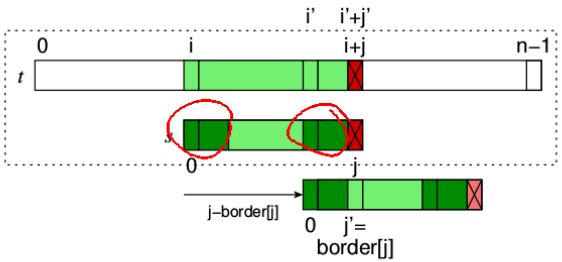
KMP-Algorithmus

Funktion KMP($t[\,]$, n, $s[\,]$, m)

```
int border[m + 1];
computeBorders(border, m, s);
int i := 0, j := 0;
while i ≤ n − m do
    while t[i + j] = s[j] do
        j++;
        if j = m then
            return TRUE;
    i := i + (j − border[j]) ;          // Es gilt j − border[j] > 0
    j := max{0, border[j]};
return FALSE;
```


Sichere Shifts

Shift um j – border[j]



Laufzeit des KMP-Algorithmus: erfolglose Vergleiche

Nach erfolglosem Vergleich (Mismatch) wird $(i + j)$ nie kleiner:

- Seien dazu i und j die Werte vor einem erfolglosen Vergleich und i' und j' die Werte nach einem erfolglosen Vergleich.
- Wert vor dem Vergleich: $i + j$
- Wert nach dem Vergleich:
  $i' + j' = (i + j - border[j]) + (max\{0, border[j]\})$.
- Fallunterscheidung: $border[j]$ negativ oder nicht.
  - $border[j] < 0$, also $border[j] = -1$, dann muss $j = 0$ sein.
    Das bedeutet $i' + j' = i' + 0 = (i + 0 - (-1)) + 0 = i + 1$.
  - $border[j] \geq 0$, dann gilt $i' + j' = i + j$
- Also wird $i + j$ nach einem erfolglosen Vergleich nicht kleiner.

KMP-Algorithmus

Funktion KMP($t[\,]$, n, $s[\,]$, m)

```
int border[m + 1];
computeBorders(border, m, s);
int i := 0, j := 0;
while i ≤ n − m do
    while t[i + j] = s[j] do
        j++;
        if j = m then
            return TRUE;
    i := i + (j − border[j]) ;          // Es gilt j − border[j] > 0
    j := max{0, border[j]};
return FALSE;
```

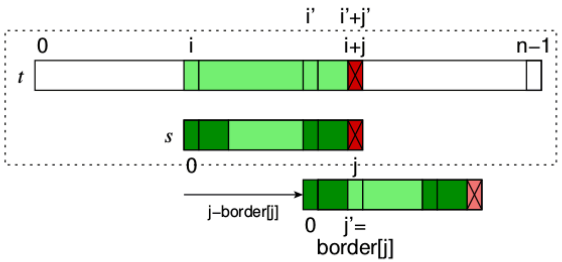

Laufzeit des KMP-Algorithmus: erfolglose Vergleiche

Nach erfolglosem Vergleich (Mismatch) wird $(i + j)$ nie kleiner:

- Seien dazu i und j die Werte vor einem erfolglosen Vergleich und i' und j' die Werte nach einem erfolglosen Vergleich.
- Wert vor dem Vergleich: $i + j$
- Wert nach dem Vergleich:
  $i' + j' = (i + j - border[j]) + (max\{0, border[j]\})$.
- Fallunterscheidung: $border[j]$ negativ oder nicht.
  - $border[j] < 0$, also $border[j] = -1$, dann muss $j = 0$ sein.
    Das bedeutet $i' + j' = i' + 0 = (i + 0 - (-1)) + 0 = i + 1$.
  - $border[j] \geq 0$, dann gilt $i' + j' = i + j$
- Also wird $i + j$ nach einem erfolglosen Vergleich nicht kleiner.

Sichere Shifts

Shift um $j - border[j]$



Laufzeit des KMP-Algorithmus: erfolglose Vergleiche

Nach erfolglosem Vergleich (Mismatch) wird $(i + j)$ nie kleiner:

- Seien dazu i und j die Werte vor einem erfolglosen Vergleich und i' und j' die Werte nach einem erfolglosen Vergleich.
- Wert vor dem Vergleich: $i + j$
- Wert nach dem Vergleich:
  $i' + j' = (i + j - border[j]) + (max\{0, border[j]\})$.
- Fallunterscheidung: $border[j]$ negativ oder nicht.
  - $border[j] < 0$, also $border[j] = -1$, dann muss $j = 0$ sein.
    Das bedeutet $i' + j' = i' + 0 = (i + 0 - (-1)) + 0 = i + 1$.
  - $border[j] \geq 0$, dann gilt $i' + j' = i + j$
- Also wird $i + j$ nach einem erfolglosen Vergleich nicht kleiner.

KMP-Algorithmus

Funktion KMP($t[\,]$, n, $s[\,]$, m)

```
int border[m + 1];
computeBorders(border, m, s);
int i := 0, j := 0;
while i ≤ n − m do
    while t[i + j] = s[j] do
        j++;
        if j = m then
            return TRUE;
    i := i + (j − border[j]) ;          // Es gilt j − border[j] > 0
    j := max{0, border[j]};
return FALSE;
```

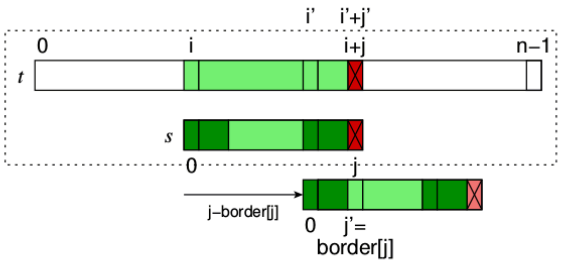

Laufzeit des KMP-Algorithmus: erfolglose Vergleiche

Nach erfolglosem Vergleich (Mismatch) wird $(i + j)$ nie kleiner:

- Seien dazu i und j die Werte vor einem erfolglosen Vergleich und i' und j' die Werte nach einem erfolglosen Vergleich.
- Wert vor dem Vergleich: $i + j$
- Wert nach dem Vergleich:
  $i' + j' = (i + j - border[j]) + (max\{0, border[j]\})$.
- Fallunterscheidung: $border[j]$ negativ oder nicht.
  - $border[j] < 0$, also $border[j] = -1$, dann muss $j = 0$ sein.
    Das bedeutet $i' + j' = i' + 0 = (i + 0 - (-1)) + 0 = i + 1$.
  - $border[j] \geq 0$, dann gilt $i' + j' = i + j$
- Also wird $i + j$ nach einem erfolglosen Vergleich nicht kleiner.

Sichere Shifts

Shift um $j - border[j]$



Laufzeit des KMP-Algorithmus: erfolglose Vergleiche

Nach erfolglosem Vergleich (Mismatch) wird $(i + j)$ nie kleiner:

- Seien dazu i und j die Werte vor einem erfolglosen Vergleich und i' und j' die Werte nach einem erfolglosen Vergleich.
- Wert vor dem Vergleich: $i + j$
- Wert nach dem Vergleich:
  $i' + j' = (i + j - border[j]) + (max\{0, border[j]\})$.
- Fallunterscheidung: $border[j]$ negativ oder nicht.
  - $border[j] < 0$, also $border[j] = -1$, dann muss $j = 0$ sein.
    Das bedeutet $i' + j' = i' + 0 = (i + 0 - (-1)) + 0 = i + 1$.
  - $border[j] \geq 0$, dann gilt $i' + j' = i + j$
- Also wird $i + j$ nach einem erfolglosen Vergleich nicht kleiner.

Laufzeit des KMP-Algorithmus

- Nach jedem erfolglosen Vergleich wird $i \in [0 : n - m]$ erhöht.
- i wird nie verkleinert.

$\Rightarrow$ maximal $n - m + 1$ erfolglose Vergleiche

- Nach einem erfolgreichen Vergleich wird $i + j$ um 1 erhöht.
- maximal n erfolgreiche Vergleiche, da $i + j \in [0 : n - 1]$.

- insgesamt maximal $2n - m + 1$ Vergleiche

Berechnung der border-Tabelle

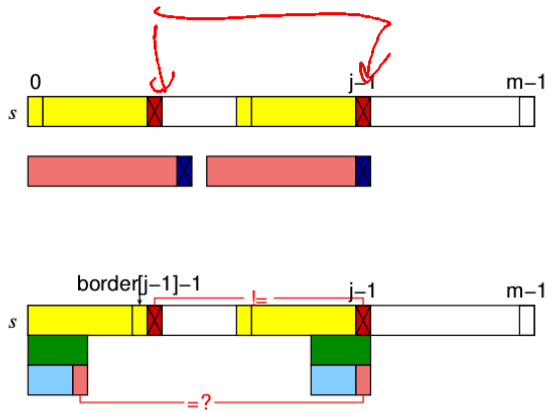
- *border*[ ]-Tabelle:
  speichert für jedes Präfix $s_0 \cdots s_{j-1}$ der Länge $j \in \{0 \ldots m\}$ von Suchstring s die Größe des eigentlichen Rands

- Initialisierung: $border[0] = -1$ und $border[1] = 0$

- Annahme: $border[0], \ldots, border[j-1]$ sind schon berechnet

- Ziel: Berechnung von $border[j]$
  (Länge des eigentlichen Randes von Präfix der Länge j)

Berechnung der border-Tabelle



Berechnung der border-Tabelle

- Der eigentliche Rand $s_0 \cdots s_k$ von $s_0 \cdots s_{j-1}$ kann um maximal ein Zeichen länger sein als der eigentliche Rand von $s_0 \cdots s_{j-2}$, denn $s_0 \cdots s_{k-1}$ ist auch ein Rand von $s_0 \cdots s_{j-2}$ (oberer Teil der Abbildung).
- Ist $s_{border[j-1]} = s_{j-1}$, so ist $border[j] = border[j-1] + 1$.
- Andernfalls müssen wir ein kürzeres Präfix von $s_0 \cdots s_{j-2}$ finden, das auch ein Suffix von $s_0 \cdots s_{j-2}$ ist.
- Der nächstkürzere Rand eines Wortes ist offensichtlich der eigentliche Rand des zuletzt betrachteten Randes dieses Wortes.
- Nach Konstruktion der Tabelle *border* ist das nächstkürzere Präfix mit dieser Eigenschaft das der Länge $border[border[j-1]]$.

Berechnung der border-Tabelle

- teste nun, ob sich dieser Rand von $s_0 \cdots s_{j-2}$ zu einem eigentlichen Rand von $s_0 \cdots s_{j-1}$ erweitern lässt
- solange wiederholen, bis wir einen Rand gefunden haben, der sich zu einem Rand von $s_0 \cdots s_{j-1}$ erweitern lässt
- Falls sich kein Rand von $s_0 \cdots s_{j-2}$ zu einem Rand von $s_0 \cdots s_{j-1}$ erweitern lässt, so ist der eigentliche Rand von $s_0 \cdots s_{j-1}$ das leere Wort und wir setzen $border[j] = 0$.

Algorithmus zur Berechnung der border-Tabelle

Prozedur computeBorders(int *border*[ ], int *m*, char *s*[ ])

```
border[0] := −1;
border[1] := 0;
int i := 0;
for (int j := 2; j ≤ m; j++) do
    // Hier gilt: i = border[j − 1]
    while (i ≥ 0) && (s[i] ≠ s[j − 1]) do
        i := border[i];
    i++;
    border[j] := i;
```


Berechnung der border-Tabelle

- Der eigentliche Rand $s_0 \cdots s_k$ von $s_0 \cdots s_{j-1}$ kann um maximal ein Zeichen länger sein als der eigentliche Rand von $s_0 \cdots s_{j-2}$, denn $s_0 \cdots s_{k-1}$ ist auch ein Rand von $s_0 \cdots s_{j-2}$ (oberer Teil der Abbildung).
- Ist $s_{border[j-1]} = s_{j-1}$, so ist $border[j] = border[j-1] + 1$.
- Andernfalls müssen wir ein kürzeres Präfix von $s_0 \cdots s_{j-2}$ finden, das auch ein Suffix von $s_0 \cdots s_{j-2}$ ist.
- Der nächstkürzere Rand eines Wortes ist offensichtlich der eigentliche Rand des zuletzt betrachteten Randes dieses Wortes.
- Nach Konstruktion der Tabelle *border* ist das nächstkürzere Präfix mit dieser Eigenschaft das der Länge $border[border[j-1]]$.

Algorithmus zur Berechnung der border-Tabelle

Prozedur computeBorders(int *border*[ ], int *m*, char *s*[ ])

```
border[0] := −1;
border[1] := 0;
int i := 0;
for (int j := 2; j ≤ m; j++) do
    // Hier gilt: i = border[j − 1]
    while (i ≥ 0) && (s[i] ≠ s[j − 1]) do
        i := border[i];
    i++;
    border[j] := i;
```

Laufzeit der Berechnung der border-Tabelle

- maximal $m-1$ erfolgreiche Vergleiche, da jedes Mal $j \in [2:m]$ um 1 erhöht und nie erniedrigt wird
- Betrachte für die Anzahl erfolgloser Vergleiche den Wert i. Zu Beginn ist $i = 0$.
- i wird genau $(m-1)$ Mal um 1 erhöht, da die for-Schleife $(m-1)$ Mal durchlaufen wird.
- Bei einem erfolglosen Vergleich wird i um mindestens 1 erniedrigt.
- i kann maximal $(m-1)+1 = m$ Mal erniedrigt werden, da immer $i \geq -1$ gilt. Es gibt also höchstens m erfolglose Vergleiche.
- Gesamtzahl der Vergleiche $\leq 2m-1$

Algorithmus zur Berechnung der border-Tabelle

Prozedur computeBorders(int *border*[ ], int m, char s[ ])

```
border[0] := −1;
border[1] := 0;
int i := 0;
for (int j := 2; j ≤ m; j++) do
    // Hier gilt: i = border[j − 1]
    while (i ≥ 0) && (s[i] ≠ s[j − 1]) do
        i := border[i];
    i++;
    border[j] := i;
```


Pattern Matching | Knuth-Morris-Pratt-Algorithmus

Laufzeit der Berechnung der border-Tabelle

- maximal $m-1$ erfolgreiche Vergleiche, da jedes Mal $j \in [2:m]$ um 1 erhöht und nie erniedrigt wird
- Betrachte für die Anzahl erfolgloser Vergleiche den Wert i. Zu Beginn ist $i = 0$.
- i wird genau $(m-1)$ Mal um 1 erhöht, da die for-Schleife $(m-1)$ Mal durchlaufen wird.
- Bei einem erfolglosen Vergleich wird i um mindestens 1 erniedrigt.
- i kann maximal $(m-1)+1 = m$ Mal erniedrigt werden, da immer $i \geq -1$ gilt. Es gibt also höchstens m erfolglose Vergleiche.
- Gesamtzahl der Vergleiche $\leq 2m-1$

H. Seidl (TUM) GAD SS'16 583

Pattern Matching | Knuth-Morris-Pratt-Algorithmus

Algorithmus zur Berechnung der border-Tabelle

Prozedur computeBorders(int *border*[ ], int m, char s[ ])

```
border[0] := −1;
border[1] := 0;
int i := 0;
for (int j := 2; j ≤ m; j++) do
    // Hier gilt: i = border[j − 1]
    while (i ≥ 0) && (s[i] ≠ s[j − 1]) do
        i := border[i];
    i++;
    border[j] := i;
```

H. Seidl (TUM) GAD SS'16 582

Laufzeit der Berechnung der border-Tabelle

- maximal $m-1$ erfolgreiche Vergleiche, da jedes Mal $j \in [2:m]$ um 1 erhöht und nie erniedrigt wird
- Betrachte für die Anzahl erfolgloser Vergleiche den Wert i. Zu Beginn ist $i = 0$.
- i wird genau $(m-1)$ Mal um 1 erhöht, da die for-Schleife $(m-1)$ Mal durchlaufen wird.
- Bei einem erfolglosen Vergleich wird i um mindestens 1 erniedrigt.
- i kann maximal $(m-1)+1 = m$ Mal erniedrigt werden, da immer $i \geq -1$ gilt. Es gibt also höchstens m erfolglose Vergleiche.
- Gesamtzahl der Vergleiche $\leq 2m-1$

Laufzeit des KMP-Algorithmus

- Nach jedem erfolglosen Vergleich wird $i \in [0:n-m]$ erhöht.
- i wird nie verkleinert.

$\Rightarrow$ maximal $n-m+1$ erfolglose Vergleiche

- Nach einem erfolgreichen Vergleich wird $i+j$ um 1 erhöht.
- maximal n erfolgreiche Vergleiche, da $i+j \in [0:n-1]$.
- insgesamt maximal $2n-m+1$ Vergleiche

Laufzeit des KMP-Algorithmus

Theorem

Der Algorithmus von Knuth, Morris und Pratt benötigt maximal $2n+m$ Vergleiche, um festzustellen, ob ein Muster s der Länge m in einem Text t der Länge n enthalten ist.

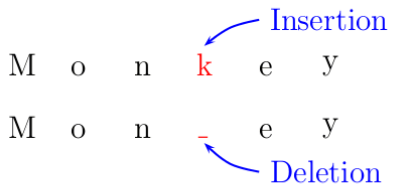
Der Algorithmus lässt sich leicht derart modifizieren, dass er alle Positionen der Vorkommen von s in t ausgibt, ohne dabei die asymptotische Laufzeit zu erhöhen.

Donald E. Knuth, James H. Morris, Jr. and Vaughan R. Pratt
Fast Pattern Matching in Strings
SIAM Journal on Computing 6(2):323–350, 1977.

Distanz und Ähnlichkeit von Sequenzen

Paarweises Sequenzenalignment

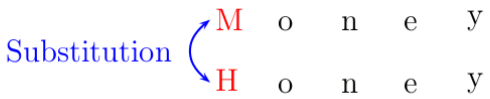
- Ähnlichkeit von 2 Sequenzen bzw.
- Wie kann die eine Sequenz aus der anderen hervorgegangen sein?



Beispiel: Veränderung durch Insertion bzw. Deletion

Distanz und Ähnlichkeit von Sequenzen

- Neben Einfügen und Löschen ist das Ersetzen von Zeichen eine weitere Möglichkeit.



Beispiel: Veränderung durch Substitution

Edit-Distanz

Definition

Sei Σ ein Alphabet und sei $-$ ein neues Zeichen, d.h. $- \notin \Sigma$.

Dann bezeichne

$$\overline{\Sigma} := \Sigma \cup \{-\}$$

das um $-$ erweiterte Alphabet.

Außerdem sei

$$\overline{\Sigma}^2_0 := \overline{\Sigma} \times \overline{\Sigma} \setminus \{(-,-)\} \ .$$

Das Zeichen $-$ werden wir auch als *Leerzeichen* bezeichnen.

Edit-Operationen

Definition

Eine Edit-Operation ist ein Paar $(x, y) \in \overline{\Sigma}^2_0$ und (x, y) heißt

- *Match*, wenn $x = y \in \Sigma$;
- *Substitution*, wenn $x \neq y$ mit $x, y \in \Sigma$;
- *Insertion*, wenn $x = -$, $y \in \Sigma$;
- *Deletion*, wenn $x \in \Sigma$, $y = -$.

Als *InDel-Operation* bezeichnet man eine Edit-Operation, die entweder eine Insertion oder Deletion ist.

Eine neutrale (NoOp)-Operation $(x, x) \in \Sigma \times \Sigma$ ist hier als Edit-Operation zugelassen. Manchmal wird dies auch nicht erlaubt.

Edit-Operationen

Definition

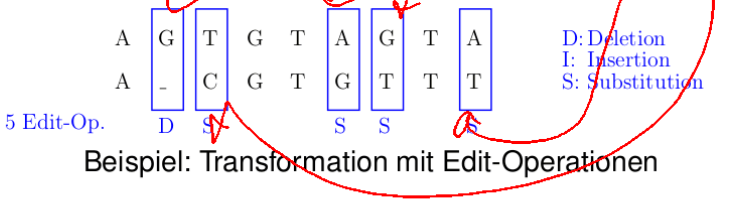
Ist (x, y) eine Edit-Operation und sind $a, b \in \Sigma^*$, dann gilt $a \xrightarrow{(x,y)} b$ (a kann durch die Edit-Operation (x, y) in b umgeformt werden), wenn

- $x, y \in \Sigma \ \wedge \ \exists\, i \in [1 : |a|] : \ (a_i = x) \wedge$
  $(b = a_1 \cdots a_{i-1} \cdot y \cdot a_{i+1} \cdots a_{|a|})$ (Substitution oder Match)
- $x \in \Sigma \ \wedge \ y = - \ \wedge \ \exists\, i \in [1 : |a|] : (a_i = x) \wedge$
  $(b = a_1 \cdots a_{i-1} \cdot a_{i+1} \cdots a_{|a|})$ (Deletion)
- $x = - \ \wedge \ y \in \Sigma \ \wedge \ \exists\, i \in [1 : |a| + 1] :$
  $(b = a_1 \cdots a_{i-1} \cdot y \cdot a_i \cdots a_{|a|})$ (Insertion)

Sei $s = ((x_1, y_1), \ldots, (x_m, y_m))$ eine Folge von Edit-Operationen mit $a_{i-1} \xrightarrow{(x_i,y_i)} a_i$, wobei $a_i \in \Sigma^*$ für $i \in [0 : m]$ und $a := a_0$ und $b := a_m$. Dann schreibt man auch kurz $a \overset{s}{\Rightarrow} b$.

Sequenz-Transformationen

$AGTGTAGTA \overset{s}{\Rightarrow} ACGTGTTT$ mit $s = ((G,-),(T,C),(A,G),(G,T),(A,T))$
oder mit $s = ((G,T),(A,G),(G,-),(A,T),(T,C))$



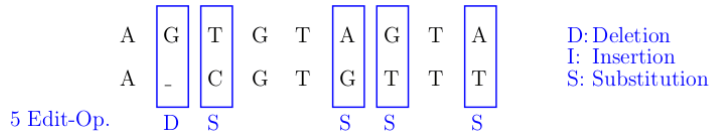
Beispiel: Transformation mit Edit-Operationen

Anmerkung: Es kommt nicht unbedingt auf die Reihenfolge der Edit-Operationen an.

Pattern Matching | Edit-Distanz

Sequenz-Transformationen

$AGTGTAGTA \overset{s}{\Rightarrow} ACGTGTTT$ mit $s = ((G,-),(T,C),(A,G),(G,T),(A,T))$
oder mit $s = ((G,T),(A,G),(G,-),(A,T),(T,C))$



Beispiel: Transformation mit Edit-Operationen

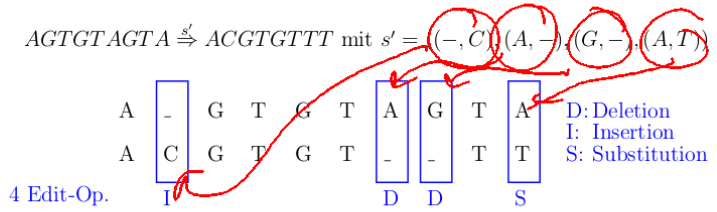
Anmerkung: Es kommt nicht unbedingt auf die Reihenfolge der Edit-Operationen an.

H. Seidl (TUM) GAD SS'16 590

Pattern Matching | Edit-Distanz

Sequenz-Transformationen

$AGTGTAGTA \overset{s'}{\Rightarrow} ACGTGTTT$ mit $s' = ((-,C),(A,-),(G,-),(A,T))$



Beispiel: Transformation mit anderen Edit-Operationen

Dieselben Sequenzen können auch mit Hilfe anderer Edit-Operationen ineinander transformiert werden:

Pattern Matching | Edit-Distanz

Kosten

- Kosten einer Transformationsfolge setzen sich zusammen aus den Kosten der einzelnen Edit-Operationen.
- Man verwendet eine Kostenfunktion $w : \overline{\Sigma} \times \overline{\Sigma} \to \mathbb{R}_+$
- Bsp.: Match-Kosten 0, Substitution/Insertion/Deletion-Kosten 1
- In der Biologie (wo die Sequenzen Basen oder insbesondere Aminosäuren repräsentieren) wird man jedoch intelligentere Kostenfunktionen wählen.

Kostenfunktion, Edit-Distanz

Definition

Sei $w : \overline{\Sigma}_0^2 \to \mathbb{R}_+$ eine Kostenfunktion. Seien $a, b \in \Sigma^*$ und sei $s = (s_1, \ldots, s_\ell)$ eine Folge von Edit-Operationen mit $a \overset{s}{\Rightarrow} b$. Dann sind die Kosten der Edit-Operationen s definiert als

$$w(s) := \sum_{j=1}^{\ell} w(s_j).$$

Die Edit-Distanz von $a, b \in \Sigma^*$ ist definiert als

$$d_w(a,b) := \min_s \left\{ w(s) \;:\; a \overset{s}{\Rightarrow} b \right\}.$$

Dreiecksungleichung

- Folgende Beziehung soll gelten:

$$\forall x, y, z \in \overline{\Sigma} : \; w(x,y) + w(y,z) \geq w(x,z)$$

- Betrachte zum Beispiel eine Mutation (x, z), die als direkte Mutation relativ selten (also teuer) ist, sich jedoch sehr leicht (d.h. billig) durch zwei Mutationen (x, y) und (y, z) ersetzen lässt.
- Dann sollten die Kosten für diese Mutation durch die beiden billigen beschrieben werden, da man in der Regel nicht feststellen kann, ob eine beobachtete Mutation direkt oder über einen Umweg erzielt worden ist.
- Diese Bedingung ist beispielsweise erfüllt, wenn w eine Metrik ist.

Pattern Matching Edit-Distanz

Dreiecksungleichung

- Folgende Beziehung soll gelten:

$$\forall x, y, z \in \overline{\Sigma} : \; w(x,y) + w(y,z) \geq w(x,z)$$

- Betrachte zum Beispiel eine Mutation (x, z), die als direkte Mutation relativ selten (also teuer) ist, sich jedoch sehr leicht (d.h. billig) durch zwei Mutationen (x, y) und (y, z) ersetzen lässt.
- Dann sollten die Kosten für diese Mutation durch die beiden billigen beschrieben werden, da man in der Regel nicht feststellen kann, ob eine beobachtete Mutation direkt oder über einen Umweg erzielt worden ist.
- Diese Bedingung ist beispielsweise erfüllt, wenn w eine Metrik ist.

H. Seidl (TUM) GAD SS'16 594

Metrik

Definition

Sei M eine beliebige Menge.
Eine Funktion $w : M \times M \to \mathbb{R}_+$ heißt Metrik auf M, wenn die folgenden Bedingungen erfüllt sind:

(M1) $\forall x, y \in M : w(x,y) = 0 \Leftrightarrow x = y$ (Definitheit)

(M2) $\forall x, y \in M : w(x,y) = w(y,x)$ (Symmetrie)

(M3) $\forall x, y, z \in M : w(x,z) \leq w(x,y) + w(y,z)$ (Dreiecksungleichung)

Lemma

Ist $w : \overline{\Sigma} \times \overline{\Sigma} \to \mathbb{R}_+$ eine Metrik, dann ist auch $d_w : \overline{\Sigma}^* \times \overline{\Sigma}^* \to \mathbb{R}_+$ eine Metrik.

Restriktion

Definition (Restriktion)

Sei $u \in \overline{\Sigma}^*$. Dann sei die Restriktion von u auf Σ mit Hilfe eines Homomorphismus h wie folgt definiert

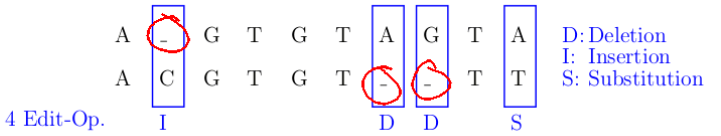
$$u|_\Sigma = h(u), \text{ wobei}$$

$$\begin{aligned} h(a) &= a && \text{für alle } a \in \Sigma, \\ h(-) &= \varepsilon, \\ h(u'u'') &= h(u')h(u'') && \text{für alle } u', u'' \in \Sigma^*. \end{aligned}$$

Die Restriktion von $u \in \overline{\Sigma}^*$ auf Σ ist also nichts anderes als das Löschen aller Leerzeichen $(-)$ aus u.

Sequenz-Transformationen

$AGTGTAGTA \overset{s'}{\Rightarrow} ACGTGTTT$ mit $s' = ((-,C),(A,-),(G,-),(A,T))$



Beispiel: Transformation mit anderen Edit-Operationen

Dieselben Sequenzen können auch mit Hilfe anderer Edit-Operationen ineinander transformiert werden:

Pattern Matching Edit-Distanz

Restriktion

Definition (Restriktion)

Sei $u \in \overline{\Sigma}^*$. Dann sei die Restriktion von u auf Σ mit Hilfe eines Homomorphismus h wie folgt definiert

$$u|_\Sigma = h(u), \text{ wobei}$$

$$\begin{aligned} h(a) &= a && \text{für alle } a \in \Sigma, \\ h(-) &= \varepsilon, \\ h(u'u'') &= h(u')h(u'') && \text{für alle } u', u'' \in \Sigma^*. \end{aligned}$$

Die Restriktion von $u \in \overline{\Sigma}^*$ auf Σ ist also nichts anderes als das Löschen aller Leerzeichen $(-)$ aus u.

Pattern Matching Edit-Distanz

Alignment

Definition (Alignment)

Ein (paarweises) Alignment ist ein Paar $(\overline{a}, \overline{b}) \in \overline{\Sigma}^* \times \overline{\Sigma}^*$ mit

- $|\overline{a}| = |\overline{b}|$ und
- $\overline{a}_i \neq - \neq \overline{b}_i$ für alle $i \in [1 : |\overline{a}|]$ mit $a_i = b_i$.

$(\overline{a}, \overline{b})$ ist ein Alignment für $a, b \in \Sigma^*$, wenn $\overline{a}|_\Sigma = a$ und $\overline{b}|_\Sigma = b$.

Beispiel

A	$-$	G	G	C	A	T	T
A	G	C	G	C	$-$	T	T

Alignment von $AGGCATT$ mit $AGCGCTT$
Dieses Alignment hat Distanz 3,
es gibt jedoch ein besseres mit Distanz 2.

Alignment

Definition (Alignment)

Ein (paarweises) Alignment ist ein Paar $(\overline{a}, \overline{b}) \in \overline{\Sigma}^* \times \overline{\Sigma}^*$ mit

- $|\overline{a}| = |\overline{b}|$ und
- $\overline{a}_i \neq - \neq \overline{b}_i$ für alle $i \in [1 : |\overline{a}|]$ mit $a_i = b_i$.

$(\overline{a}, \overline{b})$ ist ein Alignment für $a, b \in \Sigma^*$, wenn $\overline{a}|_\Sigma = a$ und $\overline{b}|_\Sigma = b$.

Beispiel

A	–	G	G	C	A	T	T
A	G	C	G	C	–	T	T

Alignment von *AGGCATT* mit *AGCGCTT*
Dieses Alignment hat Distanz 3,
es gibt jedoch ein besseres mit Distanz 2.

Alignment-Kosten und Alignment-Distanz

Definition

Sei $\overline{w} : \overline{\Sigma}_0^2 \to \mathbb{R}_+$ eine Kostenfunktion (für einzelne Zeichen).

Die Notation von $\overline{w}$ wird wie folgt auf Sequenzen erweitert, um die (Gesamt-)Kosten eines Alignments $(\overline{a}, \overline{b})$ für (a, b) zu definieren:

$$\overline{w}(\overline{a}, \overline{b}) = \sum_{i=1}^{|\overline{a}|} \overline{w}(\overline{a}_i, \overline{b}_i).$$

Die Alignment-Distanz von $a, b \in \Sigma^*$ ist definiert als

$$\overline{d}_{\overline{w}}(a, b) := \min\left\{\overline{w}(\overline{a}, \overline{b}) \;:\; (\overline{a}, \overline{b}) \text{ ist Alignment für } a, b\right\}.$$

Beziehung zwischen Edit- und Alignment-Distanz

Lemma

Sei $w : \overline{\Sigma}_0^2 \to \mathbb{R}_+$ eine Kostenfunktion (die hier sowohl für die Edit-Operationen als auch für das Alignment benutzt wird, also $w = \overline{w}$) und seien $a, b \in \Sigma^*$.

Für jedes Alignment $(\overline{a}, \overline{b})$ von a und b gibt es eine Folge s von Edit-Operationen, so dass $a \overset{s}{\Rightarrow} b$ und $w(s) = w(\overline{a}, \overline{b})$

Beziehung zwischen Edit- und Alignment-Distanz

Aus diesem Lemma folgt sofort, dass die Edit-Distanz von zwei Zeichenreihen höchstens so groß ist wie die Alignment-Distanz.

Folgerung

Sei $w : \overline{\Sigma}_0^2 \to \mathbb{R}_+$ eine Kostenfunktion, dann gilt für alle $a, b \in \Sigma^*$:

$$d_w(a, b) \leq \overline{d}_w(a, b)$$

Beziehung zwischen Edit- und Alignment-Distanz

Lemma

Sei $w : \overline{\Sigma}_0^2 \to \mathbb{R}_+$ eine metrische Kostenfunktion und seien $a, b \in \Sigma^*$.

Für jede Folge von Edit-Operationen mit $a \overset{s}{\Rightarrow} b$ gibt es ein Alignment $(\overline{a}, \overline{b})$ von a und b, so dass $w(\overline{a}, \overline{b}) \leq w(s)$.

Beziehung zwischen Edit- und Alignment-Distanz

Beweis.

(durch Induktion über $n = |s|$)

Induktionsanfang ($n = 0$):

- Aus $|s| = 0$ folgt, dass $s = \epsilon$.
- Also ist $a = b$ und $w(s) = 0$.
- Wir setzen nun $\overline{a} = a = b = \overline{b}$ und erhalten ein Alignment $(\overline{a}, \overline{b})$ für a und b mit $w(\overline{a}, \overline{b}) = 0 \leq w(s)$.

Beziehung zwischen Edit- und Alignment-Distanz

Beweis.

Induktionsschritt ($n \to n+1$):

- Sei $s = (s_1, \ldots, s_n, s_{n+1})$ eine Folge von Edit-Operationen mit $a \overset{s}{\Rightarrow} b$.
- Sei nun $s' = (s_1, \ldots, s_n)$ und $a \overset{s'}{\Rightarrow} c \overset{s_{n+1}}{\longrightarrow} b$ für ein $c \in \Sigma^*$.
- Aus der Induktionsvoraussetzung folgt nun, dass es ein Alignment $(\overline{a}, \overline{c})$ von a, c gibt, so dass $w(\overline{a}, \overline{c}) \leq w(s')$.

Beziehung zwischen Edit- und Alignment-Distanz

Beweis.

- Betrachte zuerst den Fall, dass die letzte Edit-Operation $s_{n+1} = (x, y)$ eine Substitution, ein Match oder eine Deletion ist, d.h. $x \in \Sigma$ und $y \in \overline{\Sigma}$.
- Wir können dann (wie in der Abbildung) ein Alignment für a und b erzeugen, indem wir die Zeichenreihe $\overline{b}$ geeignet aus $\overline{c}$ unter Verwendung der Edit-Operation(x, y) umformen.

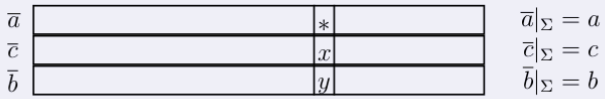
$\overline{a}$		$*$	$\overline{a}\|_\Sigma = a$
$\overline{c}$		x	$\overline{c}\|_\Sigma = c$
$\overline{b}$		y	$\overline{b}\|_\Sigma = b$

Skizze: $s_{n+1} = (x, y)$ ist eine Substitution, Match oder Deletion

Beziehung zwischen Edit- und Alignment-Distanz

Beweis.

- Betrachte zuerst den Fall, dass die letzte Edit-Operation $s_{n+1} = (x, y)$ eine Substitution, ein Match oder eine Deletion ist, d.h. $x \in \Sigma$ und $y \in \overline{\Sigma}$.
- Wir können dann (wie in der Abbildung) ein Alignment für a und b erzeugen, indem wir die Zeichenreihe $\overline{b}$ geeignet aus $\overline{c}$ unter Verwendung der Edit-Operation (x, y) umformen.



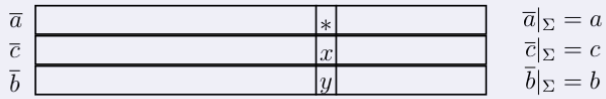
Skizze: $s_{n+1} = (x, y)$ ist eine Substitution, Match oder Deletion

Pattern Matching | Edit-Distanz

Beziehung zwischen Edit- und Alignment-Distanz

Beweis.

- Betrachte zuerst den Fall, dass die letzte Edit-Operation $s_{n+1} = (x, y)$ eine Substitution, ein Match oder eine Deletion ist, d.h. $x \in \Sigma$ und $y \in \overline{\Sigma}$.
- Wir können dann (wie in der Abbildung) ein Alignment für a und b erzeugen, indem wir die Zeichenreihe $\overline{b}$ geeignet aus $\overline{c}$ unter Verwendung der Edit-Operation (x, y) umformen.



Skizze: $s_{n+1} = (x, y)$ ist eine Substitution, Match oder Deletion

H. Seidl (TUM) GAD SS'16 605

Pattern Matching | Edit-Distanz

Beziehung zwischen Edit- und Alignment-Distanz

Beweis.

Es gilt dann:

$$\begin{aligned} w(\overline{a}, \overline{b}) &= \underbrace{w(\overline{a}, \overline{c}) - w(\overline{a}_i, \overline{c}_i)}_{\leq w(\overline{c}_i, \overline{b}_i)} + w(\overline{a}_i, \overline{b}_i) \\ &\quad \text{aufgrund der Dreiecksungleichung} \\ &\quad \text{d.h., } w(\overline{a}_i, \overline{b}_i) \leq w(\overline{a}_i, \overline{c}_i) + w(\overline{c}_i, \overline{b}_i) \\ &\leq \underbrace{w(\overline{a}, \overline{c})}_{\leq w(s')} + \underbrace{w(\overline{c}_i, \overline{b}_i)}_{= w(s_{n+1})} \\ &\leq w(s') + w(s_{n+1}) = w(s) \end{aligned}$$

Pattern Matching | Edit-Distanz

Beziehung zwischen Edit- und Alignment-Distanz

Beweis.

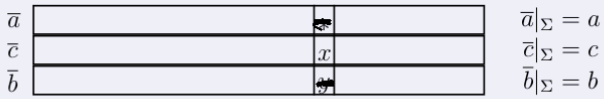
Den Fall $\overline{a}_i = \overline{b}_i = -$ muss man gesondert betrachten.
Hier wird das verbotene Alignment von Leerzeichen im Alignment $(\overline{a}, \overline{b})$ eliminiert:

$$\begin{aligned} w(\overline{a}, \overline{b}) &= w(\overline{a}, \overline{c}) - w(\overline{a}_i, \overline{c}_i) \\ &\leq w(\overline{a}, \overline{c}) + w(\overline{c}_i, \overline{b}_i) \\ &\leq w(s') + w(s_{n+1}) = w(s) \end{aligned}$$

Beziehung zwischen Edit- und Alignment-Distanz

Beweis.

- Betrachte zuerst den Fall, dass die letzte Edit-Operation $s_{n+1} = (x, y)$ eine Substitution, ein Match oder eine Deletion ist, d.h. $x \in \Sigma$ und $y \in \overline{\Sigma}$.
- Wir können dann (wie in der Abbildung) ein Alignment für a und b erzeugen, indem wir die Zeichenreihe $\overline{b}$ geeignet aus $\overline{c}$ unter Verwendung der Edit-Operation(x, y) umformen.



Skizze: $s_{n+1} = (x, y)$ ist eine Substitution, Match oder Deletion

Beziehung zwischen Edit- und Alignment-Distanz

Beweis.

Den Fall $\overline{a}_i = \overline{b}_i = -$ muss man gesondert betrachten.
Hier wird das verbotene Alignment von Leerzeichen im Alignment $(\overline{a}, \overline{b})$ eliminiert:

$$\begin{aligned} w(\overline{a}, \overline{b}) &= w(\overline{a}, \overline{c}) - w(\overline{a}_i, \overline{c}_i) \\ &\leq w(\overline{a}, \overline{c}) + w(\overline{c}_i, \overline{b}_i) \\ &\leq w(s') + w(s_{n+1}) = w(s) \end{aligned}$$

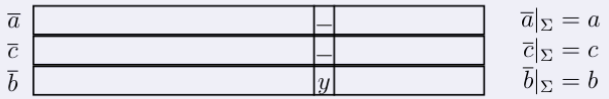
Beziehung zwischen Edit- und Alignment-Distanz

Beweis.

- Es bleibt noch der Fall, wenn $s_{n+1} = (-, y)$ mit $y \in \Sigma$ eine Insertion ist.
- Dann erweitern wir das Alignment $(\overline{a}, \overline{c})$ von a und c zu einem eigentlich *unzulässigen Alignment* $(\overline{a'}, \overline{c'})$ von a und c wie folgt.
- Es gibt ein $i \in [0 : |b|]$ mit $b_i = y$ und $b = c_1 \cdots c_i \cdot y \cdot c_{i+1} \cdots c_{|a|}$.
- Sei j die Position, nach der das Symbol y in $\overline{c}$ eingefügt wird.
- Dann setzen wir $\overline{a} = \overline{a}_1 \cdots \overline{a}_j \cdot - \cdot \overline{a}_{j+1} \cdots \overline{a}_{|\overline{c}|}$, $\overline{c} = \overline{c}_1 \cdots \overline{c}_j \cdot - \cdot \overline{c}_{j+1} \cdots \overline{c}_{|\overline{c}|}$ und $\overline{b} = \overline{c}_1 \cdots \overline{c}_j \cdot y \cdot \overline{c}_{j+1} \cdots \overline{c}_{|\overline{c}|}$.

Beziehung zwischen Edit- und Alignment-Distanz

Beweis.



Skizze: $s_{n+1} = (x, y)$ ist eine Insertion

- $(\overline{a}, \overline{c})$ ist jetzt wegen der Spalte $(-, -)$ kein Alignment mehr.
- jedoch nur noch interessant: Alignment $(\overline{a}, \overline{b})$ von a und b

$$\begin{aligned} w(\overline{a}, \overline{b}) &= w(\overline{a}, \overline{c}) + w(-, y) \\ &\quad \text{Nach Induktionsvoraussetzung} \\ &\leq w(s') + w(s_{n+1}) = w(s) \end{aligned}$$

□