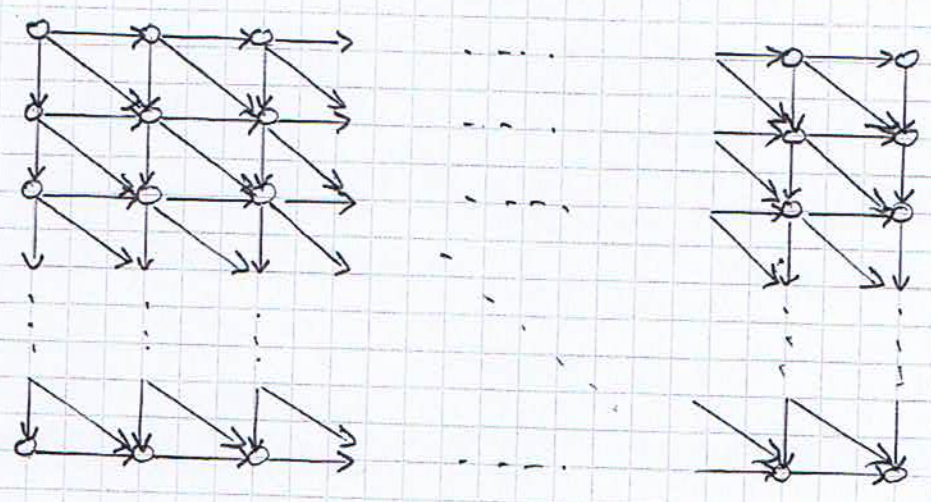


$$op(e) = \begin{cases} \text{Streichen von } x_{j+1} & \text{falls } i' = i, j' = j+1 \\ \text{Einfügen von } y_{i+1} & \text{falls } i' = i+1, j' = j \\ \text{Ersetzen bzw. Substitution von } x_{j+1} \text{ durch } y_{i+1} & \text{falls } i' = i+1, j' = j+1 \end{cases}$$

Mit der Kante  $e$  assoziieren wir die Kosten  $c_{M_1}(op(e))$ . Die Kosten  $c_{M_1}(P)$  eines Pfades  $P$  in  $E_{M_1}(x, y)$  sind definiert durch

$$c_{M_1}(P) := \sum_{e \in P} c_{M_1}(op(e)).$$



Editiergraph  $E_{M_1}(x, y)$ .

Der Distanzgraph  $D_{M_1}(x, y) = (V, E_d)$  ist denn definiert durch

$$E_d := \left\{ ([i, j], [i', j']) \in E \mid e(x^j, y^i) = e(x^{j'}, y^{i'}) + c_{M_1}(op([i, j], [i', j'])) \right\}.$$

Übung:

- a) Zeigen Sie, dass es eine Bijektion zwischen den Pfaden vom Knoten  $[0,0]$  zum Knoten  $[m,n]$  in  $D_{M_1}(x,y)$  und den Alignments  $A(x,y)$  mit  $c_{M_1}(A(x,y)) = d_{M_1}(x,y)$  gibt.
- b) Bauen Sie in den Algorithmus  $EDIS_{M_1}$  die Konstruktion des Distanzgraphen ein.
- c) Analysieren Sie die Zeit- und die Platzkomplexität des unter b) erweiterten Algorithmus  $EDIS_{M_1}$ .

Insgesamt haben wir somit folgenden Satz bewiesen:

Satz 3.2

Seien  $x = x_1x_2 \dots x_n$  und  $y = y_1y_2 \dots y_m$  zwei Strings. Dann kann der Editierabstand  $d_{M_1}(x,y)$  von  $x$  und  $y$  und der Distanzgraph  $D_{M_1}(x,y)$  in Zeit  $O(m \cdot n)$  und Platz  $O(m \cdot n)$  berechnet werden.

Übung:

Zeigen Sie, dass  $d_{M_1}(x,y)$  auch in  $O(\min\{m,n\})$  Platz und  $O(m \cdot n)$  Zeit berechnet werden kann.

$M_2$ , eine zweite Wahl für  $M$ :

In der evolutionären Geschichte genetischer Sequenzen tritt das Streichen bzw. Einfügen eines

Strings der Länge  $\geq 2$  als einzelne Mutation auf  $M_1$  erlaubt jedoch die Behandlung einer solchen Mutation nur als Folge von  $k$  Einzeloperationen, deren Kosten sich addieren. Diese Vorgehensweise wird von Molekularbiologen als nicht sinnvoll betrachtet. Daher erweitern wir  $M_1$  durch Hinzufügen dieser Operationen zu einer Operationenmenge  $M_2$ .

$M_2$  enthält demnach folgende Typen von Operationen:

- a) Streichen eines Strings der Länge  $k, k \geq 1$  aus  $x$ .
- b) Einfügen eines Strings der Länge  $k, k \geq 1$  in  $x$ .
- c) Ersetzen eines Symbols durch ein anderes in  $x$ .
- d) Substitution eines Symbols durch sich selbst in  $x$ .

Somit ist eine Operation in  $M_2$  ein Paar

$$(a,b) \in (\Sigma^+ \times \{-\}) \cup (\{-\} \times \Sigma^+) \cup (\Sigma \times \Sigma),$$

wobei

$$(a,b) = \begin{cases} \text{Streichen von } a & \text{falls } a \in \Sigma^+, b = - \\ \text{Einfügen von } b & \text{falls } a = -, b \in \Sigma^+ \\ \text{Ersetzen von } a \text{ durch } b & \text{falls } a \in \Sigma, b \in \Sigma \setminus \{a\} \\ \text{Substitution von } a \text{ für } a & \text{falls } a = b \in \Sigma \end{cases}$$

Üblicherweise sind die Kosten einer Einfüge- bzw. Streicheoperation derart definiert, dass diese nur von der Länge des Operanden und nicht von Operanden selbst abhängen. D.h., die Kosten sind definiert durch eine Funktion

(153)

$w_{id} : \mathbb{N} \rightarrow \mathbb{R}^+$  und eine Konstante  $d > 0$ .

Mit jeder Operation  $(a, b)$  assoziieren wir die Kosten  $c_{M_2}(a, b)$ , wobei

$$c_{M_2}(a, b) := \begin{cases} d + w_{id}(|a|) & \text{falls } b = - \\ |a| + w_{id}(|b|) & \text{falls } a = - \end{cases}$$

und ansonsten wie  $c_{M_1}(a, b)$  definiert ist.

$d$  ist die "Strafe" dafür, dass überhaupt eine Streiche- bzw. Einfügeoperation erfolgt und  $w_{id}(|a|)$  bzw.  $w_{id}(|b|)$  ist die "Strafe" für die Länge des Operanden.

Übung:

Erweitern Sie die bzgl.  $M_1$  entwickelten Methoden auf  $M_2$ . Genauer:

- a) Definieren Sie evolutionäre Distanz, Alignment, Editier- und Distanzgraph für  $M_2$ .
- b) Formulieren Sie bzgl.  $M_2$  einen zu Satz 3.1 analogen Satz und beweisen Sie diesen.
- c) Modifizieren Sie den Algorithmus  $EDD_{M_1}$ , so dass die evolutionäre Distanz und der Distanzgraph bzgl.  $M_2$  in Zeit und Platz  $O(m^2n + mn^2)$  berechnet werden.

M<sub>3</sub>, eine dritte Wahl für M

Üblicherweise definiert man  $c_{M_2}(a, a) := 0 \quad \forall a \in \Sigma$ .

Mitunter ist es nützlich, wenn die Belohnung für gleiche Teilsequenzen mit deren Länge wächst. Da die Strafe für Mutationen positiv ist, muss die Belohnung für Gleichheit negativ sein. Analog zum Streichen oder Einfügen eines Strings der Länge  $> 1$  betrachten wir die Substitution eines Strings der Länge  $> 1$ . Daher erweitern wir wie folgt die Operationenmenge  $M_2$  zur Operationenmenge  $M_3$ :

- a) Streichen eines Strings der Länge  $k, k \geq 1$  aus  $x$ .
- b) Einfügen eines Strings der Länge  $k, k \geq 1$  in  $x$ .
- c) Ersetzen eines Symbols durch ein anderes in  $x$ .
- d) Substitution eines Teilstrings der Länge  $k \geq 1$  durch sich selbst in  $x$ .

Wir definieren die Kosten einer Substitution derart, dass die Belohnung nur von der Länge des Operanden und nicht vom Operanden selbst abhängt. D.h., die Belohnung ist definiert durch eine Funktion

$$w_m: \mathbb{N} \rightarrow \mathbb{R}^- \text{ und einer Konstanten } 0 < g < |w_m(1)|.$$

Mit jeder Operation  $(a, a) \in M_3$  assoziieren wir die Belohnung  $c_{M_3}(a, a)$ , wobei

$$c_{M_3}(a, a) := g + w_m(|a|).$$

Biologisch sinnvoll scheint, dass der Anteil an

(155)

Belohnung mit der Länge des substituierten Teilstrings wächst. Die Konstante  $g$  stellt sicher, dass eine lange Substitution stets günstiger, als zwei aufeinanderfolgende kürzere Substitutionen ist.

Übung:

Erweitern Sie die bzgl.  $M_2$  entwickelten Methoden auf  $M_3$ .

### Biologisch relevante Kostenfunktionen

Als biologisch relevant wird von Molekularbiologen angesehen, dass der Kostenzuwachs für Streiche- bzw. Einfügeoperationen mit der Länge des Operanden abnimmt. Solche Funktionen heißen in der Mathematik konkav. Formaler

$w: \mathbb{N} \rightarrow \mathbb{R}$  heißt konkav, falls  $\forall k \geq 1$

$$w(k+1) - w(k) \geq w(k+2) - w(k+1).$$

Da Belohnung negativ ist, wäre dann eine Funktion  $w_m$ , die den Anteil an Belohnung mit der Länge des substituierten Teilstrings wachsen lässt, konkav.

Konkave Kostenfunktionen lassen effizientere Algorithmen zu.

### 3.2. Der Algorithmus von Galil und Giancarlo

Z. Galil, R. Giancarlo, Speeding up dynamic programming with applications to molecular biology, TCS 64 (1989), 107 - 118.

Wir betrachten nachfolgend nur konvexe Kostenfunktionen.

Ziel:

Beschleunigung der Algorithmen aufgrund der Konvexität der Kostenfunktionen.

Betrachten wir noch einmal die oben definierte  $(n+1) \times (n+1)$ -Matrix  $\Gamma$ .  $v(i, j)$  berechnet die minimale Kosten für das zur Position  $[i, j]$  in  $\Gamma$  korrespondierende Teilproblem.

Für  $0 \leq i \leq n, 0 \leq j \leq n$  erhalten wir

$$v(i, j) = \begin{cases} 0 & \text{falls } i=j=0 \\ \min \{ r_{\min}(i, j), c_{\min}(i, j), d_{\min}(i, j) \} & \text{sonst} \end{cases}$$

wobei

•  $r_{\min}(i, j) := \min_{0 \leq k \leq j} \{ v(i, k) + c([i, k], [i, j]) \}$ ,

•  $c_{\min}(i, j) := \min_{0 \leq l \leq i} \{ v(l, j) + c([l, j], [i, j]) \}$

und

(15)

- $d_{\min}(i, j) := \min_{1 \leq q \leq \min\{i, j\}} \{ v(i-q, j-q) + c([i-q, j-q], [i, j]) \}$

Dabei bezeichnet  $c([i', j'], [i, j])$  die zusätzlichen Kosten für eine Lösung des Teilproblems  $[i, j]$ , gestartet mit einer Lösung des Teilproblems  $[i', j']$ .

Für  $1 \leq i \leq m$  sei

$$RMIN(i) := [r_{\min}(i, 0), r_{\min}(i, 1), \dots, r_{\min}(i, n)].$$

Im folgenden betrachten wir  $i$  fest.

Jeder Pfad  $P$  im Distanzgraphen, der für  $RMIN(i)$  von Relevanz ist, hat eine letzte Kante  $e$ , so dass beide Endknoten von  $e$  in Zeile  $i$  sind. Ein derartiger Pfad heißt Zeilenlösung.

Annahme:

Für  $0 \leq k < p$  sind die Kosten  $v(i, k)$  eines optimalen Pfades vom Knoten  $[0, 0]$  zum Knoten  $[i, k]$  im Distanzgraphen berechnet.

Ziel:

Berechnung einer optimalen Zeilenlösung  $P$  mit letztem Knoten  $[i, p]$  auf  $P$ .

1. Lösungsmethode:

- Betrachte alle  $0 \leq k < p$  und berechne die Kosten



$v(i, k) + c([i, k], [i, p])$ . Nimm eine Lösung mit minimalen Kosten.

Aufwand:  $O(p)$

Gesamtaufwand für  $RMIN(i)$ :  $\sum_{p=1}^n p = O(n^2)$ .

Ziel:

Für konvexe Kostenfunktionen  $c$  möchten wir ein Verfahren zur Berechnung von  $RMIN(i)$  entwickeln, welches lediglich die Zeit  $O(n \log n)$  benötigt.

Idee:

Beschränke die Anzahl der Kandidaten  $k < p$ , die betrachtet werden müssen, auf eins.

Betrachten wir hierzu diejenige optimale Zeilenlösung nach  $[i, p-1]$  mit längster letzter Kante  $([i, k], [i, p-1])$ .

$\Rightarrow$

Für jede andere optimale Zeilenlösung nach  $[i, p-1]$  gilt  $k' > k$ , wobei  $[i, k']$  der Anfangsknoten der letzten Kante  $([i, k'], [i, p-1])$  dieser Zeilenlösung ist.

Konkavität der Kostenfunktion  $c \Rightarrow$

Eine beste Zeilenlösung, die die Kante  $([i, k], [i, p])$  verwendet, kann nicht schlechter sein, als eine Zeilenlösung, die die Kante  $([i, k'], [i, p])$ ,  $k < k' < p-1$  verwendet.

=>

Der Knoten  $[i, k]$  dominiert alle Knoten  $[i, k']$  mit  $k < k' < p-1$  bezüglich  $[i, p]$ .

Beobachtung:

Falls auf jedem optimalen Pfad von  $[0, 0]$  nach  $[i, p-1]$  als letzte Kante eine Spalten- oder Diagonalkante liegt und auch

$$v(i, p-1) + c([i, p-1], [i, p]) < v(i, k) + c([i, k], [i, p]),$$

dann gilt:

Eine Zeilenlösung, die die Kante  $([i, p-1], [i, p])$  verwendet, ist kostengünstiger, als jede Zeilenlösung, die die Kante  $([i, k], [i, p])$  verwendet.

Dann sagen wir,  $[i, p-1]$  dominiert  $[i, k]$  bzgl.  $[i, p]$ .

Da die Länge der Kante  $([i, k], [i, p])$  größer ist, als die Länge der Kante  $([i, p-1], [i, p])$ , stellt sich unmittelbar folgende Frage:

Kann es sein, dass zwar  $[i, p-1]$  den Knoten  $[i, k]$  bzgl.  $[i, p]$  dominiert, jedoch für ein  $p' > p$  gilt:  $[i, k]$  dominiert  $[i, p-1]$  bzgl.  $[i, p']$ ?

Zur Beantwortung dieser Frage betrachten wir nun die Knoten  $[i, p+1], [i, p+2], \dots$  und die Kanten  $([i, p-1], [i, p+1]), ([i, p-1], [i, p+2]), \dots$  sowie

die Kanten  $([i, k], [i, p+1]), ([i, k], [i, p+2]), \dots$

Beobachtung:

Zunächst können auch die Kanten  $([i, p-1], [i, p+1]), ([i, p-1], [i, p+2]), \dots$  eine Zeit lang zu besseren Zeilenlösungen führen, als die Kanten  $([i, k], [i, p+1]), ([i, k], [i, p+2]), \dots$

Aber:

Aufgrund der Konkavität der Kostenfunktion  $c$  kann  $p' > p$  existieren, so dass  $([i, k], [i, p'])$  zu einer Zeilenlösung führt, die nicht schlechter ist, als eine Zeilenlösung, die die Kante  $([i, p-1], [i, p'])$  verwendet.

Konkavität  $\Rightarrow$  Für  $p'' > p'$  würde dies dann immer der Fall sein.

$\Rightarrow$

Das kleinste  $p' > p$  mit der Eigenschaft, dass  $([i, k], [i, p'])$  zu einer Zeilenlösung führt, die nicht schlechter ist, als eine Zeilenlösung, die  $([i, p-1], [i, p'])$  verwendet, ist von Interesse.

Betrachten wir nun  $k' < k$ . Dann gilt aufgrund der Wahl des Knotens  $[i, k]$ :

161

$[i, k]$  dominiert  $[i, k']$  bzgl.  $[i, p-1]$

$\Rightarrow$

$[i, k']$  dominiert genau dann  $[i, k]$  bzgl.  $[i, p]$ , wenn  $p$  der kleinste Wert ist, so dass  $([i, k'], [i, p])$  zu einer Zeilenlösung führt, die nicht schlechter ist, als eine Zeilenlösung, die die Kante  $([i, k], [i, p])$  verwendet.

Somit führt die gesamte Betrachtung zu folgender Definition:

Ein Knoten  $[i, k]$   $p$ -dominiert genau dann den Knoten  $[i, k']$ , wenn eine der folgenden zwei Bedingungen erfüllt ist:

1)  $k < k'$  und  $v(i, k) + c([i, k], [i, p]) \leq v(i, k') + c([i, k'], [i, p])$

2)  $k > k'$  und  $v(i, k) + c([i, k], [i, p]) < v(i, k') + c([i, k'], [i, p])$ .

Beobachtung 1:

Falls  $[i, k]$  einen Knoten  $[i, k']$   $p$ -dominiert und  $k < k'$ , dann  $p'$ -dominiert  $[i, k]$  den Knoten  $[i, k']$  für alle  $p' > p$ .

Wir kombinieren die beiden Fälle in obiger Definition wie folgt:

Sei  $0 \leq k < j \leq p \leq n$ . Der Knoten  $[i, k]$  heißt genau dann  $\langle j, p \rangle$ -dominant, wenn für alle  $k' < j$  der Knoten  $[i, k]$  den Knoten  $[i, k']$   $p$ -dominiert.

Die Konkavität der Kostenfunktion  $c$  impliziert folgende Beobachtung:

Beobachtung 2:

- a)  $\forall 0 < j \leq p \leq n$  existiert genau ein Knoten  $[i, k]$ , der  $\langle j, p \rangle$ -dominant ist. Bezeichne  $\text{dom}(j, p)$  diesen eindeutigen Knoten.
- b) Sei  $j$  fest und  $[i, k_p] := \text{dom}(j, p)$ . Dann gilt:  $q > p \Rightarrow k_q \leq k_p$ .
- c)  $r_{\min}(i, j) = v(i, k) + c([i, k], [i, j])$ , wobei  $[i, k] := \text{dom}(j, j)$ .



Zu dem Zeitpunkt, wenn wir eine optimale Zeilenlösung nach  $[i, p]$  berechnen möchten, genügt die Kenntnis des eindeutigen  $\langle p, p \rangle$ -dominanten Knotens, um  $r_{\min}(i, p)$  in konstanter Zeit zu berechnen.

Frage:

Was ist eine vernünftige Datenstruktur zur Verwaltung derjenigen Knoten, die eventuell irgendwann dominant werden?

Ober haben wir uns überlegt, dass ein Knoten  $[i, k]$  dominiert werden kann und daher der bisherige dominante Knoten  $[i, k']$  diese Eigenschaft verliert. Wenn der Knoten  $[i, k']$  wieder dominiert werden sollte, dann wird dennoch  $[i, k]$  niemals wieder dominiert werden.

⇒

Ein Vektor scheint eine vernünftige Datenstruktur zu sein.

Idee

Anstatt für  $[i, k]$  jedes  $[i, p]$  mit der Eigenschaft, dass  $[i, k] \prec_{j, p}$ -dominant ist, zu speichern, genügt es, den Knoten  $[i, p]$  mit kleinsten  $p \geq j$  und  $[i, k]$  ist  $\prec_{j, p}$ -dominant zu speichern.

Daher definieren wir für  $0 \leq k < j \leq n$

$$p_j(k) := \min_{j \leq p \leq n} \{ p \mid [i, k] = \text{dom}(j, p) \}.$$

Falls  $p_j(k)$  definiert ist, dann nennen wir die Kante  $([i, k], [i, p_j(k)])$   $\prec_{k, j}$ -wesentlich.

Durchführung:

Zur Berechnung von  $\text{RMIN}(i)$  betrachten wir die  $i$ -te Zeile der Matrix  $\Gamma$  von links nach rechts.

Annahme:

Zum Zeitpunkt, wenn der Knoten  $[i, j]$  betrachtet

184

wird, sind in einem Keller  $S(j)$  genau die  $\langle k, j \rangle$ -wesentlichen Kanten bzgl.  $k$  monoton steigend vom Kellerboden zum obersten Kellerelement abgelegt.

$\Rightarrow$

Die zweiten Komponenten der Kantenköpfe sind monoton fallend abgelegt.

Bezeichnungen:

$TOP(S(j))$

oberste Kante in  $S(j)$

$SEC(S(j))$

zweitoberste Kante in  $S(j)$

$S(0)$

leerer Keller

Annahme:  $TOP(S(j)) = (\text{dom}(j, j), [i, j])$

Zusammen mit Beobachtung 2c folgt somit, dass  $\text{min}(i, j)$  in konstanter Zeit berechnet werden kann.

Sei

$S(j) = [([i, k_1], [i, p_1]), ([i, k_2], [i, p_2]), \dots, ([i, k_r], [i, p_r])] ]$

$\begin{matrix} TOP \\ \downarrow \end{matrix}$

Dann gilt:

1)  $p_r = j$

2)  $k_r = j-1 \Leftrightarrow$

• Jeder optimale Pfad von  $[0, 0]$  nach  $[i, j-1]$  hat eine Spalten- oder eine Diagonalkante als letzte Kante und

•  $[i, j-1]$   $j$ -dominiert  $[i, k_{r-1}]$ .

Frage:

Wie erhalten wir nach der Betrachtung des Knotens  $[i, j]$  den Keller  $S(j+1)$ ?

Hierzu modifizieren wir den Keller  $S(j)$ .

Modifikation von  $S(j)$ :

- Da die Kante  $([i, k_r], [i, j])$  nicht  $\langle k_r, j+1 \rangle$ -wesentlich sein kann, verfahren wir wie folgt:

$$\left\{ \begin{array}{ll} \text{Ersetze den Kopf dieser Kante durch } [i, j+1] & \text{falls } \text{Kopf}(S(j)) \neq [i, j+1] \\ \text{Führe Pop-Operation aus} & \text{sonst.} \end{array} \right.$$

Sei

$$S'(j) = [( [i, k_1], [i, p_1] ), ( [i, k_2], [i, p_2] ), \dots, ( [i, k_t], [i, p_t] )]$$

der auf diese Art und Weise modifizierte Keller.

$\Rightarrow$

$$p_t = j+1.$$

Falls  $v(i, j)$  bezüglich einer Zeilenlösung definiert wurde, dann gilt  $S(j+1) = S'(j)$  und wir sind fertig.

Annahme:

$v(i, j)$  wurde nicht bezüglich einer Zeilenlösung definiert.

$\Rightarrow$



(16)

Oberhalb auf  $S'(j)$  sind möglicherweise einige Kanten  $([i, k_e], [i, p_e])$ , die nicht  $\langle k_e, j+1 \rangle$ -wesentlich sind.

Ziel:

Entfernung dieser Kanten aus dem Keller.

Beobachtungen 1 und 2  $\Rightarrow$

Wir können den Keller von oben nach unten inspizieren und entscheiden, welche Kanten zu entfernen sind. Dies sieht man folgendermaßen ein:

Sei

$([i, k_s], [i, p_s])$

die oberste Kante in  $S'(j)$ , die  $\langle k_s, j+1 \rangle$ -wesentlich ist.

$\Rightarrow$

$([i, k_s], [i, p_s])$  ist die oberste Kante, so dass der Knoten  $[i, k_s]$  nicht durch  $[i, j]$   $p_s - 0$  = minimiert wird.

$\Rightarrow$

Folgende Bedingungen sind erfüllt:

$$1) v(i, k_s) + c([i, k_s], [i, p_s]) \leq v(i, j) + c([i, j], [i, p_s])$$

und

2) Falls  $s < t$ , dann gilt

$$v(i, k_{s+1}) + c([i, k_{s+1}], [i, p_{s+1}]) > v(i, j) + c([i, j], [i, p_{s+1}]).$$

⇒

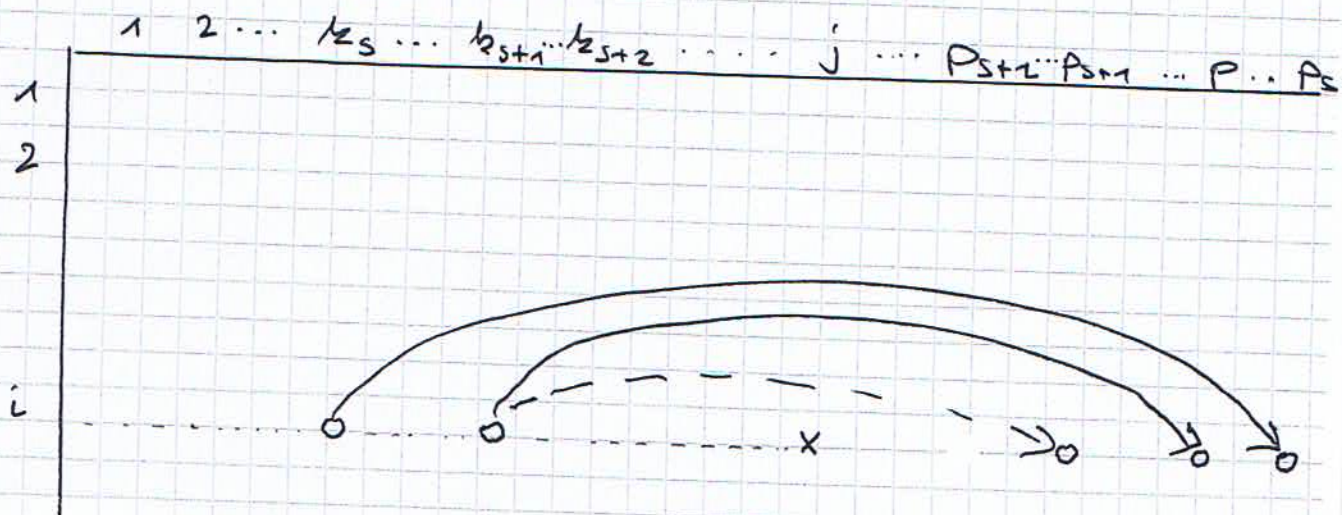
- i) Genau die Kanten oberhalb  $([i, k_s], [i, p_s])$  in  $S(j)$  sind zu entfernen.
- ii) Falls  $s < t$ , dann kann eine der Kanten mit Anfangsknoten  $[i, k_{s+1}]$  und Kopf  $[i, p]$ ,  $p_{s+1} < p < p_s < k_{s+1}, j+1$  - wesentlich sein.
- iii) Sei  $p \in \{p_{s+1}+1, p_{s+1}+2, \dots, p_s-1\}$  minimal, so dass  $[i, k_{s+1}] < j+1, p$  - dominant.

Beobachtungen 1 und 2 ⇒

$([i, k_{s+1}], [i, p])$  ist die einzige  $< k_{s+1}, j+1 >$  - wesentliche Kante.

Falls  $p$  nicht existiert, dann gibt es keine solche Kante.

Situation:



Die Kante  $([i, j], [i, j+1])$  ist genau dann  $\langle j, j+1 \rangle$ -wesentlich, wenn  $s < t$ . Dann muss diese noch auf den Keller abgelegt werden.

Insgesamt wird somit  $S(j+1)$  durch folgenden Algorithmus aus  $S(j)$  berechnet:

Algorithmus  $S(j) \rightsquigarrow S(j+1)$

Eingabe:  $S(j)$

Ausgabe:  $S(j+1)$

Methode:

(1) if  $\text{Kopf}(S \text{ECK}(S(j))) = [i, j+1]$

then

POP

else

Ersetze  $\text{Kopf}(\text{TOP}(S(j)))$  durch  $[i, j+1]$   
fi; (\* Sei  $t$  die Höhe des resultierenden Kellers \*)

(2) Berechne größtes  $s \leq t$  mit

$$v(i, k_s) + c([i, k_s], [i, p_s]) \leq v(i, j) + c([i, j], [i, p_s]);$$

if  $s < t$

then

Entferne alle  $([i, k_q], [i, p_q])$  mit  $q > s$   
fi;

(3) if  $s < t$

then

Berechne  $p \in \{p_{s+1}, \dots, p_{s-1}\}$  mit

$$v(i, k_{s+1}) + c([i, k_{s+1}], [i, p-1])$$

$$> v(i, j) + c([i, j], [i, p-1])$$

und

$$v(i, k_{s+1}) + c([i, k_{s+1}], [i, p])$$

$$\leq v(i, j) + c([i, j], [i, p])$$

~~fi~~ ;

if  $p$  definiert

then

PUSH  $([i, k_{s+1}], [i, p])$

~~fi~~ ;

(4) if  $s < t$

then

PUSH  $([i, j], [i, j+1])$

~~fi~~.

Zeitanalyse:

Berechne GZ die für die Konstruktion von  $RMIN(i)$  benötigte Gesamtzeit und KZ diejenige Zeit, die zur Manipulation des Kellers benötigt wird.

Dann gilt:  $GZ = O(n) + KZ$ .

(17)

Ziel: Abschätzung von KZ.

Beobachtung:

Nach jeder Betrachtung eines Knotens der Matrix werden maximal zwei Kanten auf den Keller abgelegt.

⇒

Die für PUSH- und POP-Operationen benötigte Gesamtzeit ist  $O(n)$ .

Falls wir im Schritt (3) binäre Suche verwenden, dann genügt für eine Ausführung im Schritt (3)  $O(\log n)$  Zeit.

⇒

$$KZ = O(n \cdot \log n).$$

Bemühtlich den Spalten und den Diagonalen des Distanzgraphen verfahren wir analog.

Übung:

Entwickeln Sie einen Algorithmus, der bei konvexen Kostenfunktionen für  $0 \leq i \leq m$ ,  $0 \leq j \leq n$  alle  $v(i, j)$  in  $O(m \cdot n \log(m+n))$  Zeit berechnet.

Um eine kompakte Repräsentation des Distanzgraphen in  $O(m \cdot n \log(m+n))$  Zeit zu berechnen muss noch einiges an Arbeit investiert werden.

hn

(17)

N. Blum, Speeding up dynamic programming...

ist folgendes Satz bewiesen:

### Satz 3.3

Seien  $x := x_1 x_2 \dots x_n$  und  $y = y_1 y_2 \dots y_m$  zwei Sequenzen. Dann kann bei konkaven Kostenfunktionen eine kompakte Repräsentation des Distanzgraphen  $D_{D_3}(x, y)$  in  $O(m \cdot n \cdot \log(m+n))$  Zeit und  $O(m \cdot n)$  Platz berechnet werden.

### 3.3 Algorithmen zur Mustererkennung in Sequenzen

Seien  $x = x_1 x_2 \dots x_n$  und  $y = y_1 y_2 \dots y_m$  zwei Strings über ein endliches Alphabet  $\Sigma$ . Ein Alignment zwischen  $y$  und einem Teilstring von  $x$  heißt lokales Alignment. Ein Alignment zwischen einem Teilstring von  $y$  und einem Teilstring von  $x$  heißt Subalignment.

Folgende Probleme werden wir betrachten:

#### Problem 1:

Gegeben zwei Sequenzen  $x = x_1 x_2 \dots x_n$ ,  $y = y_1 y_2 \dots y_m \in \Sigma^+$ ,  $n \gg m$ , eine Operationenmenge  $M$  und eine Kostenfunktion  $c_M$  sollen alle Teilsequenzen  $\tilde{x}$  von  $x$  mit lokaler minimaler Distanz von  $y$  und für jedes  $\tilde{x}$  alle korrespondierende lokal optimale lokale Alignments berechnet werden.

## Problem 2:

Gegeben zwei Sequenzen  $x := x_1 x_2 \dots x_n$ ,  $y := y_1 y_2 \dots y_m \in \Sigma^+$ , eine Operationenmenge  $M$  und eine Kostenfunktion  $c_M$  sollen alle Paare  $(\tilde{x}, \tilde{y})$  von Teilsequenzen von  $x$  und von  $y$  mit lokal maximaler Ähnlichkeit und für jedes dieser Paare alle korrespondierenden lokal optimale Subalignments  $\tilde{a}$  berechnet werden.

Wir betrachten in der Vorlesung nur die Operationenmenge  $M_1$ . Zunächst müssen wir den Lokalitätsbegriff formal definieren. Die erste solche Definition wurde 1980 von Sellers gegeben:

Eine Teilsequenz  $\tilde{x}$  von  $x$  hat schwach lokal minimale Distanz von  $y$ , falls für alle Teilsequenzen  $x'$  von  $x$  mit  $x' \subset \tilde{x}$  oder  $\tilde{x} \subset x'$  gilt:

$$c_{M_1}(\tilde{x}, y) \leq c_{M_1}(x', y).$$

Da wir nur die Operationenmenge  $M_1$  zugrunde legen, lassen wir ab sofort den Index  $M_1$  weg.

Ein lokales Alignment  $A(\tilde{x}, y)$  mit  $c(A(\tilde{x}, y)) = c(\tilde{x}, y)$  heißt schwach lokal optimal.

### Kritik:

Folgende Situation kann eintreten:

Eine Teilsequenz  $\tilde{x}$  von  $x$  hat schwach lokal minimale Distanz von  $y$ , obwohl ein Teilstring

$x'$  von  $x$  existiert mit:

- $\tilde{x} \cap x' \neq \emptyset$ . D.h.,  $\tilde{x}$  und  $x'$  überlappen.
- $c(x', y) < c(\tilde{x}, y)$ .

Dies bedeutet insbesondere, dass in einer sinnvollen lokalen Region von  $x$  die Teilsequenz  $\tilde{x}$  nicht lokal minimale Distanz von  $y$  hat, obwohl  $\tilde{x}$  schwach lokal minimale Distanz von  $y$  hat.

Lösung: Verbot der oben beschriebenen Situation.

Dies führt zur folgenden Definition:

Eine Teilsequenz  $\tilde{x}$  von  $x$  hat lokal minimale Distanz von  $y$ , falls für alle Teilsequenzen  $x'$  von  $x$  mit  $\tilde{x} \cap x' \neq \emptyset$  gilt:

$$c(\tilde{x}, y) \leq c(x', y).$$

Ein lokales Alignment  $A(\tilde{x}, y)$  mit  $c(A(\tilde{x}, y)) = c(\tilde{x}, y)$  heißt lokal optimal.

Bezüglich Subalignments ist folgende analoge Definition sinnvoll:

Ein Paar  $(\tilde{x}, \tilde{y})$  von Teilsequenzen von  $x$  und  $y$  hat lokal maximale Ähnlichkeit, falls für alle Paare  $(x', y')$  von Teilsequenzen von  $x$  und  $y$  mit  $x' \cap \tilde{x} \neq \emptyset$  oder  $y' \cap \tilde{y} \neq \emptyset$  gilt:

$$c(\tilde{x}, \tilde{y}) \leq c(x', y').$$



Ein Subalignement  $A(\tilde{x}, \tilde{y})$  mit  $c(A(\tilde{x}, \tilde{y})) = c(\tilde{x}, \tilde{y})$  heißt dann lokal optimal.

Genauso wie früher definieren wir den Editiergraphen  $\Sigma_{M_1}(x, y) = (V, E)$ . Analog zu früher lässt sich auch zur Lösung von Problem 1 sowie auch zur Lösung von Problem 2 ein korrespondierender Distanzgraph  $D'_{M_1}(x, y)$  bzw.  $D''_{M_1}(x, y)$  definieren und auch berechnen.

Übung:

Definieren Sie  $D'_{M_1}(x, y)$  und  $D''_{M_1}(x, y)$  und modifizieren Sie den Algorithmus EDDIS $_{M_1}$ , derart, dass  $D'_{M_1}(x, y)$  bzw.  $D''_{M_1}(x, y)$  berechnet wird. Was ist die Zeitkomplexität Ihres Algorithmus?

Zunächst werden wir uns mit Problem 1 beschäftigen. Hierzu sind folgende Definitionen nützlich:

Ein Pfad  $P$  in  $D'_{M_1}(x, y) = (V, E_d)$  heißt (schwach) lokal optimal, falls  $P$  zu einem (schwach) lokal optimalen Alignment korrespondiert.

Seien  $E_{se}(x, y) := (V_{se}, E_{se})$  und  $E_e(x, y) := (V_e, E_e)$  definiert durch:

$$E_{se} := \{ e \in E_d \mid e \text{ liegt auf einem schwach lokal optimalen Pfad in } D'_{M_1}(x, y) \}$$

$$V_{se} := \{v \in V \mid \exists w \in V: (v,w) \in E_{se} \text{ oder } (w,v) \in E_{se}\}.$$

und

$$E_e := \{e \in E_d \mid e \text{ liegt auf einem lokal optimalen Pfad } P \text{ in } D_{M_1}^1(x,y)\}$$

$$V_e := \{v \in V \mid \exists w \in V: (v,w) \in E_e \text{ oder } (w,v) \in E_e\}.$$

Berechne  $x(P)$  denjenigen Teilstring von  $x$ , der zum Pfad  $P$  korrespondiert.  $y(P)$  ist analog definiert. Sei

$$e(x^j, y^i) := \begin{cases} 0 & \text{falls } i=0 \\ \min\{c(\tilde{x}, y^i) \mid \tilde{x} \text{ ist Suffix von } x^j\} & \text{falls } i>0 \end{cases}$$

Folgendes Lemma ist bei der Lösung von Problem 1 hilfreich:

### Lemma 3.1

Sei  $P := P_1, [i,j], P_2$  ein (schwach) lokal optimaler Pfad. Dann gilt

$$c(P_1, [i,j]) = e(x^j, y^i).$$

Beweis:

Definition  $\Rightarrow$

$$c(P_1, [i,j]) \geq e(x^j, y^i).$$

Annahme:  $c(P_1, [i, j]) > e(x^j, y^i)$ .

Dann existiert ein Pfad  $P'$ , der in Zeile 0 beginnt, mit

$$c(P', [i, j]) < c(P_1, [i, j]).$$

Betrachten wir

$$\tilde{P} := P', [i, j], P_2.$$

Es gilt:  $c(\tilde{P}) < c(P)$ .

Sei  $[m, s]$  der letzte Knoten auf  $P$  und somit auch auf  $\tilde{P}$ .

$\Rightarrow$

$\tilde{P}$  korrespondiert zu einem Alignment zwischen einem Suffix von  $x_1 x_2 \dots x_s$  und  $y$ .

Wegen  $c(\tilde{P}) < c(P)$  folgt somit:

$P$  ist nicht (schwach) lokal optimal.

Dies ist ein Widerspruch zur Voraussetzung des Lemmas.

$\Rightarrow$

Annahme ist falsch, d.h.,  $c(P_1, [i, j]) = e(x^j, y^i)$ . ■

Lemma 3.1 impliziert direkt folgendes Lemma:

Lemma 3.2

$$E_{se}(x, y) \subseteq D'_{M_1}(x, y).$$

Aus den Definitionen von schwach lokal optimal und lokal optimal folgt direkt

$$E_e(x, y) \subseteq E_{se}(x, y).$$

Bezeichne  $P_{e\epsilon}$  einen Pfad vom Knoten  $[0, e]$  zum Knoten  $[m, \epsilon]$  in  $D'_{M_1}(x, y)$ . In den nächsten drei Lemmata beweisen wir einige Eigenschaften von  $E_{se}(x, y)$  und von  $E_e(x, y)$ .

Lemma 3.3

Seien  $P_{e_1 \epsilon_1}$  und  $P_{e_2 \epsilon_2}$  zwei Pfade in  $D'_{M_1}(x, y)$  mit  $\epsilon_1 \leq \epsilon_2$  und  $\epsilon_2 \leq \epsilon_1$ . Falls  $P_{e_i \epsilon_i}$ ,  $i \in \{1, 2\}$  schwach lokal optimal ist, dann gilt:

$$e(x^{\epsilon_i}, y^m) = \min \{ e(x^{\epsilon_1}, y^m), e(x^{\epsilon_2}, y^m) \}.$$

Beweis:

Wegen  $\epsilon_1 \leq \epsilon_2$  und  $\epsilon_2 \leq \epsilon_1$  gilt:

$$x(P_{e_2 \epsilon_2}) \subseteq x(P_{e_1 \epsilon_1}).$$

Also folgt die Behauptung direkt aus der Definition von schwach lokal optimal. ■

Sei

$$e_{e, \epsilon} := \min_{1 \leq j \leq n} \{ e(x^j, y^m) \mid \exists \text{ Pfad von } [e, \epsilon] \text{ nach } [m, j] \text{ in } D'_{M_1}(x, y) \}$$

Dann gilt

$$e_{r,z} = \min \{ e_{r',z'} \mid ([r,z], [r',z']) \in E_{o1} \}$$

Aus Lemma 3.3 folgt direkt folgendes Lemma:

Lemma 3.4

Sei  $([r,z], [r',z']) \in E_d$ . Falls  $e_{r,z} < e_{r',z'}$ ,  
dann gilt  $([r,z], [r',z']) \notin E_{se}$ .

Aus Lemma 3.3 und Lemma 3.4 folgt folgendes  
Lemma:

Lemma 3.5

Sei CC eine Zusammenhangskomponente von  
 $E_{se}(x,y)$  oder  $E_e(x,y)$ . Dann haben alle Pfade  
 $P_{r,z}$  in CC dieselben Kosten.

Folgender Satz besagt, dass es zur Lösung von  
Problem 1 genügt  $E_{se}(x,y)$  bzw.  $E_e(x,y)$  zu  
berechnen.

Satz 3.4

Es existiert eine Bijektion zwischen den Pfaden  
von Zeile 0 zur Zeile  $m$  in

- a)  $E_{se}(x,y)$  und den schwach lokal optimalen Pfaden.
- b)  $E_e(x,y)$  und den lokal optimalen Pfaden.

Beweis:

Konstruktion  $\Rightarrow$

- a)  $E_{se}(x,y)$  enthält alle schwach lokal optimale Pfade.
- b)  $E_c(x,y)$  enthält alle lokal optimale Pfade.

Also genügt es zu zeigen, dass der Graph jeweils keine weitere Pfade von Zeile 0 zur Zeile  $m$  enthält.

Sei  $P_{rk} := e_1, e_2, \dots, e_s$  ein beliebiges Pfad in  $E_{se}(x,y)$  bzw.  $E_c(x,y)$ . Sei

$$e_s := ([m-1, e], [m, k]).$$

Lemma 3.1 und 3.2  $\Rightarrow$

$$c(P_{rk}) = e(x^k, y^m).$$

Falls  $P_{rk}$  ein Pfad in  $E_{se}(x,y)$  ist, dann folgt die Behauptung direkt aus Lemma 3.5, der Definition von  $E_{se}(x,y)$  und der Tatsache, dass es auf  $P_{rk}$  liegt.

Annahme:

$P_{rk}$  ist ein Pfad in  $E_c(x,y)$ , der nicht lokal optimal ist.

Dann existiert ein Pfad  $P_{r'k'}$  in  $D_{0,1}(x,y)$ , so dass

- i)  $c(P_{r'k'}) < c(P_{rk})$  und
- ii)  $x(P_{rk})$  und  $x(P_{r'k'})$  überlappen. D.h.,  $r' \leq r \leq k'$  oder  $r \leq r' \leq k$ .

Da  $e_s \in E_e$ , liegt  $e_s$  auf mindestens einem lokal optimalen Pfad.

Somit folgt aus i) und der Definition der lokalen Optimalität:

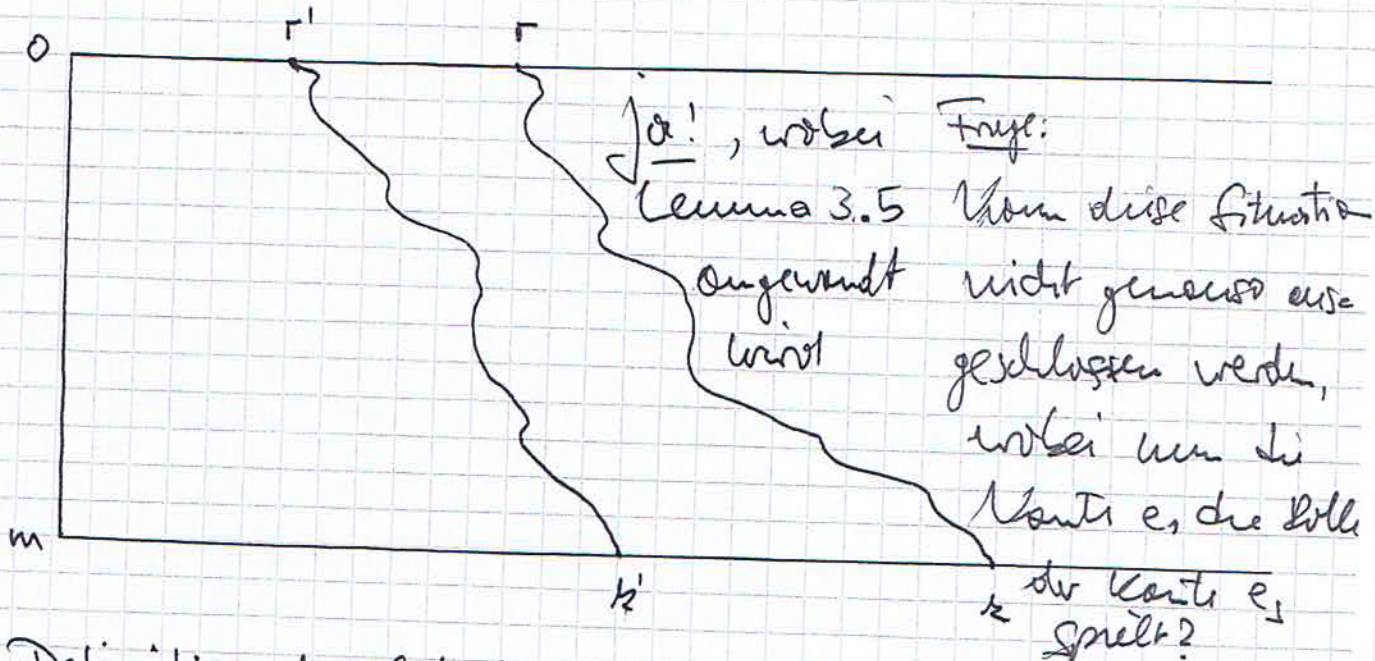
- Das linke Ende von  $x(P_{r \leq z})$  überlappt das rechte Ende von  $x(P_{r' \leq z'})$ .

Ansonsten wäre  $e_s \notin E_e$ .

$\Rightarrow$

$$r' \leq r \leq z'$$

Somit liegt folgende Situation vor:



Definition der lokalen Optimalität impliziert somit:

Für jeden lokal optimalen Pfad  $P_{r \leq z}$ , der  $e_s$  enthält, gilt:  $c(P_{r \leq z}) \leq c(P_{r' \leq z'})$ .

Sei  $e_t$  die erste Kante auf  $P_{r \leq z}$ , die auf einem lokal optimalen Pfad

$$P_{\bar{F}\bar{E}} := Q_1, e_t, Q_2 \text{ mit } c(P_{\bar{F}\bar{E}}) > c(P_{r\bar{k}'})$$

liegt. Da  $e_s$  diese Eigenschaft hat, existiert  $e_t$ .

$\Rightarrow$

$$\exists \text{ Pfad } P_{r\bar{k}''} = e_1, e_2, \dots, e_{t-1}, R \text{ mit } c(P_{r\bar{k}''}) \leq c(P_{r\bar{k}'})$$

Betrachten wir den Pfad  $T := Q_1, R$ . Dann gilt:

- i)  $T$  ist ein Pfad von  $[0, \bar{F}]$  nach  $[m, \bar{k}']$  in  $D'_m(x, y)$  und  $c(T) = c(P_{r\bar{k}''})$ .
- ii)  $x(T)$  ist ein Teilstring von  $x(P_{\bar{F}\bar{E}})$  oder umgekehrt.

Wegen

$$c(P_{\bar{F}\bar{E}}) > c(P_{r\bar{k}'}) \geq c(P_{r\bar{k}''}) = c(T)$$

ist dies ein Widerspruch zur lokalen Optimalität von  $P_{\bar{F}\bar{E}}$ .

Also war unsere Annahme falsch. D.h., jeder Pfad von Zeile 0 zur Zeile  $m$  in  $E_c(x, y)$  ist lokal optimal.



Ziel:

Berechnung von  $E_{se}(x, y)$  und dann von  $E_c(x, y)$ .



Berechnung von  $E_{se}(x,y)$ :

Idee:

(1) (Anwendung von Lemma 3.4)

Führe eine Rückwärts-topologische Suche auf  $D'_{H_1}(x,y)$  durch und entferne dabei alle Kanten  $([r,k], [r',k'])$  mit  $e_{r,k} < e_{r',k'}$ .

(2) Streiche alle Kanten  $([i,j], [i',j'])$  mit  $i > 0$  und  $\text{Eingangsgrad}([i,j]) = 0$ , bis keine solche Kante mehr existiert.

Im folgenden Algorithmus verwenden wir die Variable  $\text{num}([i,j])$ , um die Anzahl der nicht betrachteten direkten Nachfolger des Knotens  $[i,j]$  zu speichern. Zu Beginn ist die Anzahl der direkten Nachfolger des Knotens  $[i,j]$  der Wert der Variablen  $\text{num}([i,j])$ . Nach Betrachten des Knotens  $[i,j]$  setzen wir die Variable  $\text{num}([i,j])$  auf den Wert 4. Damit soll ausgedrückt werden, dass der Knoten  $[i,j]$  bereits betrachtet ist.

Algorithmus BERECHNUNG  $E_{se}(x,y)$

Eingabe: Distanzgraph  $D'_{H_1}(x,y) = (V, E_d)$   
wobei Werte  $e(x^i, y^j) \forall [i,j] \in V$ .

Ausgabe:  $E_{se}(x,y) = (V_{se}, E_{se})$ .

18

## Methode:

(1)  $K := \emptyset;$

(2) for  $j$  from 1 to  $n$

do

for  $i$  from 0 to  $m-1$

do

$e_{i,j} := \infty;$

$\text{num}([i,j]) := \text{outdeg}_{E_d}([i,j])$

od;

$e_{m,j} := e(x^j, y^m);$

$\text{num}([m,j]) := 0$

od;

(3) while  $\exists [r,k]$  mit  $\text{num}([r,k]) = 0$

do

Wähle solch einen Knoten  $[r,k];$

$\text{num}([r,k]) := 4;$

for alle  $([r,k], [r',k']) \in E_d$

do

if  $e_{r,k} = e_{r',k'}$

then

$K := K \cup \{([r,k], [r',k'])\}$

fi

od;

for alle  $([r',k'], [r,k]) \in E_d$

do

$e_{r',k'} := \min \{ e_{r',k'}, e_{r,k} \};$

$\text{num}([r',k']) := \text{num}([r',k']) - 1$

od

od;

(4) while  $\exists ([i,j], [i',j']) \in K$  mit  $i > 0$  und  $\text{indeg}_K([i,j]) = 0$

do

$$K := K \setminus \{([i,j], [i',j'])\}$$

od ;

(5)  $E_{se} := K$  ;

(6)  $V_{se} := \{v \in V \mid \exists w \in V : (v,w) \in E_{se} \text{ oder } (w,v) \in E_{se}\}$

Übung:

Beweisen Sie die Korrektheit des Algorithmus BERECHNUNG  $E_{se}(x,y)$  und entwickeln Sie Datenstrukturen für die Implementierung. Was ist die von dem Algorithmus benötigte Zeit?

Insgesamt haben wir folgenden Satz bewiesen:

Satz 3.5

Sei  $M$ , die Operationenmenge und sei  $D^1_{M_1}(x,y)$  gegeben. Dann kann  $E_{se}(x,y) = (V_{se}, E_{se})$  in  $O(m \cdot n)$  Zeit berechnet werden, wobei  $|x| = n$  und  $|y| = m$ .

Berechnung von  $E_e(x,y)$ :

Idee:

Ausgehend vom Graphen  $E_{se}(x,y)$  erhalten wir  $E_e(x,y)$ , indem wir aus  $V_{se}$  alle Knoten in  $V_{se} \setminus V_e$  und dann aus  $E_{se}$  alle Kanten in  $E_{se} \setminus E_e$  entfernen.

Für die Durchführung benötigen wir einige Bezeichnungen.

Sei  $CC$  eine Zusammenhangskomponente in  $E_{se}(x, y)$  (18)

Dann bezeichnet

- $c(CC)$  die eindeutig bestimmten Kosten  $c(P_{r,z})$  für  $P_{r,z} \in CC$ .

- $\text{Left}(CC)$  die erste Position von  $CC$  bzgl.  $x$ . D.h.

$$\text{Left}(CC) := \min_{1 \leq k \leq n} \{ k \mid [0, k-1] \in CC \}.$$

- $\text{Right}(CC)$  die letzte Position von  $CC$  bzgl.  $x$ . D.h.,

$$\text{Right}(CC) := \max_{1 \leq k \leq n} \{ k \mid [n, k] \in CC \}.$$

Als nächstes werden wir mittels einigen Lemmata exakt diejenigen Knoten charakterisieren, die aus  $U_{se}$  zu streichen sind.

### Lemma 3.6

Sei  $CC$  eine Zusammenhangskomponente in  $E_{se}(x, y)$ .

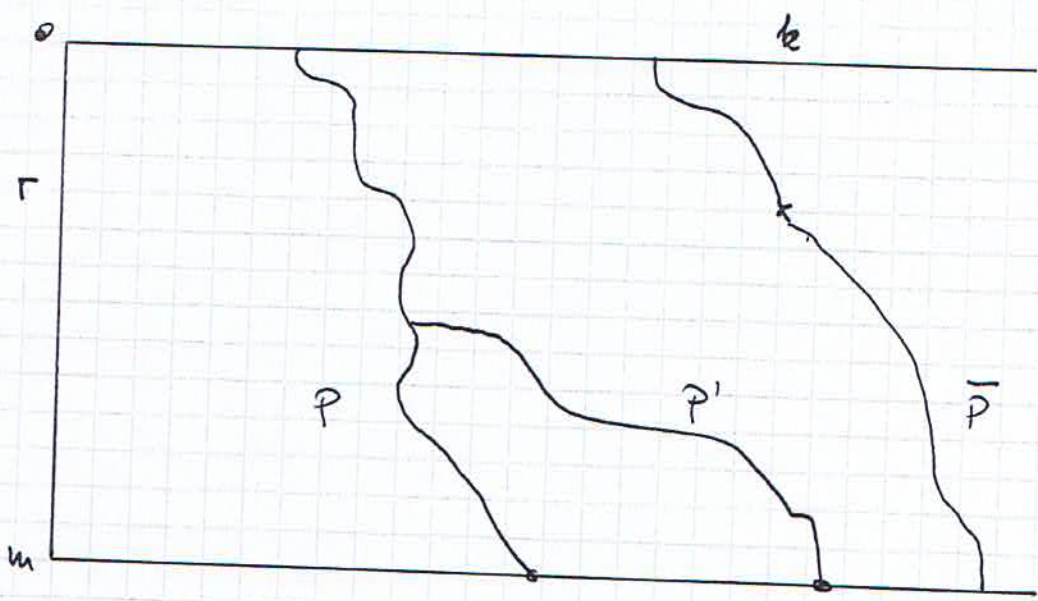
Sei  $[r, k] \in CC$ . Falls es eine Zusammenhangskomponente  $CC'$  in  $E_{se}(x, y)$  mit  $c(CC') < c(CC)$ ,  $\text{Left}(CC') \leq k$  und  $\text{Right}(CC') \geq r$  gibt, dann ist  $[r, k] \notin U_{se}$ .

### Beweis:

Die Behauptung folgt direkt aus der Definition eines lokal optimalen Pfades und der Struktur von  $E_{se}(x, y)$ .

Lemma 3.6 charakterisiert Knoten  $[r, k] \in CC$ , die nicht in  $V_e$  sein können, da  $x_k$  von einem Teilstring von  $x$  überlappt wird, der zu  $y$  eine kleinere Distanz als  $c(CC)$  hat.

Folgendes Beispiel beschreibt eine weitere Situation, in der ein Knoten  $[r, k] \in CC$  nicht in  $V_e$  sein kann, da  $x_k$  von einem Teilstring von  $x$  überlappt wird, der zu  $y$  eine kleinere Distanz als  $c(CC)$  hat.



$$c(P) < c(P') < c(\bar{P})$$

Der zu  $P'$  korrespondierende Teilstring von  $x$  wird von  $c(P)$  ausgeschlossen, ist also nicht in der Zusammenhangskomponente, die  $P$  enthält.

Aber:  $x(P')$  eliminiert  $[r, k]$  aus  $V_e$ !

Wir sagen, ein Knoten  $[r, k] \in CC$  wird direkt ausgeschlossen, falls  $x_k$  von einem Teilstring

18

von  $x$  überlappt wird, der zu  $y$  eine kleinere Distanz als  $c(CCC)$  hat.

Des Weiteren müssen wir alle Knoten  $[r, k]$  ausschließen, die die Eigenschaft besitzen, dass für alle Pfade  $P$  in  $E_{se}(x, y)$  mit  $[r, k] \in P$  gilt:  $x(P)$  wird von einem Teilstrang von  $x$  überlappt, welcher kleinere Distanz als  $c(CCC)$  von  $y$  hat. Daher definieren wir induktiv, dass  $[i, j] \in V_{se}$  ausgeschlossen wird, falls einer der folgenden drei Fälle erfüllt ist:

1.  $[i, j]$  wird direkt ausgeschlossen.
2. Alle direkte Vorgänger von  $[i, j]$  in  $E_{se}(x, y)$  werden ausgeschlossen.
3. Alle direkte Nachfolger von  $[i, j]$  in  $E_{se}(x, y)$  werden ausgeschlossen.

Folgendes Lemma charakterisiert exakt die Knoten in  $V_{se} \setminus V_e$ .

### Lemma 3.7

Sei  $[i, j] \in V_{se}$ . Dann gilt genau dann  $[i, j] \notin V_e$ , wenn  $[i, j]$  ausgeschlossen wird.

## Beweis:

Es ist klar, dass  $[i, j] \notin V_e$ , falls  $[i, j]$  ausgeschlossen wird.

Zum Beweis der anderen Richtung betrachten wir

$$[i, j] \in V_{se} \setminus V_e.$$

Sei  $CC$  diejenige Zusammenhangskomponente in  $E_{se}(x, y)$ , die den Knoten  $[i, j]$  enthält.

$$[i, j] \notin V_e \Rightarrow$$

Alle Teilstrings  $x'$  von  $x$  mit  $x' = x(P)$  für einen Pfad  $P = P_1, [i, j], P_2 \in CC$  von Zeile 0 nach Zeile  $n$  werden von einem Teilstring von  $x$  überlappt, dessen Distanz zu  $y$  kleiner als  $c(CC)$  ist.

$\Rightarrow$

Auf all diesen Pfaden  $P$  gibt es einen Knoten, der direkt ausgeschlossen wird.

## Annahme:

Es existieren Pfade  $Q = Q_1, [i, j], Q_2$  und  $R = R_1, [i, j], R_2$  in  $CC$ , so dass kein Knoten auf  $Q_1, [i, j]$  und kein Knoten auf  $[i, j], R_2$  direkt ausgeschlossen wird.

$\Rightarrow$

$x(Q_1, [i, j], R_2)$  wird nicht von einem Teilstring von  $x$  überlappt, dessen Distanz zu  $y$

kleiner als  $c(CC)$  ist.

$\Rightarrow$

$[i, j] \in V_e$ , Widerspruch.

Also ist mindestens einer der folgenden zwei Fälle erfüllt:

- i) Auf allen Pfaden  $Q_1, [i, j]$  gibt es einen Knoten, der direkt ausgeschlossen wird.
- ii) Auf allen Pfaden  $[i, j], R_2$  gibt es einen Knoten, der direkt ausgeschlossen wird.

In beiden Fällen kann leicht mittels Induktion bewiesen werden, dass  $[i, j]$  ausgeschlossen wird.

Übung:

Durchführung des Induktionsbeweises. ■

Mit Hilfe des Lemmas 3.7 ist es nun einfach,  $E_e(x, y) = (V_e, E_e)$  zu berechnen. Wir erhalten folgenden Algorithmus:

Algorithmus BERECHNUNG  $E_e(x, y)$

Eingabe:  $x = x_1 x_2 \dots x_n, y = y_1 y_2 \dots y_m \in \Sigma^+,$   
Kostenfunktion  $c_{M_1}, D_{M_1}'(x, y).$

Ausgabe:  $E_e(x, y) = (V_e, E_e).$



## Methode:

- (1) Berechne gleichzeitig  $E_{se}(x, y) = (V_{se}, E_{se})$ , die Zusammenhangskomponenten  $CC_1, CC_2, \dots$  und für jede Zusammenhangskomponente  $CC_p$  die benötigte Information  $c(CC_p)$ ,  $Left(CC_p)$  und  $Right(CC_p)$ .
- (2) Berechne alle Knoten, die direkt ausgeschlossen werden. Sei  $K_0$  die Menge dieser Knoten.
- (3) Mittels einer topologischen Suche auf  $E_{se}(x, y)$ , wobei wir in den Knoten, die direkt "von links" ausgeschlossen werden, starten, berechne alle Knoten  $[i, j]$ , die ausgeschlossen werden, da alle direkte Vorgänger von  $[i, j]$  ausgeschlossen werden. Sei  $K_1$  die Menge dieser Knoten.
- (4) Mittels einer Rückwärts-topologischen Suche auf  $E_{se}(x, y)$ , wobei wir in den Knoten, die direkt "von rechts" ausgeschlossen werden, starten, berechne alle Knoten  $[i, j]$ , die ausgeschlossen werden, da alle direkte Nachfolger von  $[i, j]$  ausgeschlossen werden. Sei  $K_2$  die Menge dieser Knoten.
- (5)  $V_e := V_{se} \setminus (K_0 \cup K_1 \cup K_2)$ .
- (6)  $E_e := E_{se} \cap (V_e \times V_e)$ .

## Übung:

Arbeiten Sie den Algorithmus BERECHNUNG  $E_e(x, y)$  aus und analysieren Sie die benötigte Zeit.

Insgesamt erhalten wir somit folgenden Satz:

### Satz 3.6

Sei  $M_1$  die Operationenmenge und sei  $D'_{M_1}(x, y)$  gegeben. Dann kann  $E_e(x, y) = (V_e, E_e)$  in  $O(m \cdot n)$  Zeit, wobei  $|x| = n$  und  $|y| = m$ , berechnet werden.

## Übung:

Modifizieren Sie die bzgl. der Operationenmenge  $M_1$  entwickelte Lösung von Problem 1, so dass Problem 2 gelöst wird.

(Hinweis: Definieren Sie analog zu Left(CC) und Right(CC) Upper(CC) und Lower(CC) und modifizieren Sie Lemma 3.7 entsprechend).

## Bemerkung:

Eine Lösung von Problem 1 und von Problem 2 bzgl. der Operationenmenge  $M_3$  kann in

N. Blum, Speeding up dynamic programming... gefunden werden.