Fluss: Funktion $f: V^2 \to R$
(1) $f \le k$
(2) A x,y e V: f(x,y) = -f(y,x)
(3) A x e V\{q,s}: 0 = Sum(f(x,V))

bei präfluss statt (3):
A x e V\{q}: ü(x) = Sum(f(V,x)) >= 0

Höhenfunktion h: V -> N0:
(1) h(q) = |V|
(2) h(s) = 0
(3) A (x,y) e E_f: h(x) - h(y) <= 1

An die Spitze
generiere L
x = head(L)
A x e V: i_x = Grad(x)
While x != NIL
h_alt = h(x)
LEERE(x)
if h_alt < h(x)
x in front of L
x = nachfolger von x in L

Pusch(x,y) bis zu kapazität des überschusses aus x nach y nur wenn,
ü(x) > 0
x e V\{q,s}
h(x) - h(y) = 1

Bem Residualkapazität k_f = k - f
k_f(x,y) + k_f(y,x) = k(x,y) + k(y,x)

Lifte(x)
h(x) = 1 + min{h(y) | (x,y) e E_f} nur wenn,
x e V\{q,s}
ü(x) > 0
(3) h(x) <= min {h(y) | (x,y) e E_f}
zu (3): keine tiefergelegenen nachbarknoten im residualgraph

Sym Diff D = (P\Q) u (Q\P)
besteht aus Wegen, deren Kanten alternierend in P und Q liegen
mit P eine Paarung und Q eine maximale Paarung. Zyklen haben daher gerade Länge

SCHNITT
While |V| >= 3
wähle zufällig Kante (x,y) e E
entferne alle (x,y) aus E
verschmelze x mit y
gib S = E aus

LEERE
solange ü(x) > 0
if i_x > 0
y = n_x(i_x)
if (x,y) puschbar, pushe es
else i_x --
else Lifte(x)
i_x = Grad(x)

mit n_x array der nachbarn von x

E[Sum(X_i)] = Sum(E[X_i])
Pr[E1 | E2] = Pr[E1 ^ E2] / Pr[E2] bedingte wahrscheinlichkeit für eintreten von Ereignis1 unter Voraussetzung E2
Wenn: Pr[E_1 | E_2] = Pr[E_1] oder Pr[E_1] * Pr[E_2] = Pr[E_1 ^ E_2] heißen E_1 und E_2 unabhängig

Folgerung: Pr[Cut(E_i)] = Pr[E_1 | E_2 ^ ... ^ E_n] * ... * Pr[E_n-1 | E_n] * Pr[E_n]

WERT
1: (y,z) = Paar der Kinder von x in zufälliger Reihenfolge
2:if (WERT(y) == 1) return 0
3: else return 0 NOR WERT(z)

VonNeumannMinMax W := max_p(min_q(p^t*M*q)) = min_q(max_p(p^t*M*q)) gilt immer
und ein paar optimaler Strategien (p,q) heißt Lösung

Für festes p wird p^t*M*q minimal, wenn q = e_j <- d.h. wähle minimalen Eintrag aus p^t*M
damit Loomis Satz: W := max_p(min_j(p^t*M*e_j)) = min_q(max_i(e_i^t*M*q))

Wert und Lösung bei reinen Strategien
falls W := min_j(max_i(m_ij)) =
max_i(min_j(m_ij))
heißt W wert des spiels und das dazugehörige (i,j) Lösung

Strategien
-Zeile i ist optimale Strategie von A, falls sie A's Mindestgewinn: min_j(m_ij) maximiert
-Spalte j ist optimale Strategie von B, falls sie B's Maximalverlust: max_i(m_ij) minimiert
-Wählen also beide optimale Strategien gilt:
max_i(min_j(m_ij)) <= Gewinn(A) <= min_j(max_i(m_ij))

Bei gemischten Strategien A verfolt p und B verfolgt q:
-E[Gewinn(A)] = p^t*M*q
-Strategie von A optimal, wenn min_q(p^t*M*q) maximal
-Strategie von B optimal, wenn max_p(p^t*M*q) minimal
-Wählen beide optimal:
max_p(min_q(p^t*M*q)) <= E[Gewinn(A)] <= min_q(max_p(p^t*M*q))

Komplexitäten
m_ij ist Laufzeit von A_j(E_i)
max_i(m_ij) ist worst-case Laufzeit von A_j
min_j(max_i(m_ij)) ist deterministische komplexität von P bzgl A =: K_d
min_j(m_ij) ist Laufzeit des schnellsten Algos für E_i
max_i(min_j(m_ij)) ist stochastische Komplexität von P bzgl A =: K_s

K_s <= K_d

gemischte Strategien:
E[Laufzeit] = p^t*M*q
max_p(min_q(p^t*M*q)) heißt Verteilungskomplexität =: K_v
K_s <= K_v <= K_d (mit Loomi und Neumann)

Yaos Technik Für alle p und q:
min_j p^t*M*e_j <= K_v <= max_i e_i^t*M*q
damit kann die worst-case Laufzeit aller stochastischen Algorithmen durch Laufzeit des schnellsten deterministischen nach unten abgeschätzt werden

BRZ

```
Eingabe:
        Ein Polyeder P ⊂ ℝ³
        orientierte, planare, disjunkte
        Polygone P_1,...,P_n ⊂ ℝ³
Ausgabe:
        Ein RZB fuer P_1,...,P_n

        k ← 1
        for i = 1,...,n
        Q_k ← P_i ∩ P
        falls Q_k ≠ ∅
                k ← k + 1
l ← 1
falls ∃j: Q_j zerlegt P
        l ← j
Wurzel leftarrowQ_l

        ⋮ // rekursiver Aufruf mit Haelften
```

Dualität

$\mathbf{u}^t\mathbf{x} = 1,\ \mathbf{u} \in \mathbb{R}^n$

$\mathbf{u}* := \{\mathbf{x} \in A | \mathbf{u}^t\mathbf{x} = 1\}$ heißt Hyperebene

$\mathbf{u} \in A$ wird der zu $\mathbf{u}*$ *duale Punkt* genannt. $\mathbf{x}*$ die zum Punkt $\mathbf{x}$ *duale HE.*

$A* := \{\mathbf{x}* | \mathbf{x} \in A\}$ ist der *Dualraum* zu A. $\mathbf{u}^t\mathbf{x} \leq 1,\ \mathbf{u} \neq 0$ bildet den *Halbraum* $\mathbf{u}^{\leq}$.

Satz.: $[[M]] :=$ closure $[M]$

$$M_p := \{\mathbf{u} | M \subset \mathbf{u}^{\leq}\} \text{ Polarmenge von } M$$

$$M_{pp} = \{\mathbf{x} | M_p \subset \mathbf{x}^{\leq}\} = [[M]]$$

Konvexe Hüllen

$$M \subset A \text{ heißt konvex } :\Leftrightarrow$$
$$\forall \mathbf{a}, \mathbf{b} \in M \forall t \in [0,1]:$$
$$\mathbf{x} := \mathbf{a}(1-t) + \mathbf{b}t \in M$$

Def.: $[M] :=$ konv $M =$ konv Hülle M

PLEDGE–STRATEGIE

```
Solange R ∈ L
        gehe vorwaerts,
        bis Wand kontaktiert wird

        gehe links der Wand,
        bis ∉ L oder φ=0
```

Drehwinkel immer im Vergleich zur Ausgangsposition. Immer Aufsummieren $\Rightarrow$ mehr als 2π möglich. Wenn $\phi = 0$ bleibt R nicht stehen, sondern geht geradeaus.

Polygone

Polygonzug: $P : p_1, ..., p_m, p_i$ Ecken

P geschlossen $\Leftrightarrow p_1 = p_m$

P einfach $\Leftrightarrow$ Kanten schneiden sich nicht

planares Gebiet$\hat{=}$einfaches Polygon $\Leftrightarrow$ Rand einf. Polygonzug

WANZE

```
Solange r ≠ z
        lauf in Richtung z
        bis r = z oder ∃i: r ∈ P_i
        falls r ≠ z
                umlaufe P_i und suche
                ein q ∈ min_{x∈P_i} ||x − z||_2
                gehe zu q
```

Bewegung von WANZE kann mit z, l, r beschrieben werden. $\exists$ universelles Steuerwort.

Suche in Polygonen

Bem.: nicht kompetitiv, aber $\exists$ Strategien für die gilt $\frac{l}{d} \in O(\#$ Ecken von $P)$.

Es gibt nur einen kürzesten Weg von $\mathbf{s}$ nach $\mathbf{z}$, dessen Ecken außer $\mathbf{s}$ und $\mathbf{z}$ nur Ecken von P sind.

Kompetitivität

Problem P, Eingabemenge E, Algorithmus A. $k_{opt} : E \rightarrow N$ und $k_A : E \rightarrow N$ Größe der optimalen und der mit A berechneten Lsg.

Falls $k_A \leq a + ck_{opt} \forall$ Eingaben E, heißt A c-kompetitiv.

TUERSUCHE

```
i ← 1
Bis Tuer gefunden
        gehe i Meter der Wand entlang
        und zurueck (Laufrichtung aendert sich)
        (1) i = i + 1 // nicht kompetitiv
        (2) i = 2 * i // 9-kompetitiv
```

Bsp.: Competitivitätsbeweis

$$\begin{aligned}\text{Weglänge } &\leq 2\sum_{i=0}^{n+1} 2^i + 2^{n+\delta} \\ &\leq 2^{n+3+\delta} + 2^{n+\delta} \\ &\leq 9 * 2^{n+\delta}\end{aligned}$$

```
Sternsuche
∀i ∈ ℕ_0 : f_i ← (m/(m−1))^i
i ← 0
Bis z gefunden
        gehe f_i Einheiten auf H_{i mod m}
        entlang und zurueck
        i ← i + 1
```

$$\text{Kompetitivität: } c := 2m\left(\frac{m}{m-1}\right)^{m-1} + 1$$
$$= 2m\left(1 + \frac{1}{m-1}\right)^{m-1} + 1 < 2me + 1$$

Summen

$$\sum_{n=1}^{N} n = \frac{N(N+1)}{2}$$

$$\sum_{n=0}^{N} q^n = \frac{1-q^{n+1}}{1-q},\ q \neq 1$$

$$\sum_{k=-\infty}^{\infty} \binom{n}{k} a^k b^{n-k} = (a+b)^n$$

$$\sum_{k=0}^{n} k * k! = (n+1)! - 1$$

$$\sum_{i=1}^{n} if(i) = \sum_{i=1}^{n}\sum_{j=1}^{n} f(j)$$

$$\sum_{i=1}^{n} \frac{1}{j} < 1 + \text{logn}$$

Lineare Ungleichung:

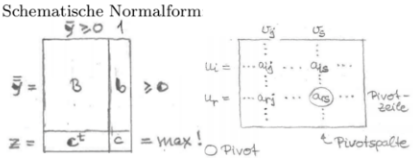
$$y := \mathbf{u}^t\mathbf{x} + u \ge 0$$

Ungleichungssystem mit l linearen Ungleichungen

$$y_i := \mathbf{a}_i^t\mathbf{x} + a_i \ge 0$$

bilden konvexes Polyeder S, *Simplex!*

Schematische Normalform



AUSTAUSCH:

Eingabe:
    $A = [a_{ij}]_{i,j=1,1}^{m,n}$
    r, s
Ausgabe:
    $A' = [a'_{ij}]_{i,j=1,1}^{m,n}$

For $i \neq r,\ j \neq s$
    $a'_{ij} \leftarrow a_{ij} - \frac{a_{is}a_{rj}}{a_{rs}}$
For $i = r,\ j \neq s$ (Pivotzeile)
    $a'_{ij} \leftarrow -\frac{a_{ij}}{a_{rs}}$
For $i \neq r,\ j = s$ (Pivotspalte)
    $a'_{ij} \leftarrow \frac{a_{ij}}{a_{rs}}$
For $i = r,\ j = s$ (Pivot)
    $a'_{ij} \leftarrow \frac{i}{a_{rs}}$

Algorithmus:

Eingabe:
    Normalform B eines lin. Programms
Ausgabe:
    Normalform geaendert, sodass $z(\mathbf{0})$=c=max

solange ein $c_s > 0$
    falls alle $b_{is} \ge 0$
        keine Loesung − Ende
    sonst
        bestimme r so, documentclass
        $\frac{b_r}{b_{rs}} = \max_{b_{is}<0} \frac{b_i}{b_{is}}$
    B ← AUSTAUSCH(B, r, s)

Aufwand im worst-case $\Omega\left(m^{\frac{n}{2}}\right)$. In Praxis meist in $O\left(m^2 n\right)$.

Berechnung der Normalform

Gegeben ein lineares Programm

- finde zulässigen Punkt
- mache Punkt zum Ursprung
- führe n Tauschs x_r mit y_r durch

Damit Ursprung beim Tausch von x_r mit y_r gültig bleibt muss für alle $i > r$ gelten:

$$a_i - \frac{a_{ir}a_r}{a_{rr}} \ge 0$$

Duale lineare Programme

Das lineare Programm

$$\mathbf{y} \ge 0$$
$$B^t\mathbf{y} + \mathbf{c} \le 0$$
$$\mathbf{b}^t\mathbf{y} + c = min!$$

ist dual zu

$$\mathbf{x} \ge 0$$
$$B\mathbf{x} + \mathbf{b} \le 0$$
$$\mathbf{c}^t\mathbf{y} + c = min!$$

Ersteres kann transformiert werden zu

$$\mathbf{y} \ge 0$$
$$-B^t\mathbf{y} - \mathbf{c} \ge 0$$
$$-\mathbf{b}^t\mathbf{y} - c = max!$$

Ausgleichen mit Maximumsnorm

$A\mathbf{x} = \mathbf{a}$ überbestimmtes LGS. $\forall\mathbf{x}$ ist das Residuum

$$\mathbf{r} := (r_1 ... r_n)^t := A\mathbf{x} - \mathbf{a} \neq 0$$

Wir definieren $x_0 := \frac{1}{r}$ und $\bar{\mathbf{x}} := \mathbf{x}x_0$. Führt zu linearem Programm:

$$-A\bar{\mathbf{x}} + \mathbf{a}x_0 + \mathbf{e} \ge 0$$
$$A\bar{\mathbf{x}} - \mathbf{a}x_0 + \mathbf{e} \ge 0$$
$$x_0 = max!$$

1 Gradientenverfahren

Ausgehend vom Anfangspunkt $\mathbf{x_0} = [x_1, .., x_n]$, berechne Gradient der Kostenfunktion an dieser Stelle: $\bigtriangledown c(\mathbf{x_0}) = [\partial c/\partial x_1, ..., \partial c/\partial x_n]$. Der Nachfolger ist dann, $\mathbf{x_{i+1}} = \mathbf{x_i} + h * \bigtriangledown c(\mathbf{x_i})$, wobei $h > 0$, falls maximiert werden soll und $h < 0$, wenn minimiert wird.

3 stochastische verfahren

3.1 Allgemein

startzustand $\mathbf{q}$ -¿ einen nachbar $\mathbf{p}$ zufällig wählen -¿ schritt von ausgang zum neuen nachbar akzeptabel?, falls nein neuer nachbar, sonst von akzeptiertem zustand weiter, bis zurfieden

3.2 simuliertes tempern

Hier: Maximalzahl betrachteter nachbarn insgesamt (Bedingung fuer Terminieren), Folge sinkender temperaturen $t_1, t_2, ...$, Akzeptanzbedingung:
$c(\mathbf{p}) > c(\mathbf{q})\ \ OR\ \ exp(\frac{c(\mathbf{p}) - c(\mathbf{q})}{t_i}) > Zuffalszahl \in [0,1]$, nächste Temperatur immer wenn akzeptiert wurde.

3.3 Schwellwert-Algorithmus

Akzeptanzbedingung: $c(\mathbf{p}) > c(\mathbf{q}) - \sigma$, wobei σ immer weiter abgesenkt wird.

3.4 Sintflut-Maximierung

Akzeptanzbedingung: $c(\mathbf{p}) > F$, mit steigender unterer grenze F.

3.5 Rekordjagd-Algorithmus

Akzeptanzbedingung: $c(\mathbf{p}) \ge Rekord - \sigma$, bisher bester gesehener Wert, darf nicht um sinkende Toleranz unterschritten werden.

4 Evolutionäre Algorithmen

4.1 Allgemein

Population aus Individuen mit Merkmalsvektor $\mathbf{q}$ und Fitness c($\mathbf{q}$). Außerdem gibt es noch einen globalen Streuungsvektor σ, aus dem normal oder geometrisch verteilt mutiert wird. Individuen koennen sich fortpflanzen, bei mehreren Eltern Kreuzung, sonst Klon. Populationsgräße wird einigermaßen konstant gehalten, d.h. es werden immer wieder Loesungen verworfen.

4.2 Plus-Evolutionsstrategie

Erzeuge λ Nachkommen, nur die μ fittesten Individuen aus den μ Eltern und λ Nachkommen ueberleben.
Eltern werden pro Kind zufällig aus Population gewählt.

4.3 Komma-ES

Wie bei plus, allerdings sterben alle Eltern garantiert und nur μ fittesten Kinder ueberleben.

4.4 Klonen vs. Mehrere Eltern

Beim Klonen mutiere einfach mithilfe von σ (s.o.).
Bei mehreren Eltern:

- Mischen: Es wird fuer jedes Merkmal (q_i) zufaellig bestimmt, von welchem Elternteil dieses Merkmal uebernommen wird.
- Mitteln: Es wird fuer jedes Merkmal der Mittelwert ueber die Werte der Eltern gebildet.

5 Genetische Algorithmen

Binäre Merkmalsvektoren, nur Nachkommen ueberleben.
Individuum klont sich mit Wahrscheinlichkeit $W(\mathbf{q}) = c(\mathbf{q}) / \sum_{\mathbf{p} \in Pop} c(\mathbf{p})$. μ Eltern erzeugen immer μ Klon-Nachkommen.
Wähle unter den μ Klonen p% Individuen, die gekreuzt werden. Wähle daraus zufällige Paare. Aus a = (a1,...,an) und b = (b1,...,bn) entstehen c = (a1,...,aj,bj+1,...,bn) und d = (b1,...,bj,aj+1,...,an).
Dann kippe jedes Bit des Mermalsvektors mit sehr geringer Wahrscheinlichkeit.
Zusammengehörende Mermalsbits sollten möglichst nahe beisammen stehen, damit sie bei der Kreuzung nicht zerbrechen.

6 Partikel-Schwarm-Optimierung

Partikel jeweils mit Position $p_i(t)$, Fitness $f_i(p_i)$, Geschwindigkeit $v_i(t) = p_i(t) - p_i(t-1)$.
Algorithmus:
Initiale Position und Geschwindigkeit zufällig. Dann sei q_i die bisher beste Position eines Partikels, q die global bisher beste Position.
Dann nächste Werte:
$v_i = v_i * \omega + (q_i - p_i) * c_1 * r_1 + (q - p_i) * c_2 * r_2$ r jeweils aus [0,1], c und ω konstant.
Die Konstanten können per Superschwarm optimiert werden.

Schreibweise

C ist De-Casteljau, C^* Unterteilungsoperator mit Stufe k.

$$C\left(B_0^0, t\right) = B_0^1 B_1^1$$
$$C^*\left(B_0^0, k\right) := B_0^k ... B_{2^k-1}^k$$

Eingabe:
    $\mathbf{b}_0^0 ... \mathbf{b}_n^0 \subset \mathbb{R}^d$
    $t \in \mathbb{R}$
Ausgabe:
    $\mathbf{b}_0^0 ... \mathbf{b}_0^n\ \mathbf{b}_0^n ... \mathbf{b}_n^0 \subset \mathbb{R}^d$

For k = 1, ..., n
    For i = 0, ..., n − k
        $\mathbf{b}_i^k \leftarrow \mathbf{b}_i^{k-1}(1-t) + \mathbf{b}_{i+1}^{k-1} t$

Konvergenz

Satz: Zu jedem Polygon $B = b_0 ... b_n$ gibt es ein Polygon $L_\infty(B)$ vom Grad $\le n$, sodass

$$\sup_{[0,1]} |L_k(B) - L_\infty(B)| \in O\left(2^{-k}\right)$$

Folgerung:

$$\frac{d}{dt} L_\infty(B) = n L_\infty(\Delta B)$$

Differenzenpolygon

$$\Delta B := \Delta\mathbf{b}_0 ... \mathbf{b}_n$$
$$:= (\Delta\mathbf{b}_0) ... (\Delta\mathbf{b}_{n-1})$$
$$:= (\mathbf{b}_1 - \mathbf{b}_0) ... (\mathbf{b}_n - \mathbf{b}_{n-1})$$

Zusammenhang von Differenzen mit normalem Polygon

$$B_0 B_1 := C^*(B, 1)$$
$$\Delta B_0 \Delta B_1 = \frac{1}{2} C^*(\Delta B, 1)$$
$$B_0 ... B_{2^k} := C^*(B, k)$$
$$\Delta B_0 ... \Delta B_{2^k} = 2^{-k} C^*(\Delta B, k)$$

Abhängigkeit von ΔB_0 und ΔB_1 bzgl. C: (Übung 9.2)

$$tL = \Delta B_0 \quad (1-t) R = \Delta B_1$$
$$\Rightarrow \frac{\Delta B_0}{t} \frac{\Delta B_1}{(1-t)} = C(\Delta B, t)$$

Ableitung unterteilter Polygone

$$b(t) := L_k(B)$$
$$\beta_i^{k,j} := \frac{jn + i}{n 2^k}$$
$$b_i^{k,j} := b^k\left(\beta_i^{k,j}\right)$$
$$\dot{b}^k(t) = \frac{\Delta b_i^{k,j}}{\Delta \beta_i^{k,j}} = n 2^k \Delta b_i^{k,j}$$

Kanonische Parametrisierung

Unterteilung eines Polygons in $P^k := \mathbf{P}_0^k ... \mathbf{P}_{2^k-1}^k$ mit

$$P_j^k := \mathbf{p}_0^{k,j} ... p_n^{k,j}$$
$$\mathbf{p}_i^{k,j} := \begin{bmatrix} \beta_i^{k,j} \\ \mathbf{b}_i^{k,j} \end{bmatrix}$$
$$\beta_i^{k,j} := \frac{j + \frac{i}{n}}{2^k}$$

Zu P^k it die stückweise lineare Funktion $L_k(\mathbf{b}_0 ... \mathbf{b}_n)$

Biinfinite Kontrollpolygone: $\mathbf{c}_\mathbb{Z} := (\mathbf{c}_i)_{i\in\mathbb{Z}} = (...\mathbf{c}_{-1}\mathbf{c}_0\mathbf{c}_1...)$

Algorithmus

```
Eingabe:
          c⁰_Z ein Polygon ⊂ R^d
          n ∈ N_0 der Grad
Ausgabe:
          c^m_Z ⊂ R^d

For k = 1, ..., m
          For i ∈ Z //verdoppele
                    d⁰_{2i} ← d⁰_{2i+1} ← c^{k-1}_i
          For j = 1, ..., n
                    For i ∈ Z //mittele
                              d^j_i ← (d^{j-1}_{i-1} + d^{j-1}_i) 1/2
          For i ∈ Z //bennen um
                    c^k_i ← d^n_i
```

$$D = \begin{bmatrix} \ddots & & & \\ & 1 & & \\ & 1 & & \\ & & 1 & \\ & & 1 & \\ & & & \ddots \end{bmatrix} \quad M = \frac{1}{2}\begin{bmatrix} \ddots & & & & \\ & 1 & 1 & & \\ & & 1 & 1 & \\ & & & 1 & 1 \\ & & & & \ddots \end{bmatrix}$$

$$U_n^m \mathbf{c} = (U_n)^m \mathbf{c} = (M^n D)^m \mathbf{c}$$

Allgemein enthält U_n in den Spalten die Einträge α_i, jeweils um zwei Zeilen versetzt:

$$\alpha_i = \begin{cases} \frac{1}{2^n}\binom{n+1}{i}, & i = 0, ..., n+1 \\ 0, & sonst \end{cases}$$

Unterteilungsgleichung:

$$b_i = \sum_{k\in\mathbb{Z}} \alpha_{i-2k} c_k,\ i \in \mathbb{Z}$$

Bsp. α eines stationären Unterteilungsalgorithmus:

$$\alpha(z) := \sum_{j\in\mathbb{Z}} \alpha_j z^j$$

$$\alpha_n(z) := \frac{1}{2^n}\sum_{i=0}^{n+1}\binom{n+1}{i} z^i = \frac{1}{2^n}(1+z)^{n+1}$$

Differenzenschema

Rückwärtsdifferenzen: $\nabla \mathbf{c} := (\nabla c_i)_{i\in\mathbb{Z}}$

$$\nabla c_i := c_i - c_{i-1}$$

Symbol: $v(z) := 1 - z$

$$\nabla \mathbf{c}(z) = v(z)\, c(z)$$

Differenzenpolygone: $b_\nabla(z) = (1-z)\, b(z)$

$$= (1-z)\,\alpha(z)\,\frac{c_\nabla(z^2)}{(1-z^2)}$$

$$= \frac{\alpha(z)}{1+z} c_\nabla(z^2)$$

Bem.: Das Differenzenschema zu $\alpha(z)$ existiert nur, wenn gilt: $\alpha(-1) = \sum_{i\in\mathbb{Z}} \alpha_{2i} - \sum_{i\in\mathbb{Z}} \alpha_{2i+1} = 0$

Bem.: Es muss $\sum_{i\in\mathbb{Z}} \alpha_{2i} = \sum_{i\in\mathbb{Z}} \alpha_{2i+1} = 1$ gelten, damit die unterteilten Polygone gegen eine Kurve konvergieren.

Allgemein

Regelmäßiges biinfinites Kontrollnetz C und das unterteilte Netz B mit den Unterteilungsmatrizen U und V:

$$C := [\mathbf{c}_{ij}]_{i,j\in\mathbb{Z}} = [\mathbf{c_i}]_{\mathbf{i}\in\mathbb{Z}^2} =: \mathbf{c}_{\mathbb{Z}^2}$$
$$B := \mathbf{b}_{\mathbb{Z}^2} := UCV^t$$
$$(U, V) \text{ heißt Tempus}$$

Bem.: Wenn U, V konvergente Kurvenunterteilungsalgorithmen, dann konvergiert $U^k C (V^t)^k$ gegen eine Fläche.

Symbole

$$C = \mathbf{c}_{\mathbb{Z}^2} \Rightarrow \mathbf{c}(\mathbf{x}) := \mathbf{c}(x, y) := \sum_{i\in\mathbb{Z}}\sum_{j\in\mathbb{Z}} \mathbf{c}_{ij} x^i y^j := \sum_{\mathbf{i}\in\mathbb{Z}^2} \mathbf{c_i x^i}$$

$$B := UCV^t \Rightarrow \mathbf{b}(x, y) := \alpha(x)\,\mathbf{c}(x^2, y^2)\,\beta(y)$$

$$(U, V) \Rightarrow \gamma(x, y) := \alpha(x)\,\beta(y) \text{ (Symbol des Tempus)}$$

Unterteilungsgleichung: $\mathbf{b}(\mathbf{x}) = \gamma(\mathbf{x})\,\mathbf{c}(\mathbf{x}^2)$ bzw. komponentenweise: $\mathbf{b_i} = \sum_{\mathbf{j}\in\mathbb{Z}^2} \gamma_{\mathbf{i}-2\mathbf{j}} \mathbf{c_j}$

Masken

Wenn $\Gamma := \gamma_{\mathbb{Z}^2}$ biinfinite Matrix. Also

$$\gamma(x, y) := [...x^{-1}x^0x^1...]\Gamma[...y^{-1}y^0y^1...]^t$$

$\Gamma_{\mathbf{i}} := [\gamma_{\mathbf{i}-2\mathbf{j}}]_{\mathbf{j}\in\mathbb{Z}^2}$ heißen Masken für $\mathbf{i} = (0,0), (1,0), (0,1), (1,1)$

```
Eingabe:
          t = t_1...t_n
          s = s_1...s_m  // Suchtext
Ausgabe:
          kleinstes i mit t_{i+1}...t_{i+m} = s

i <- 0
Solange i ≤ n − m
          // vgl. s mit t_{i+1}...t_{i+m}
          j <- m
          Solange j > 0 und s_j = t_{i+j}
                    j <- j − 1
          Falls j = 0
                    gib i aus; stop
          sonst i <- i + 1
```

Vorkommensfunktion:

$$v : A \to 0, ..., m$$
$$a \mapsto v(a),$$
$$v(a) := min\{\ k | a \notin s_{k+1}...s_m \ und\ (a = s_k \lor k = 0)\}$$

Def.

Ein Präfix eines Wortes w, das zugleich ein Suffix von w ist, heißt *Präsuffix*. *geps*(w) ist das größte echte Präsuffix von w. $w_j := s_{j+i}...s_m$.

$$\gamma(j) := |geps(w_j)|$$

Suffixfunkion σ wird dargestellt:

$$\sigma(j) = min(\{k \in \{1, ..., j\} | \gamma(j-k) = m - j\} \cup \{m - \gamma(0)\})$$

Änderungen in „Naive Suche“

Sei $t_{i+j} \neq s_j$ und $t_{i+j+1}...t_{i+m} = s_{j+1}...s_m$.
Dann ist $v := v(t_{i+j}) \neq j$ und

- falls $v < j$, kann i um $j - v$ erhöht werden
- falls $v > j$, kann i um $m - v + 1$ erhöht werden

Bem.

Für kleines m und großes Alphabet A nun Laufzeit $O(\frac{n}{m})$

```
Eingabe:
          γ(0 ... m)
Ausgabe:
          σ(1 ... m)

For j = 1, ..., m
          σ(j) ← m − gamma(0)
For i = 0, ..., m−1
          k ← m − γ(i) − i
          j ← m − γ(i)
          // Es gilt jetzt γ(j−k) = m − j
          falls σ(j) > k
                    dann σ(j) ← k
```

```
// wie in "Naiver Suche", letzte Zeile
// durch folgendes ersetzt
v ← v(t_{i+j})
falls v < j
          i ← i + max{j−v, σ(j)}
sonst
          i ← i + max{m−v+1, σ(j)}
```