

Algorithmen und Datenstrukturen

Stefan Roth, SS 2025

07

Fortgeschrittene Algorithmen-Entwurfsmethoden

Auswahl Algorithmischer Entwurfsmethoden

Divide & Conquer

Löse rekursiv
(disjunkte) Teilprobleme

(Quicksort, Mergesort)

Backtracking

durchsuche iterativ
Lösungsraum

Dynamisches Programmieren

Löse rekursiv
(überlappende) Teilprobleme
durch Wiederverwenden

Greedy

baue Lösung aus Folge
lokal bester Auswahlen zusammen

(Kruskal, Prim, Dijkstra)

+Metaheuristiken: übergeordnete Methoden für Optimierungsprobleme

Divide & Conquer

(Fast Fourier Transformation, FFT)

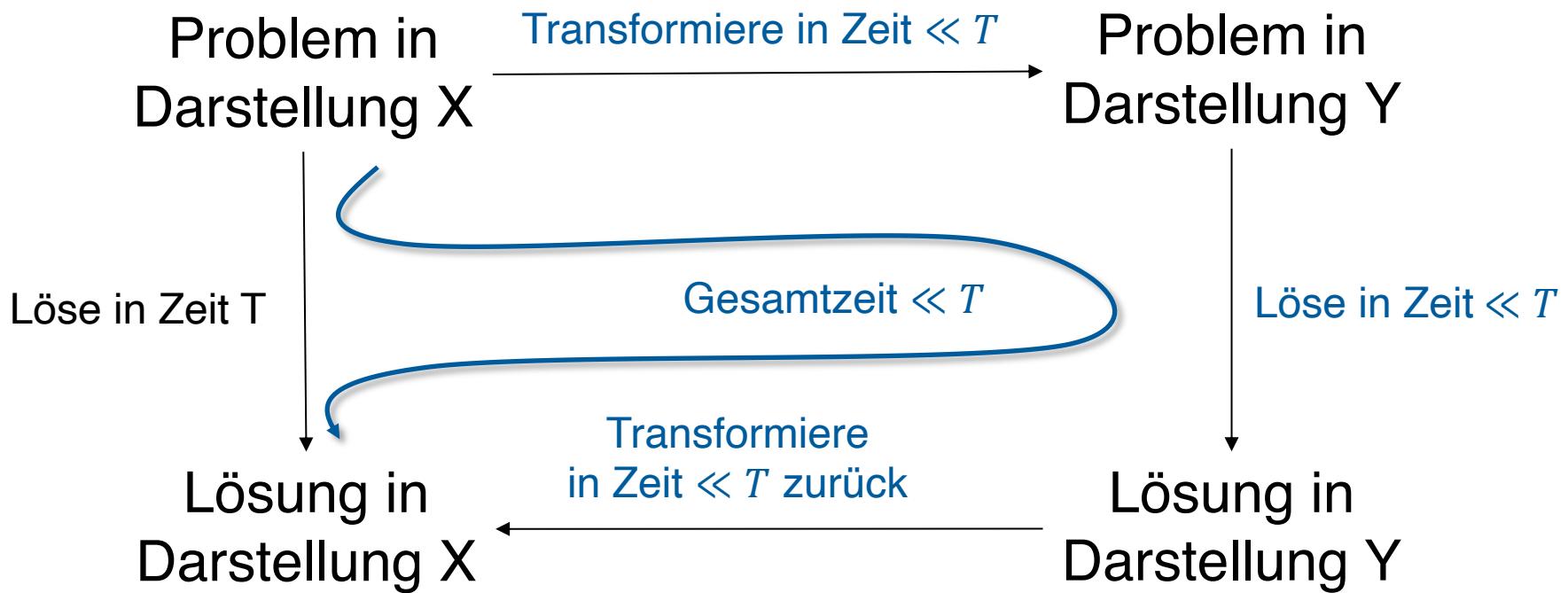
Idee der Fourier-Transformation

Beispiel:
Polynommultiplikation

Beispiel:
 $T = \Omega(n^2)$

Beispiel:
FFT und inverse FFT
in Zeit $O(n \log n)$
per Divide & Conquer

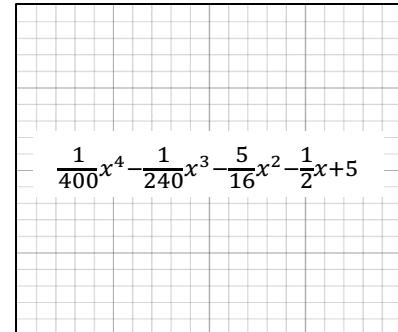
Beispiel:
in Zeit $O(n)$



Polynom: Koeffizientendarstellung (I)

$$p(x) = p_0 + p_1x + p_2x^2 + \cdots + p_{n-2}x^{n-2} + p_{n-1}x^{n-1}$$

mit $p_{n-1} \neq 0$ und $\text{grad}(p(x)) = n - 1$



Gegeben beispielsweise als Array **p[]** der Koeffizienten **p[i]**

(Annahme im Folgenden: **p[i]=0** für **i>=n**)

Schnelle Auswertung an Stelle **w** bei Koeffizientendarstellung (Horner-Methode):

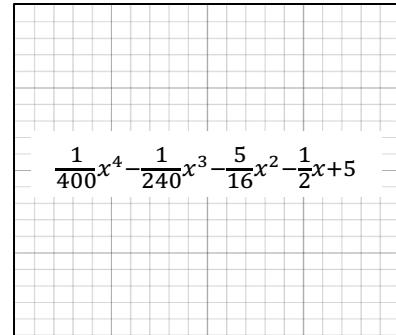
$$p(x) = (((p_{n-1}x + p_{n-2})x + p_{n-3})x + \cdots p_1)x + p_0$$

```
PolyEval(p,n,w) // p[] array of n entries
1 y=p[n-1];
2 FOR i=n-2 DOWNTO 0 DO y=y*w+p[i];
3 return y;
```

Laufzeit $\Theta(n)$

Polynom: Koeffizientendarstellung (II)

Multiplikation zweier Polynome $p(x), q(x)$, beide vom Grad $n - 1$



$$p(x) \cdot q(x) = \left(\sum_{i=0}^{n-1} p_i x^i \right) \cdot \left(\sum_{i=0}^{n-1} q_i x^i \right) \quad \text{vom Grad } 2n - 2$$

$$= \sum_{k=0}^{2n-2} \left(\sum_{j=0}^k p_j \cdot q_{k-j} \right) x^k$$

Faltung / „Konvolution“
der Koeffizienten

Geht das schneller?

```
PolyMult(p,q,n) // p,q arrays of n entries
1 r=ALLOC(2n-1);
2 FOR k=0 TO 2n-2 DO
3   r[k]=0;
4   FOR j=0 TO k DO r[k]=r[k]+p[j]*q[k-j];
5 return r;
```

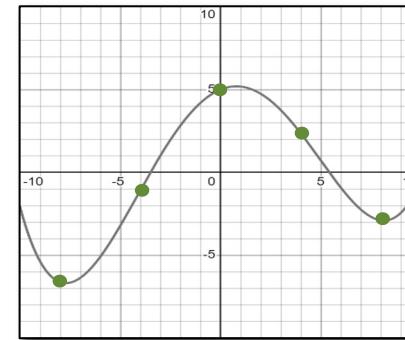
Laufzeit
 $\Theta\left(\sum_{k=0}^{2n-2} k\right) = \Theta(n^2)$

Fourier-Transformation

(I)DFT=(Inverse) Diskrete Fourier-Transformation
(I)FFT=(Inverse) Fast Fourier Transformation

$$\frac{1}{400}x^4 - \frac{1}{240}x^3 - \frac{5}{16}x^2 - \frac{1}{2}x + 5$$

„Transformiere in geeignete Darstellung, rechne dort und transformiere zurück“



Multiplikation von $p(x), q(x)$
vom Grad $n - 1$
in Koeffizientendarstellung

DFT

Darstellung von $p(x), q(x)$
vom Grad $n - 1$
in Punkt/Wert-Darstellung

in Zeit $\Theta(n \log n)$
per FFT

per Faltung
in Zeit $\Theta(n^2)$

$r(x) = p(x) \cdot q(x)$
vom Grad $2n - 2$

Gesamtzeit $\Theta(n \log n)$

in Zeit $\Theta(n)$

in Koeffizientendarstellung

IDFT

$r(x) = p(x) \cdot q(x)$
vom Grad $2n - 2$

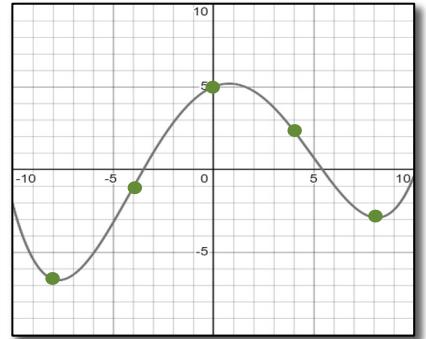
in Punkt/Wert-Darstellung

in Zeit $\Theta(n \log n)$
per IFFT

Polynome: Punkt/Wert-Darstellung (I)

$$p(x) = p_0 + p_1x + p_2x^2 + \cdots + p_{n-2}x^{n-2} + p_{n-1}x^{n-1}$$

mit $p_{n-1} \neq 0$ und $\text{grad}(p(x)) = n - 1$



Gegeben z.B. als Array **p[]** der Punkte/Werte **p[i].x**, **p[i].y**

Eindeutigkeit der Punkt/Wert-Darstellung

Jedes Polynom $p(x)$ über Körper vom Grad $\leq n - 1$ lässt sich eindeutig durch n Punkt/Wert-Paare $(x_j, y_j)_{j=0, \dots, n-1}$ für verschiedene x_j durch $y_j = p(x_j)$ beschreiben.

Zu $(x_j, y_j)_{j=0, \dots, n-1}$ betrachte lineares Gleichungssystem in Variablen p_0, p_1, \dots, p_{n-1} :

$$\begin{aligned} p(x_0) &= p_0 + p_1 x_0 + p_2 x_0^2 + \cdots + p_{n-1} x_0^{n-1} = y_0 \\ p(x_1) &= p_0 + p_1 x_1 + p_2 x_1^2 + \cdots + p_{n-1} x_1^{n-1} = y_1 \\ &\vdots \\ p(x_{n-1}) &= p_0 + p_1 x_{n-1} + p_2 x_{n-1}^2 + \cdots + p_{n-1} x_{n-1}^{n-1} = y_{n-1} \end{aligned}$$

In Matrixform:

$$\begin{pmatrix} 1 & x_0 & \cdots & x_0^{n-1} \\ 1 & x_1 & \cdots & x_1^{n-1} \\ \vdots & \ddots & & \\ 1 & x_{n-1} & \cdots & x_{n-1}^{n-1} \end{pmatrix} \begin{pmatrix} p_0 \\ p_1 \\ \vdots \\ p_{n-1} \end{pmatrix} = \begin{pmatrix} y_0 \\ y_1 \\ \vdots \\ y_{n-1} \end{pmatrix}$$

Vandermonde-Matrix $V(x_0, \dots, x_{n-1})$ mit
 $\det V(x_0, \dots, x_{n-1}) = \prod_{0 \leq i < j \leq n-1} (x_j - x_i)$
 $\det V(x_0, \dots, x_{n-1}) \neq 0$ für verschiedene x_j ,
also eindeutige Lösung p_0, p_1, \dots, p_{n-1}

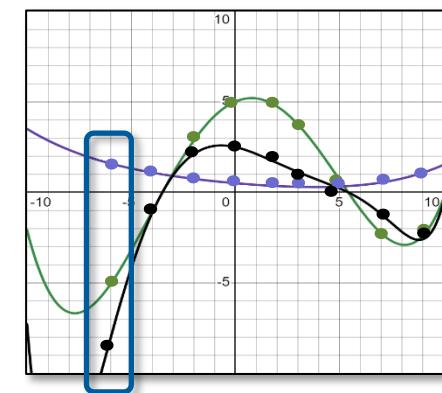
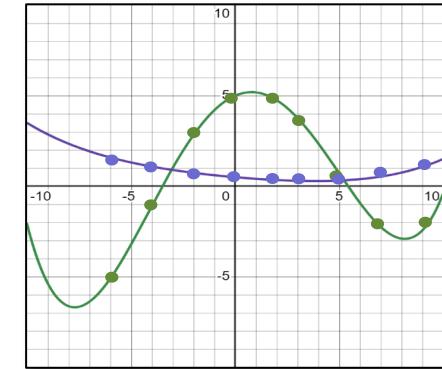
Polynome: Punkt/Wert-Darstellung (II)

Ziel: berechne Produkt $r(x) = p(x) \cdot q(x)$

Polynom-Multiplikation in Punkt/Wert-Darstellung einfach
(sofern gleiche x-Koordinaten x_0, x_1, x_2, \dots für p und q):

$$r(x_j) = p(x_j) \cdot q(x_j) \text{ für alle } j = 0, 1, \dots, 2n - 2$$

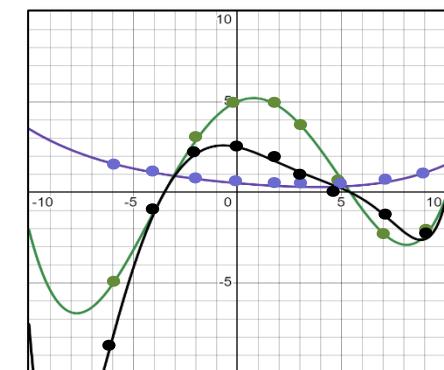
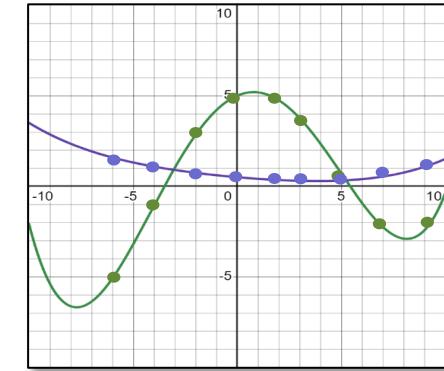
benötigen $2n - 1$ Paare
(damit auch schon für p, q ,
da r vom Grad $2n - 2$)



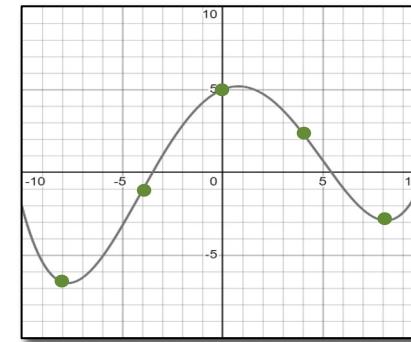
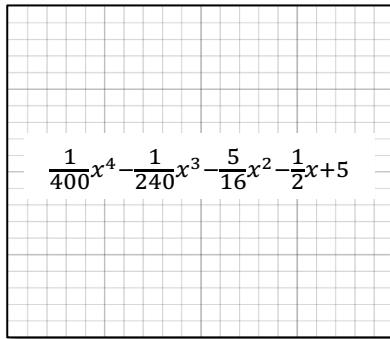
Polynome: Punkt/Wert-Darstellung (III)

```
PolyMult(p,q,n)
// p,q arrays of 2n-1 entries x,y
// p[i].x=q[i].x for all i
1 r=ALLOC(2n-1);
2 FOR i=0 TO 2n-2 DO
3     r[i].x = p[i].x;
4     r[i].y = p[i].y * q[i].y;
5 return r;
```

Laufzeit $\Theta(n)$



Diskrete Fourier-Transformation berechnen (I)



$$p(x) = p_0 + p_1 x + \cdots + p_{n-1} x^{n-1}$$

$$DFT_n(p) = (x_j, p(x_j))_{j=0, \dots, 2n-2}$$

Problem:

Wir benötigen $2n - 1$ Punkt/Werte-Paare

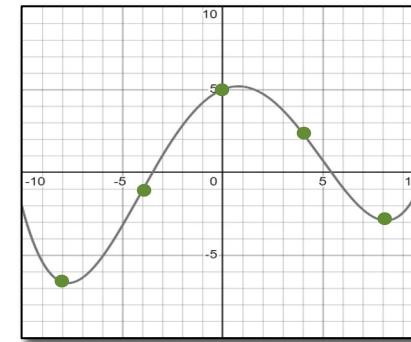
Auswertung des Polynoms nach Horner-Methode
kostet jeweils $\Theta(n)$ Schritte für jeden Punkt x_j

Gesamtaufwand für alle $2n - 1$ Punkte wäre $\Theta(n^2)$!

**Lösung: Verwende spezielle Werte x_j ,
so dass schneller per Divide & Conquer berechenbar**

Diskrete Fourier-Transformation berechnen (II)

$$\frac{1}{400}x^4 - \frac{1}{240}x^3 - \frac{5}{16}x^2 - \frac{1}{2}x + 5$$



$$p(x) = p_0 + p_1 x + \cdots + p_{n-1} x^{n-1}$$

Schreibe $p(x)$ in folgender Form (n gerade):

$$p(x) = p_{even}(x^2) + x \cdot p_{odd}(x^2)$$

wobei

$$p_{even}(x) = p_0 + p_2 x + \cdots + p_{n-2} x^{(n-2)/2}$$

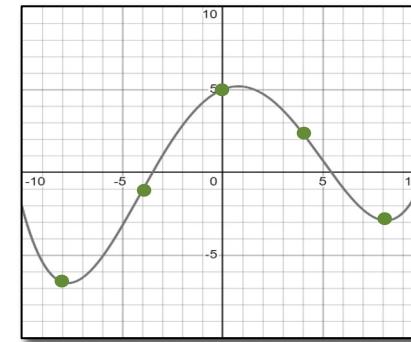
$$p_{odd}(x) = p_1 + p_3 x + \cdots + p_{n-1} x^{(n-2)/2}$$

noch nicht (ganz)
Problem halber Größe,
um Divide & Conquer
anzuwenden

p_{even}, p_{odd} sind Polynome von ca. halbem Grad
– aber leider immer noch $2n - 1$ Punkte x_j nötig

Diskrete Fourier-Transformation berechnen (III)

$$\frac{1}{400}x^4 - \frac{1}{240}x^3 - \frac{5}{16}x^2 - \frac{1}{2}x + 5$$



$$p(x) = p_0 + p_1 x + \cdots + p_{n-1} x^{n-1}$$

Schreibe $p(x)$ in folgender Form (n gerade):

$$p(x) = p_{even}(x^2) + x \cdot p_{odd}(x^2)$$

aber immer noch

$$x_j \neq x_{j+n}$$

Verwende Werte x_j , so dass $x_j^2 = x_{j+n}^2$ für alle $j = 0, 1, \dots, n-1$

Dann müssen p_{even}, p_{odd} vom Grad $\frac{n-2}{2} = \frac{n}{2} - 1$ nur an n Stellen $x_0^2, x_1^2, \dots, x_{n-1}^2$ ausgewertet werden

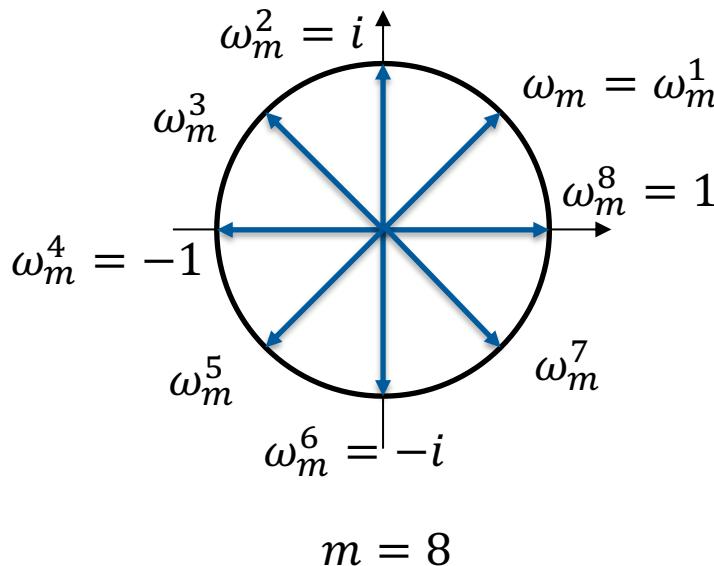
Problemgröße halbiert \Rightarrow Divide & Conquer anwendbar

m -te primitive Einheitswurzeln

Eine m -te primitive Einheitswurzel ω_m über einem Körper erfüllt $\omega_m^m = 1$ und $\omega_m^j \neq 1$ für $j = 1, 2, \dots, m - 1$.

Beispiel für komplexe Zahlen:

$$\omega_m = \exp\left(\frac{2\pi i}{m}\right) = \cos \frac{2\pi}{m} + i \cdot \sin \frac{2\pi}{m}$$



Für m -te primitive Einheitswurzel ω_m gilt für gerades m :

$$(\omega_m^j)^2 = (\omega_m^{j+m/2})^2 \text{ für alle } j = 0, 1, \dots, \frac{m}{2} - 1$$

denn:

$$(\omega_m^{j+m/2})^2 = \underbrace{\omega_m^{2j+m}}_{= 1} = \omega_m^{2j} \cdot \underbrace{\omega_m^m}_{= 1} = (\omega_m^j)^2$$

Ferner ist ω_m^2 eine $\frac{m}{2}$ -te primitive Einheitswurzel,
da $(\omega_m^2)^{m/2} = \omega_m^m = 1$
und $(\omega_m^2)^j = \omega_m^{2j} \neq 1$ für $j = 1, 2, \dots, \frac{m}{2} - 1$
(und somit $2j = 2, 4, \dots, m - 2$)

FFT – Algorithmen (I)

Wrapper, der für rekursive Aufrufe die aktuelle primitive Einheitswurzel hinzufügt:

```
FFTWrap(p,n) // n=2^k =#entries in p, k>=0  
1 return FFT(p,n,w); //w 2n-th primitive root of unity
```

Zur Vereinfachung bestimmen wir $2n$ (statt $2n - 1$) Punkt-Werte-Paare, so dass jeweils n Einträge im Array **p[]** und jeweils doppelt so viele Punkt-Werte-Paare berechnet werden

Wir benötigen, dass n Zweierpotenz ist; erreichen wir notfalls, indem wir n maximal verdoppeln und die zusätzlichen Einträge im Array **p[]** auf 0 setzen

(Übergang $n \rightarrow 2n$ wird später in der asymptotischen Laufzeit nicht ins Gewicht fallen)

FFT – Algorithmen (II)

```
FFT(p,n,w) // n=2^k =#entries in p, k>=0
1 pEven=ALLOC(n/2); pOdd=ALLOC(n/2);
2 pVal=ALLOC(2n); pEvenVal=ALLOC(n); pOddVal=ALLOC(n);
3 x=ALLOC(2n); x[0]=1; //input values x_j
4 FOR j=1 TO 2n-1 DO x[j]=w*x[j-1]; //x[j]= w^j

5 IF n==1 THEN //constant polynom
6     pVal[0].x=x[0]; pVal[0].y=p[0].y;
7     pVal[1].x=x[1]; pVal[1].y=p[0].y;
8 ELSE
9     FOR j=0 TO (n-2)/2 DO //create peven,podd
10        pEven[j]=p[2j]; pOdd[j]=p[2j+1];
11        pEvenVal=FFT(pEven,n/2,w*w); //evaluate at xj2
12        pOddVal =FFT(pOdd,n/2,w*w);
13        FOR j=0 TO 2n-1 DO //p(xj) = peven(xj2) + xj · podd(xj2)
14            pVal[j].x=x[j];
15            pVal[j].y=pEvenVal[j mod n].y + x[j]*pOddVal[j mod n].y;
16 return pVal;
```

FFT – Beispiel (I)

$$p(x) = 5 + x - x^2 + 2x^3$$

$p=[5, 1, -1, 2], \ n=4, \ w=\omega_8$

FFT([5,1,-1,2],4, ω_8)

$x[] = [1, \omega_8, \omega_8^2, \omega_8^3, \omega_8^4, \omega_8^5, \omega_8^6, \omega_8^7]$

pEvenVal[]

→ **FFT([5,-1],2, ω_8^2)**

$x[] = [1, \omega_8^2, \omega_8^4, \omega_8^6]$

pEvenVal[]

→ **FFT([5],1, ω_8^4)**

return [(1,5), (ω_8^4 ,5)]

Basisfall für
 $n = 1$
(konstantes
Polynom)

```
10 pEvenVal=FFT(pEven,n/2,w*w); //evaluate at  $x_j^2$ 
11 pOddVal =FFT(pOdd,n/2,w*w);
12 FOR j=0 TO 2n-1 DO // $p(x_j) = p_{even}(x_j^2) + x_j \cdot p_{odd}(x_j^2)$ 
13   pVal[j].x=x[j];
14   pVal[j].y=pEvenVal[j mod n].y + x[j]*pOddVal[j mod n].y;
```

$$p(x) = 5 + x - x^2 + 2x^3$$

$p=[5, 1, -1, 2], \ n=4, \ w=\omega_8$

FFT – Beispiel (II)

FFT([5,1,-1,2],4, ω_8)

$x[] = [1, \omega_8, \omega_8^2, \omega_8^3, \omega_8^4, \omega_8^5, \omega_8^6, \omega_8^7]$

pEvenVal[]

→ **FFT([5,-1],2, ω_8^2)**

$x[] = [1, \omega_8^2, \omega_8^4, \omega_8^6]$

pEvenVal[] = [(1,5), (ω_8^4 ,5)]

pOddVal[]

→ **FFT([-1],1, ω_8^4)**

return [(1,-1), (ω_8^4 ,-1)]

```

10 pEvenVal=FFT(pEven,n/2,w*w); //evaluate at  $x_j^2$ 
11 pOddVal =FFT(pOdd,n/2,w*w);
12 FOR j=0 TO 2n-1 DO // $p(x_j) = p_{even}(x_j^2) + x_j \cdot p_{odd}(x_j^2)$ 
13   pVal[j].x=x[j];
14   pVal[j].y=pEvenVal[j mod n].y + x[j]*pOddVal[j mod n].y;

```

FFT – Beispiel (III)

$$p(x) = 5 + x - x^2 + 2x^3$$

$p=[5, 1, -1, 2], \ n=4, \ w=\omega_8$

FFT([5,1,-1,2],4, ω_8)

$x[] = [1, \omega_8, \omega_8^2, \omega_8^3, \omega_8^4, \omega_8^5, \omega_8^6, \omega_8^7]$

pEvenVal[]

→ **FFT([5,-1],2, ω_8^2)**

$x[] = [1, \omega_8^2, \omega_8^4, \omega_8^6]$

pEvenVal[] = [(1,5), ($\omega_8^4, 5$)]

pOddVal[] = [(1,-1), ($\omega_8^4, -1$)]

**pVal[] = [(1,4), ($\omega_8^2, 5-\omega_8^2$),
($\omega_8^4, 5-\omega_8^4$), ($\omega_8^6, 5-\omega_8^6$)]**

$5 + 1 \cdot (-1)$

$5 + \omega_8^2 \cdot (-1)$

$5 + \omega_8^4 \cdot (-1)$

$5 + \omega_8^6 \cdot (-1)$

```

10 pEvenVal=FFT(pEven,n/2,w*w); //evaluate at  $x_j^2$ 
11 pOddVal =FFT(pOdd,n/2,w*w);
12 FOR j=0 TO 2n-1 DO // $p(x_j) = p_{even}(x_j^2) + x_j \cdot p_{odd}(x_j^2)$ 
13   pVal[j].x=x[j];
14   pVal[j].y=pEvenVal[j mod n].y + x[j]*pOddVal[j mod n].y;

```

FFT – Beispiel (IV)

$$p(x) = 5 + x - x^2 + 2x^3$$

$p=[5, 1, -1, 2], \ n=4, \ w=\omega_8$

FFT([5,1,-1,2],4, ω_8)

$$x[] = [1, \omega_8, \omega_8^2, \omega_8^3, \omega_8^4, \omega_8^5, \omega_8^6, \omega_8^7]$$

$$pEvenVal[] = [(1, 4), (\omega_8^2, 5 - \omega_8^2), (\omega_8^4, 5 - \omega_8^4), (\omega_8^6, 5 - \omega_8^6)]$$

$$pOddVal[] = [(1, 3), (\omega_8^2, 1+2\omega_8^2), (\omega_8^4, 1+2\omega_8^4), (\omega_8^6, 1+2\omega_8^6)]$$

$$pVal = [(1, 4+1\cdot 3), (\omega_8^1, (5-\omega_8^2)+\omega_8^1 \cdot (1+2\omega_8^2)), (\omega_8^2, (5-\omega_8^4)+\omega_8^2 \cdot (1+2\omega_8^4)), \dots]$$

$$7 = p(1)$$

$$5 + \omega_8^1 - \omega_8^2 + 2\omega_8^3 = p(\omega_8^1)$$

$$5 + \omega_8^2 - \omega_8^4 + 2\omega_8^6 = p(\omega_8^2)$$

return pVal[]

```

10  pEvenVal=FFT(pEven,n/2,w*w); //evaluate at  $x_j^2$ 
11  pOddVal =FFT(pOdd,n/2,w*w);
12  FOR j=0 TO 2n-1 DO  // $p(x_j) = p_{even}(x_j^2) + x_j \cdot p_{odd}(x_j^2)$ 
13      pVal[j].x=x[j];
14      pVal[j].y=pEvenVal[j mod n].y + x[j]*pOddVal[j mod n].y;

```

FFT – Laufzeit

Laufzeit $\Theta(n \log n)$

```
FFT(p,n,w) // n=2^k =#entries in p, k>=0
1 pEven=ALLOC(n/2); pOdd=ALLOC(n/2);
2 pVal=ALLOC(2n); pEvenVal=ALLOC(n); pOddVal=ALLOC(n);
3 x=ALLOC(2n); x[0]=1; //input values x_j
4 FOR j=1 TO 2n-1 DO x[j]=w*x[j-1]; //x[j]= w^j  $\Theta(n)$ 
5 IF n==1 THEN //constant polynom
6   pVal[0].x=x[0]; pVal[0].y=p[0].y; also  $T(n) = 2T(n/2) + \Theta(n)$ 
7   pVal[1].x=x[1]; pVal[1].y=p[0].y;
8 ELSE
9   FOR j=0 TO (n-2)/2 DO //create peven, podd
10    pEven[j]=p[2j]; pOdd[j]=p[2j+1];  $\Theta(n)$ 
11    pEvenVal=FFT(pEven,n/2,w*w); //evaluate at  $x_j^2$ 
12    pOddVal =FFT(pOdd,n/2,w*w);
13    FOR j=0 TO 2n-1 DO //p(xj) = peven(xj2) + xj · podd(xj2)
14      pVal[j].x=x[j];
15      pVal[j].y=pEvenVal[j mod n].y + x[j]*pOddVal[j mod n].y;
16 return pVal;
```

Die Diskrete Fourier-Transformation wird abstrakt definiert durch:



$$\begin{pmatrix} a_0 \\ a_1 \\ \vdots \\ a_{n-1} \end{pmatrix} \xrightarrow{\text{DFT}} \begin{pmatrix} \hat{a}_0 \\ \hat{a}_1 \\ \vdots \\ \hat{a}_{n-1} \end{pmatrix} \text{ mit } \hat{a}_j = \sum_{k=0}^{n-1} a_k \cdot \omega_n^{jk}$$

Interpretieren Sie diese Formel bezüglich unserer Anwendung zur Polynommultiplikation.

Inverse DFT berechnen (I)

Zur Erinnerung:

$$\begin{pmatrix} 1 & x_0 & \cdots & x_0^{m-1} \\ 1 & x_1 & \cdots & x_1^{m-1} \\ \vdots & \ddots & & \vdots \\ 1 & x_{m-1} & \cdots & x_{m-1}^{m-1} \end{pmatrix} \begin{pmatrix} r_0 \\ r_1 \\ \vdots \\ r_{m-1} \end{pmatrix} = \begin{pmatrix} y_0 \\ y_1 \\ \vdots \\ y_{m-1} \end{pmatrix}$$

$r = pq$ Polynom vom Grad $m \leq 2n - 1$

$x_j = \omega_m^j$ j -te Potenz der m -ten primitiven Einheitswurzel ω_m

Vandermonde-Matrix $V = V(x_0, \dots, x_{m-1})$

$$(V)_{jk} = \omega_m^{jk}$$

Bestimme Koeffizienten von r durch inverse Matrix V :

$$\begin{pmatrix} r_0 \\ r_1 \\ \vdots \\ r_{m-1} \end{pmatrix} = \underbrace{\begin{pmatrix} 1 & x_0 & \cdots & x_0^{m-1} \\ 1 & x_1 & \cdots & x_1^{m-1} \\ \vdots & \ddots & & \vdots \\ 1 & x_{m-1} & \cdots & x_{m-1}^{m-1} \end{pmatrix}}_{= V^{-1}}^{-1} \cdot \begin{pmatrix} y_0 \\ y_1 \\ \vdots \\ y_{m-1} \end{pmatrix}$$

Inverse DFT berechnen (II)

Es gilt: $V^{-1} = \left(\frac{\omega^{-jk}}{m} \right)_{jk}$, denn:

$$V^{-1} \cdot V = \frac{1}{m} \cdot \begin{pmatrix} 1 & \omega_m^{-j} & \omega_m^{-2j} & \dots & \omega_m^{-(m-1)j} \\ & \dots & & & \end{pmatrix} \cdot \begin{pmatrix} 1 \\ \omega_m^k \\ \omega_m^{2k} \\ \vdots \\ \omega_m^{(m-1)k} \end{pmatrix}$$

$$= \left(\frac{1}{m} \cdot \sum_{l=0}^{m-1} \omega_m^{l(-j+k)} \right)_{jk} = \begin{cases} \frac{1}{m} \cdot \sum_{l=0}^{m-1} \omega_m^0 = \frac{1}{m} \cdot \sum_{l=0}^{m-1} 1 = 1 & \text{für } j = k \\ \frac{1}{m} \cdot \frac{\omega_m^{m(-j+k)} - 1}{\omega_m^{(-j+k)} - 1} = \frac{1}{m} \cdot \frac{1 - 1}{\omega_m^{(-j+k)} - 1} = 0 & \text{für } j \neq k, \\ & 0 \leq j, k < m \end{cases}$$

verwendet $\sum_{l=0}^{m-1} q^l = \frac{q^m - 1}{q - 1}$ für $q \neq 1$, hier $q = \omega_m^{-j+k} \neq 1$ für $j \neq k$

Inverse DFT berechnen (III)

```
IFFTWrap(rVal,n) // n=2^k =#entries in rVal, k>=0  
  //w n-th primitive root of unity  
1  r[] = IFFT(rVal,n,w);  
2  FOR j=0 TO n-1 DO r[j]=r[j]/n;  
3  return r;
```

rechne Faktor
1/n heraus

Laufzeit $\Theta(n \log n)$

```
IFFT(rVal,n,w) // n=2^k =#entries in rVal, k>=0  
1  r=ALLOC(n); rEvenVal=ALLOC(n); rOddVal=ALLOC(n);  
2  x=ALLOC(n); x[0]=1;  
3  FOR j=1 TO n-1 DO x[j]=w*x[j-1]; //x[j]= w^j  
4  IF n==1 THEN  
5    r[0]=rVal[0].y;  
6  ELSE  
7    rEven=ALLOC(n/2); rOdd=ALLOC(n/2);  
8    FOR j=0 TO (n-2)/2 DO  
9      rEvenVal[j]=rVal[2j]; rOddVal[j]=rVal[2j+1];  
10   rEven=IFFT(rEvenVal,n/2,w*w);  
11   rOdd =IFFT(rOddVal,n/2,w*w);  
12   FOR j=0 TO n-1 DO  
13     r[j]=rEven[j mod n/2] + rOdd[j mod n/2]/x[j];  
14 return r;
```

Division, da wir inverses
 ω_n^{-ij} benötigen

Inverse FFT: Beispiel (I)

$$r(x) = 5 + x - x^2 + 2x^3$$

$$\omega_4 = i$$

IFFT([(1,7),(ω_4 ,6-i),(ω_4^2 ,1),(ω_4^3 ,6+i)],4, ω_4)

x []=[1, ω_4 , ω_4^2 , ω_4^3]

rEven[]

↳ **IFFT([(1,7),(ω_4^2 ,1)],2, ω_4^2)**

x []=[1, ω_4^2]

rEven[]

↳ **IFFT([(1,7)],1,1)**

return [7]

x	$r(x)$
1	7
$\omega_4 = i$	$6 - i$
$\omega_4^2 = -1$	1
$\omega_4^3 = -i$	$6 + i$

```

10  rEven=IFFT(rEvenVal,n/2,w*w);
11  rOdd =IFFT(rOddVal,n/2,w*w);
12  FOR j=0 TO n-1 DO
13    r[j]=rEven[j mod n/2] + rOdd[j mod n/2]/x[j];

```

Inverse FFT: Beispiel (II)

$$r(x) = 5 + x - x^2 + 2x^3$$

$$\omega_4 = i$$

IFFT([(1,7),(ω_4 ,6-i),(ω_4^2 ,1),(ω_4^3 ,6+i)],4, ω_4)

x []=[1, ω_4 , ω_4^2 , ω_4^3]

rEven[]

→ **IFFT([(1,7),(ω_4^2 ,1)],2, ω_4^2)**

x []=[1, ω_4^2]

rEven[]]=[7]

rOdd[]

→ **IFFT([ω_4^2 ,1]),1,1)**

return [1]

x	$r(x)$
1	7
$\omega_4 = i$	$6 - i$
$\omega_4^2 = -1$	1
$\omega_4^3 = -i$	$6 + i$

```

10  rEven=IFFT(rEvenVal,n/2,w*w);
11  rOdd =IFFT(rOddVal,n/2,w*w);
12  FOR j=0 TO n-1 DO
13    r[j]=rEven[j mod n/2] + rOdd[j mod n/2]/x[j];

```

Inverse FFT: Beispiel (III)

$$r(x) = 5 + x - x^2 + 2x^3$$

$$\omega_4 = i$$

IFFT([(1,7), $(\omega_4,6-i)$, $(\omega_4^2,1)$, $(\omega_4^3,6+i)$],4, ω_4)

x []=[1, ω_4 , ω_4^2 , ω_4^3]

rEven[]

→ **IFFT([(1,7), $(\omega_4^2,1)$],2, ω_4^2)**

x []=[1, ω_4^2]

rEven []=[7]

rOdd []=[1]

r []=[7+1/1,7+1/-1]=[8,6]

return r []

x	r(x)
1	7
$\omega_4 = i$	$6 - i$
$\omega_4^2 = -1$	1
$\omega_4^3 = -i$	$6 + i$

```
10  rEven=IFFT(rEvenVal,n/2,w*w);  
11  rOdd =IFFT(rOddVal,n/2,w*w);  
12  FOR j=0 TO n-1 DO  
13    r[j]=rEven[j mod n/2] + rOdd[j mod n/2]/x[j];
```

Inverse FFT: Beispiel (IV)

$$r(x) = 5 + x - x^2 + 2x^3$$

$$\omega_4 = i$$

IFFT([(1,7),($\omega_4, 6-i$),($\omega_4^2, 1$),($\omega_4^3, 6+i$)],4, ω_4)

x []=[1, ω_4 , ω_4^2 , ω_4^3]

rEven []=[8,6]

rOdd []

 → **IFFT([($\omega_4, 6-i$),($\omega_4^3, 6+i$)],2, ω_4^2)**

 x []=[1, ω_4^2]

 rEven []

 → **IFFT([($\omega_4, 6-i$)],1,1)**

return [6-i]

x	$r(x)$
1	7
$\omega_4 = i$	$6 - i$
$\omega_4^2 = -1$	1
$\omega_4^3 = -i$	$6 + i$

```

10   rEven=IFFT(rEvenVal,n/2,w*w);
11   rOdd =IFFT(rOddVal,n/2,w*w);
12   FOR j=0 TO n-1 DO
13     r[j]=rEven[j mod n/2] + rOdd[j mod n/2]/x[j];

```

Inverse FFT: Beispiel (V)

$$r(x) = 5 + x - x^2 + 2x^3$$

$$\omega_4 = i$$

IFFT([(1,7),($\omega_4, 6-i$),($\omega_4^2, 1$),($\omega_4^3, 6+i$)],4, ω_4)

x []=[1, ω_4 , ω_4^2 , ω_4^3]

rEven []=[8,6]

rOdd []

 → **IFFT([($\omega_4, 6-i$),($\omega_4^3, 6+i$)],2, ω_4^2)**

 x []=[1, ω_4^2]

 rEven []=[6-i]

 rOdd []

 → **IFFT([($\omega_4^3, 6+i$)],1,1)**

return [6+i]

x	$r(x)$
1	7
$\omega_4 = i$	$6 - i$
$\omega_4^2 = -1$	1
$\omega_4^3 = -i$	$6 + i$

```

10   rEven=IFFT(rEvenVal,n/2,w*w);
11   rOdd =IFFT(rOddVal,n/2,w*w);
12   FOR j=0 TO n-1 DO
13     r[j]=rEven[j mod n/2] + rOdd[j mod n/2]/x[j];

```

Inverse FFT: Beispiel (VI)

$$r(x) = 5 + x - x^2 + 2x^3$$

$$\omega_4 = i$$

```
IFFT([(1,7),(\omega_4,6-i),(\omega_4^2,1),(\omega_4^3,6+i)],4,\omega_4)
```

```
x []=[1,\omega_4,\omega_4^2,\omega_4^3]
```

```
rEven []=[8,6]
```

```
rOdd []
```

```
→ IFFT([(ω_4,6-i),(ω_4^3,6+i)],2,ω_4^2)
```

```
x []=[1,\omega_4^2]
```

```
rEven []=[6-i]
```

```
rOdd []=[6+i]
```

```
r []=[12,-2i]
```

```
return r []
```

x	$r(x)$
1	7
$\omega_4 = i$	$6 - i$
$\omega_4^2 = -1$	1
$\omega_4^3 = -i$	$6 + i$

```
10    rEven=IFFT(rEvenVal,n/2,w*w);  
11    rOdd =IFFT(rOddVal,n/2,w*w);  
12    FOR j=0 TO n-1 DO  
13        r[j]=rEven[j mod n/2] + rOdd[j mod n/2]/x[j];
```

Inverse FFT: Beispiel (VII)

$$r(x) = 5 + x - x^2 + 2x^3$$

$$\omega_4 = i$$

```
IFFT([(1,7),(\omega_4,6-i),(\omega_4^2,1),(\omega_4^3,6+i)],4,\omega_4)
```

```
x []=[1,\omega_4,\omega_4^2,\omega_4^3]
```

```
rEven []=[8,6]
```

```
rOdd []=[12,-2i]
```

```
r[0]=8+12/1=20
```

```
r[1]=6-2i/i=4
```

```
r[2]=8+12/-1=-4
```

```
r[3]=6-2i/-i=8
```

```
return r []=[20,4,-4,8]
```

Alle Werte werden in
IFFTWrap noch durch
 $n = 4$ dividiert und ergeben
daher Koeffizienten von $r(x)$

x	$r(x)$
1	7
$\omega_4 = i$	$6 - i$
$\omega_4^2 = -1$	1
$\omega_4^3 = -i$	$6 + i$

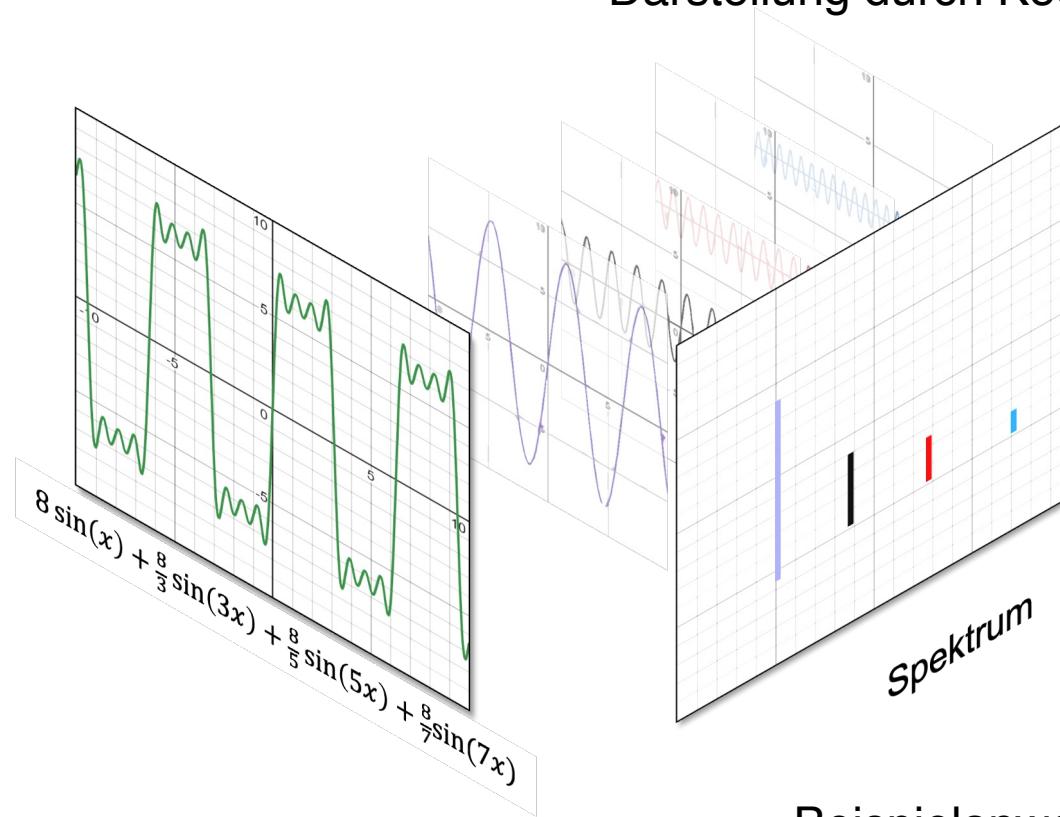
```
10    rEven=IFFT(rEvenVal,n/2,w*w);  
11    rOdd =IFFT(rOddVal,n/2,w*w);  
12    FOR j=0 TO n-1 DO  
13        r[j]=rEven[j mod n/2] + rOdd[j mod n/2]/x[j];
```

Fourier-Transformation, woanders

$f: \mathbb{R} \rightarrow \mathbb{R}$ mit Periode T

$$f(t) = \frac{a_0}{2} + \sum_{n=1}^{\infty} (a_n \cdot \cos\left(\frac{2n\pi t}{T}\right) + b_n \cdot \sin\left(\frac{2n\pi t}{T}\right))$$

Darstellung durch Koeffizienten a_0, a_1, b_1, \dots



Beispielanwendung: Frequenz-Filter

Backtracking

Backtracking

Prinzip Backtracking:

Finde Lösungen $x = (x_1, x_2, \dots, x_n)$ per „Trial-and-Error“,

indem Teillösung $(x_1, x_2, \dots, x_{i-1})$ durch Kandidaten x_i ergänzt wird, bis Gesamtlösung erhalten,

oder bis festgestellt, dass keine Gesamtlösung erreichbar, und Kandidat x_{i-1} revidiert wird

Beispiel: 4x4-Sudoku (I)

4			
			2
	4	3	1
3			

in jeder Zeile die Ziffern 1,2,3,4

in jeder Spalte die Ziffern 1,2,3,4

in jedem Quadranten die Ziffern 1,2,3,4

prüft, ob Board
komplett ausgefüllt

nächste freie Position
(zeilenweise)

prüft Kriterien oben

löscht Feld vor Rückkehr

```
SUDOKU-BACKTRACKING(B) // B[0...3][0...3] board
1 IF isFull(B) THEN
2   print „solution: “+B;
3 ELSE
4   (i,j)=nextFreePos(B);
5   FOR v=1 TO 4 DO
6     IF isAdmissible(B,i,j,v) THEN
7       B[i,j]=v;
8       SUDOKU-BACKTRACKING(B);
9       B[i,j]=empty;
```

Beispiel: 4x4-Sudoku (II)

4	1		
			2
	4	3	1
	3		

zwei Möglichkeiten: 1 und 2

```
SUDOKU-BACKTRACKING(B) // B[0...3][0...3] board
1 IF isFull(B) THEN
2   print „solution: “+B;
3 ELSE
4   (i,j)=nextFreePos(B);
5   FOR v=1 TO 4 DO
6     IF isAdmissible(B,i,j,v) THEN
7       B[i,j]=v;
8       SUDOKU-BACKTRACKING(B);
9       B[i,j]=empty;
```

Beispiel: 4x4-Sudoku (III)

4	1		
			2
	4	3	1
	3		

keine zulässige Option → Backtracking

```
SUDOKU-BACKTRACKING(B) // B[0...3][0...3] board
1 IF isFull(B) THEN
2   print „solution: “+B;
3 ELSE
4   (i,j)=nextFreePos(B);
5   FOR v=1 TO 4 DO
6     IF isAdmissible(B,i,j,v) THEN
7       B[i,j]=v;
8       SUDOKU-BACKTRACKING(B);
9       B[i,j]=empty;
```

Beispiel: 4x4-Sudoku (IV)

4	1		
			2
	4	3	1
	3		

→ Backtracking

```
SUDOKU-BACKTRACKING(B) // B[0...3][0...3] board
1 IF isFull(B) THEN
2   print „solution: “+B;
3 ELSE
4   (i,j)=nextFreePos(B);
5   FOR v=1 TO 4 DO
6     IF isAdmissible(B,i,j,v) THEN
7       B[i,j]=v;
8       SUDOKU-BACKTRACKING(B);
9       B[i,j]=empty;
```

Beispiel: 4x4-Sudoku (III)

4	2	1	3
			2
	4	3	1
	3		

einzig zulässige Option

```
SUDOKU-BACKTRACKING(B) // B[0...3][0...3] board
1 IF isFull(B) THEN
2   print „solution: “+B;
3 ELSE
4   (i,j)=nextFreePos(B);
5   FOR v=1 TO 4 DO
6     IF isAdmissible(B,i,j,v) THEN
7       B[i,j]=v;
8       SUDOKU-BACKTRACKING(B);
9       B[i,j]=empty;
```

Beispiel: 4x4-Sudoku (VI)

4	2	1	3
1			2
	4	3	1
	3		

zwei Möglichkeiten: 1 und 3

keine zulässige Option → Backtracking

```
SUDOKU-BACKTRACKING(B) // B[0...3][0...3] board
1 IF isFull(B) THEN
2   print „solution: “+B;
3 ELSE
4   (i,j)=nextFreePos(B);
5   FOR v=1 TO 4 DO
6     IF isAdmissible(B,i,j,v) THEN
7       B[i,j]=v;
8       SUDOKU-BACKTRACKING(B);
9       B[i,j]=empty;
```

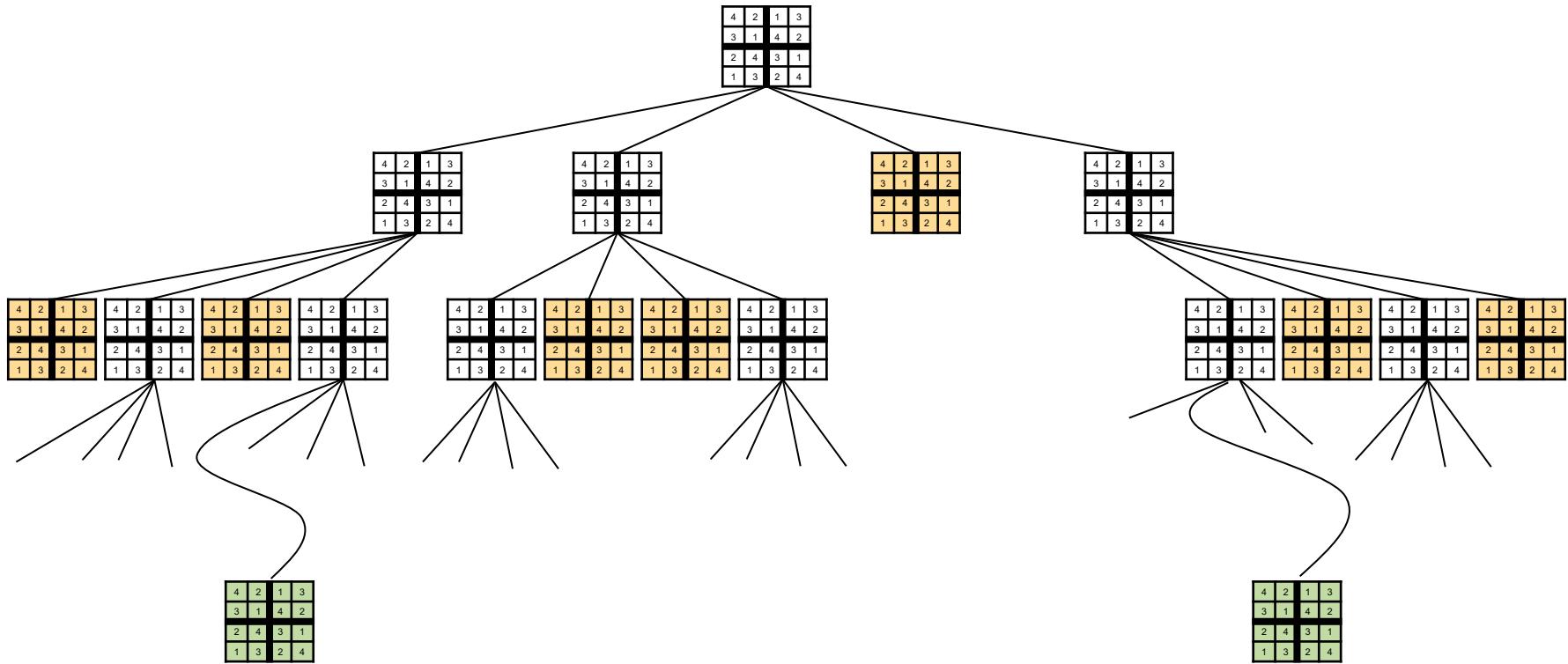
Beispiel: 4x4-Sudoku (VII)

4	(2)	1	3
(3)	1	4	2
2	4	3	1
1	3	2	4

einzig zulässige Option

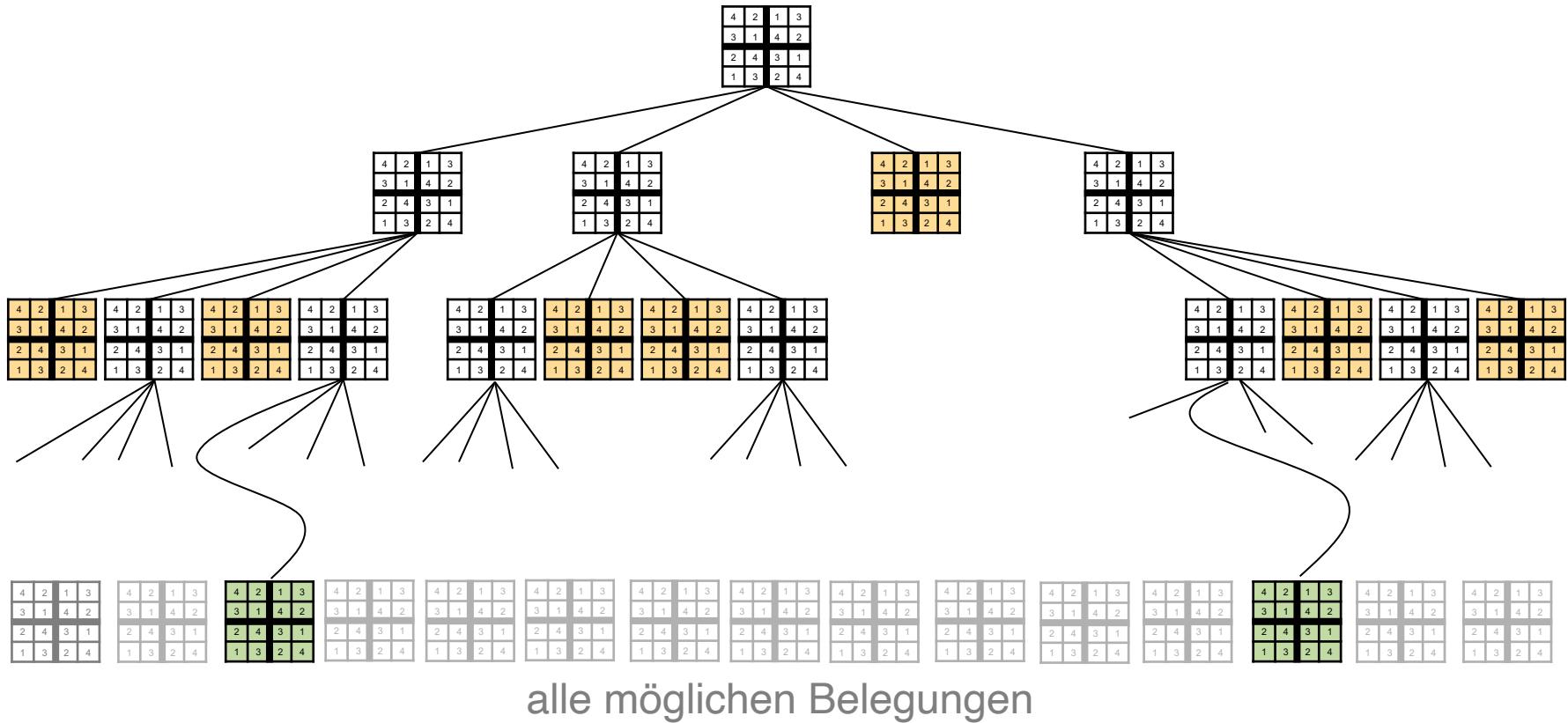
```
SUDOKU-BACKTRACKING(B) // B[0...3][0...3] board
1  IF isFull(B) THEN
2      print „solution: “+B;
3  ELSE
4      (i,j)=nextFreePos(B);
5      FOR v=1 TO 4 DO
6          IF isAdmissible(B,i,j,v) THEN
7              B[i,j]=v;
8              SUDOKU-BACKTRACKING(B);
9              B[i,j]=empty;
```

Backtracking vs. DFS



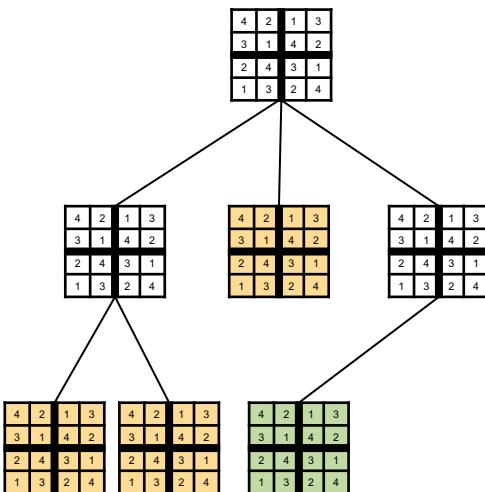
Backtracking kann man als Tiefensuche auf Rekursionsbaum betrachten,
wobei aussichtslose Lösungen evtl. frühzeitig abgeschnitten werden

Backtracking vs. Brute-Force Search

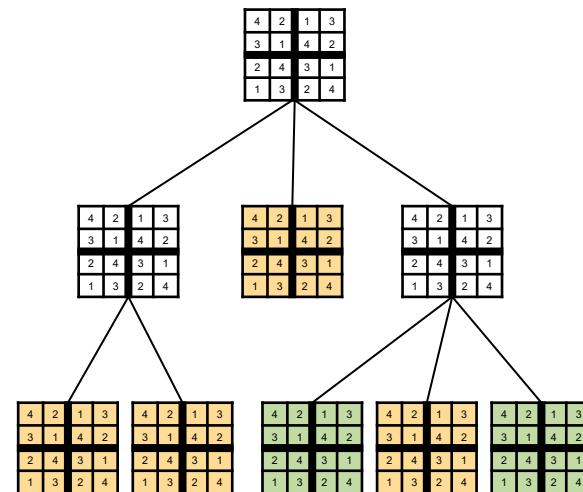


Backtracking kann man als „intelligentere“ erschöpfende Suche ansehen, die aussichtslose Lösungen vorher aussortiert

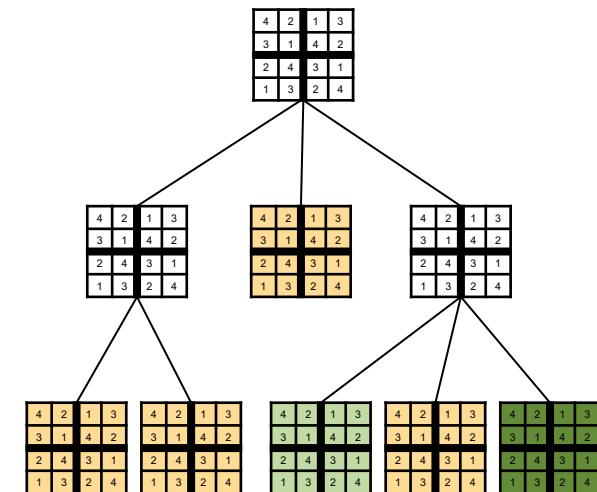
Backtracking: Lösungssuche



finde eine Lösung



finde alle Lösungen



finde beste Lösung

Beispiel: Regulärer Ausdruck (I)

Mustersuche in Strings

regulärer Ausdruck:

a (**b** | **c**) +d

...der mit **a** beginnt

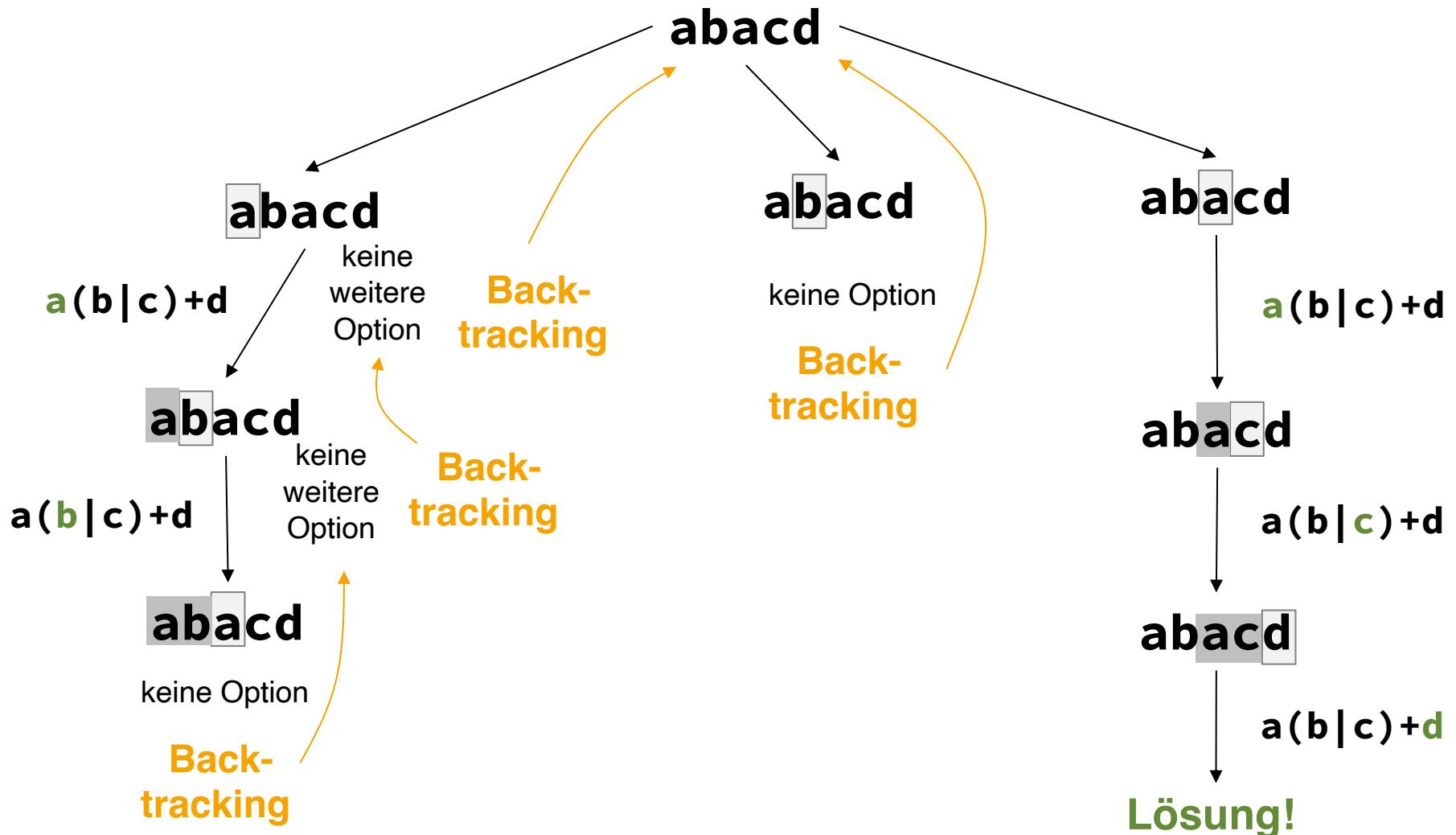
...dann ein oder mehrere
Zeichen aus der Menge
 $\{b, c\}$ enthält

...und noch auf **d** endet

durchsuche String **abacd** per Backtracking „gegen“ regulären Ausdruck

Beispiel: Regulärer Ausdruck (II)

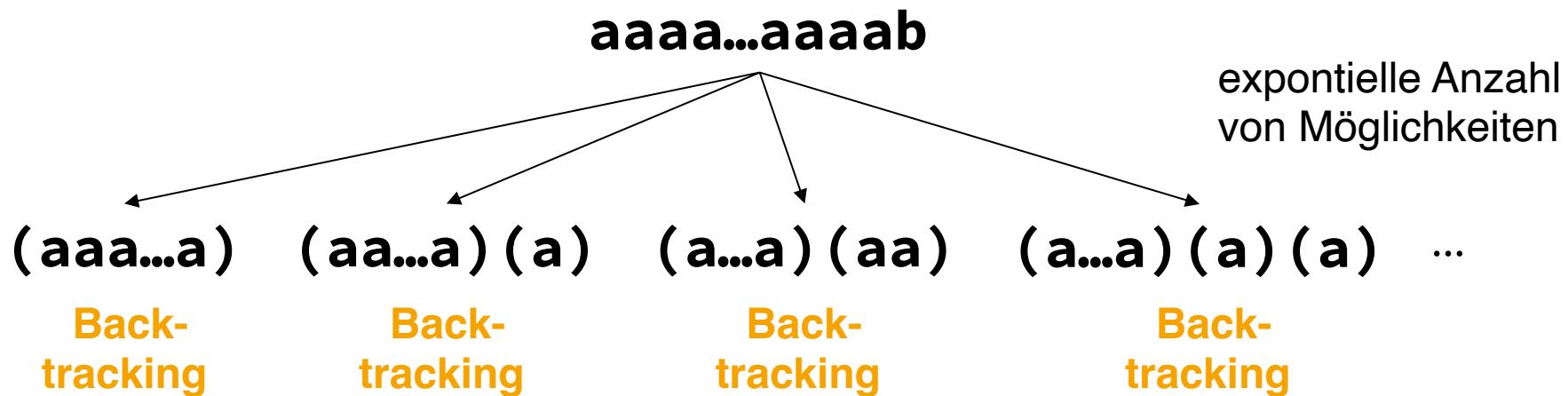
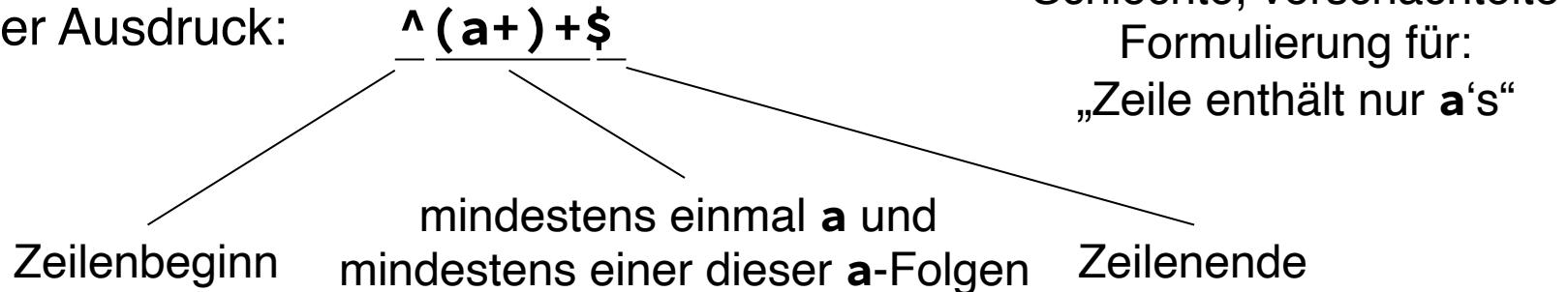
$a(b|c)^+d$



Evtl. exponentieller Aufwand

Aufwand Backtracking
kann exponentiell werden

regulärer Ausdruck:



Dynamische Programmierung

Dynamisches Programmieren

Prinzip Dynamisches Programmieren:

Teile Problem in (überlappende) Teilprobleme

Löse rekursiv Teilprobleme,
verwende dabei Zwischenergebnisse wieder
(„Memoization“)

Rekonstruiere Gesamtlösung

Achtung: es geht primär um das Auffinden geeigneter Rekursionen!

Beispiel: Fibonacci-Zahlen

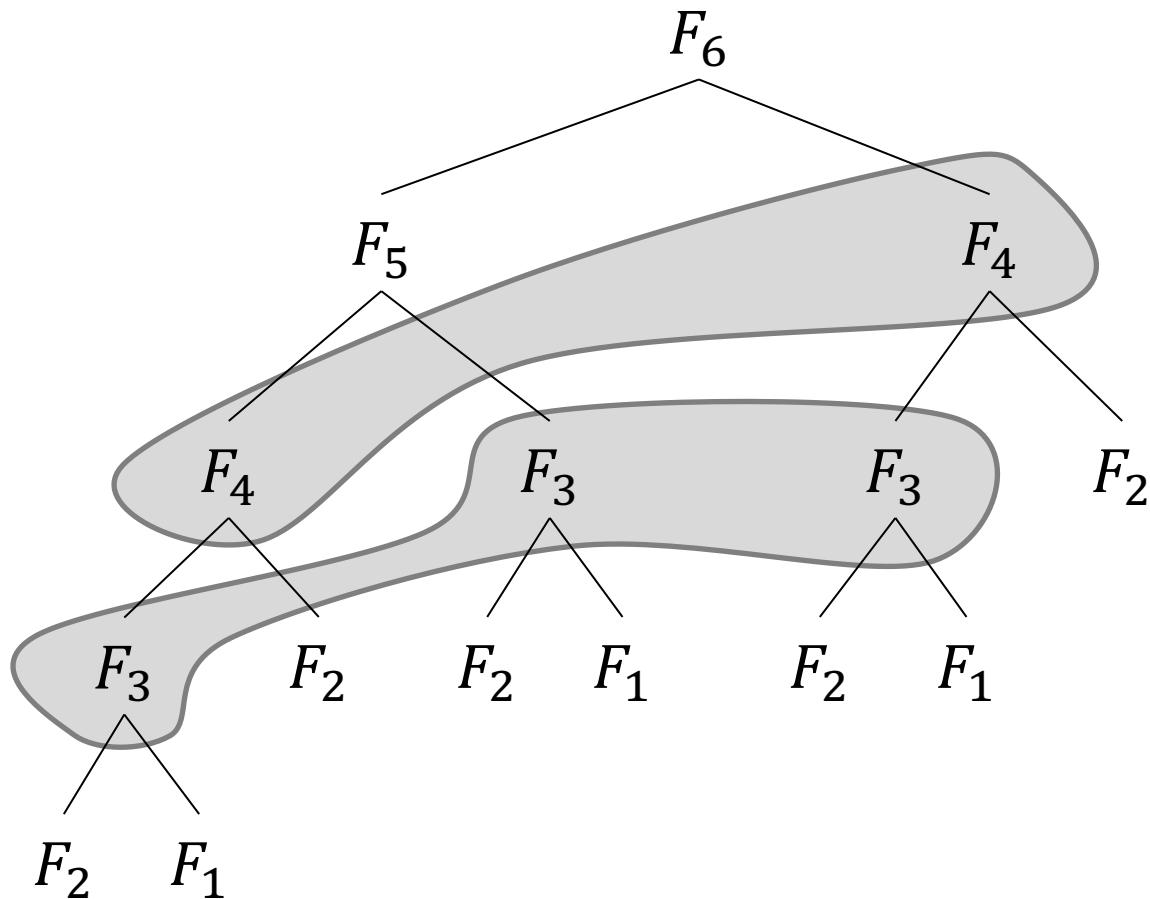
$$F_1 = 1, \quad F_2 = 1, \quad F_n = F_{n-1} + F_{n-2} \text{ für } n \geq 3$$

```
Fib-Rek(n) // n>=1
1 IF n<=2 THEN
2     return 1;
3 ELSE
4     return Fib-Rek(n-1)+Fib-Rek(n-2);
```

(vereinfachte) Laufzeitabschätzung mittels $T(n - 1) \approx T(n - 2)$:

$$\begin{aligned} T(n) &= T(n - 1) + T(n - 2) + c \\ &\approx 2 \cdot T(n - 1) + c \\ &= 4 \cdot T(n - 2) + 2 \cdot c + c \\ &= \dots \\ &= \Theta(2^n) \end{aligned}$$

Fibonacci-Berechnungsbaum

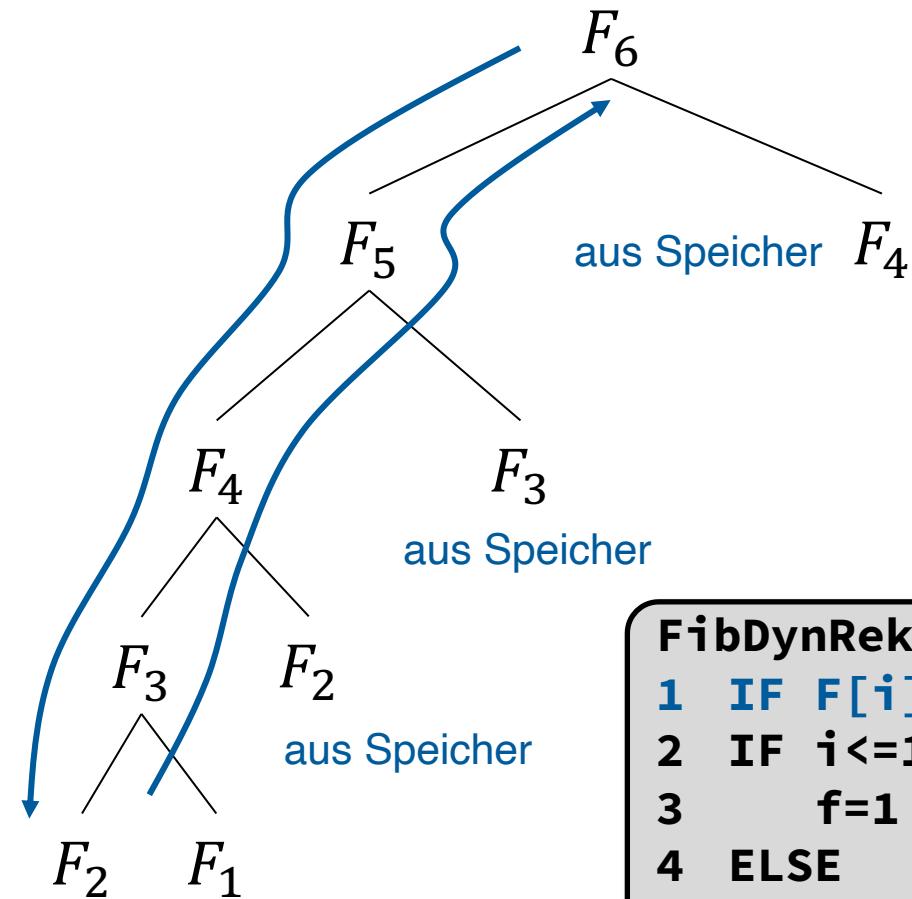


viele Werte
werden mehrfach
berechnet

Ansatz:
speichere stattdessen
Werte zwischen
(„Memoization“)

Fibonacci mit Memoization

Laufzeit $\Theta(n)$



Wenn Basisfall erreicht,
nur noch Addieren und Auslesen

```
FibDyn(n) // n>=1
1 F[] = ALLOC(n); // F[i] at F[i-1]
2 FOR i=0 TO n-1 DO F[i]=0;
3 return FibDynRek(n-1,F);
```

```
FibDynRek(i,F) // i>=0
1 IF F[i]!=0 THEN return F[i];
2 IF i<=1 THEN
3   f=1
4 ELSE
5   f=FibDynRek(i-1,F)+FibDynRek(i-2,F);
6 F[i]=f;
7 return f;
```

schon berechnet,
dann zurückgeben

Minimum Edit Distance (I) (Levenshtein-Distanz)

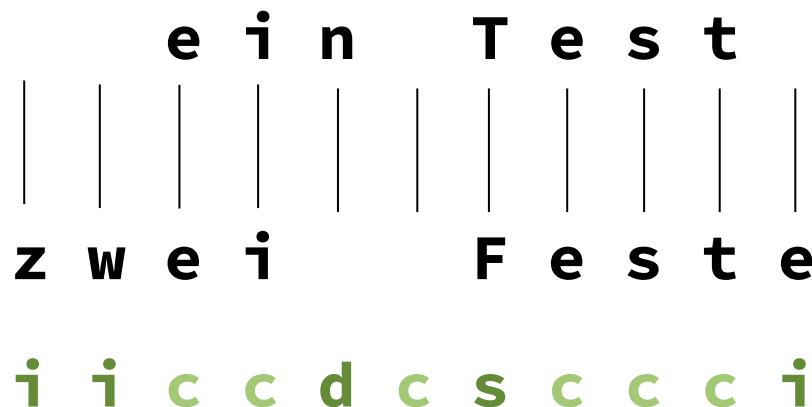
Misst Ähnlichkeit von Texten: Wie viele (Buchstaben-)Operationen

ins(S, i, b) – fügt an i -ter Position Buchstabe b in String S ein

del(S, i) – löscht an i -ter Position Buchstabe in String S

sub(S, i, b) – ersetzt an i -ter Position in String S den Buchstaben durch b

sind nötig, um Texte ineinander zu überführen?



Levenshtein-Distanz=5

(manchmal Kosten 2 für
Substitution, dann Distanz hier 6)

i i c c d c s c c c i

c für **copy(S, i)**-ohne Kosten

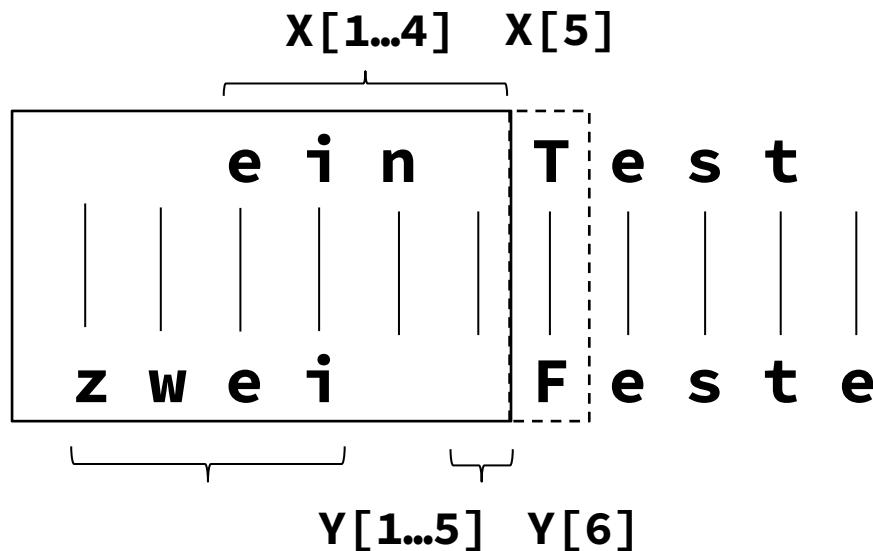
Minimum Edit Distance (II)

Algorithmische Sichtweise:

Überführe schrittweise String $X[1..m]$ von links nach rechts in String $Y[1..n]$

Jeweils Teil $X[1..i]$ bereits in Teil $Y[1..j]$ transformiert

beginnen hier jeweils bei 1



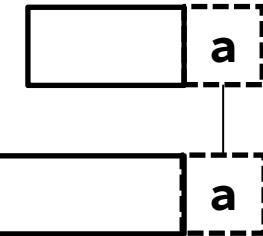
Achtung:
Optimalität dieser
Herangehensweise
wäre zu zeigen

Minimum Edit Distance (III)

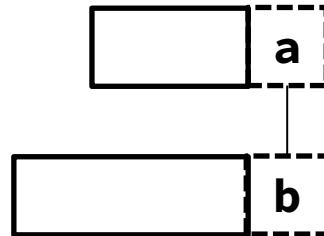
$D[i][j]$ sei Distanz, um $X[1..i]$ in $Y[1..j]$ zu überführen ($i, j \geq 1$)

Betrachte nächstens Schritt, um X in Y zu überführen:

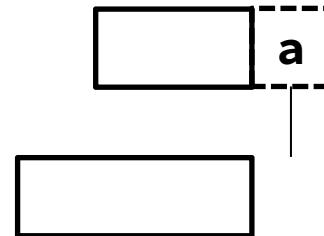
copy



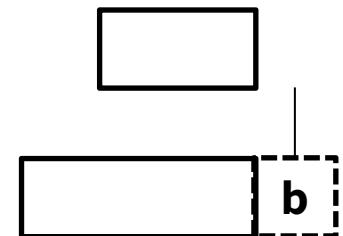
sub



del



ins



„vorher $X[1..i-1]$
in $Y[1..j-1]$
überführt, und jetzt
(kostenfrei) kopieren“

„vorher $X[1..i-1]$
in $Y[1..j-1]$
überführt, und jetzt
ersetzen“

„vorher $X[1..i-1]$
in $Y[1..j]$
überführt, und jetzt
 $X[i]$ löschen“

„vorher $X[1..i]$
in $Y[1..j-1]$
überführt, und jetzt
 $Y[j]$ einfügen“

$$D[i][j] =$$

$$D[i-1][j-1]$$

$$D[i][j] =$$

$$D[i-1][j-1] + 1$$

$$D[i][j] =$$

$$D[i-1][j] + 1$$

$$D[i][j] =$$

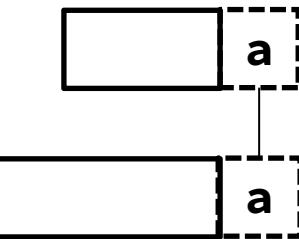
$$D[i][j-1] + 1$$

Minimum Edit Distance (IV)

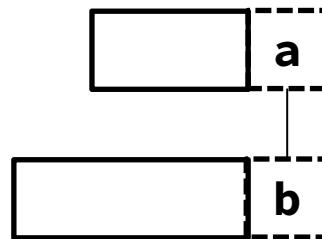
$D[i][j]$ sei Distanz, um $X[1..i]$ in $Y[1..j]$ zu überführen ($i, j \geq 1$)

Betrachte nächstens Schritt, um X in Y zu überführen:

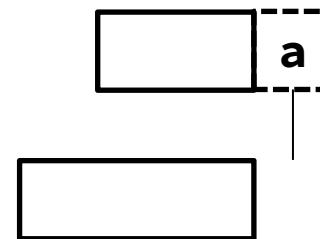
copy



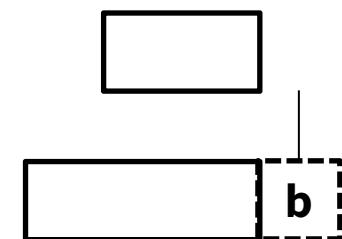
sub



del



ins



„vorher $X[1..i-1]$
in $Y[1..j-1]$
überführt, und jetzt
(kostenfrei) kopieren“

„vorher $X[1..i-1]$
in $Y[1..j-1]$
überführt, und jetzt
ersetzen“

„vorher $X[1..i-1]$
in $Y[1..j]$
überführt, und jetzt
 $X[i]$ löschen“

„vorher $X[1..i]$
in $Y[1..j-1]$
überführt, und jetzt
 $Y[j]$ einfügen“

$$D[i][j]$$

=

$$D[i-1][j-1] + (X[i] \neq Y[j])$$

1, wenn wahr, 0 sonst

$$D[i][j]$$

=

$$D[i-1][j] + 1$$

$$D[i][j]$$

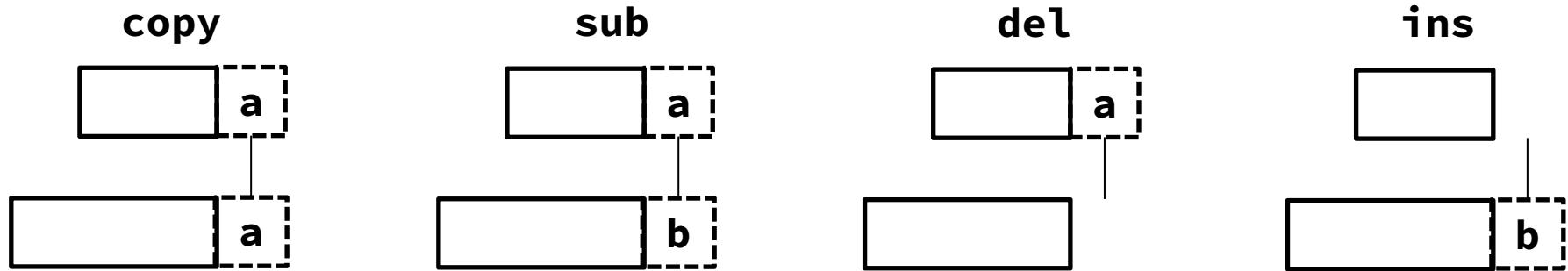
=

$$D[i][j-1] + 1$$

Minimum Edit Distance (IV)

$D[i][j]$ sei Distanz, um $X[1..i]$ in $Y[1..j]$ zu überführen ($i, j \geq 1$)

Betrachte nächstens Schritt, um X in Y zu überführen:



Da wir beste Ersetzungsstrategie suchen:

$$\begin{aligned} D[i][j] \\ = \\ \min \end{aligned}$$

$$\{ D[i-1][j-1] + (X[i] \neq Y[j]),$$

$$D[i-1][j] + 1,$$

$$D[i][j-1] + 1 \}$$

Minimum Edit Distance (V)

$D[i][j]$ sei Distanz, um $X[1..i]$ in $Y[1..j]$ zu überführen ($i, j \geq 1$)

„Randbedingungen“:

$D[0][j] = j$ – füge j Buchstaben $Y[1..j]$ zu leerem String $X[1..0]$ hinzu

$D[i][0] = i$ – lösche i Buchstaben $X[1..i]$, um leeren String $Y[1..0]$ zu erhalten

Da wir beste Ersetzungsstrategie suchen:

$$\begin{aligned} D[i][j] \\ = \\ \min \end{aligned}$$

$$\{ D[i-1][j-1] + (X[i] \neq Y[j]),$$

$$D[i-1][j] + 1,$$

$$D[i][j-1] + 1 \}$$

Minimum Edit Distance: Algorithmus

mittels dynamischer Programmierung und Memoization

```
MinEditDist(X,Y,m,n) // X=X[1..m], Y=Y[1..n]
1 D[][]=ALLOC(m,n);
2 FOR i=0 TO m DO D[i][0]=i;
3 FOR j=0 TO n DO D[0][j]=j;
4 FOR i=1 TO m DO
5   FOR j=1 TO n DO
6     IF X[i]=Y[j] THEN s=0 ELSE s=1;
7     D[i][j]=min{D[i-1][j-1]+s,D[i-1][j]+1,D[i][j-1]+1};
8 return D[m][n];
```

Laufzeit und
Speicher
 $\Theta(mn)$

Minimum Edit Distance: Beispiel (I)

X[1... 8] = ein Test

Y[1...10] = zwei Feste

Initialisierung (2 und 3)

```
MinEditDist(X,Y,m,n) // X=X[1...m]; Y=Y[1...n];
1 D[][]=ALLOC(m,n);
2 FOR i=0 TO m DO D[i][0]=i;
3 FOR j=0 TO n DO D[0][j]=j;
4 FOR i=1 TO m DO
5     FOR j=1 TO n DO
6         IF X[i]=Y[j] THEN s=0 ELSE s=1;
7         D[i][j]=min{D[i-1][j-1]+s,D[i-1][j]+1,D[i][j-1]+1};
8 return D[m][n];
```

D	0	1	2	3	4	5	6	7	8
0	0	1	2	3	4	5	6	7	8
1	1								
2	2								
3	3								
4	4								
5	5								
6	6								
7	7								
8	8								
9	9								
10	10								

Minimum Edit Distance: Beispiel (II)

X[1... 8] = ein Test

Y[1...10] = zwei Feste

i=1: FOR-Schleife für j (5-7)

j=1: X[1]=e ≠ z=Y[1] ⇒ s=1

D[1][1]=min{0+1,1+1,1+1}=1

```
MinEditDist(X,Y,m,n) // X=X[1...
1 D[][]=ALLOC(m,n);
2 FOR i=0 TO m DO D[i][0]=i;
3 FOR j=0 TO n DO D[0][j]=j;
4 FOR i=1 TO m DO
5   FOR j=1 TO n DO
6     IF X[i]=Y[j] THEN s=0 ELSE s=1;
7     D[i][j]=min{D[i-1][j-1]+s,D[i-1][j]+1,D[i][j-1]+1};
8 return D[m][n];
```

D	0	1	2	3	4	5	6	7	8
0	0	1	2	3	4	5	6	7	8
1	1	1							
2	2								
3	3								
4	4								
5	5								
6	6								
7	7								
8	8								
9	9								
10	10								

Minimum Edit Distance: Beispiel (III)

X[1... 8] = ein Test

Y[1...10] = zwei Feste

i=1: FOR-Schleife für j (5-7)

j=2: X[1]=e ≠ w=Y[2] ⇒ s=1

$$D[1][2] = \min\{1+1, 2+1, 1+1\} = 2$$

```
MinEditDist(X,Y,m,n) // X=X[1...
1 D[][]=ALLOC(m,n);
2 FOR i=0 TO m DO D[i][0]=i;
3 FOR j=0 TO n DO D[0][j]=j;
4 FOR i=1 TO m DO
5   FOR j=1 TO n DO
6     IF X[i]=Y[j] THEN s=0 ELSE s=1;
7     D[i][j]=min{D[i-1][j-1]+s,D[i-1][j]+1,D[i][j-1]+1};
8 return D[m][n];
```

D	0	1	2	3	4	5	6	7	8
0	0	1	2	3	4	5	6	7	8
1	1	1							
2	2	2							
3	3								
4	4								
5	5								
6	6								
7	7								
8	8								
9	9								
10	10								

Minimum Edit Distance: Beispiel (IV)

X[1... 8] = ein Test

Y[1...10] = zwei Feste

i=1: FOR-Schleife für j (5-7)

j=3: X[1]=e = e=Y[3] \Rightarrow s=0

$$D[1][3] = \min\{2+0, 3+1, 2+1\} = 2$$

```
MinEditDist(X,Y,m,n) // X=X[1...
1 D[][]=ALLOC(m,n);
2 FOR i=0 TO m DO D[i][0]=i;
3 FOR j=0 TO n DO D[0][j]=j;
4 FOR i=1 TO m DO
5   FOR j=1 TO n DO
6     IF X[i]=Y[j] THEN s=0 ELSE s=1;
7     D[i][j]=min{D[i-1][j-1]+s,D[i-1][j]+1,D[i][j-1]+1};
8 return D[m][n];
```

D	0	1	2	3	4	5	6	7	8
0	0	1	2	3	4	5	6	7	8
1	1	1							
2	2	2							
3	3	2							
4	4								
5	5								
6	6								
7	7								
8	8								
9	9								
10	10								

Minimum Edit Distance: Beispiel (V)

X[1... 8] = ein Test

Y[1...10] = zwei Feste

i=2: FOR-Schleife für j (5-7)

j=4: X[2]=i = i=Y[4] \Rightarrow s=0

$$D[2][4] = \min\{2+0, 3+1, 3+1\} = 2$$

```
MinEditDist(X,Y,m,n) // X=X[1...
1 D[][]=ALLOC(m,n);
2 FOR i=0 TO m DO D[i][0]=i;
3 FOR j=0 TO n DO D[0][j]=j;
4 FOR i=1 TO m DO
5   FOR j=1 TO n DO
6     IF X[i]=Y[j] THEN s=0 ELSE s=1;
7     D[i][j]=min{D[i-1][j-1]+s,D[i-1][j]+1,D[i][j-1]+1};
8 return D[m][n];
```

D	0	1	2	3	4	5	6	7	8
0	0	1	2	3	4	5	6	7	8
1	1	1	2						
2	2	2	2						
3	3	2	3						
4	4	3	2						
5	5	4							
6	6	5							
7	7	6							
8	8	7							
9	9	8							
10	10	9							

Minimum Edit Distance: Beispiel (VI)

X[1... 8] = ein Test

Y[1...10] = zwei Feste

```
MinEditDist(X,Y,m,n) // X=X[1...
1 D[][]=ALLOC(m,n);
2 FOR i=0 TO m DO D[i][0]=i;
3 FOR j=0 TO n DO D[0][j]=j;
4 FOR i=1 TO m DO
5     FOR j=1 TO n DO
6         IF X[i]=Y[j] THEN s=0 ELSE s=1;
7         D[i][j]=min{D[i-1][j-1]+s,D[i-1][j]+1,D[i][j-1]+1};
8 return D[m][n];
```

D	0	1	2	3	4	5	6	7	8	X[1...i]
0	0	1	2	3	4	5	6	7	8	
1	1	1	2	3	4	5	6	7	8	
2	2	2	2	3	4	5	6	7	8	
3	3	2	3	3	4	5	5	6	7	
4	4	3	2	3	4	5	6	6	7	
5	5	4	3	3	3	4	5	6	7	
6	6	5	4	4	4	4	5	6	7	
7	7	6	5	5	5	5	4	5	6	
8	8	7	6	6	6	6	5	4	5	
9	9	8	7	7	7	7	6	5	4	
10	10	9	8	8	8	8	7	6	5	5

Minimum Edit Distance: Beispiel (VII)

Rückwärtsschrittfolge
 $D[m][n]$ zu $D[0][0]$ entlang
der ausgewählten Minima
(bzw. bis Rand erreicht)
gibt Operationen an:

- \swarrow : **copy** (± 0)
- \searrow : **sub** (+1)
- \rightarrow : **del**
- \downarrow : **ins**

D	0	1	2	3	4	5	6	7	8
0	0	1	2	3	4	5	6	7	8
1	1	1	2	3	4	5	6	7	8
2	2	2	2	3	4	5	6	7	8
3	3	2	3	3	4	5	5	6	7
4	4	3	2	3	4	5	6	6	7
5	5	4	3	3	3	4	5	6	7
6	6	5	4	4	4	4	5	6	7
7	7	6	5	5	5	5	4	5	6
8	8	7	6	6	6	6	5	4	5
9	9	8	7	7	7	7	6	5	4
10	10	9	8	8	8	8	7	6	5

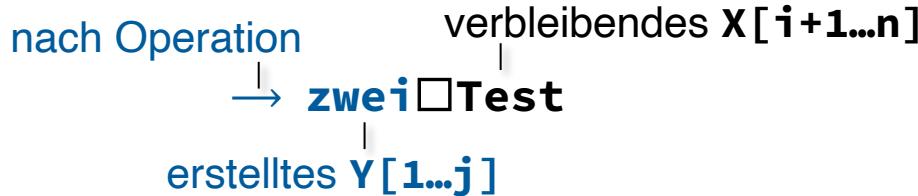
(Im Allgemeinen gibt es
mehrere mögliche Sequenzen!)

Minimum Edit Distance: Beispiel (VIII)

(\downarrow) c , $\downarrow s$, $\rightarrow d$,

	ein□Test
↓	zein□Test
↓	zwein□Test
(↓)	zwein□Test
(↓)	zwein□Test
→	zwei□Test
(↓)	zwei□Test
↓	zwei□Fest
(↓)	zwei□Fest
(↓)	zwei□Fest
(↓)	zwei□Fest
↓	zwei□Feste

D	0	1	2	3	4	5	6	7	8
0	0	1	2	3	4	5	6	7	8
1	1	1	2	3	4	5	6	7	8
2	2	2	2	3	4	5	6	7	8
3	3	2	3	3	4	5	5	6	7
4	4	3	2	3	4	5	6	6	7
5	5	4	3	3	3	4	5	6	7
6	6	5	4	4	4	4	5	6	7
7	7	6	5	5	5	5	4	5	6
8	8	7	6	6	6	6	5	4	5
9	9	8	7	7	7	7	6	5	4
10	10	9	8	8	8	8	7	6	5





Geben Sie zwei Strings an, die durch verschiedene Sequenzen von Operationen mit gleicher Levenshtein-Distanz ineinander übergeführt werden können

Greedy-Algorithmen

Greedy-Algorithmen

Prinzip Greedy:

Finde Lösung $x = (x_1, x_2, \dots, x_n)$

indem Teillösung $(x_1, x_2, \dots, x_{i-1})$ durch Kandidaten x_i
ergänzt wird,

der lokal am günstigsten erscheint.

„Unsere“ Greedy-Algorithmen...

Dijkstra-SSSP(G, s, w)

```
1  initSSSP( $G, s, w$ );
2   $Q = V$ ; // let  $S = V - Q$ 
3  WHILE !isEmpty( $Q$ ) DO
4       $u = \text{EXTRACT-MIN}(Q)$ ; // wrt. dist
5      FOREACH  $v$  in  $\text{adj}(u)$  DO
6          relax( $G, u, v, w$ );
```

(auch Prim)

wähle jeweils Knoten,
der kürzeste Distanz hat

MST-Kruskal(G, w) // $G = (V, E)$ undirected, connected graph
 w weight function

```
1   $A = \emptyset$ 
2  FOREACH  $v$  in  $V$  DO  $\text{set}(v) = \{v\}$ ;
3  Sort edges according to weight in nondecreasing order
4  FOREACH  $\{u, v\}$  in  $E$  according to order DO
5      IF  $\text{set}(u) \neq \text{set}(v)$  THEN
6           $A = A \cup \{\{u, v\}\}$ 
7          UNION( $G, u, v$ );
8  return  $A$ 
```

Wähle jeweils
leichteste Kante

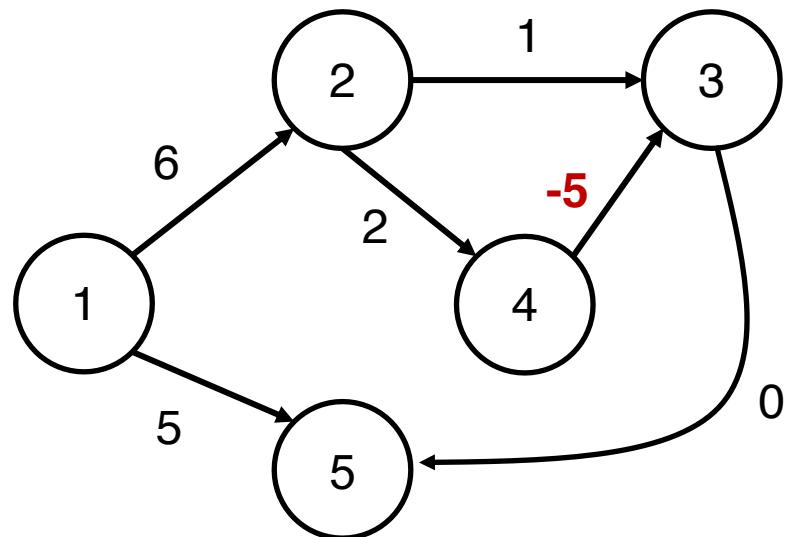
...funktionieren oft, aber nicht immer

Dijkstra-SSSP(G, s, w)

```
1  initSSSP( $G, s, w$ );
2   $Q = V$ ; // let  $S = V - Q$ 
3  WHILE !isEmpty( $Q$ ) DO
4       $u = \text{EXTRACT-MIN}(Q)$ ; // wrt. dist
5      FOREACH  $v$  in  $\text{adj}(u)$  DO
6          relax( $G, u, v, w$ );
```

kürzester Weg $1 \rightarrow 5$ via
 $1 \rightarrow 2 \rightarrow 4 \rightarrow 3 \rightarrow 5$
mit Gewicht $6+2-5+0=3$

Dijkstra-Algorithmus:
 $1 \rightarrow 5$ mit Gewicht 5



Traveling Salesperson Problem (TSP)

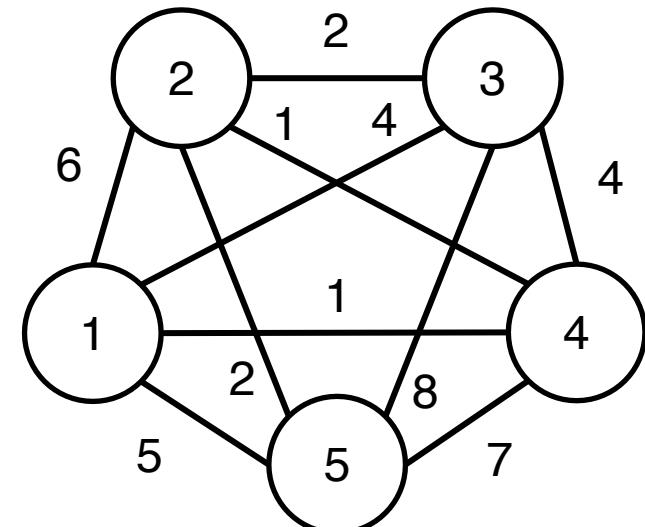
Traveling Salesperson Problem:

Gegeben vollständiger (un-)gerichteter Graph $G = (V, E)$ mit Kantengewichten $w: E \rightarrow \mathbb{R}$, finde Tour p mit minimalem Kantengewicht $w(p)$.

Eine Tour ist ein Weg $p = (v_0, v_1, \dots, v_{n-1}, v_n)$ entlang der Kanten $(v_i, v_{i+1}) \in E$ für $i = 0, 1, 2, \dots, n - 1$, der bis auf Start- und Endknoten $v_0 = v_n$ jeden Knoten genau einmal besucht.

Graph $G = (V, E)$ ist vollständig, wenn es für alle Knoten $u, v \in V$ mit $u \neq v$ eine Kante $(u, v) \in E$ gibt

Wenn Graph nicht vollständig, aber Tour hat, kann man „verboten teure“ Kanten (u, v) mit $w((u, v)) = |V| \cdot \max_{e \in E} \{w(e)\} + 1$ hinzufügen



Tour $1 \rightarrow 5 \rightarrow 2 \rightarrow 3 \rightarrow 4 \rightarrow 1$ mit Gewicht 14

TSP vs. Dijkstra

Allgemeiner TSP-Algorithmus:

finde optimale Route,
die durch jeden
Knoten geht und zum
Ausgangspunkt zurückkehrt

löst anderes Problem

Dijkstra-Algorithmus:

finde optimalen Pfad vom
Ausgangspunkt aus

(besucht evtl. nicht alle Knoten
und betrachtet auch nicht Rückkehr)

Ansatz Greedy-Algorithmus für TSP:

Starte mit beliebigem oder gegebenem Knoten.
Nimm vom gegenwärtigen Knoten aus die Kante
zu noch nicht besuchtem Knoten, die kleinstes Gewicht hat.
Wenn kein Knoten mehr übrig, gehe zu Startpunkt zurück.

Greedy-TSP

```
Greedy-TSP(G,s,w) // |V|=n, s starting node
```

```
1 FOREACH v in V DO v.color=white;
2 tour[] = ALLOC(n); tour[0] = s; tour[0].color=gray;
3 FOR i=1 TO n-1 DO
4   tour[i] = EXTRACT-MIN(adj(tour[i-1]));
    //get (white) neighbor with minimum edge weight
5   tour[i].color=gray;
6 return tour;
```

Ansatz Greedy-Algorithmus für TSP:

Starte mit beliebigem oder gegebenem Knoten.

Nimm vom gegenwärtigen Knoten aus die Kante zu noch nicht besuchtem Knoten, die kleinstes Gewicht hat.
Wenn kein Knoten mehr übrig, gehe zu Startpunkt zurück.

Greedy-TSP ist zu gierig (I)

Betrachte vollständigen Graphen mit Knoten $V = \{1, 2, 3, \dots, n\}$, $n \geq 3$, und folgenden Kantengewichten für beliebig große Konstante N :

$$w(\{i, j\}) = \begin{cases} 1 & 1 \leq i, j \leq n, j = i + 1 \\ N & i = 1, j = n \\ 2 & \text{sonst} \end{cases}$$

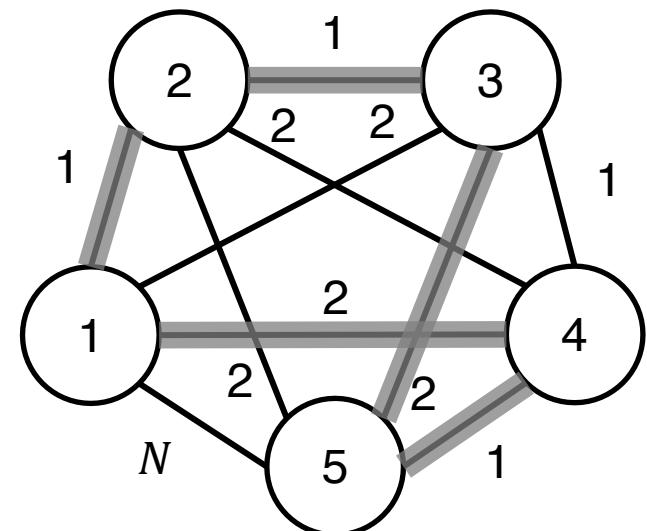
Startknoten $s = 1$

Optimale Tour mit Startknoten $s = 1$

$1 \rightarrow 2 \rightarrow \dots \rightarrow n - 2 \rightarrow n \rightarrow n - 1 \rightarrow 1$

hat Gewicht

$$1 + 1 + \dots + 1 + 2 + 1 + 2 = n + 2$$



Beispiel: $n = 5$

Greedy-TSP ist zu gierig (II)

Betrachte vollständigen Graphen mit Knoten $V = \{1, 2, 3, \dots, n\}$, $n \geq 3$, und folgenden Kantengewichten für beliebig große Konstante N :

$$w(\{i, j\}) = \begin{cases} 1 & 1 \leq i, j \leq n, j = i + 1 \\ N & i = 1, j = n \\ 2 & \text{sonst} \end{cases}$$

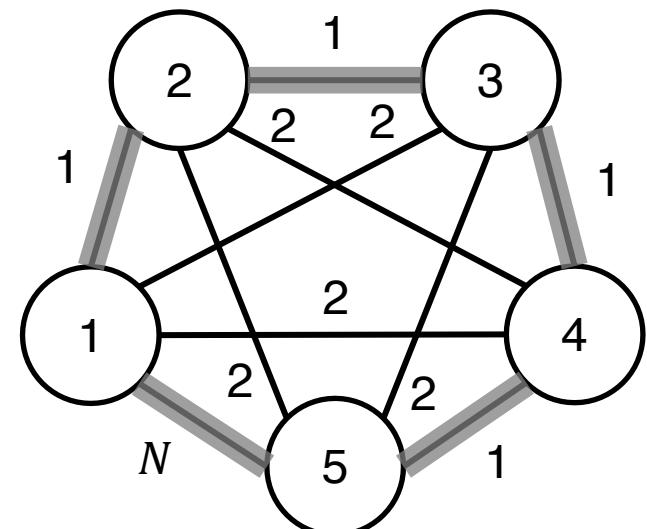
Startknoten $s = 1$

Greedy-Algorithmus läuft zunächst

$1 \rightarrow 2 \rightarrow \dots \rightarrow n - 1 \rightarrow n$,

muss dann die „teure“ Kante $n \rightarrow 1$ nehmen, erreicht also nur Gewicht

$$1 + 1 + \dots + 1 + N = N + n - 1$$



Beispiel: $n = 5$

Effizienter Algorithmus für TSP?

$TSP(G, w, s)$

???

Vermutlich schwierig, effizienten Algorithmus zu finden

→ siehe Kapitel 8 über NP

Metaheuristiken

Heuristiken und Metaheuristiken

Heuristik:

dedizierter Suchalgorithmus
für Optimierungsproblem,

der gute (aber evtl. nicht optimale)
Lösung für **spezielles** Problem findet

problem-abhängig,
arbeitet direkt „am“ Problem

Metaheuristik:

allgemeine Vorgehensweise,
um Suche für beliebige
Optimierungsprobleme
zu leiten

problem-unabhängig,
arbeitet mit abstrakten Problemen

Lokale Suche

Güte der Lösung



„Hill-Climbing-Strategie“

Hill-Climbing-Algorithmus

```
HillClimbing(P) // maxTime constant
```

```
1 sol=initialSol(P);
2 time=0;
3 WHILE time<maxTime DO
4     new=perturb(P,sol);
5     IF quality(P,new)>quality(P,sol) THEN
6         sol=new;
7         time=time+1;
8 return sol;
```

wähle
initiale Lösung

ändere Lösung
leicht ab

ersetze aktuelle
Lösung durch neu,
falls diese besser ist

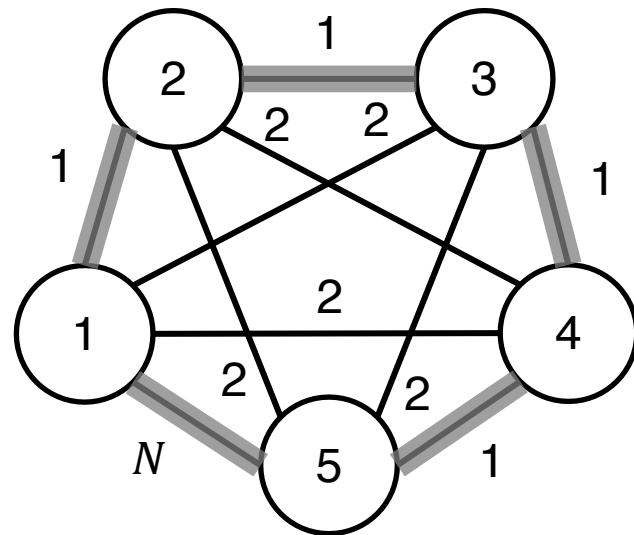
Beispiel TSP:

initSol: wähle beliebige Tour, z.B. per Greedy-Algorithmus

perturb: wähle zwei Knoten **u**, **v** zufällig und tausche sie in Tour

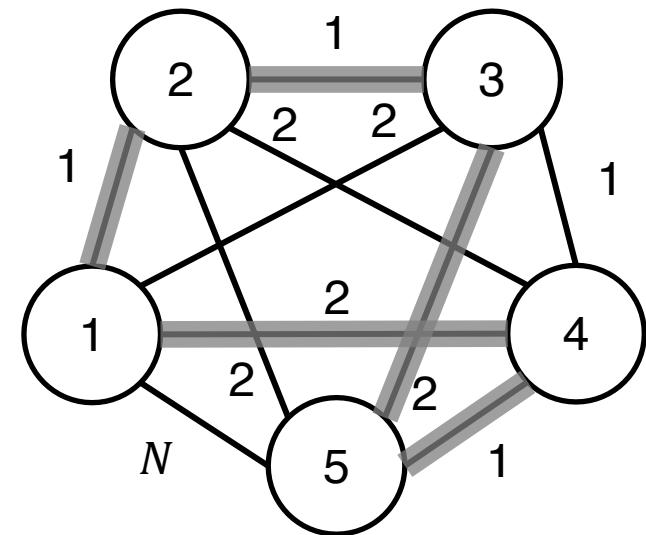
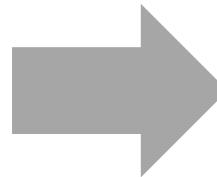
quality: (negatives) Gewicht der aktuellen Tour

Hill-Climbing-Algorithmus für TSP



Greedy-Lösung

wenn 4,5 gewählt



optimale Lösung

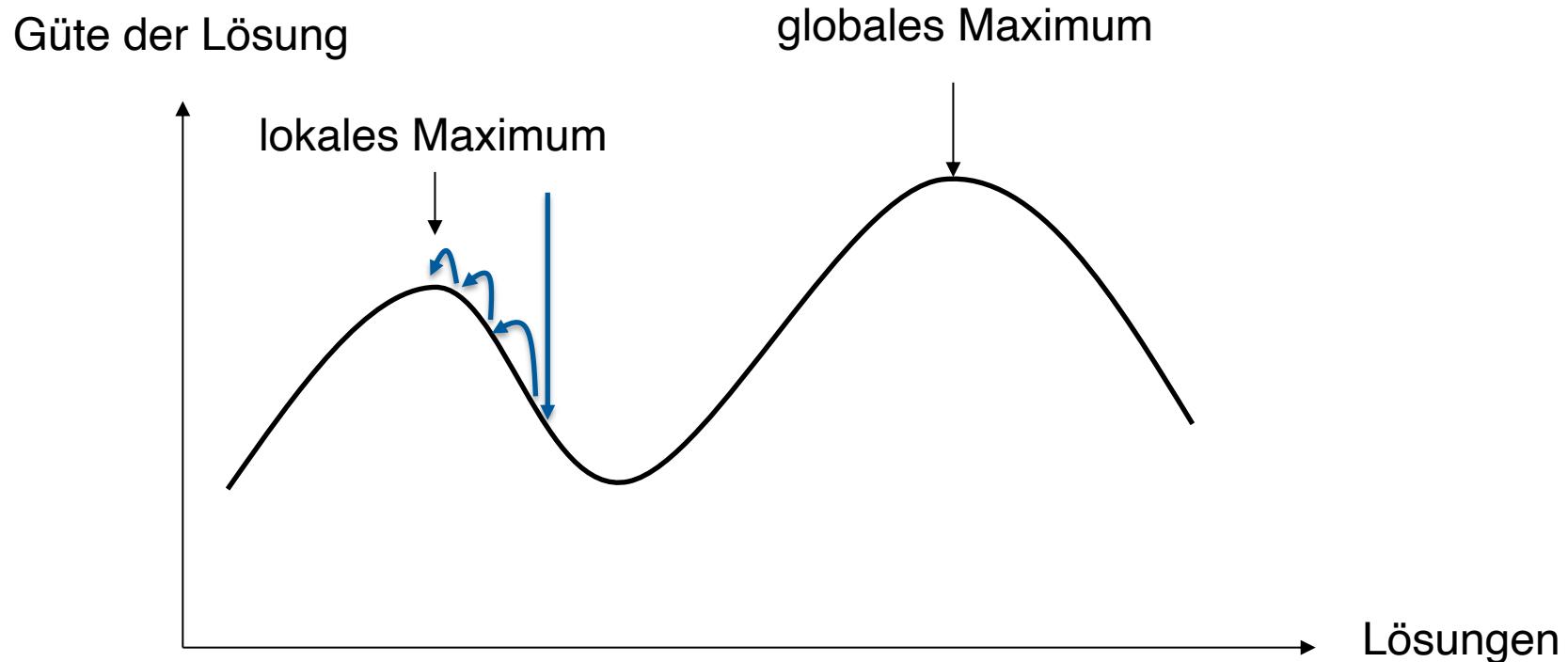
Beispiel TSP:

initSol: wähle beliebige Tour, z.B. per Greedy-Algorithmus

perturb: wähle zwei Knoten u, v zufällig und tausche sie in Tour

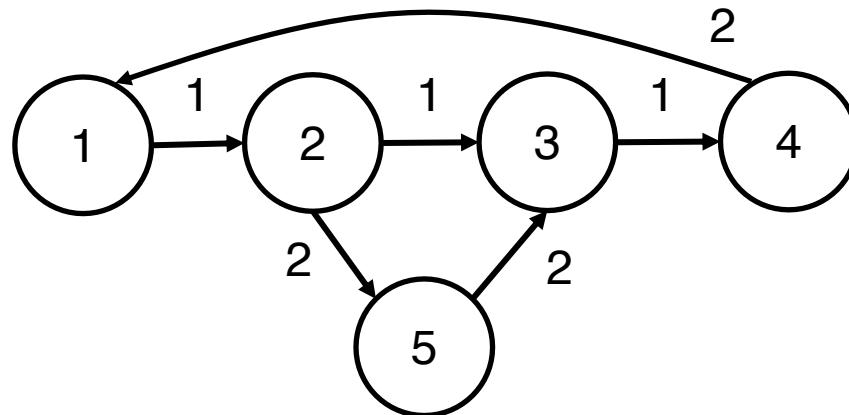
quality: (negatives) Gewicht der aktuellen Tour

Lokale vs. globale Maxima



Eventuell bleibt Hill-Climbing-Algorithmus in lokalem Maximum hängen,
da stets nur leichte Lösungsänderungen in aufsteigender Richtung!

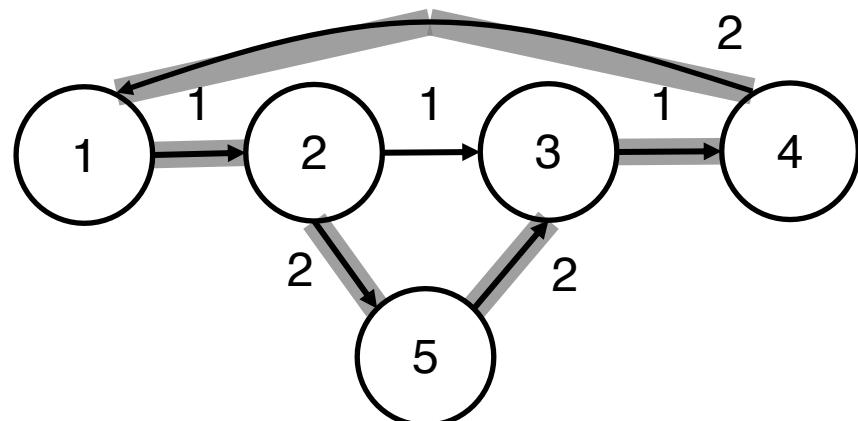
Lokale vs. globale Maxima beim TSP (I)



TSP auf gerichtetem Graphen
Alle anderen Kanten Gewicht N
Startknoten 1

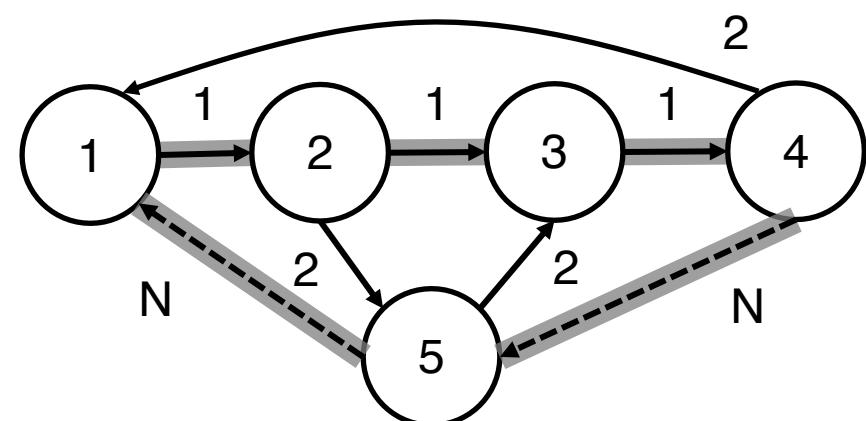
optimale Lösung

$1 \rightarrow 2 \rightarrow 5 \rightarrow 3 \rightarrow 4 \rightarrow 1$ mit Gewicht 8



Greedy-Lösung

$1 \rightarrow 2 \rightarrow 3 \rightarrow 4 \rightarrow 5 \rightarrow 1$ mit Gewicht $2N+3$



Lokale vs. globale Maxima beim TSP (II)

Umgebung der Greedy-Lösung durch Vertauschen zweier Knoten gegeben (Startknoten bleibt fix):

$1 \rightarrow 3 \rightarrow 2 \rightarrow 4 \rightarrow 5 \rightarrow 1$ mit Gewicht $5N$

$1 \rightarrow 4 \rightarrow 3 \rightarrow 2 \rightarrow 5 \rightarrow 1$ mit Gewicht $4N+2$

$1 \rightarrow 5 \rightarrow 3 \rightarrow 4 \rightarrow 2 \rightarrow 1$ mit Gewicht $3N+3$

$1 \rightarrow 2 \rightarrow 4 \rightarrow 3 \rightarrow 5 \rightarrow 1$ mit Gewicht $4N+1$

$1 \rightarrow 2 \rightarrow 5 \rightarrow 4 \rightarrow 3 \rightarrow 1$ mit Gewicht $3N+3$

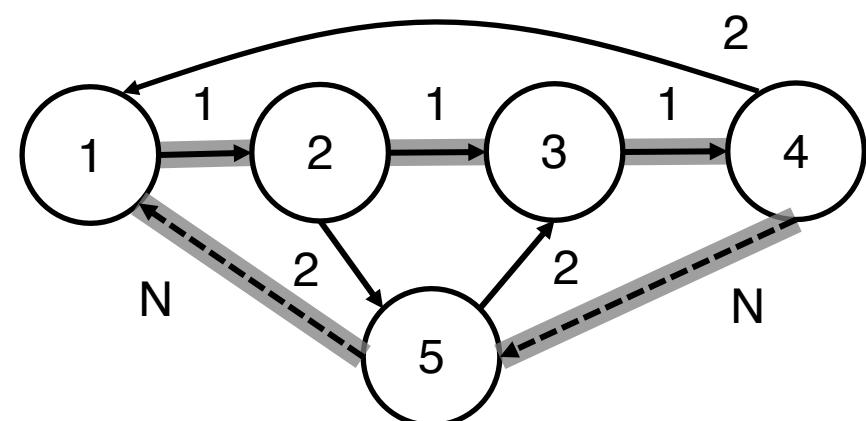
$1 \rightarrow 2 \rightarrow 3 \rightarrow 5 \rightarrow 4 \rightarrow 1$ mit Gewicht $2N+4$

Hill-Climbing findet

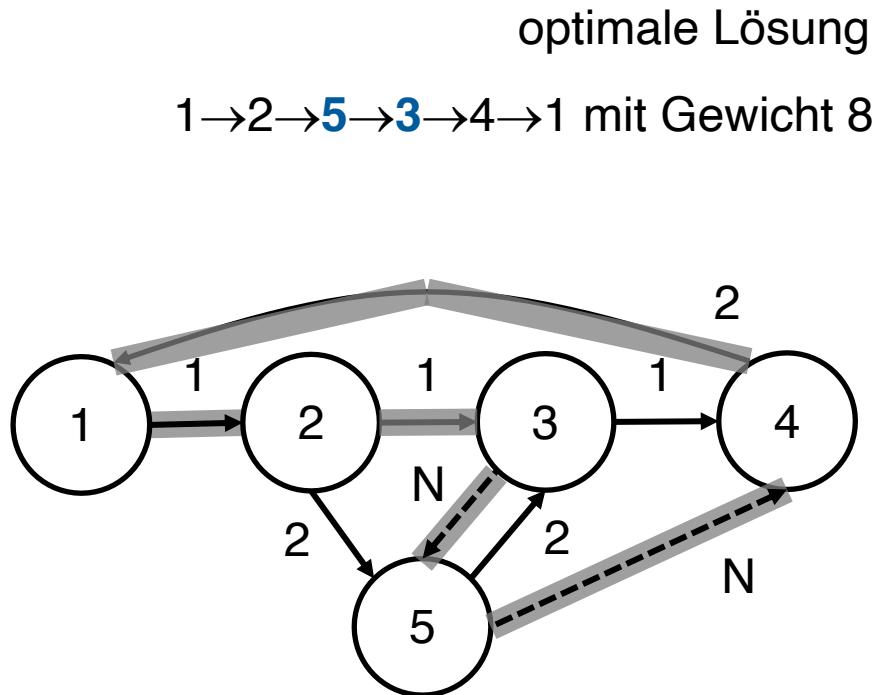
keine bessere Lösung in der Nähe

→ Greedy-Lösung lokales Max (bzw. Min)

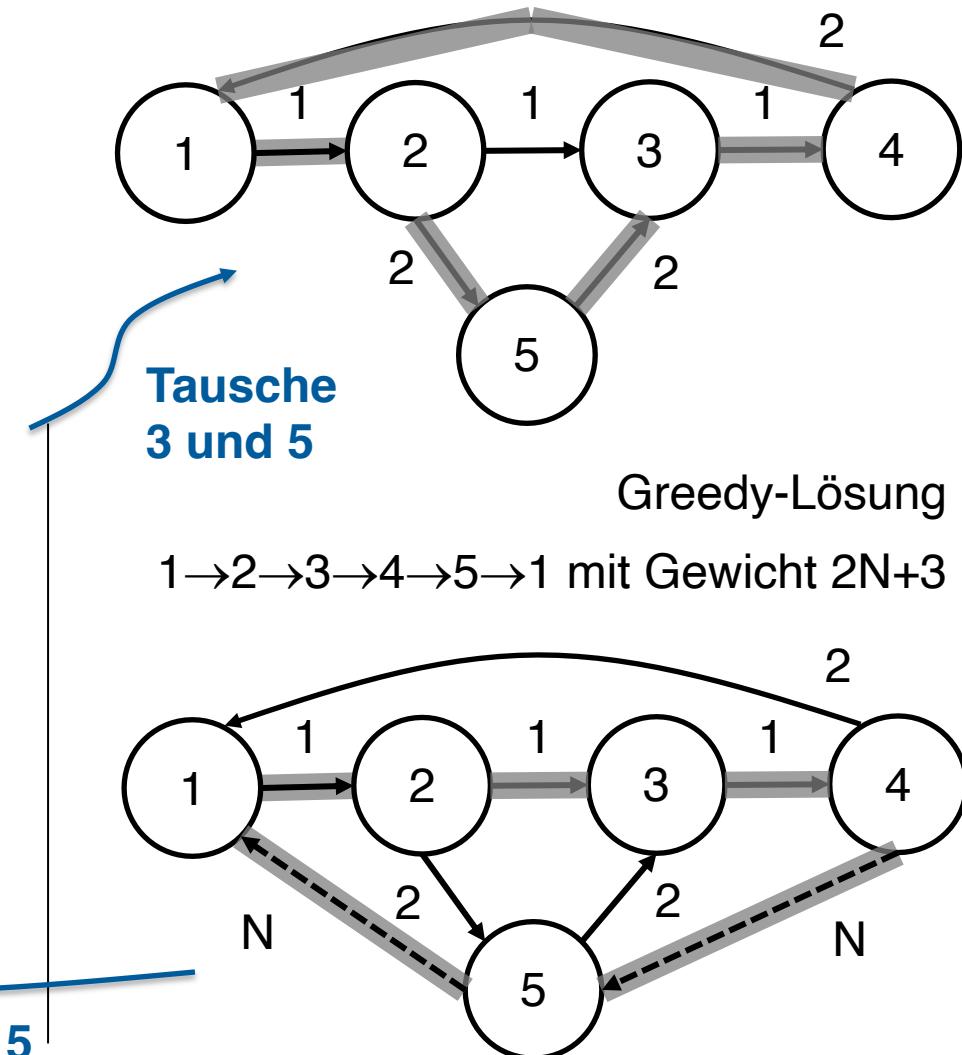
Greedy-Lösung
 $1 \rightarrow 2 \rightarrow 3 \rightarrow 4 \rightarrow 5 \rightarrow 1$ mit Gewicht $2N+3$



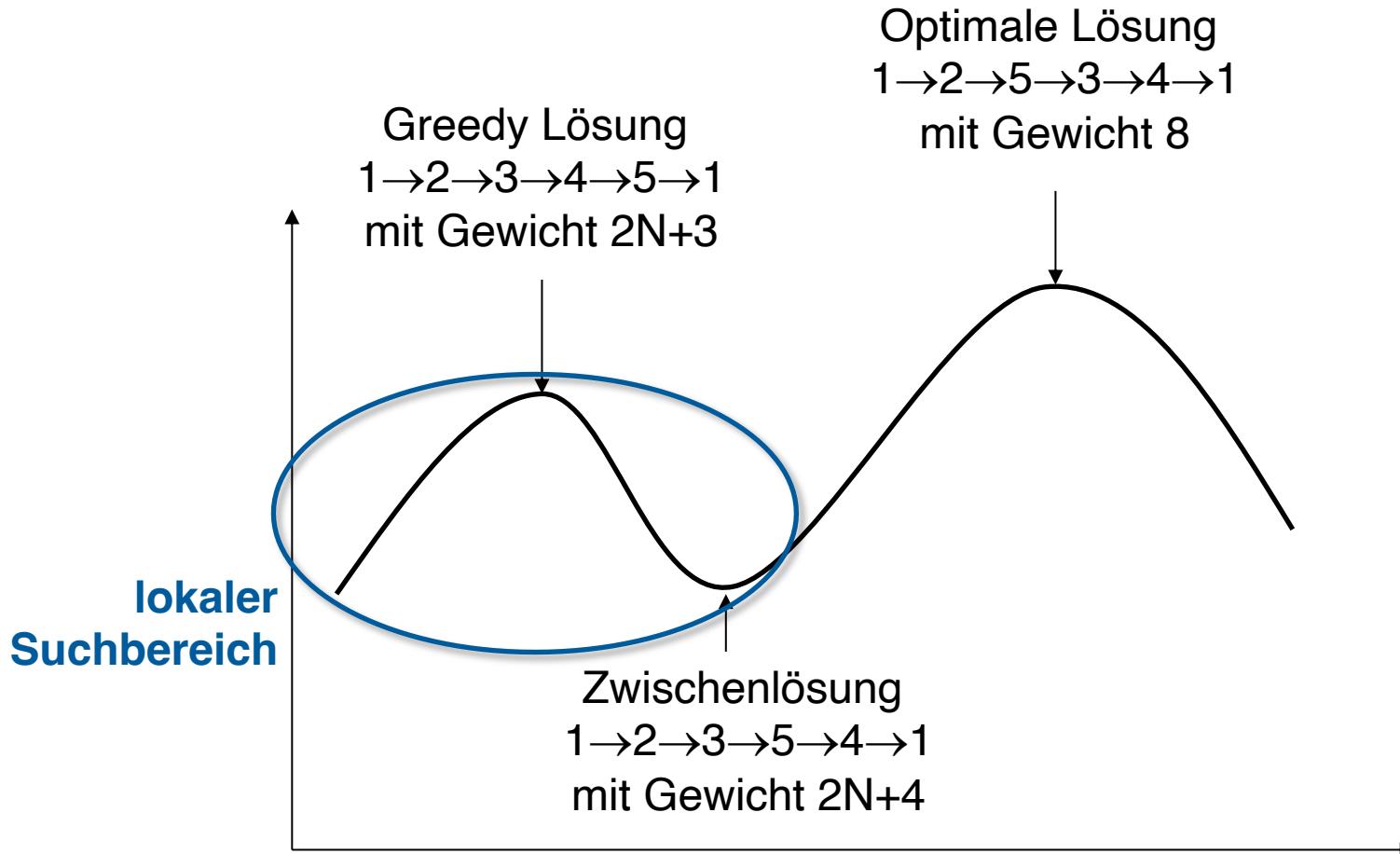
Lokale vs. globale Maxima beim TSP (II)



Tausche 4 und 5

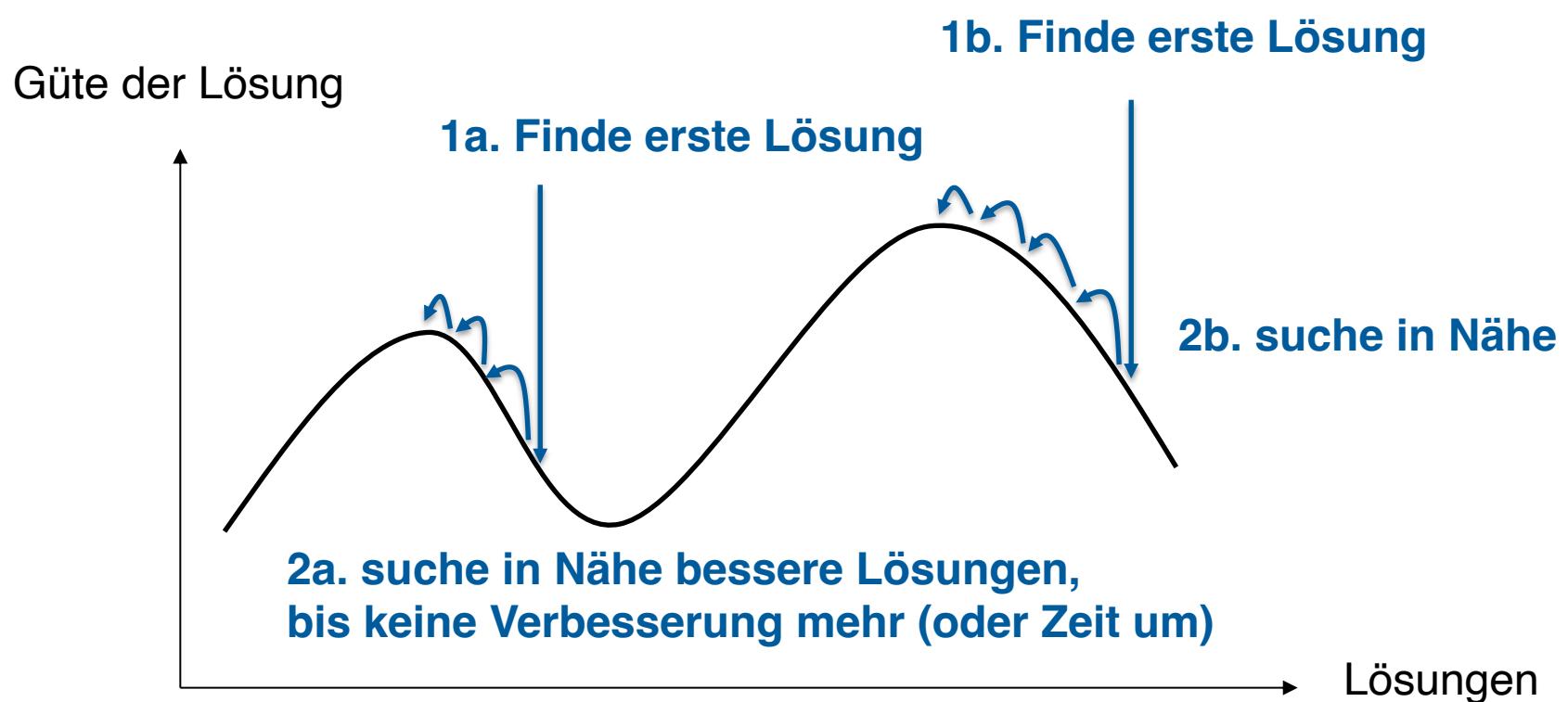


Lokale vs. globale Maxima beim TSP (IV)



Iterative Lokale Suche

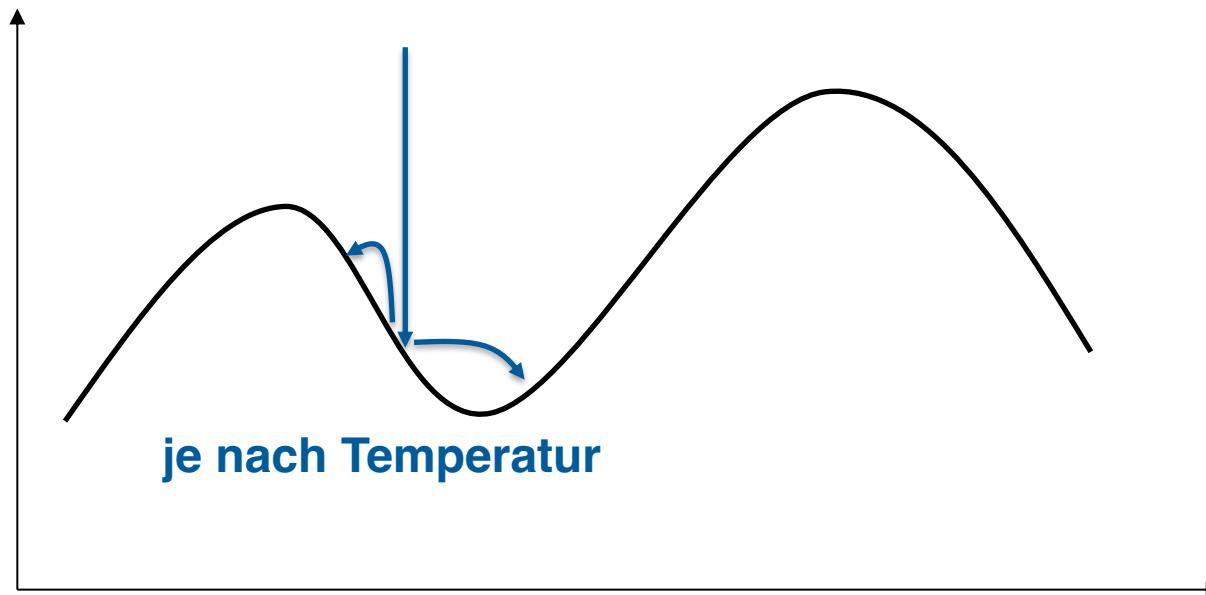
Problem: „zufällige“ (?) Lösungen könnten auch schlecht sein



Beginne Suche nochmal von vorne, z.B. mit neuer zufälliger Lösung,
akzeptiere beste gefundene Lösung

Simulated Annealing

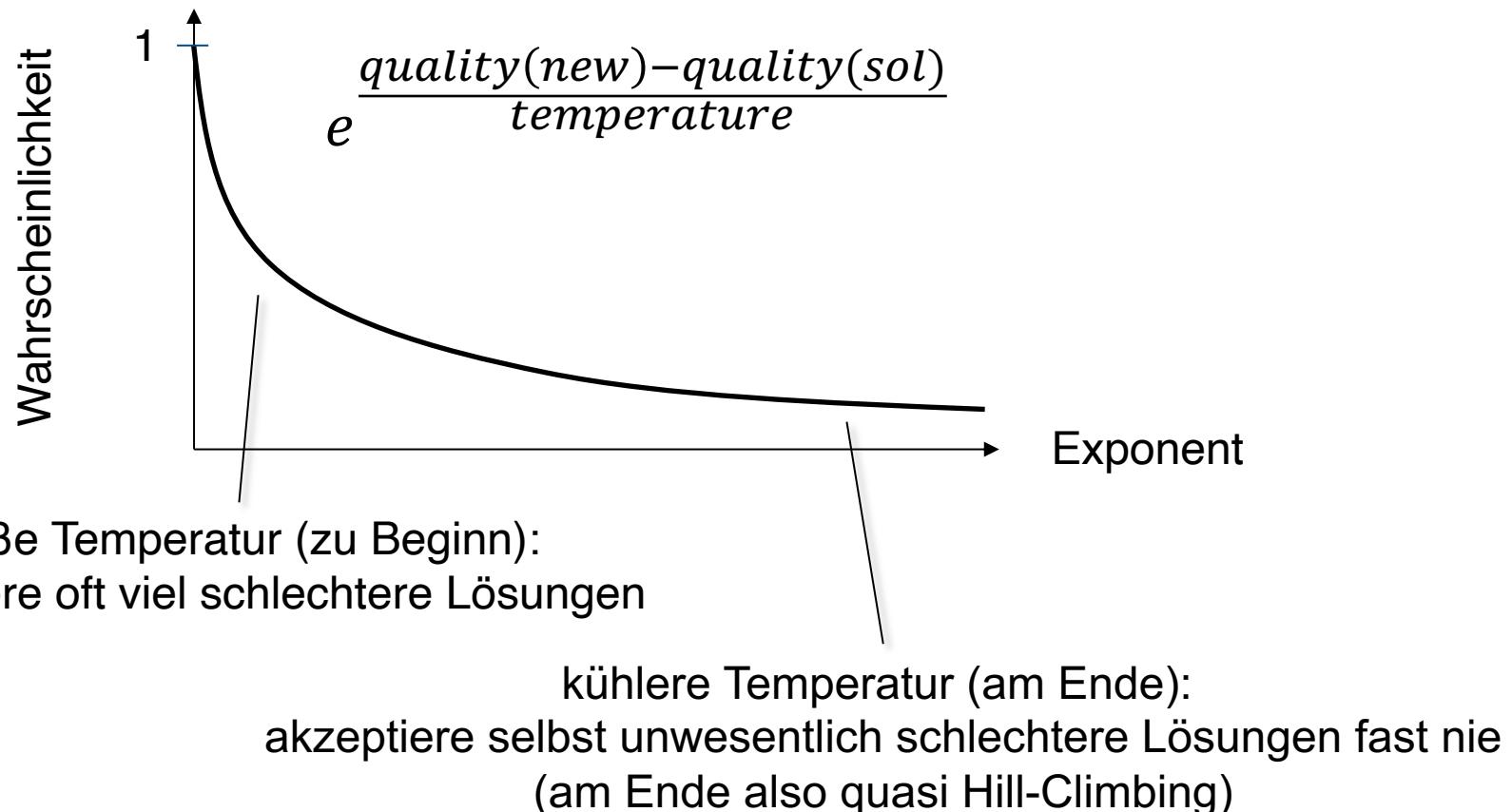
„Annealing“ in Metallverarbeitung:
Härten von Metallen durch Erhitzen auf hohe Temperatur
und langsames Abkühlen



1. Temperatur zu Beginn hoch, kühlt langsam ab
2. Je höher Temperatur, desto wahrscheinlicher „Sprung in schlechte Richtung“

„In schlechte Richtung“ mit Wahrscheinlichkeit

Ansatz: akzeptiere auch schlechtere Lösung **new** mit
quality(new) < quality(sol) mit Wahrscheinlichkeit:



Simulated Annealing

```
SimulatedAnnealing(P) // maxTime constant
                      // TempSched[] temparature annealing

1  sol=initialSol(P);
2  time=0;
3  WHILE time<maxTime DO
4      new=perturb(P,sol);
5      temperature=TempSched[time];
6      d=quality(P,new)-quality(P,sol); r=random(0,1);
7      IF d>0 OR r<=exp(d/temperature) THEN
8          sol=new;
9          time=time+1;
10 return sol;
```

uniformer Wert
zwischen 0 und 1

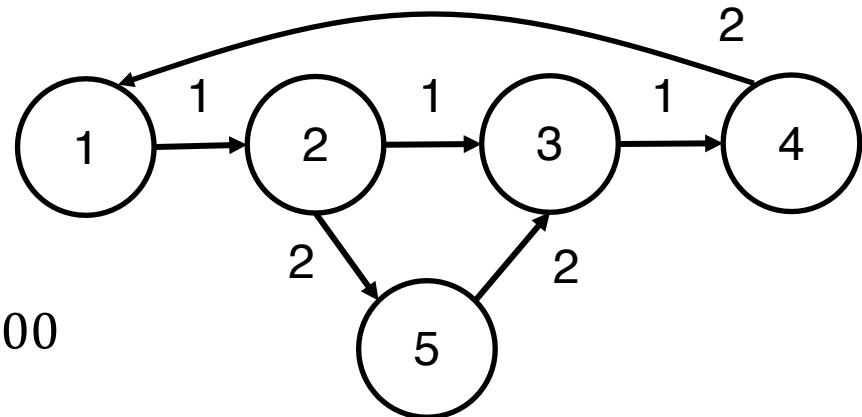
Bestimmung eines guten „Annealing schedule“ (Starttemperatur und Abnahme)
betrachten wir hier in der Vorlesung nicht weiter

Simulated Annealing beim TSP

Starttemperatur

$$= \max_{e \in E} \{w(e)\} = N = 1000$$

lineare Temperaturabnahme = $N/2n = 100$



	Aktuelle Lösung	Gewicht	Zuf. Tausch	Gew. nach Tausch	Temp	Zuf. Wert r	Exp (-d/temp)	Neue Löung akzeptiert?
1	1→2→3→4→5→1	2N+3	2 und 3	5N	1000	0.37	0.0498	nein
2	1→2→3→4→5→1	2N+3	2 und 4	4N+2	900	0.60	0.1084	nein
3	1→2→3→4→5→1	2N+3	4 und 5	2N+4	800	0.17	0.9986	ja (Zufall)
4	1→2→3→5→4→1	2N+4	2 und 4	5N	700	0.34	0.0138	nein
5	1→2→3→5→4→1	2N+4	3 und 4	3N+3	600	0.45	0.1889	nein
6	1→2→3→5→4→1	2N+4	3 und 5	8	500	0.78	–	ja (besser)
7	1→2→5→3→4→1	8	2 und 5	2N+4	400	0.51	0.0067	nein

Weitere Metaheuristiken

Tabu Search

Ausgehend von aktueller Lösung, suche bessere Lösung in der Nähe

Speichere eine Zeit lang schon besuchte Lösungen, und vermeide diese Lösungen

Wenn keine bessere Lösung in der Nähe, akzeptiere auch schlechtere Lösung

+Schwarmoptimierung, Ameisenkolonialisierung,...

Evolutionäre Algorithmen

Beginne mit Lösungspopulation

Wähle beste Lösungen zur Reproduktion aus

Bilde durch Überkreuzungen und Mutationen der besten Lösungen neue Lösungen

Ersetze schlechteste Lösungen durch diese neuen Lösungen