Algorithmen und Datenstrukturen 2

Sommersemester 2007
10. Vorlesung

Peter Stadler

Universität Leipzig
Institut für Informatik
studla@bioinf.uni-leipzig.de

Inhaltliche Satzähnlichkeit

Frage: Wie ähnlich sind 2 Sätze, welche beide genau die Wörter

stehst Wenn klar gewinnen alles natürlich du Finale willst

Enthalten? Wieso ist es besser bei

Staatsanwaltschaft Christoph_Schlingensief Volksverhetzung ermittelt Verdachts

Antwort:

- Seltene Wörter sind aussagekräftiger als häufige Wörter
- Nicht alle seltenen Wörter sind inhaltlich aussagekräftig, aber ihr gemeinsames Auftreten spricht trotzdem für große Übereinstimmung.
- Stoppwörter sagen gar nichts aus

Satzähnlichkeit: Beispiel 1

Beispielsätze mit der Übereinstimmung

Staatsanwaltschaft Christoph_Schlingensief Volksverhetzung ermittelt Verdachts wegen

bsp_nr=16042: *Gegen den Theatermacher Christoph Schlingensief ermittelt die Staatsanwaltschaft wegen des Verdachts der Volksverhetzung.*

bsp_nr=15634: *Düsseldorf (dpa) Die Staatsanwaltschaft ermittelt gegen den Regisseur Christoph Schlingensief wegen des Verdachts der Volksverhetzung.*

Satzähnlichkeit: Beispiel 2

Beispielsätze mit der Übereinstimmung

stehst Wenn klar gewinnen alles natürlich du Finale willst

bsp_nr=8244: *Wenn du im Finale stehst, willst du natürlich alles gewinnen, das ist klar.*

bsp_nr=15758: *Oliver Kahn: "Wenn du im Finale stehst, willst du natürlich alles gewinnen, das ist klar.*

Aber auch:

bsp_nr=15744: *"Wenn du im Finale stehst, willst du natürlich alles gewinnen, das ist doch klar", formulierte Torhüter Oliver Kahn selbstbewusst das nächste Ziel und Christoph Metzelder meinte: "Wir haben Sensationelles geleistet."*

Algorithmus Satzähnlichkeit 1

- Berücksichtige nur Wörter mit Anzahl <256 und Anzahl >2
- D.h. wir lassen folgende Stoppwörter weg: *Auch Das Der Deutschland Die Er Es Euro Ich Im In Jahr Jahren Prozent Sie Und aber als am an auch auf aus bei bis dann das dass dem den der des die durch ein eine einem einen einer er es für gegen habe haben hat hatte ich ihre im in ist kann keine können man mehr mit nach nicht noch nur oder sagt sagte schon sei sein seine sich sie sind so soll um und unter vom von vor war was wenn werden wie wieder will wir wird wurde zu zum zur zwei über*
- Insgesamt 9393 Wörter
- Das Gewicht eines Wortes liegt zwischen 40 und 90 und wird berechnet zu

  wert(wort) = int(10*log(anzahl(‚der‘)/anzahl(wort))
- Die Ähnlichkeit zweier Sätze ergibt sich als Summe der Werte für die gemeinsamen Wörter

Algorithmus Satzähnlichkeit 2

Schritt 1: Inverse Liste exportieren, Format: wort, anzahl, satz_nr

Schritt 2: Zeilenweise sortieren

Schritt 3: Dubletten entfernen

Schritt 4: Für jedes Wort alle Paare von Sätzen ermitteln, die dieses Wort enthalten. Format: satz_nr1, satz_nr2, wert

Schritt 5: Zeilenweise sortieren

Schritt 6: Werte für gleiche Paare summieren und ausgeben, falls Schwellwert überschritten.

Zusätzlich werden noch die Wörter durchgeschleift und zusammengefasst, damit man die Ausgabe beurteilen kann.

Der Aufwand wächst mehr als quadratisch. Für Wörter mit großer Anzahl ist Schritt 4 tödlich. Das ist kein Problem bei 20.000 Sätzen (Wörter des Tages, wortschatz.uni-leipzig.de/wort-des-tages/), wohl aber bei 25.000.000 Sätzen (wortschatz.uni-leipzig.de).

Helfen würde das Weglassen aller Wörter über einer gewissen Anzahl, aber „Helmut Kohl“ hat im Wortschatz Anzahl 10561, und ist sicher trotzdem wichtig.

Dokumentenähnlichkeit

1. Versuch

Identischer Algorithmus wie bei Satzähnlichkeit, nur wird statt Satznummer die Dokumentnummer genommen.

Die klassifizierenden Wörter werden genau wie eben ausschließlich nach Häufigkeit ausgewählt.

Beobachtung: Dokumente sind stark ähnlich wegen vieler gemeinsamer nicht aussagkräftiger Wörter. Dies bevorzugt speziell längere Dokumente.

2. Versuch

Die klassifizierenden Wörter werden semantisch ausgewählt: Genau die, die schon einmal „Wort des Tages“ waren. Das sind momentan ca. 8.000 Wörter, von denen kommen pro Tag ca. 400 vor. Das schränkt die Wortanzahl und den Aufwand drastisch ein.

Beispiele ähnlicher Dokumente

Doc.-Nr. 1	**Doc.-Nr. 2**	**Ähnlichkeit**
60910293	51923690	9558.00
Weltmeister Südkorea Michael_Ballack Oliver_Kahn Brasilien Yokohama Ballack DFB Rudi_Völler		
60910293	552389133	7946.00
Weltmeister Südkorea Dietmar_Hamann Thomas_Linke Weltmeisterschaft Carsten_Ramelow Rudi_Völler		
60910293	588749685	7278.00
Südkorea Michael_Ballack Weltmeisterschaft Ballack Elf Seoul		
734389933	1313082725	11073.00
Israel Arafat Hebron Jericho Palästinenser Terror Bush Frieden US-Präsident_George_W._Bush		
734389933	1598295465	7344.00
Israel Arafat Palästinenser Terror Bush US-Präsident_George_W._Bush		
242550748	1598295465	7344.00
Israel Arafat Palästinenser Terror Bush US-Präsident_George_W._Bush		
242550748	734389933	12691.00
Israel Autonomiebehörde Arafat Hebron Palästinenser Terror Bush Frieden Jassir_Arafat US-Präsident_George_W._Bush		

Themen des Tages

Wenn mehrere Paare von Dokumenten durch ähnliche Begriffe charakterisiert sind, lassen sich diese Gruppen vielleicht clustern:

Für alle Paare ähnlicher Dokumente (ca. 200 Stück) werden die Mengen gemeinsamer Begriffe (nur „Wörter des Tages“) gebildet.

Schrittweise werden die ähnlichsten Mengen zusammengefasst. Das passiert solange, bis alle Paare von Mengen unter einer gewissen Mindestähnlichkeit liegen.

Ähnlichkeitsmaß: $sim(A,B)= |A \cap B| / (|A|+|B|)$

Clustering: Falls $sim(A,B)>0.4$, dann wird B ersetzt durch $A \cup B$ und A wird entfernt.

Cluster des 25.6.2002

(Titel der Cluster von Hand eingefügt)

NAHOST 1 Gaza-Streifen Arafat Außenminister_Schimon_Peres Jerusalem Terroristen Westjordanland Hebron Israel Panzer Attentäter Palästinenser Ramallah Selbstmordanschläge Israelis Ausgangssperre Autonomiebehörde Tulkarem Bethlehem Dschenin Gazastreifen Terror US-Präsident_George_W._Bush Anschlägen Nablus

FORMEL1 2 Grand_Prix Rubens_Barrichello Ferrari Ralf_Schumacher McLaren-Mercedes Coulthard Barrichello Montoya Großbritannien Schumacher Michael_Schumacher Nürburgring Nürburgri Großen_Preis Stallorder Weltmeister Brasilien Fußball-WM

STOLPE 3 Potsdam SPD-Generalsekretär_Franz_Müntefering Lothar_Späth Stolpe Bundestagswahlkampf Matthias_Platzeck Müntefering Schönbohm Bundesrat Platzeck Brandenburg Manfred_Stolpe Cottbus Zuwanderungsgesetz PDS Wittenberge Ministerpräsident_Manfred_Stolpe Jörg_Schönbohm Bundestagswahl Schröder

WM 4 Korea Südkorea Skibbe Seoul Oliver_Kahn Südkoreaner Michael_Ballack Koreaner Spanier Hitze Nationalmannschaft Elfmeterschießen Viertelfinale Paraguay WM-Halbfinale Miroslav_Klose Völler Jens_Jeremies Karl-Heinz_Rummenigge Klose Golden_Goal Weltmeister Türken Senegal Fußball Verlängerung Brasilien Elf Weltmeisterschaft Entschuldigung Rudi_Völler Portugal Ronaldo Rivaldo Achtelfinale Argentinien Fifa Dietmar_Hamann

PISA 5 Nordrhein-Westfalen Gymnasien Pisa-E Naturwissenschaften Brandenburg Rheinland-Pfalz Sachsen-Anhalt

BÖRSE 6 T-Aktie Allzeittief Neuen_Markt DAX France_Télécom Moody's Tarifrunde

HARTZ 7 Hartz SPD-Generalsekretär_Franz_Müntefering Bundeswirtschaftsminister_Werner_Müller Florian_Gerster Arbeitslosenzahl FDP-Chef_Guido_Westerwelle Hartz-Kommission

Cluster des 26.6.2002

WM 1 Fußball Ilhan_Mansiz Golden_Goal WM-Halbfinale Senegal Türken Schröder Weltmeister Bundesinnenminister_Otto_Schily Völler Bundespräsident_Johannes_Rau Brasilien Bundeskanzler_Gerhard_Schröder Ballack Frings Neuville Bierhoff Jeremies Klose Ramelow Korea Südkorea Michael_Ballack Oliver_Kahn Beckham Weltmeisterschaft Zidane Pelé Ronaldo Rivaldo Miroslav_Klose Viertelfinale Paraguay Jens_Jeremies FC_Liverpool Seoul Christian_Ziege Spanier Sebastian_Kehl Elf Saudi-Arabien Thomas_Linke Nationalmannschaft Rudi_Völler Seo Carsten_Ramelow Christoph_Metzelder Foul WM-Finale Koreaner Südkoreaner Oliver_Bierhoff Dietmar_Hamann Yokohama Schiedsrichter Franz_Beckenbauer Portugal Guus_Hiddink DFB Oliver_Neuville Marco_Bode Gelbe_Karte Fifa Franzosen Yoo

NAHOST 2 Ariel_Scharon Israel Arafat Palästinenser Bush Nahen_Osten Autonomiebehörde Hebron Terror Frieden Jassir_Arafat US-Präsident_George_W._Bush Jericho Ramallah Israelis Scharon Weiße_Haus George_Bush Palästina Westjordanland Jerusalem Ministerpräsident_Ariel_Scharon Panzer Großbritannien Gewalt Palästinenserpräsident_Jassir_Arafat US-Regierung Anschläge Waffen Intifada

ERFURT 3 Schule Massaker Erfurt Lehrer Steinhäuser Rainer_Heise Robert_Steinhäuser

STOLPE 4 Brandenburg Bundesrat Stolpe Bundespräsident_Johannes_Rau PDS Jörg_Schönbohm Schönbohm Lothar_Späth Platzeck Matthias_Platzeck Manfred_Stolpe

FORMEL1 5 Weltmeister Rubens_Barrichello Nürburgring Großen_Preis McLaren-Mercedes Barrichello Ralf_Schumacher Ferrari Michael_Schumacher Schumacher Jean_Todt

BÖRSE 6 Moody's Neuen_Markt DAX ABN_Amro Goldman_Sachs France_Télécom

BABCOCK 7 Babcock Nordrhein-Westfalen Oberhausen Bürgschaft Babcock_Borsig IG_Metall Stellenabbau Indien

HOLZMANN 8 Philipp_Holzmann_AG Ottmar_Hermann Baukonzern Philipp_Holzmann Holzmann Insolvenz Niederländer

Vergleich der zwei Tage

25.6.02:
WM
NAHOST
FORMEL1
STOLPE
BÖRSE
PISA
HARTZ

26.6.02:
WM
NAHOST
FORMEL1
STOLPE
BÖRSE
ERFURT
BABCOCK
HOLZMANN

Einige Themen lassen sich so über mehrere Tage verfolgen, andere Themen sind nur einen Tag lang aktuell, kehren allerdings manchmal wieder (z.B. PISA).

Werden die Themen einmal von Hand benannt, ist für viele Themen die Benennung am nächsten Tag automatisch möglich.

Grundlegende Strategien von Algorithmen

- brute force
- divide & conquer
- dynamische Programmierung
- greedy
- evolutionär $\rightarrow$ genetische Algorithmen

Dynamische Programmierung

Grundlage: Das Prinzip „Teile und Herrsche“

> Um ein umfangreiches Problem zu lösen, zerlege man es in kleinere Probleme, die unabhängig voneinander gelöst werden können.

In der dynamischen Programmierung wird dieses Prinzip bis zum Extrem weiterentwickelt:

> Wenn nicht bekannt ist, welche kleineren Probleme zu lösen sind, löst man einfach alle und speichert dann die Ergebnisse zum Zwecke der späteren Verwendung bei der Lösung größerer Probleme.

Dieser Ansatz ist in „Operations Research“ weit verbreitet. Hierbei bezieht sich der Begriff „Programmierung“ auf den Prozess der Formulierung der Bedingungen eines Problems, um das Verfahren anwendbar zu machen.

Prinzip und Voraussetzung

1. Charakterisiere den Lösungsraum und die Struktur der intendierten optimalen Lösung

2. Definiere rekursiv, wie sich eine optimale Lösung (und der ihr zugeordnete Wert) aus kleineren optimalen Lösungen (und deren Werte) zusammensetzt.

3. Konzipiere den Algorithmus in einer bottom-up Weise so, dass für n=1,2,3, ... tabellarisch optimale Teillösungen (und deren zugeordnete Werte) gefunden werden. Beim Finden einer bestimmten optimalen Teillösung der Größe k>1 ist auf alle optimalen Teillösungen der Größe < k zurückzugreifen.

Voraussetzung: Die optimale Lösung für ein Problem der Größe n setzt sich aus optimalen Teillösungen kleinerer Größe zusammen. (Bellmannsches Optimalitäts-prinzip)

Wann funktioniert dynamische Programmierung?

Bei der Anwendung der dynamischer Programmierung können zwei Schwierigkeiten auftreten:

- Erstens muss es nicht immer möglich sein, die Lösungen kleinerer Probleme so zu kombinieren, dass sich die Lösung eines größeren Problems ergibt.

- Zweitens kann die Anzahl der zu lösenden Probleme unvertretbar groß sein.

Es ist noch nicht gelungen, genau anzugeben, welche Probleme mit Hilfe der dynamischen Programmierung in effizienter Weise gelöst werden können. Es gibt

- viele „schwierige“ Probleme, für die sie nicht anwendbar zu sein scheint,
- aber auch viele „leichte“ Probleme, für die sie weniger effizient ist als Standardalgorithmen.

Das Produkt mehrerer Matrizen

Ein klassischer Anwendungsfall der dynamischen Programmierung ist das Problem der Minimierung des Rechenaufwands, der für die Multiplikation einer Reihe von Matrizen unterschiedlicher Dimension erforderlich ist.

$$\begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \\ a_{31} & a_{32} \\ a_{41} & a_{42} \end{pmatrix} \quad \begin{pmatrix} b_{11} & b_{12} & b_{13} \\ b_{21} & b_{22} & b_{23} \end{pmatrix} \quad \begin{pmatrix} c_{11} \\ c_{21} \\ c_{31} \end{pmatrix} \quad \begin{pmatrix} d_{11} & d_{12} \end{pmatrix} \quad \begin{pmatrix} e_{11} & e_{12} \\ e_{21} & e_{22} \end{pmatrix} \quad \begin{pmatrix} f_{11} & f_{12} & f_{13} \\ f_{21} & f_{22} & f_{23} \end{pmatrix}$$

4x2 2x3 3x1 1x2 2x2 2x3

Ziel: Multiplikation der 6 Matrizen

Natürlich muss die Anzahl der Spalten in einer Matrix stets mit der Anzahl der Zeilen in der folgenden Matrix übereinstimmen, damit die Multiplikation ausführbar ist.

Die Gesamtanzahl der erforder-lichen Multiplikationen

Diese hängt von der Reihenfolge ab, in der die Matrizen multipliziert werden.

Im Beispiel könnte man von links nach rechts vorgehen:

A x B	24 Skalar-Multipl.	neue Matrix	M1	4 x 3
M1 x C	12 Skalar-Multipl.	neue Matrix	M2	4 x 1
M2 x D	8 Skalar-Multipl.	neue Matrix	M3	4 x 2
. . .				
nach	84 Skalar-Multipl.	Ergebnismatrix		4 x 3

Geht man statt dessen von rechts nach links vor, erhält man als Ergebnis die gleiche Matrix der Dimension 4 x 3 nach nur 69 Skalarmultiplikationen.

Es sind auch noch viele andere Reihenfolgen möglich.

Reihenfolge der Multiplikationen

Die Reihenfolge der Multiplikation kann durch das Setzen von Klammern ausgedrückt werden.

- Reihenfolge von links nach rechts entspricht Ausdruck
  (((((AB) C) D) E) F).
- Reihenfolge von rechts nach links entspricht Ausdruck
  (A (B (C (D (EF)))))).
- Jedes zulässige Setzen von Klammern führt zum richtigen Ergebnis.

→ Wann ist die Anzahl der Multiplikationen am kleinsten?

Wenn große Matrizen auftreten, können beträchtliche Einsparungen erzielt werden:

Wenn z. B. die Matrizen B, C und F im obigen Beispiel eine Dimension von 300 statt von 3 besitzen, sind bei der Reihenfolge von links nach rechts 6024 Multiplikationen erforderlich, bei der Reihenfolge von rechts nach links dagegen die viel größere Zahl von 274 200 Multiplikationen auszuführen.

Das Reihenfolgeproblem

Dimensionen der Ausgangsmatrizen: $p \times q$ und $q \times r$

$\rightarrow$ Dimension der Zielmatrix: $p \times r$

$\rightarrow$ Anzahl Multiplikationen für 1 Element der Zielmatrix: q

$\rightarrow$ Anzahl Multiplikationen insgesamt: $p \times q \times r$

Allgemeiner Fall: n Matrizen sind miteinander zu multiplizieren:

$$M_1 \, M_2 \, M_3 \, ... \, M_n$$

wobei für jede Matrix M_i, $1 \leq i < n$, gilt:

M_i hat r_i Zeilen und r_{i+1} Spalten.

Ziel: Diejenige Reihenfolge der Multiplikation der Matrizen zu finden, für die die Gesamtzahl der auszuführenden Skalar-Multiplikationen minimal wird.

Beschreibung des Algorithmus I

Die Lösung des Problems mit Hilfe der dynamischen Programmierung besteht darin,

- „von unten nach oben“ vorzugehen und
- berechnete Lösungen kleiner Teilprobleme zu speichern, um eine wiederholte Rechnung zu vermeiden.

1. Berechnung der Kosten für Multiplikation benachbarter Matrizen und Speicherung in einer Tabelle.
2. Berechnung der Kosten für Multiplikation von 3 aufeinanderfolgenden Matrizen.

   z. B. beste Möglichkeit, $M_1M_2M_3$ zu multiplizieren:

   - Entnimm der gespeicherten Tabelle die Kosten der Berechnung von M_1M_2 und addiere die Kosten der Multiplikation dieses Ergebnisses mit M_3.
   - Entnimm der gespeicherten Tabelle die Kosten der Berechnung von M_2M_3 und addiere die Kosten der Multiplikation dieses Ergebnisses mit M_1.
   - Vergleich der Ergebnisse und Abspeichern der besseren Variante.

Beschreibung des Algorithmus II

3. Berechnung der Kosten für Multiplikation von 4 aufeinanderfolgenden Matrizen.

   z. B. beste Möglichkeit, $M_1M_2M_3M_4$ zu multiplizieren:

   - Entnimm der gespeicherten Tabelle die Kosten der Berechnung von $M_1M_2M_3$ und addiere die Kosten der Multiplikation dieses Ergebnisses mit M_4.
   - Entnimm der gespeicherten Tabelle die Kosten der Berechnung von $M_2M_3M_4$ und addiere die Kosten der Multiplikation dieses Ergebnisses mit M_1.
   - Entnimm der gespeicherten Tabelle die Kosten der Berechnung von M_1M_2 sowie von M_3M_4 und addiere dazu die Kosten der Multiplikation dieser beiden Ergebnisse.
   - Vergleich der Ergebnisse und Abspeicherung der besseren Variante.

...

Indem man in dieser Weise fortfährt, findet man schließlich die beste Möglichkeit, alle Matrizen miteinander zu multiplizieren.

Beschreibung des Algorithmus III

allgemein:

Kosten für Multiplikation $M_i M_{i+1} \ldots M_{i+j}$ mit $1 \leq j \leq n-1$ und $1 \leq i \leq n-j$ ergeben sich indem:

- die Gruppe an der Stelle k mit $i < k \leq i+j$ in zwei Teile $M_i M_{i+1} \ldots M_{k-1}$ und $M_k M_{k+1} \ldots M_{i+j}$ zerlegt wird,
- die Kosten für die Berechnung der Teilergebnisse stets aus der Tabelle entnommen werden (nicht neu zu berechnen!) und die Kosten für die Multiplikation der Teilergebnisse hinzu addiert werden und
- von allen Möglichkeiten für k die günstigste ausgewählt (und in der Tabelle gespeichert) wird.

Programm

```
for ( i = 1 ; i <= n ; i++ )
   for ( j = i+1 ; j <= n ; j++ )
      cost[i][j] = INT_MAX ;
for ( i = 1 ; i <= n ; i++ )
   cost[i][i] = 0 ;
for ( j = 1 ; j < n ; j++ )
   for ( i = 1 ; i <= n-j ; i++ )
      for ( k = i+1 ; k <= i+j ; k++ ) {
         t = cost[i][k-1]+cost[k][i+j]+r[i]*r[k]*r[i+j+1];
         if ( t < cost[i][i+j] )
         { cost[i][i+j] = t ; best[i][i+j] = k ; }
      }
```

Programmbeschreibung 1

cost [l][r] gibt die minimalen Kosten der Berechnung von $M_l\ M_{l+1}\ \ldots\ M_r$ an;

- die Kosten für die erste im Programm angegebene Teilgruppe betragen cost [i][k-1],
- die Kosten für die zweite Teilgruppe betragen cost [k][i + j].
- Die Kosten für die abschließenden Multiplikationen lassen sich leicht bestimmen:
  - $M_i\ M_{i+1}\ \ldots\ M_{k-1}$ ist eine Matrix der Dimension $r_i \times r_k$,
  - und $M_k\ M_{k+1}\ \ldots\ M_{i+j}$ ist eine Matrix der Dimension $r_k \times r_{i+j+1}$,
  - so dass die Kosten der Multiplikation dieser beiden Matrizen $r_i\ r_k\ r_{i+j+1}$ betragen.

Auf diese Weise berechnet das Programm cost[i][i+j] für $1 \leq i \leq n\text{-}j$, wobei j von 1 bis n-1 wächst. Wenn man j = n-1 erreicht (und i = 1), hat man die gesuchten minimalen Kosten der Berechnung $M_1 M_2\ \ldots\ M_n$ gefunden.

Programmbeschreibung 2

Die getroffenen Entscheidungen werden in einem getrennten Feld `best` registriert, um sie später, wenn die tatsächliche Folge der Multiplikationen erzeugt werden soll, wieder bestimmen zu können.

Das folgende Programm stellt die Implementation dieses Prozesses der Ermittlung der optimalen Anordnung der Klammern anhand der mit Hilfe des obigen Programms berechneten Felder `cost` und `best` dar.

```
order (int i, int j ) {
   if (i == j ) printf ( „%c“ , name( i ) ) ;
   else  {
      printf ( „(„ ) ;
      order ( i, best [i][j] -1) ;
      order (best[i][j], j );
      printf ( „)“ ) ;
   }
}
```

Lösung des Problems der Multiplikation mehrerer Matrizen

	A	*B*	*C*	*D*	*E*	*F*
A	0	24 [A][B]	14 [A][BC]	22 [ABC][D]	26 [ABC][DE]	36 [ABC][DEF]
B		0	6 [B][C]	10 [BC][D]	14 [BC][DE]	22 [BC][DEF]
C			0	6 [C][D]	10 [C][DE]	19 [C][DEF]
D				0	4 [D][E]	10 [DE][F]
E					0	12 [E][F]
F						0

Beschreibung zur Tabelle

Sie zeigt den Ablauf der oben gezeigten Programme für das angegebene Beispiel.

Sie gibt die Gesamtkosten und die optimale „letzte“ Multiplikation für jede Teilfolge in der Liste der Matrizen an. Z. B. besagt die Eintragung in der Zeile A und der Spalte F, dass 36 Skalar-Multiplikationen erforderlich sind, um die Matrizen A bis F zu multiplizieren, und dass dies erreicht werden kann, indem A bis C auf optimale Weise multipliziert werden, dann D bis F auf optimale Weise multipliziert werden und danach die erhaltenen Matrizen miteinander multipliziert werden. Nur D ist in dem Feld best enthalten.

Für obiges Beispiel ist die ermittelte Anordnung der Klammern ((A (BC))((DE) F), wofür nur 36 Skalar-Multiplikationen benötigt werden.

Für das Beispiel, wo die Dimension von 3 auf 300 geändert wurde, ist die gleiche Anordnung der Klammern optimal, wobei hier 2412 Skalar-Multiplikationen erforderlich sind.

Eigenschaft

Mit Hilfe der dynamischen Programmierung kann das Problem der Multiplikation mehrerer Matrizen in einer zu N^3 proportionalen Zeit und mit einem zu N^2 proportionalen Speicheraufwand gelöst werden.

Beispiel: Traveling Salesman Problem (TSP)

Gegeben: n Städte, Entfernungsmatrix $M = (m_{ij})$, m_{ij} Entfernung von Stadt i nach Stadt j.

Gesucht: Rundreise über alle Städte mit minimaler Länge, also Permutation π: {1,...,n} -> {1,...,n} so dass

$$c(p) = \left(\sum_{i=1}^{n-1} m_{\pi(i),\pi(i+1)} \right) + m_{\pi(n),\pi(1)} \quad \text{minimal.}$$

NP-vollständig (Rechenzeit mit großer Sicherheit exponentiell)

Naiver Algorithmus: alle (n-1)! Reihenfolgen betrachten.

Dynamische Programmierung für TSP

Sei g(i,S) Länge des kürzesten Weges von Stadt i über jede Stadt in der Menge S (jeweils genau 1 Besuch) nach Stadt 1.

Lösung des TSP also g(1, {2,...,n}).
(Stadt 1 kann beliebig gewählt werden, da Rundreise gesucht wird.)

Es gilt:

$$g(i,S) = \begin{cases} m_{i1} & \text{falls } S = \{\} \\ \min_{j \in S} (m_{ij} + g(j, S - \{j\})) & \text{sonst} \end{cases}$$

Algorithmus für TSP

```
for ( i = 2 ; i = n; i++)  g[i,{}] = m_i1
for ( k = 1; k =  n-2 ; k++ )
   for (S, |S| = k, 1∉S )
      for (i ∈ {2,...,n}-S )
         Berechne g[i,S] gemäß Formel
Berechne g[1,{ 2,...,n }] gemäß Formel
```

Komplexität: Tabellengröße * Aufwand je Tabelleneintrag
Größe: $< n\,2^n$ (Anzahl der i's mal Anzahl betrachtete Teilmengen)
Tabelleneintrag: Suche nach Minimum unter j aus S: O(n)
Insgesamt: $O(n^2 2^n)$, deutlich besser als *(n-1)!* .

TSP-Beispiel: 4 Städte I

	1	2	3	4
1	0	4	9	8
2	4	0	12	2
3	9	12	0	10
4	8	2	10	0

Tabelleneinträge:

g[2,{}] = 4
g[3,{}] = 9
g[4,{}] = 8

g[2,{3}] = 12 + 9 = 21
g[2,{4}] = 2 + 8 = 10
g[3,{2}] = 12 + 4 = 16
g[3,{4}] = 10 + 8 = 18
g[4,{2}] = 2 + 4 = 6
g[4,{3}] = 10 + 9 = 19

TSP-Beispiel: 4 Städte II

$g[2,\{3,4\}] = \min(m_{23} + g[3,\{4\}] , m_{24} + g[4,\{3\}]) = \min(30,21) = 21$
$g[3,\{2,4\}] = \min(m_{32} + g[2,\{4\}] , m_{34} + g[4,\{2\}]) = \min(22,16) = 16$
$g[4,\{2,3\}] = \min(m_{42} + g[2,\{3\}] , m_{43} + g[3,\{2\}]) = \min(23,26) = 23$

$g[1,\{2,3,4\}] = \min(m_{12} + g[2,\{3,4\}] , m_{13} + g[3,\{2,4\}], m_{14} + g[4,\{2,3\}])$
$= \min(25,25,31) = 25$

Lösung: 1, 2, 4, 3, 1 oder
1, 3, 4, 2, 1