

Kap. 6.6: Kürzeste Wege ff. Kap. 7: Optimierung

Petra Mutzel

Lehrstuhl für Algorithm Engineering, LS11

22. VO

27. Juni 2006

Überblick

- All-Pairs-Shortest Path
- Algorithmus von Floyd-Warshall

- Optimierung
 - Einführung
 - Einige Klassische Probleme:
 - TSP, Packen von Kisten, Rucksackproblem
 - Greedy-Heuristiken

2

Motivation

„Warum soll ich heute hier bleiben?“

Einfacher Algorithmus für APSP

„Und was noch?“

Optimierung ist Klasse!

3

6.6.2 All Pairs Shortest Paths

All-Pairs Shortest Paths (APSP)

Gegeben:	gerichteter Graph $G = (V, A)$ Gewichtsfunktion $w : A \rightarrow \mathbb{R}_0^+$
Gesucht:	ein kürzester Weg von u nach v für jedes Paar $u, v \in V$

4

Algorithmen für APSP

Idee:

- Aufruf von Dijkstra für alle Knoten $v \in V$
- Laufzeit: $O(|V| (|V|+|E|) \log |V|)$
 - für dünne Graphen: $O(|V|^2 \log |V|)$
 - für dichte Graphen: $O(|V|^3 \log |V|)$

jetzt: schnellerer Algorithmus für dichte Graphen

5

Algorithmus von Floyd-Warshall

Idee: Löse eingeschränkte Teilprobleme:

- Sei $V_k := \{v_1, \dots, v_k\}$ die Menge der ersten k Knoten
- Finde kürzeste Wege, die nur Knoten aus V_k als Zwischenknoten benutzen dürfen
- Zwischenknoten sind alle Knoten eines Weges außer die beiden Endknoten
- Sei $d_{ij}^{(k)}$ die Länge eines kürzesten Weges von v_i nach v_j , der nur Knoten aus V_k als Zwischenknoten benutzt

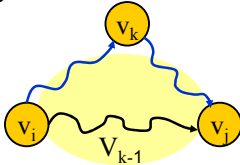
6

Berechnung von $d_{ij}^{(k)}$

Berechnung von $d_{ij}^{(k)}$:

- $d_{ij}^{(0)} = w(v_i, v_j)$
- Bereits berechnet: $d_{ij}^{(k-1)}$ für alle $1 \leq i, j \leq n$

Für einen kürzesten Weg von v_i nach v_j gibt es zwei Möglichkeiten:



7

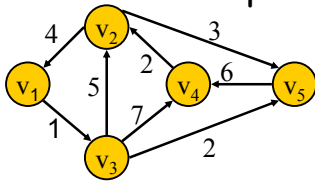
Berechnung von $d_{ij}^{(k)}$

Zwei Möglichkeiten für kürzesten Weg von v_i nach v_j :

- Der Weg p benutzt v_k nicht als Zwischenknoten: dann ist $d_{ij}^{(k)} = d_{ij}^{(k-1)}$
- Der Weg p benutzt v_k als Zwischenknoten: dann setzt sich p aus einem kürzesten Weg von v_i nach v_k und einem von v_k nach v_j zusammen, die jeweils nur Zwischenknoten aus V_{k-1} benutzen: $d_{ij}^{(k)} = d_{ik}^{(k-1)} + d_{kj}^{(k-1)}$

- $d_{ij}^{(k)} := \min(d_{ij}^{(k-1)}, d_{ik}^{(k-1)} + d_{kj}^{(k-1)})$
- Für $k=n$: optimale Wege gefunden

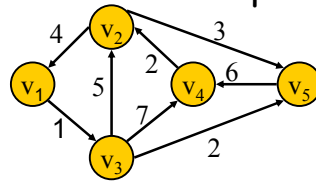
Beispiel



	v_1	v_2	v_3	v_4	v_5
v_1	0	∞	1	∞	∞
v_2	4	0	∞	∞	3
v_3	∞	5	0	7	2
v_4	∞	2	∞	0	∞
v_5	∞	∞	∞	6	0

9

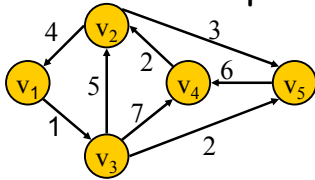
Beispiel



$$D^{(0)} = \begin{pmatrix} 0 & \infty & 1 & \infty & \infty \\ 4 & 0 & \infty & \infty & 3 \\ \infty & 5 & 0 & 7 & 2 \\ \infty & 2 & \infty & 0 & \infty \\ \infty & \infty & \infty & 6 & 0 \end{pmatrix} \quad D^{(1)} = \begin{pmatrix} 0 & \infty & 1 & \infty & \infty \\ 4 & 0 & 5 & \infty & 3 \\ \infty & 5 & 0 & 7 & 2 \\ \infty & 2 & \infty & 0 & \infty \\ \infty & \infty & \infty & 6 & 0 \end{pmatrix}$$

10

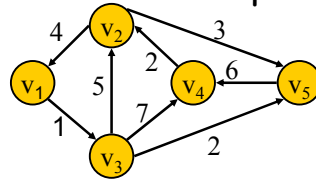
Beispiel



$$D^{(1)} = \begin{pmatrix} 0 & \infty & 1 & \infty & \infty \\ 4 & 0 & 5 & \infty & 3 \\ \infty & 5 & 0 & 7 & 2 \\ \infty & 2 & \infty & 0 & \infty \\ \infty & \infty & \infty & 6 & 0 \end{pmatrix} \quad D^{(2)} = \begin{pmatrix} 0 & \infty & 1 & \infty & \infty \\ 4 & 0 & 5 & \infty & 3 \\ 9 & 5 & 0 & 7 & 2 \\ 6 & 2 & 7 & 0 & 5 \\ \infty & \infty & \infty & 6 & 0 \end{pmatrix}$$

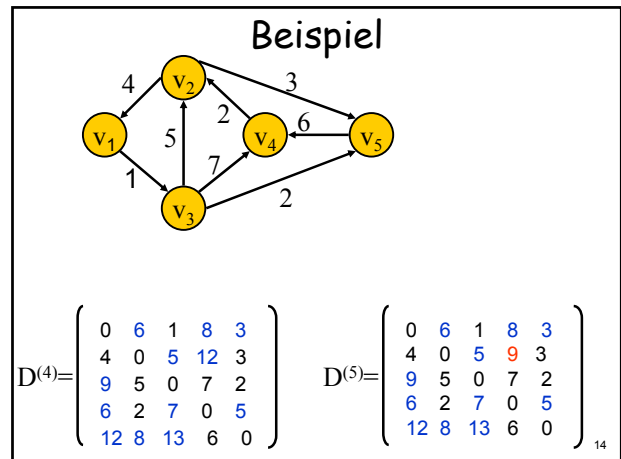
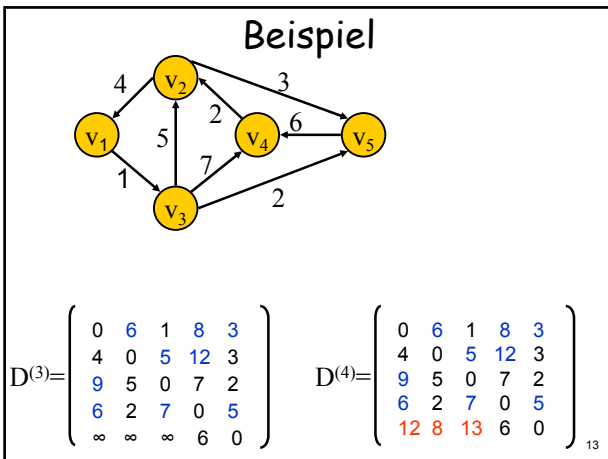
11

Beispiel



$$D^{(2)} = \begin{pmatrix} 0 & \infty & 1 & \infty & \infty \\ 4 & 0 & 5 & \infty & 3 \\ 9 & 5 & 0 & 7 & 2 \\ 6 & 2 & 7 & 0 & 5 \\ \infty & \infty & \infty & 6 & 0 \end{pmatrix} \quad D^{(3)} = \begin{pmatrix} 0 & 6 & 1 & 8 & 3 \\ 4 & 0 & 5 & 12 & 3 \\ 9 & 5 & 0 & 7 & 2 \\ 6 & 2 & 7 & 0 & 5 \\ \infty & \infty & \infty & 6 & 0 \end{pmatrix}$$

12



Algorithmus von Floyd-Warshall

```

(1) for i:=1 to n do
(2)   for j:=1 to n do
(3)     dij(0) := w(vi, vj)
(4)   } }
(5) for k:=1 to n do
(6)   for i:=1 to n do
(7)     for j:=1 to n do
(8)       dij(k) := min (dij(k-1), dik(k-1) + dkj(k-1))
(9)     } } }

```

15

Diskussion

- Ist man auch an den Wegen selbst interessiert, dann: **Vorgängermatrix**: $\pi_{ij}^{(k)}$ für jedes $0 \leq k \leq n$
- Initialisierung**:
 - Falls $i=j$ oder $w(v_i, v_j) = \infty$, dann: $\pi_{ij}^{(0)} := \text{nil}$
 - Sonst: $\pi_{ij}^{(0)} := i$
- Aktualisierung**:
 - Falls $\pi_{ij}^{(k-1)} \leq \pi_{ik}^{(k-1)} + \pi_{kj}^{(k-1)}$ dann: $\pi_{ij}^{(k)} := \pi_{ij}^{(k-1)}$
 - Sonst: $\pi_{ij}^{(k)} := \pi_{kj}^{(k-1)}$

16

Analyse

- Speicherplatzverbrauch für d-Matrizen: $\Theta(|V|^3)$
- Man benötigt zur Berechnung von $d_{ij}^{(k)}$ nur die Werte von $d_{ij}^{(k-1)} \rightarrow \Theta(|V|^2)$ Speicherplatz,
- genauso für die π -Matrizen
- Laufzeit: $\Theta(|V|^3)$

17

Algorithmus von Floyd-Warshall

- Der Algorithmus von Floyd-Warshall berechnet das APSP-Problem in einer Laufzeit von $\Theta(|V|^3)$ mit einem Speicherverbrauch von $\Theta(|V|^2)$.

18

Diskussion

- Bemerkung: Dijkstra-Algorithmus kann durch eine andere Implementierung der Priority-Queue in Laufzeit $O(|V|^2 + |E|)$ realisiert werden
- Dann: APSP für dichte Graphen: $O(|V|^3)$
- Bemerkung: Dijkstra-Algorithmus kann durch eine andere Implementierung der Priority-Queue (Fibonacci-Heaps) in Laufzeit $O(|E| + |V| \log |V|)$ realisiert werden (aber eher theoretisch)
- Dann: APSP für dichte Graphen: $O(|V|^2)$

Aber: Floyd-Warshall konzeptionell einfacher!

19

Kapitel 7: Optimierung

20

Optimierung: Einführung

- Optimierungsprobleme sind Probleme, die i.A. viele zulässige Lösungen besitzen
- Jeder Lösung ist ein bestimmter Wert (**Zielfunktionswert, Kosten**) zugeordnet.
- **Optimierungsalgorithmen** suchen in der Menge aller zulässigen Lösungen diejenige mit dem **optimalen** Wert („die beste“ bzgl. Zielfunktion)
- **Heuristiken** sind Algorithmen, die irgendeine zulässige Lösung berechnen

1

Rundreiseprobleme (TSP)



Du möchtest in Deinem Urlaub verschiedene Städte Europas besuchen und nicht zu viel Zeit im Auto verbringen.

DEIN ZIEL:
Finde die kürzeste Rundtour durch alle Städte.

Bei 12 Städten gibt es 19.958.400 verschiedene Touren.

Beispiele für Optimierungsprobleme

- Minimal aufspannender Baum (MST)
- Kürzeste Wege in Graphen
- Längste Wege in Graphen
- Handlungsreisendenproblem (TSP)
- Rucksackproblem
- Scheduling (z.B. Maschinen, Crew)
- Zuschneideprobleme (z.B. Bilderrahmen)
- Packungsprobleme

23

Optimierungsprobleme (OP)

- Wir unterscheiden zwischen
 - OP, für die wir Algorithmen mit **polynomieller** Laufzeit kennen, und
 - OP, für die noch **kein polynomieller** Algorithmus bekannt ist.
- Die Klasse der **NP-schwierigen** Optimierungsprobleme: das Finden eines polynomiellen Algorithmus für eines dieser OPs zieht automatisch polynomielle Algorithmen für alle anderen OPs dieser Klasse nach sich.
- Allgemeine Vermutung: es existieren keine polynomielle Algorithmen für NP-schwierige OPs

4

Optimierungsprobleme (OP)

- Ein Algorithmus hat polynomielle Laufzeit, wenn die Laufzeitfunktion $T(n)$ durch ein Polynom in n beschränkt ist.
- Dabei steht n für die Eingabegröße der Instanz.

• **Beispiel:** Kürzestes Wegeproblem:
 Laufzeitfunktion in $|V|+|E|$:
 $T(|V|+|E|)=O((|V|+|E|) \log |V|)=O(|V|+|E|)^2$

• **Beispiel:** Längstes Wegeproblem: NP-schwierig
 (Denn: Transformation von TSP)

25

Beispiele für Optimierungsprobleme

- Minimal aufspannender Baum (MST) **polynomiell**
- Kürzeste Wege in Graphen **polynomiell**
- Längste Wege in Graphen **NP-schwierig**
- Handelsreisendenproblem (TSP) **NP-schwierig**
- Rucksackproblem **NP-schwierig**
- Scheduling (z.B. Maschinen, Crew) **NP-schwierig**
- Zuschneideprobleme (z.B. Bilderrahmen) **NP-schwierig**
- Packungsprobleme **NP-schwierig** **NP-schwierig**

26

Strategien zur Lösung von OP

- **Polynomielle OP:**
 - oft: speziell entwickelte Algorithmen
 - manchmal auch allgemeine Strategien, wie z.B. Greedy-Algorithmen oder Dynamische Programmierung
- **NP-schwierige OP:**
 - Exakte Verfahren, Enumeration, Branch-and-Bound, Dynamische Programmierung
 - Heuristiken: Konstruktionsheuristiken (z.B. Greedy, approximative Algorithmen), Verbesserungsheuristiken (z.B. Local Search)

7

Kap. 7.1: Heuristiken

- Vorstellung typischer OP:
 - Rundreiseproblem (TSP)
 - Rucksack-Problem
 - Bin-Packing Problem
- und einfache Greedy-Heuristiken

28

Greedy-Algorithmen

Greedy-Verfahren treffen **lokale** Entscheidungen
 Sie wählen in jedem Schritt die aktuell günstigste Auswahl ohne Vorausschau



Greedy-Algorithmen sind in der Regel nicht optimal, können aber in einigen wichtigen Anwendungen dennoch gute oder gar optimale Lösungen erzeugen!

29

Greedy-Algorithmen

Greedy-Algorithmus („gefräßig“): iterative Konstruktion einer Lösung, die immer um die momentan besten Kandidaten erweitert wird.

Greedy-Algorithmen führen

- bei manchen OP immer zu optimalen Lösungen (Bsp.: MST, SSSP)
- bei manchen OP können sie auch zu Lösungen führen, die „beliebig“ weit von der optimalen Lösung entfernt sind.

30

Das Travelling Salesman Problem

Auch: Handlungsreisendenproblem, Rundreiseproblem, TSP

- **Gegeben:** Vollständiger ungerichteter Graph $G=(V,E)$ mit Kantenkosten c_e
- **Gesucht:** Tour T (Kreis, der jeden Knoten genau einmal enthält) mit minimalen Kosten $c(T)=\sum c_e$

31

Buch von F. Voigt, Ilmenau 1831

Der Handlungsreisende, wie er sein soll und was er zu thun hat, um Aufträge zu erhalten und eines glücklichen Erfolgs in seinen Geschäften gewiß zu sein. Von einem alten Commis-Voyageur.

... Durch geeignete Auswahl und Planung der Tour kann man oft so viel Zeit sparen, daß wir einige Vorschläge zu machen haben. ... Der Wichtigste Aspekt ist, so viele Orte wie möglich zu erreichen, ohne einen Ort zweimal zu besuchen. ... "

32

Wie schwierig ist das TSP?

Anzahl der Städte	Anzahl der möglichen Touren
- 3 Städte	1 Tour
- 4 Städte	3 Touren
- 5 Städte	12 Touren
- 6 Städte	60 Touren
- 7 Städte	360 Touren
- 8 Städte	2520 Touren
- 9 Städte	20160 Touren
- 10 Städte	181440 Touren
- 11 Städte	1814400 Touren
- 12 Städte	19958400 Touren

33

Das Travelling Salesman Problem

Ann.: Rechner schafft 40 Mio. Touren pro Sekunde
Anzahl der verschiedenen Touren: $(|V|-1)! / 2$

n	#Tour	Zeit
10	181440	0.0045 Sek.
17	ca. 10^{13}	3 Tage
19	ca. 10^{15}	2,5 Jahre
20	ca. 10^{17}	48 Jahre
25	ca. 10^{23}	10^8 Jahre
60	ca. 10^{80}	10^{64} Jahre

Anzahl der Atome im Weltall: ca. 10^{80}

34

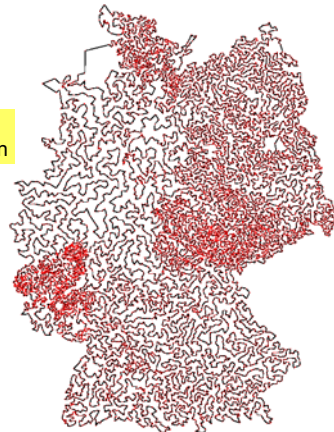
Meilensteine für TSP Lösungen

1954	Dantzig, Fulkerson, Johnson	49
1971	Held, Karp	64
1977	Grötschel	120
1980	Crowder, Padberg	318
1987	Padberg, Rinaldi	532
1987	Grötschel, Holland	666
1987	Padberg, Rinaldi	2392
1994	Applegate, Bixby, Chvátal, Cook	7397
1998	Applegate, Bixby, Chvátal, Cook	13509
2001	Applegate, Bixby, Chvátal, Cook	15112
2004	Applegate, Bixby, Chvátal, Cook, Helsgaun	24978

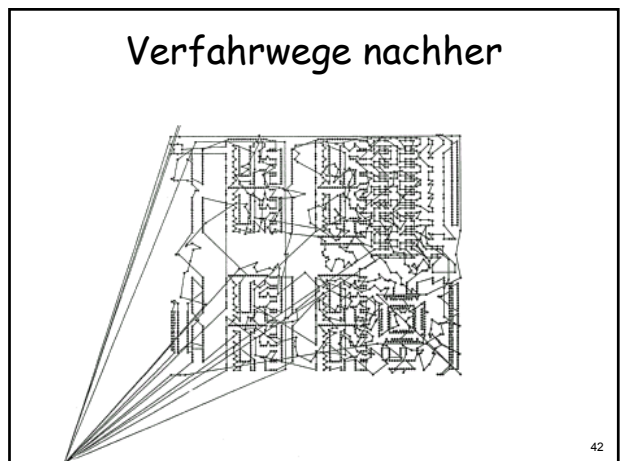
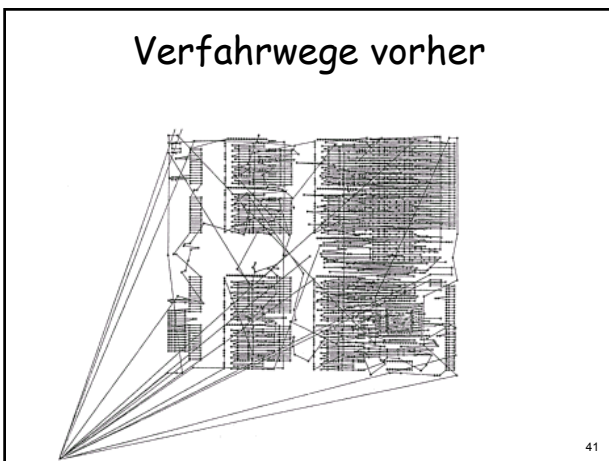
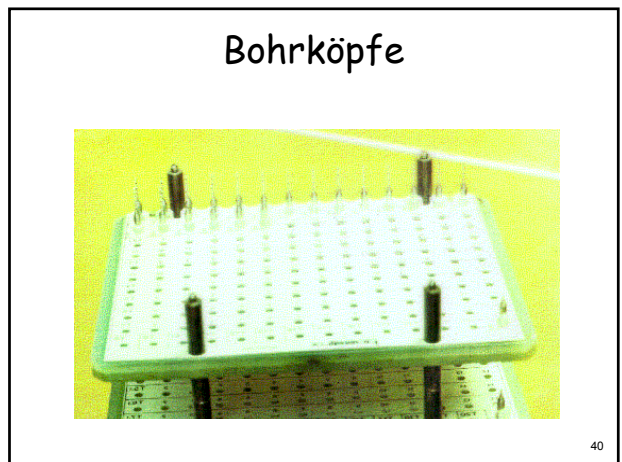
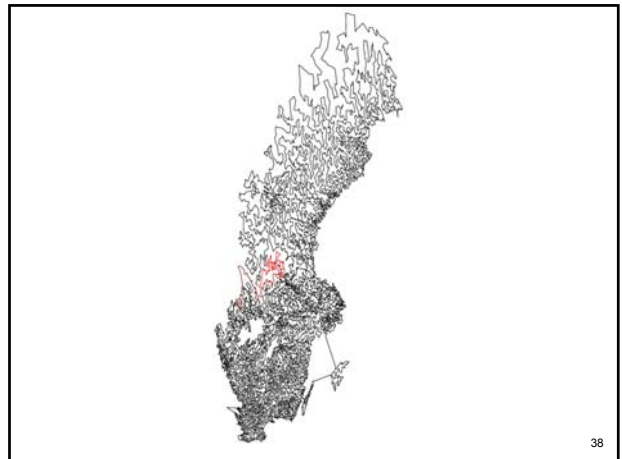
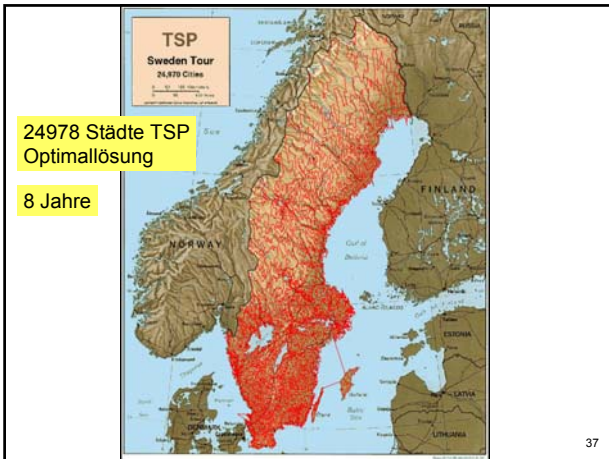
35

15112 Städte TSP

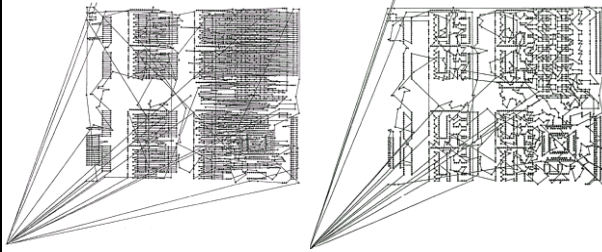
22,6 CPU Jahre
auf 110 Prozessoren



36



Vergleich



43

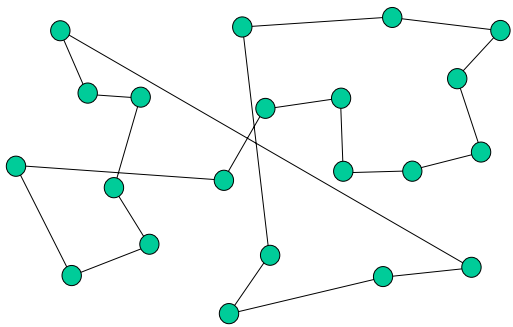
Greedy-Heuristiken für TSP

Nearest-Neighbor Heuristik:

- Beginne mit leerer Tour $T := \emptyset$
- Beginne an einem Knoten $v = v_0$: Ende der Tour
- Solange noch „freie“ Knoten existieren:
 - Suche den nächsten freien (noch nicht besuchten) Knoten zu v (d.h. die billigste Kante (v, w)) und addiere Kante (v, w) zu T .
- Addiere die Kante (v, v_0)

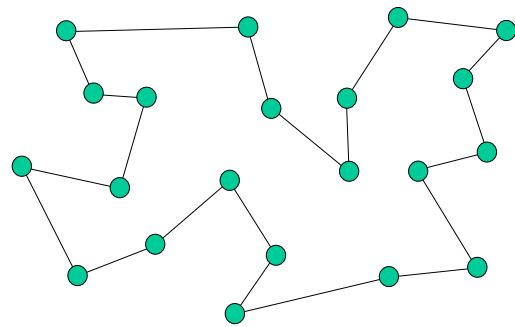
44

Nearest-Neighbor Heuristik für TSP



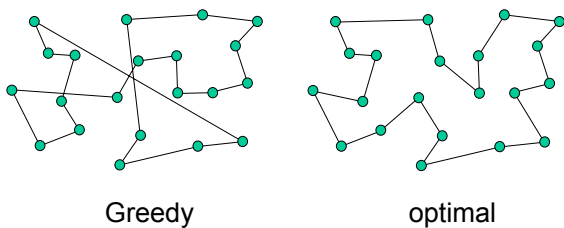
45

Optimale Lösung der TSP-Instanz



46

TSP - Vergleich der Lösungen



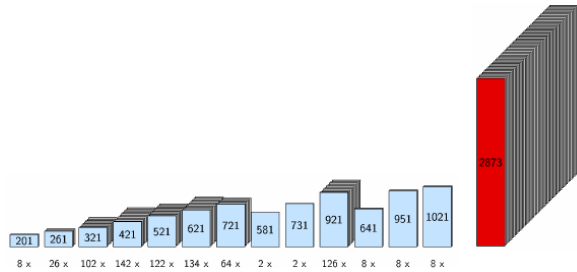
47

Eindimensionale Verschnittoptimierung

- **Geg.:** Gegenstände $1, \dots, N$ der Größe w_i ; und beliebig viele Rohlinge der Größe K .
- **Gesucht:** Finde die kleinste Anzahl von Rohlingen, aus denen alle Gegenstände geschnitten werden können.

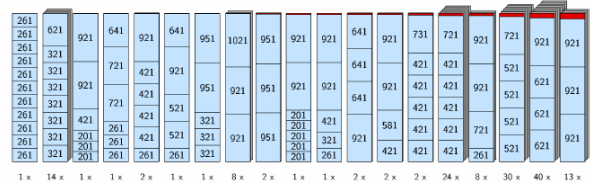
48

Bilderrahmenproblem: Beispiel



49

Minimaler Rohmaterialeinsatz



50

Bin-Packing / Packen von Kisten

- **Geg.:** Gegenstände $1, \dots, N$ der Größe w_i und beliebig viele Kisten der Größe K .
- **Gesucht:** Finde die kleinste Anzahl von Kisten, die alle Gegenstände aufnehmen.

First-Fit Heuristik: Jeder Gegenstand wird in die erstmögliche Kiste gelegt, in die er passt.

51

First-Fit Heuristik für Bin-Packing

- Gegeben sind beliebig viele Kisten der Größe 101 und 37 Gegenstände der Größen:
- 7x Größe 6
- 7x Größe 10
- 3x Größe 16
- 10x Größe 34
- 10x Größe 51

Insgesamt: 17 Kisten

Kiste 1:	7x Größe 6	5x Größe 10	Summe=92
Kiste 2:	2x Größe 10	3x Größe 16	Summe=68
Kisten 3-7:	2x Größe 34		Summe=68
Kiste 8-17:	1x Größe 51		Summe=51

52

Hausaufgabe bis Donnerstag:

- Finden Sie Beispiele bei denen die heute besprochenen Greedy-Algorithmen möglichst schlecht abschneiden.
- Bringen Sie am Donnerstag je eine Folie mit Ihrem Beispiel (für Tageslichtprojektor) mit.

53