PROF. DR. SÁNDOR FEKETE

Abteilung Algorithmik, TU Braunschweig

THEMA

GEHT'S NICHT NOCH ETWAS SCHNELLER?

Eine Rundreise durch gelöste und ungelöste Probleme des algorithmischen Alltags

Algorithmen

Datenstrukturen

Locker drauf,
zielorientiert

Ordentlich,
hält Regeln ein

Algorithmen

Datenstrukturen

I get the job done. What the hell do you want?

Can you make it without killing yourself?

I don‘t make things difficult. That‘s the way they get, all by themselves.

I don‘t want to work with you.

Ain‘t got no choice.

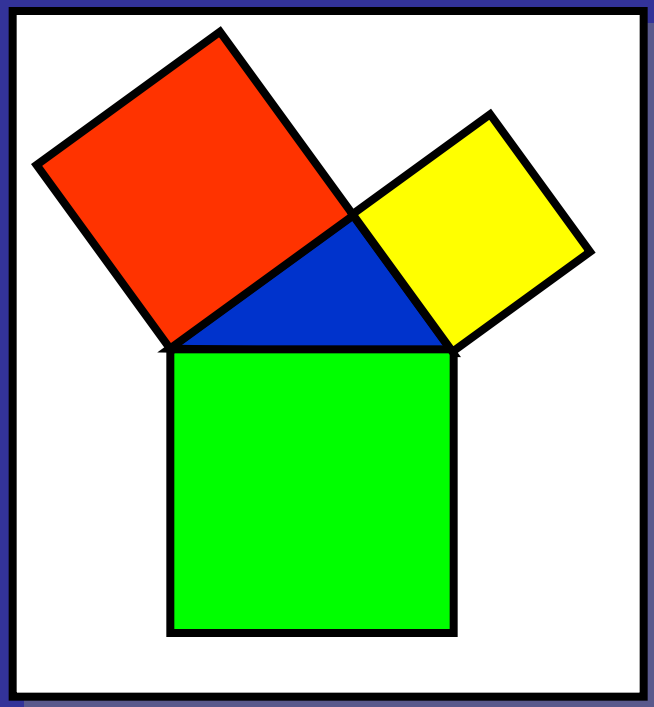
Then don‘t.

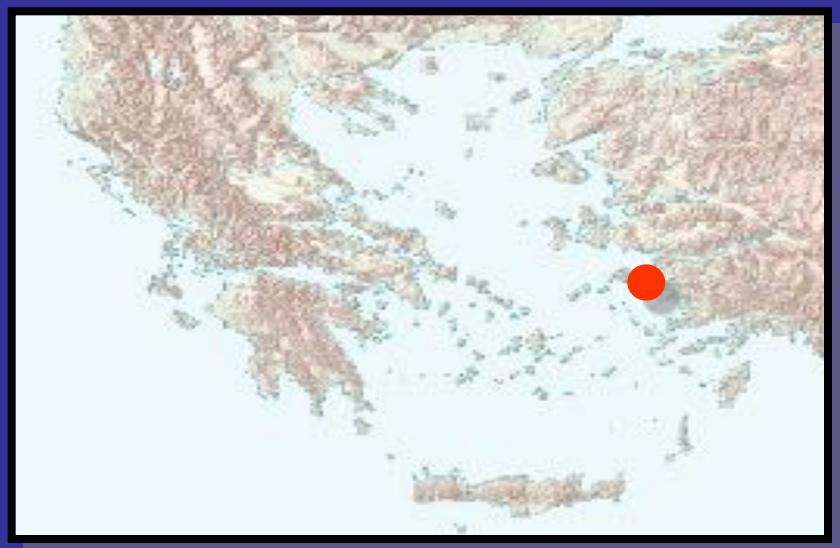
I‘m getting too old for this…

mathematische

Die bekannteste Formel aller Zeiten

$$a^2 + b^2 = c^2$$



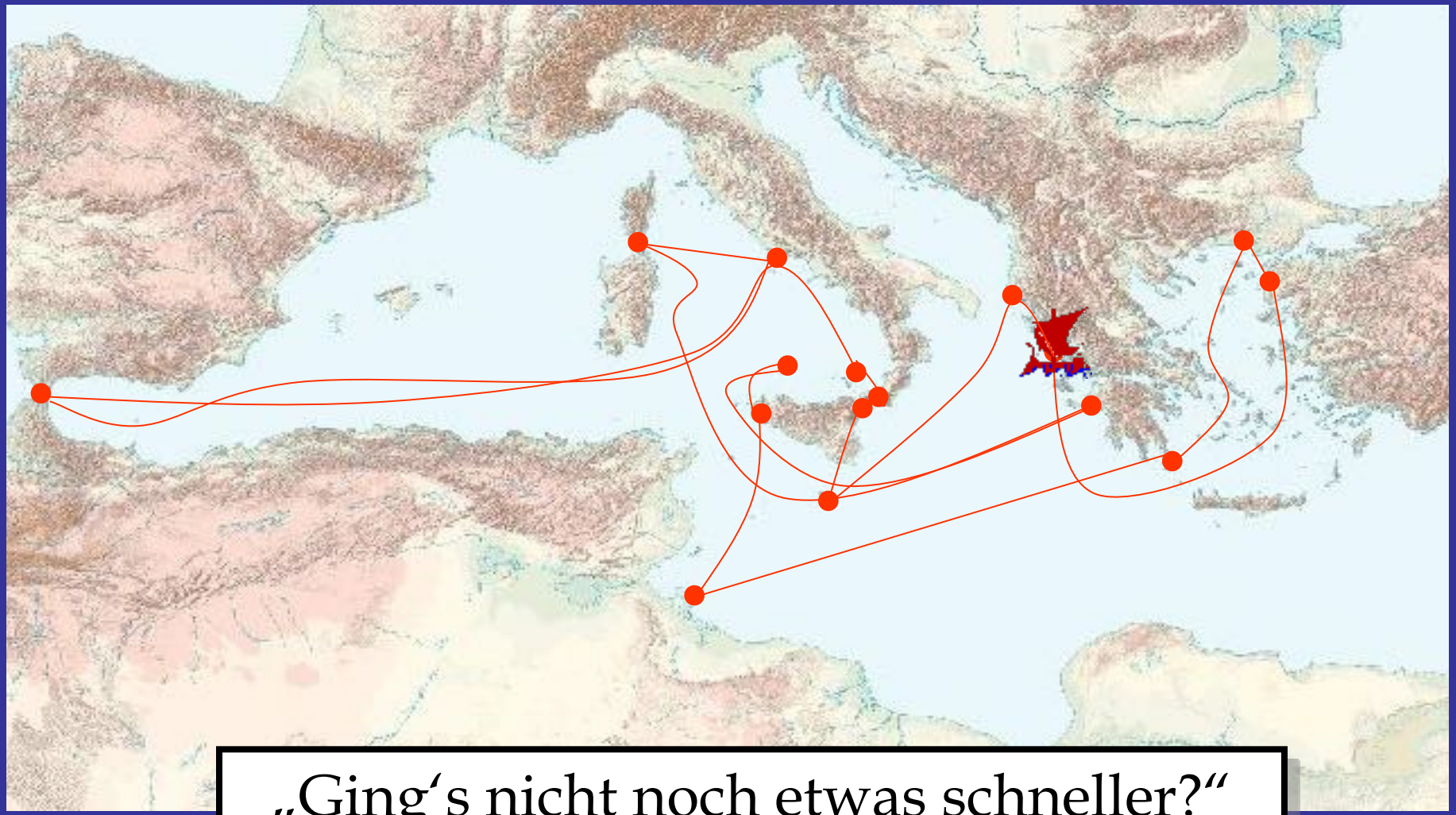


Pythagoras (ca. 500 v.Chr.)

Einige Jahre früher

Einige Jahre früher

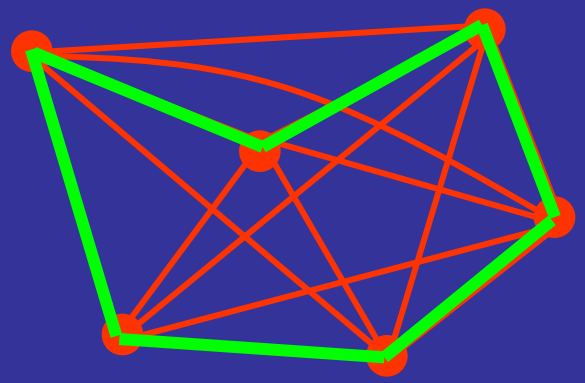
20 Jahre später!



„Ging‘s nicht noch etwas schneller?“

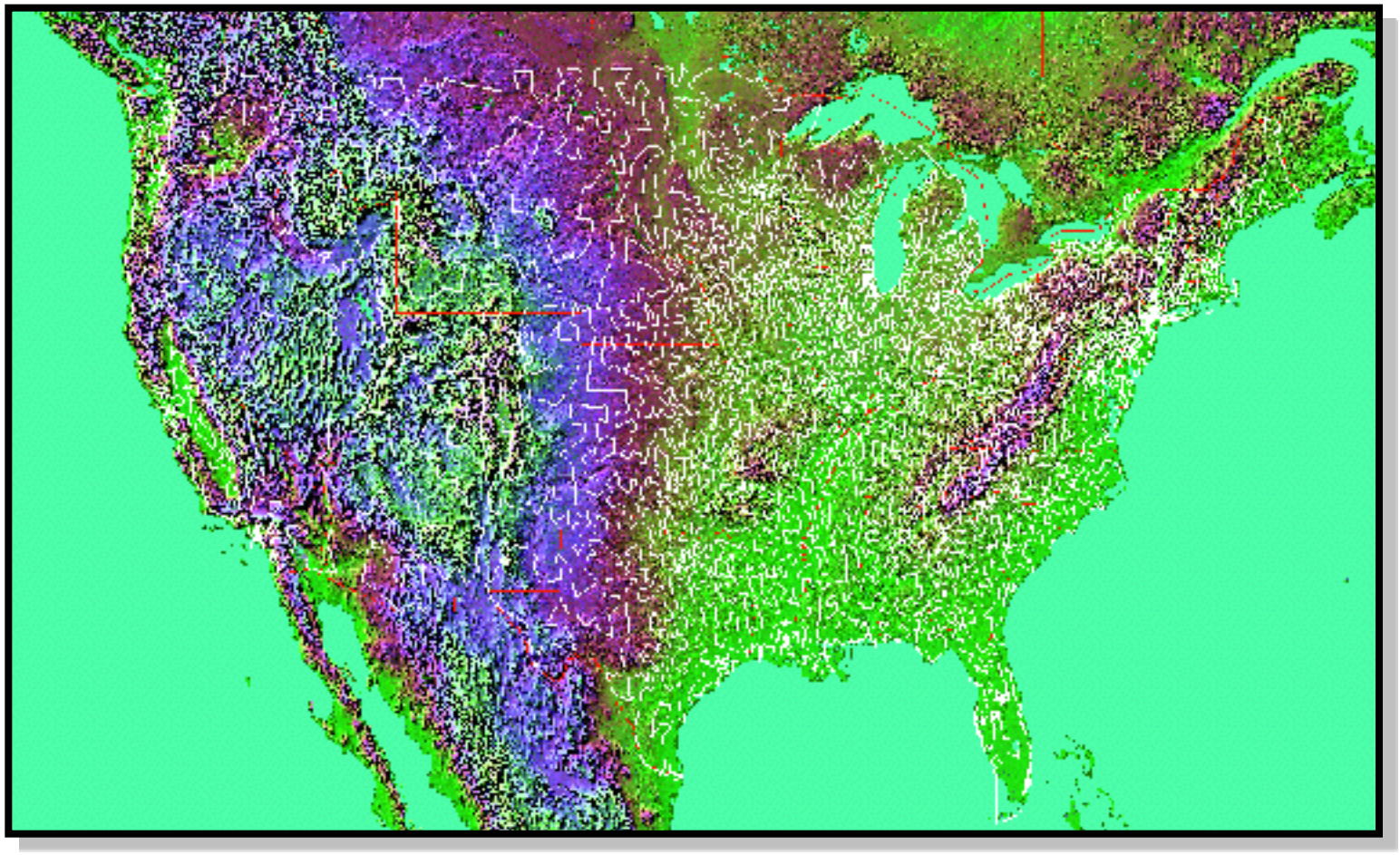
Das Rundreiseproblem

Gegeben: Ein Graph $G = (V, E)$
mit Kantenlängen w_e

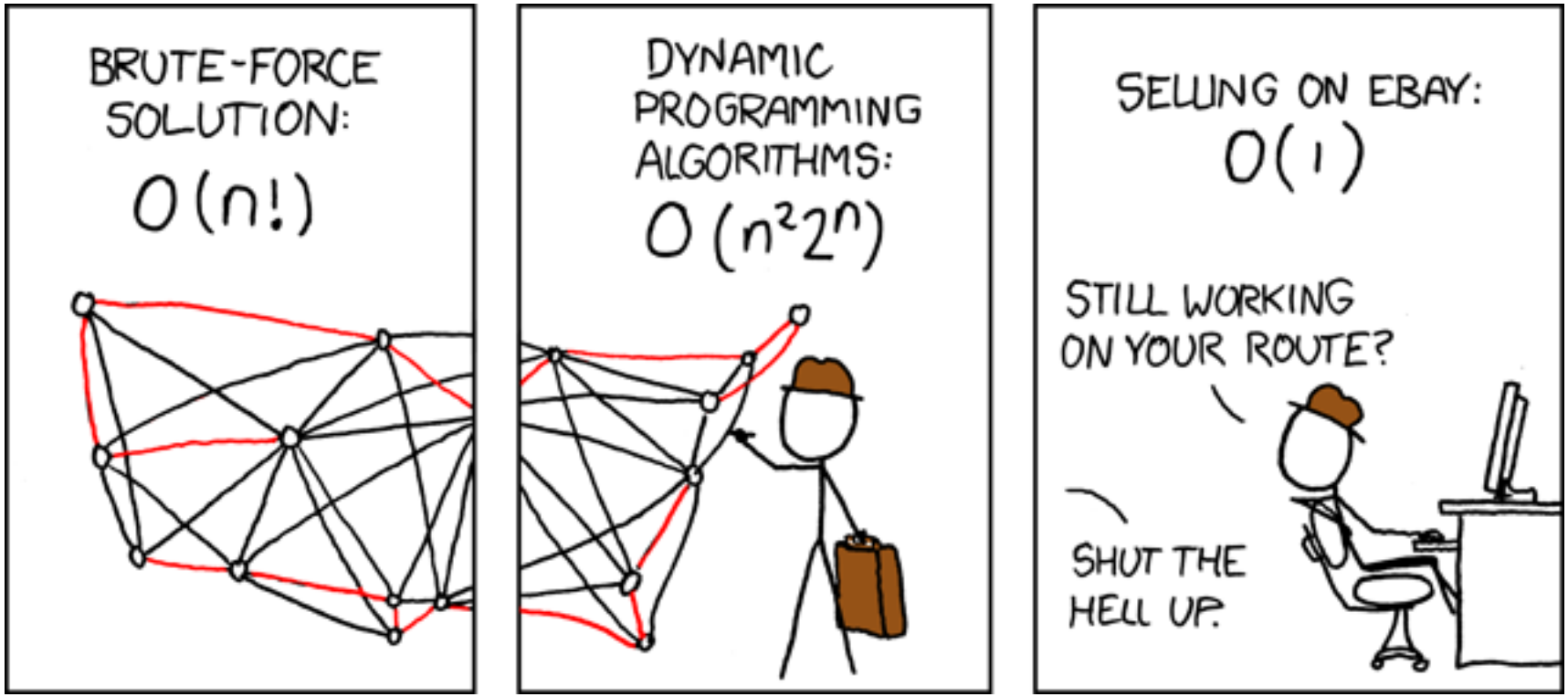


Gesucht: Eine kürzeste Rundreise

Eine optimale Rundreise in der Neuen Welt



Eine optimale Rundreise in der Neuen Welt



„Geht‘s nicht noch etwas schneller?“

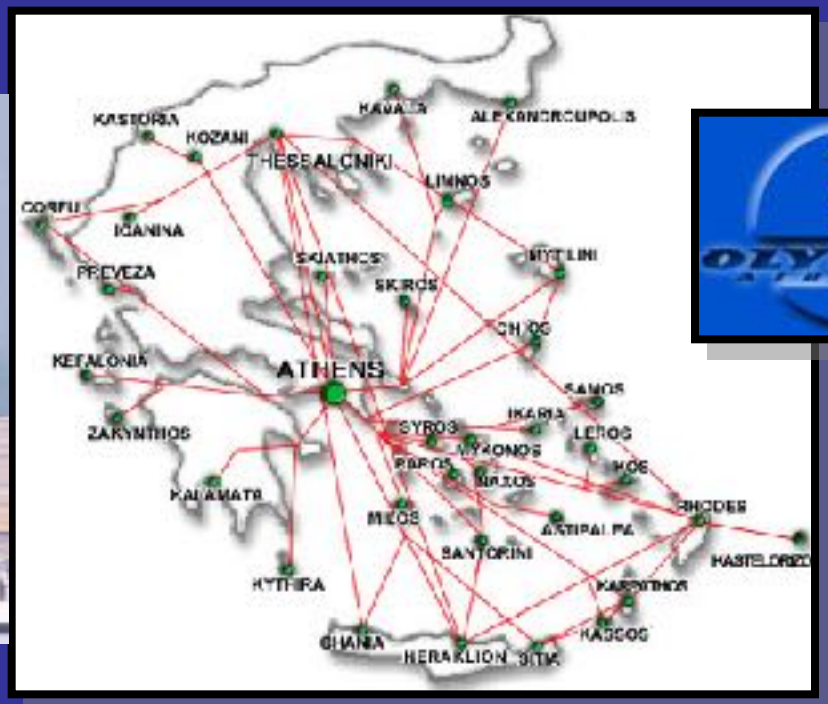
Probleme optimaler Reisegestaltung

Tourenplanung

Probleme optimaler Reisegestaltung

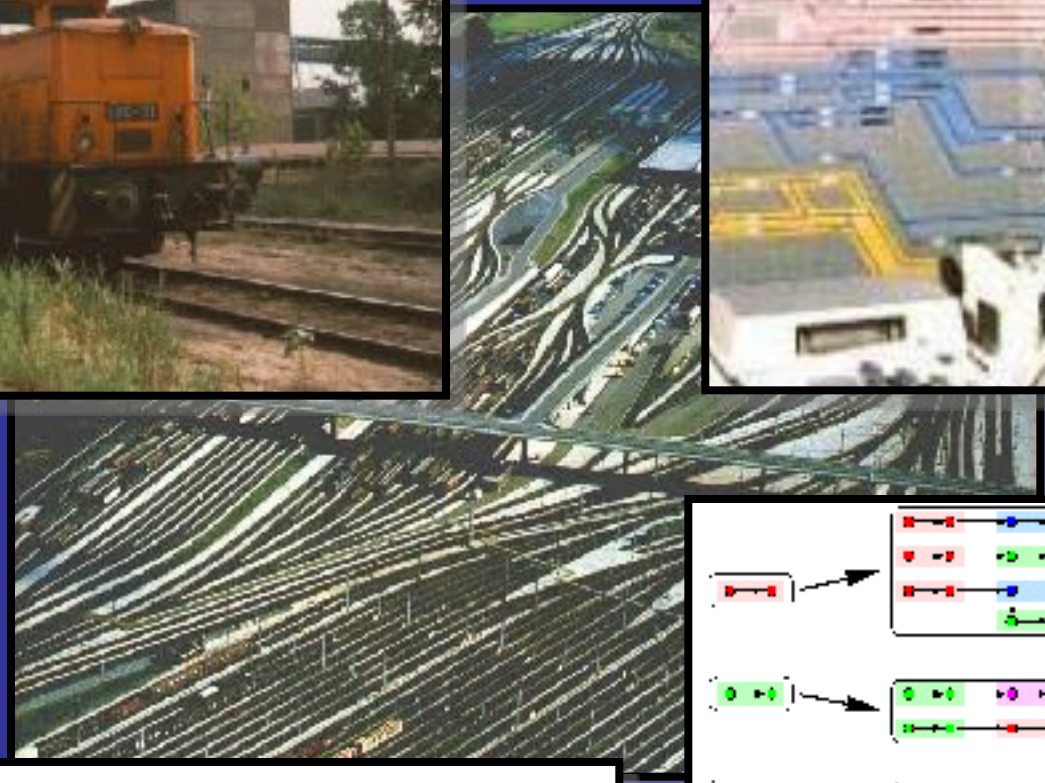
Tourenplanung

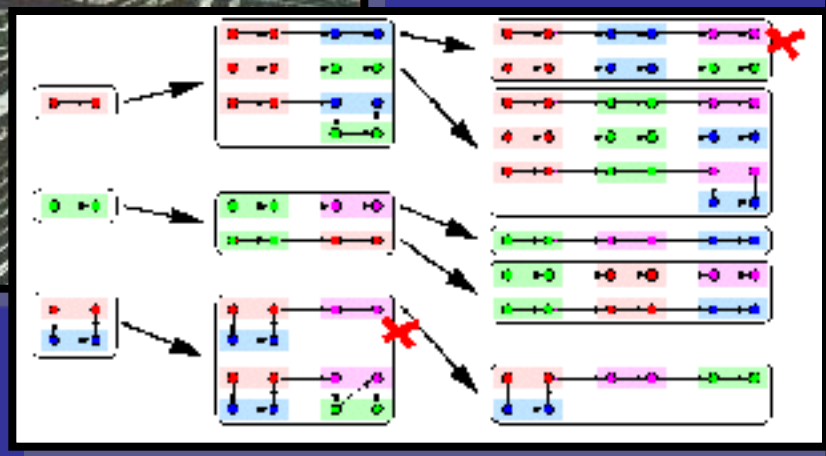
Probleme optimaler Reisegestaltung



Flugplanoptimierung

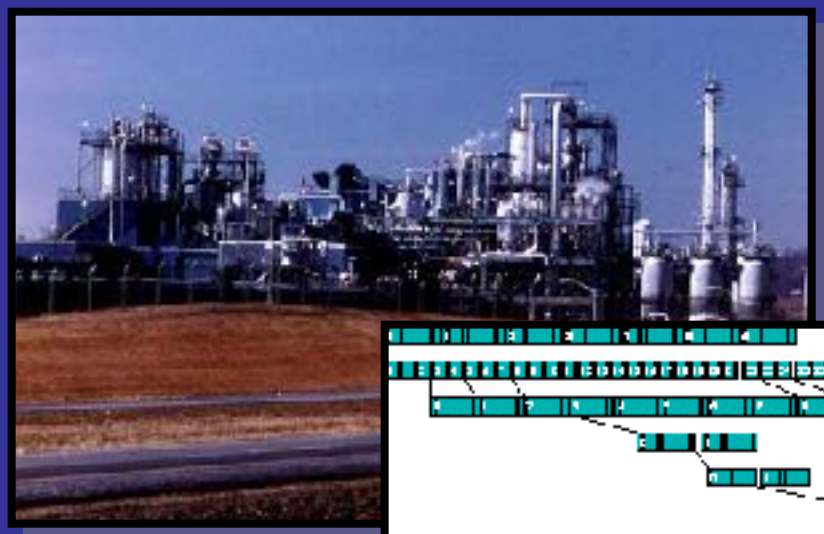
Probleme optimaler Reisegestaltung

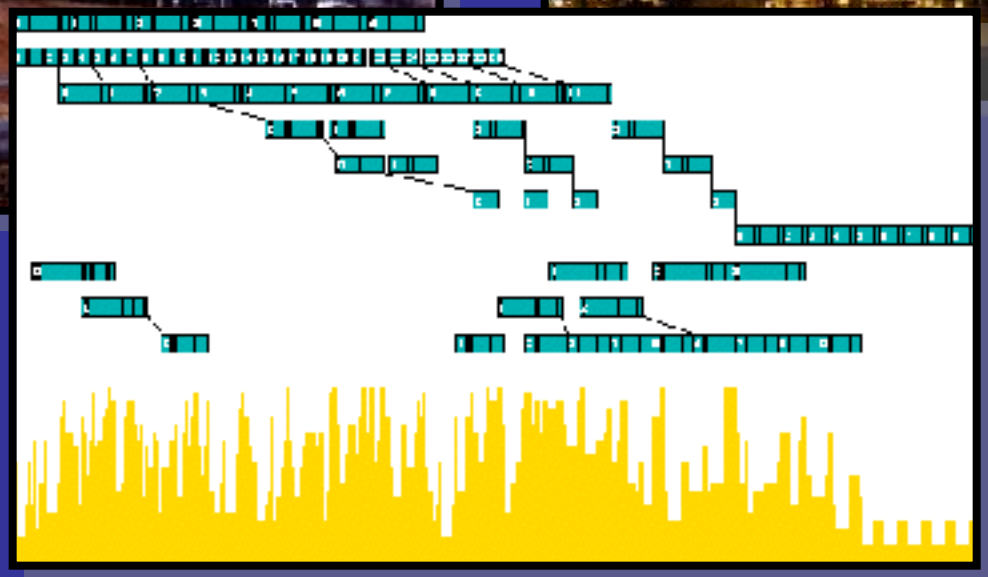




Eisenbahnplanung

Andere diskrete Optimierungsprobleme





Ablaufoptimierung

Andere diskrete Optimierungsprobleme

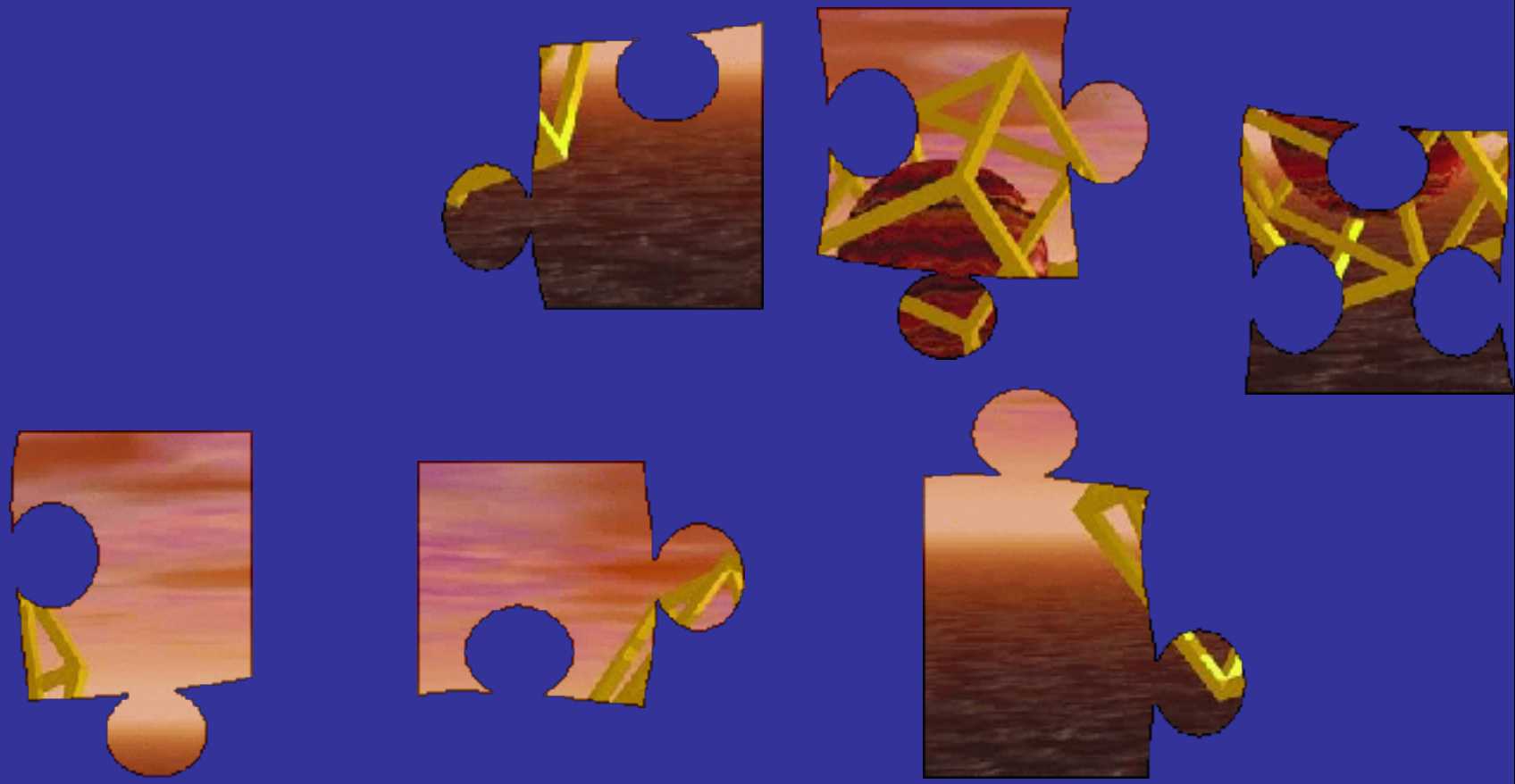
Ablaufoptimierung

Andere diskrete Optimierungsprobleme

Stabile Kommunikationsnetzwerke

Ein algorithmisches Problem

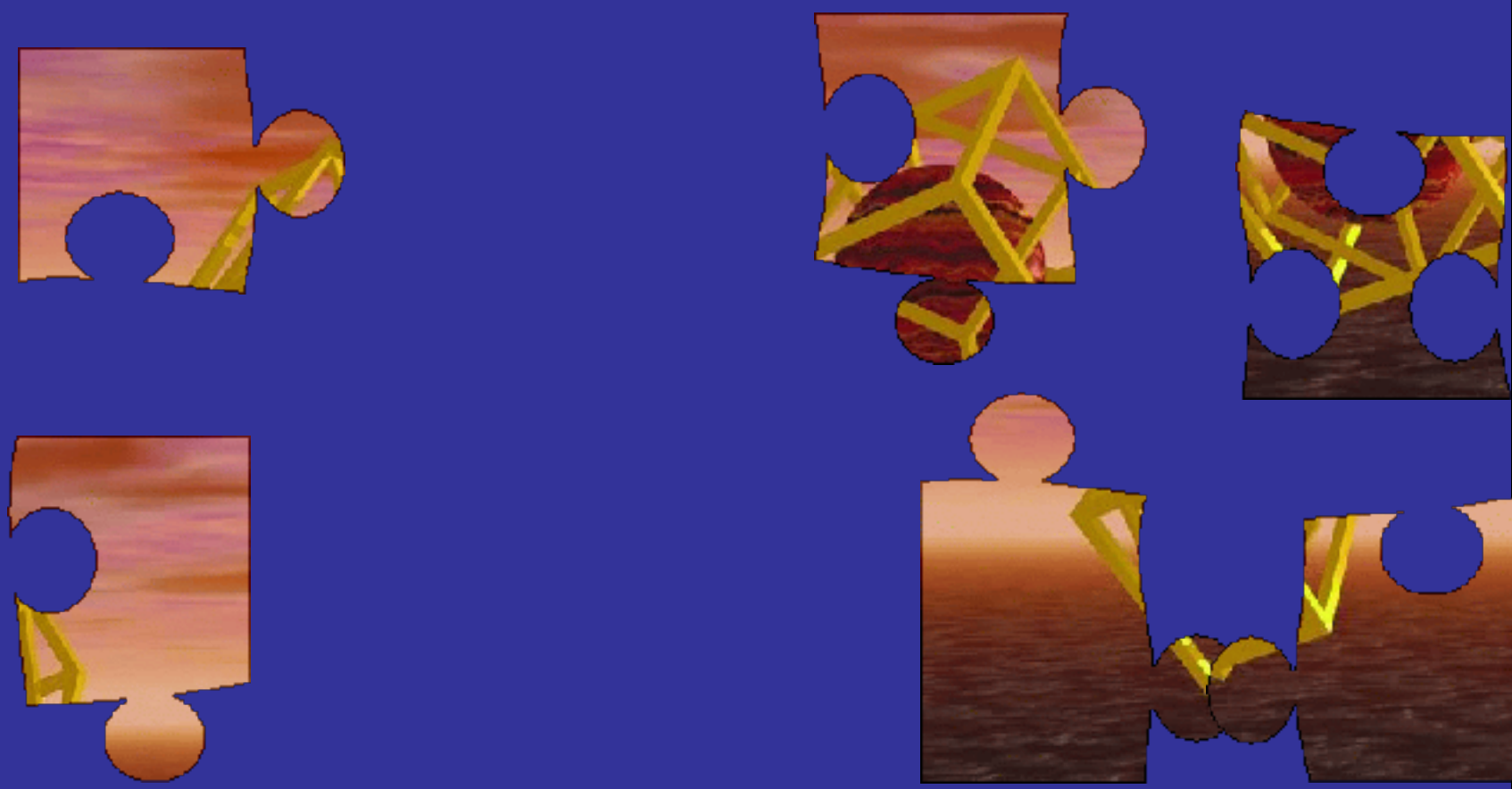
Gegeben: n Puzzleteile



Gesucht: Eine „richtige“ Anordnung der Teile

Ein algorithmisches Problem

Gegeben: n Puzzleteile



Gesucht: Eine „richtige“ Anordnung der Teile

Ein algorithmisches Problem

Gegeben: n Puzzleteile

Gesucht: Eine „richtige“ Anordnung der Teile

Ein algorithmisches Problem

Gegeben: n Puzzleteile

Gesucht: Eine „richtige“ Anordnung der Teile

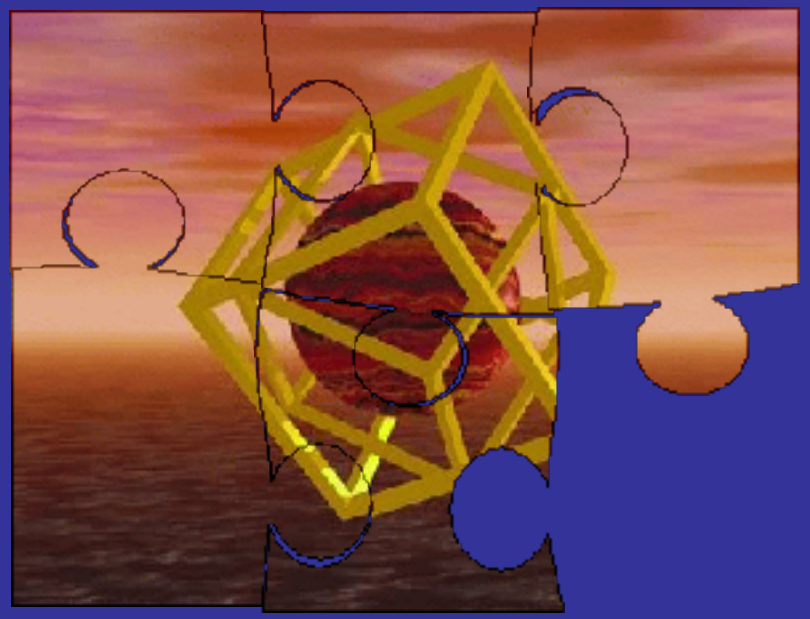
Ein algorithmisches Problem

Gegeben: n Puzzleteile

Gesucht: Eine „richtige“ Anordnung der Teile

Ein algorithmisches Problem

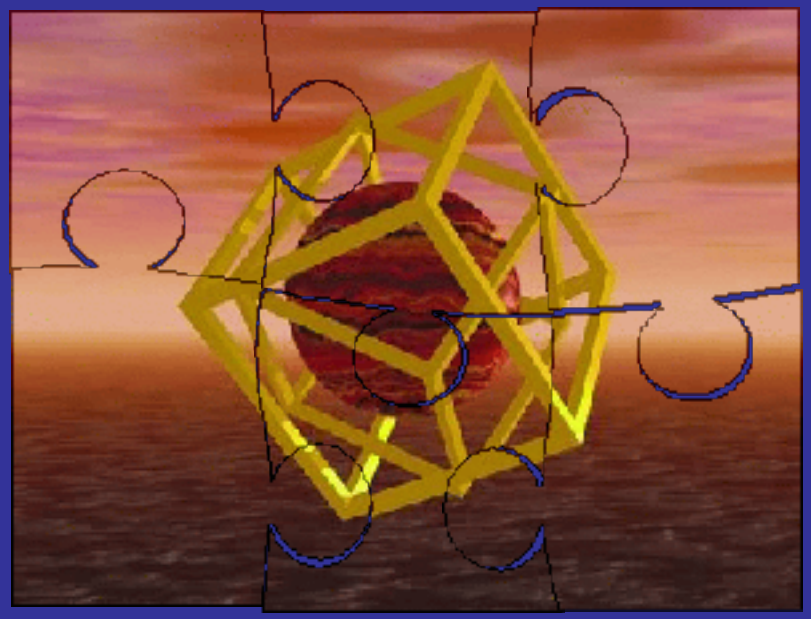
Gegeben: n Puzzleteile



Gesucht: Eine „richtige“ Anordnung der Teile

Ein algorithmisches Problem

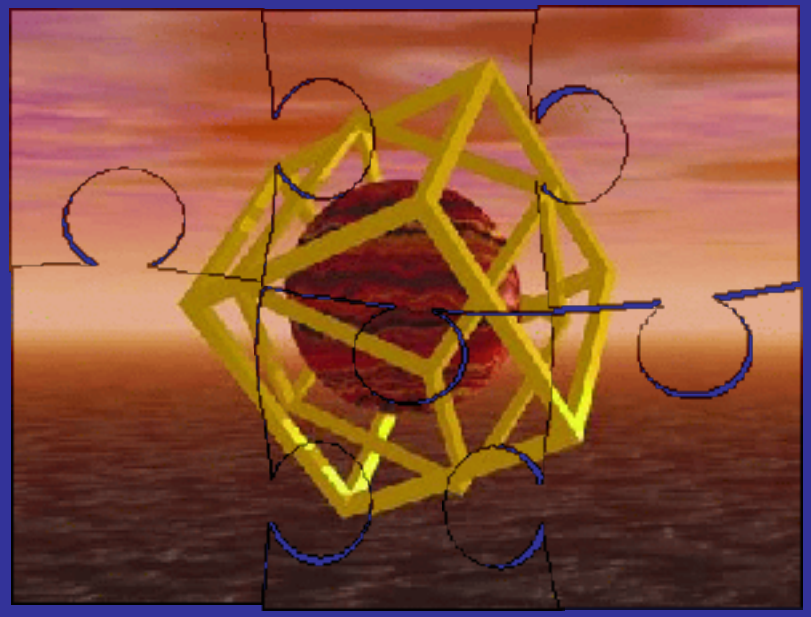
Gegeben: n Puzzleteile



Gesucht: Eine systematische Methode zum Puzzeln

Ein algorithmisches Problem

Gegeben: n Puzzleteile



Einfach so: $O(n^2)$

Für n=6: 21

Für n=100: 5.050

Für n=5000: 12.502.500

Raffiniert sortiert:

$O(n \log n)$

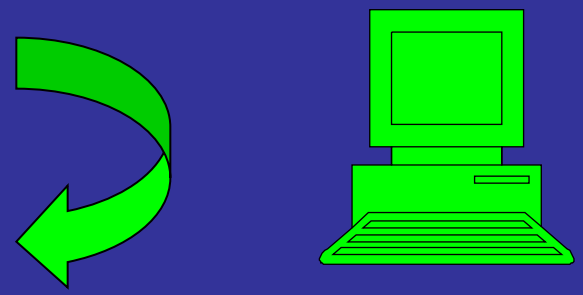
Gesucht: Eine systematische Methode zum Puzzeln

Konstruktives Lösen von Problemen

Gegeben: Eine Problembeschreibung

Gegeben: Eingabedaten

Gesucht: Ausgabedaten



Gesucht: Ein Algorithmus, der das Problem löst

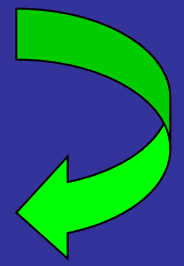
„Geht‘s nicht noch etwas schneller?“

Die Laufzeit von Algorithmen

Gegeben: Eine Problembeschreibung der Größe O(n)

Gegeben: O(n) Zahlen

Gesucht: O(n) Zahlen



$O(n^k)$

Die Klasse P

Gesucht: Ein polynomialer Algorithmus

Sortieren von Objekten

Gegeben: n Objekte unterschiedlicher Größe

23 17 13 19 33 28 15

Gesucht: Eine Sortierung nach Größe

Sortieren von Objekten

Gegeben: n Objekte unterschiedlicher Größe

13 15 17 19 23 28 33

Gesucht: Eine Sortierung nach Größe

Sortieren von Objekten

Gegeben: n Objekte unterschiedlicher Größe

13 15 17 19 23 28 33

Zahl der Vergleiche: $O(n^2)$

„Geht‘s nicht noch etwas schneller?“

Sortieren von Objekten

Gegeben: n Objekte unterschiedlicher Größe

13 15 17 19 23 28 33

Zahl der Vergleiche: $O(n \log n)$

Ein untere Laufzeitschranke

Gegeben: n Objekte in einer beliebigen Anordnung

(1) Insgesamt gibt es n*(n-1)*…*2*1 = n! mögliche Anordungen

(2) Jeder Vergleich teilt die verbleibenden Anordnungen in zwei Mengen

(3) Im schlechtesten Fall bleibt die größere Menge übrig

(4) Insgesamt benötigen wir $\log(n!) = \Omega(n \log n)$ Vergleiche.

Ein algorithmisches Problem

Gegeben: n Puzzleteile

Gesucht: Eine „richtige“ Anordnung der Teile

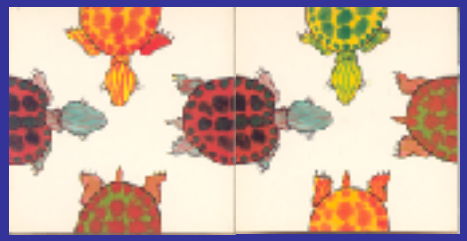
Ein algorithmisches Problem

Gegeben: n Puzzleteile

Gesucht: Eine „richtige“ Anordnung der Teile

Ein algorithmisches Problem

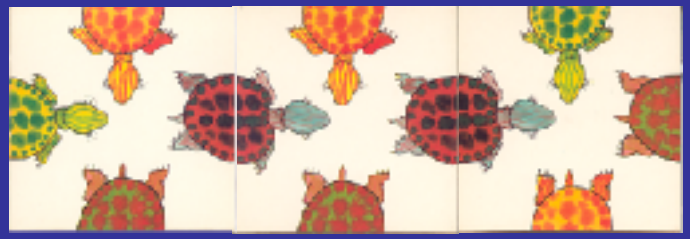
Gegeben: n Puzzleteile



Gesucht: Eine „richtige“ Anordnung der Teile

Ein algorithmisches Problem

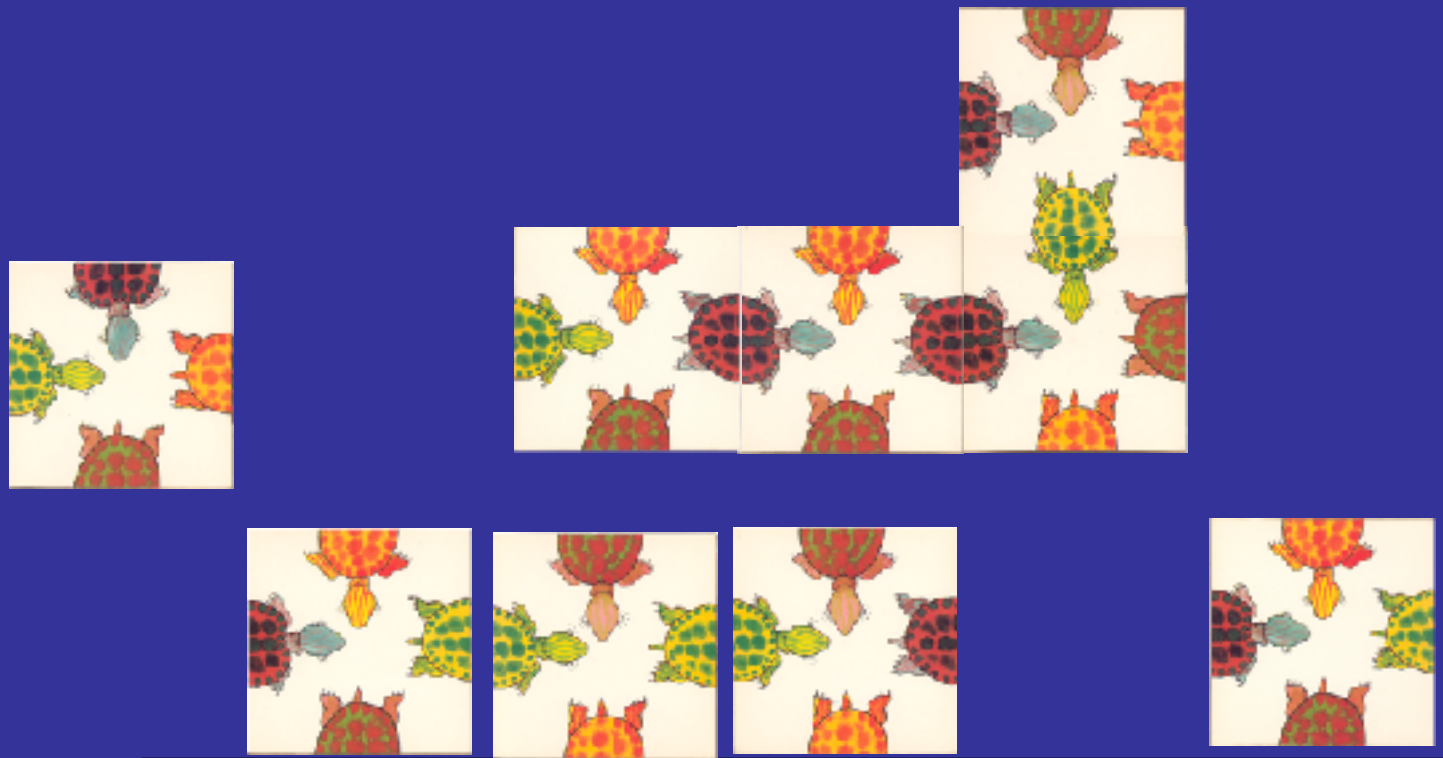
Gegeben: n Puzzleteile



Gesucht: Eine „richtige“ Anordnung der Teile

Ein algorithmisches Problem

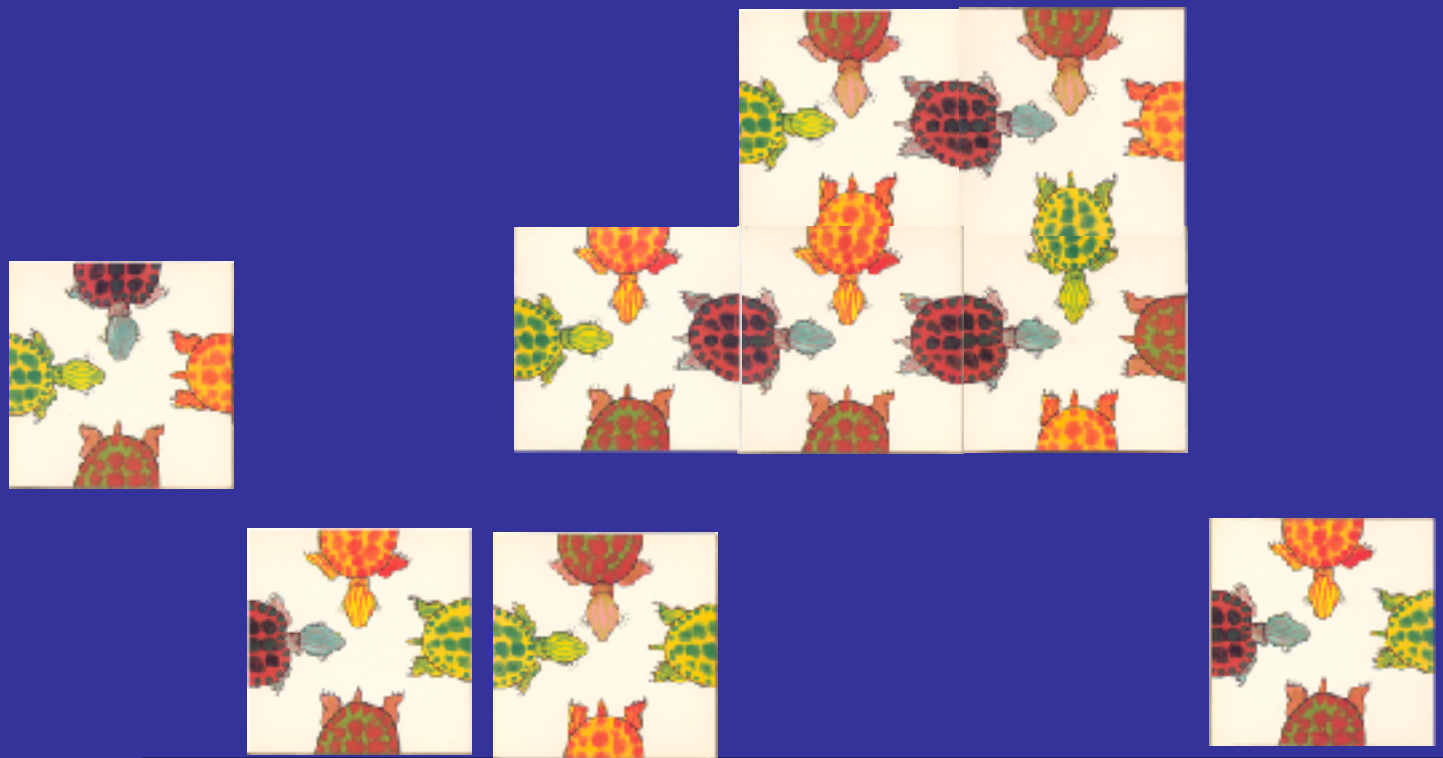
Gegeben: n Puzzleteile



Gesucht: Eine „richtige“ Anordnung der Teile

Ein algorithmisches Problem

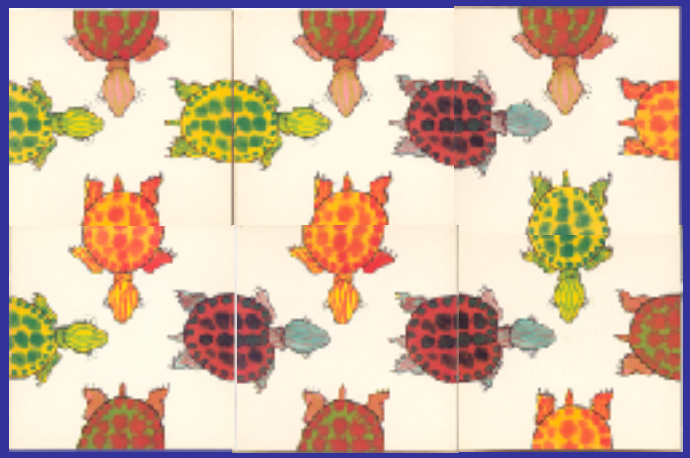
Gegeben: n Puzzleteile



Gesucht: Eine „richtige“ Anordnung der Teile

Ein algorithmisches Problem

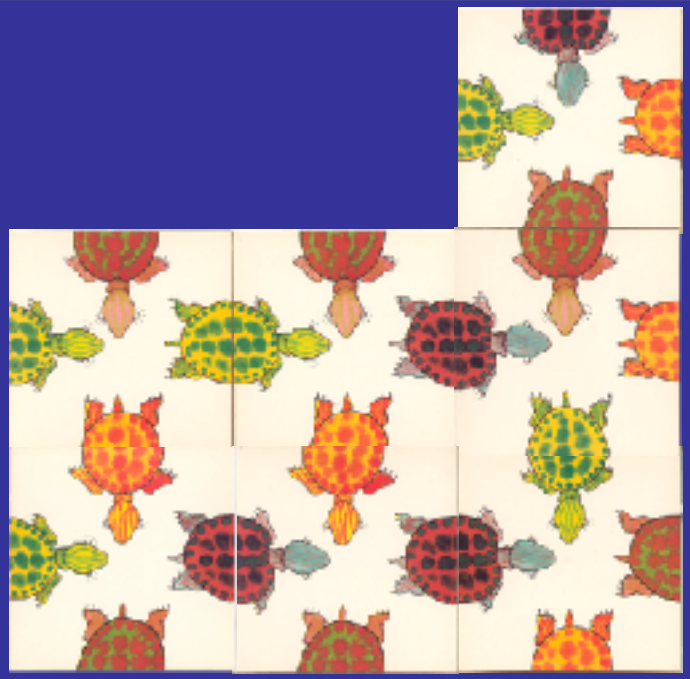
Gegeben: n Puzzleteile



Gesucht: Eine „richtige“ Anordnung der Teile

Ein algorithmisches Problem

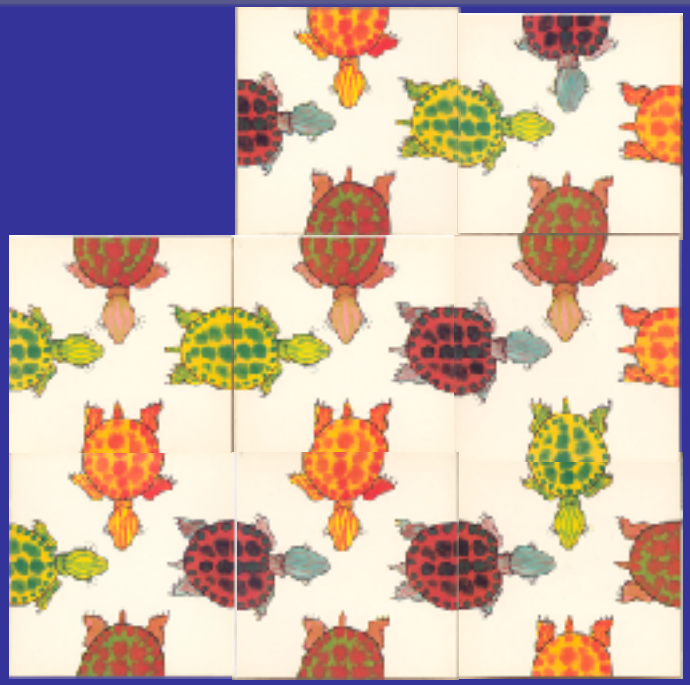
Gegeben: n Puzzleteile



Gesucht: Eine „richtige“ Anordnung der Teile

Ein algorithmisches Problem

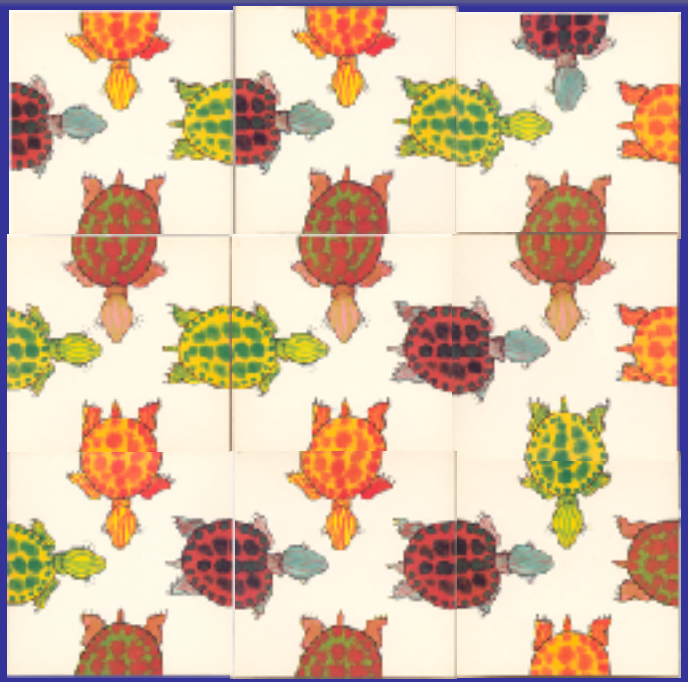
Gegeben: n Puzzleteile



Gesucht: Eine „richtige“ Anordnung der Teile

Ein algorithmisches Problem

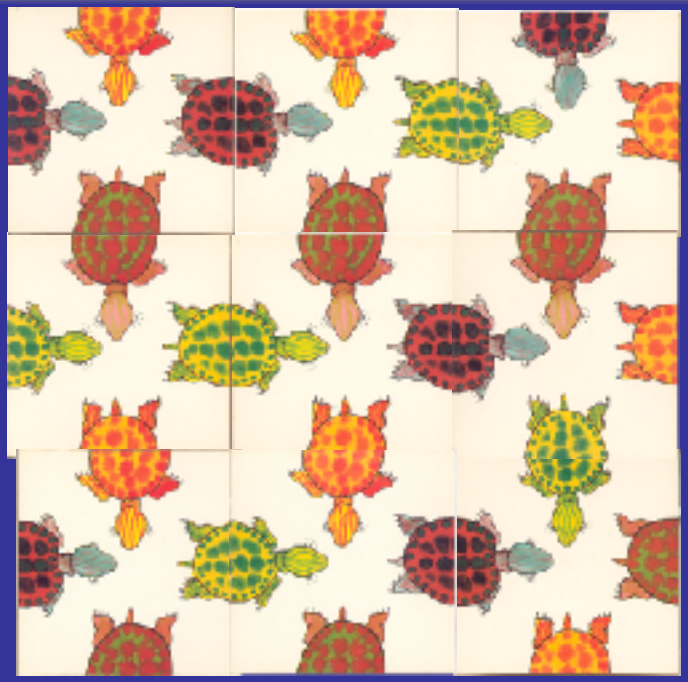
Gegeben: n Puzzleteile



Gesucht: Eine „richtige“ Anordnung der Teile

Ein algorithmisches Problem

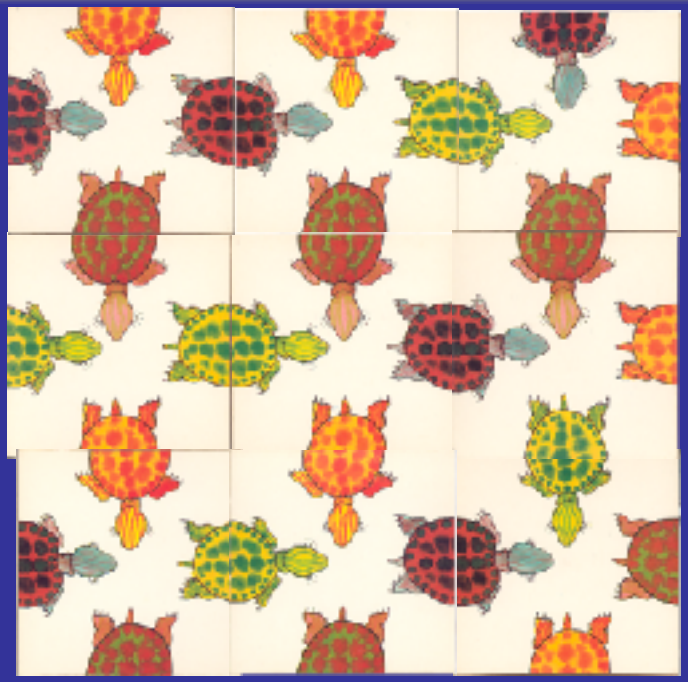
Gegeben: n Puzzleteile



Gesucht: Eine „richtige“ Anordnung der Teile

Ein algorithmisches Problem

Gegeben: n Puzzleteile



Probieren:

$$\Omega(n!4^{n-1})$$

Für n=9: 23.781.703.680

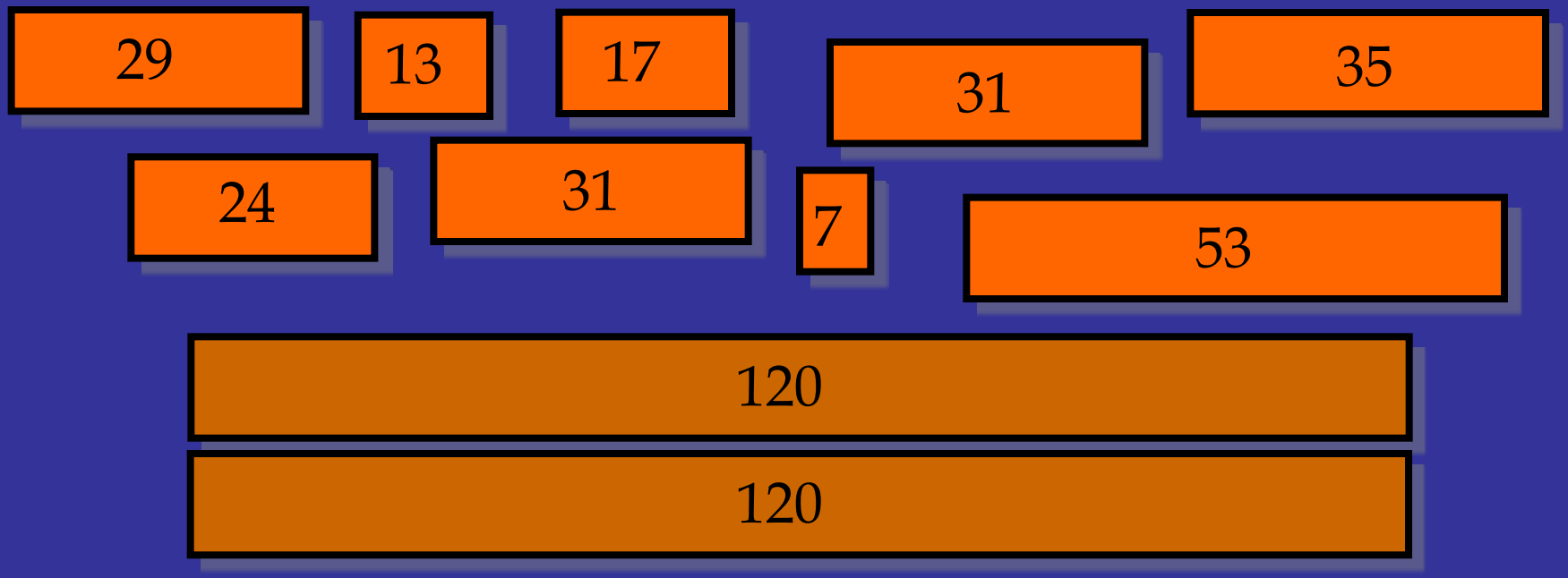
Raffiniert?!

Problem: „Sackgassen“!

Gesucht: Eine systematische Methode zum Puzzeln

Kofferpacken für die Reise

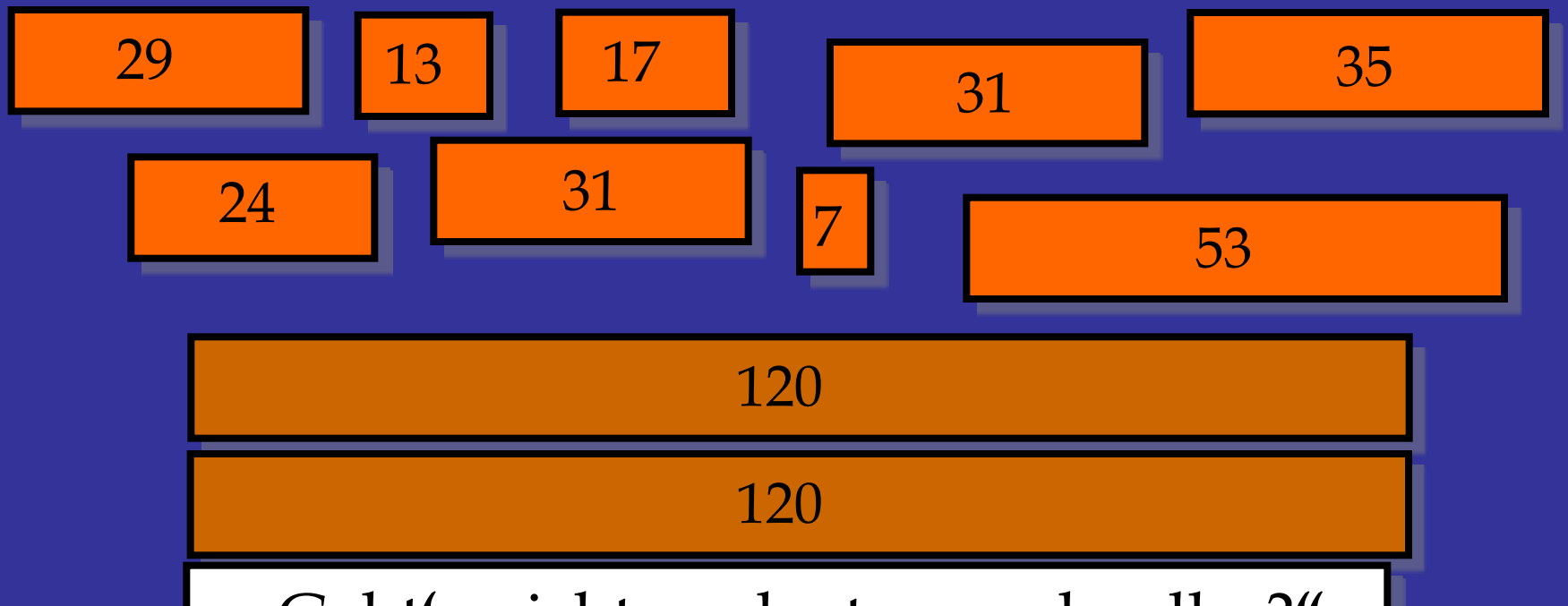
Gegeben: Eine Menge von n Objekten, jedes mit einer Größe l_i; Gesamtgröße $\sum_{i=1}^{n} l_i = 2K$



Gesucht: Eine Verteilung auf zwei Koffer der Größe K

Kofferpacken für die Reise

Gegeben: Eine Menge von n Objekten, jedes mit einer Größe l_i; Gesamtgröße $\sum_{i=1}^{n} l_i = 2K$

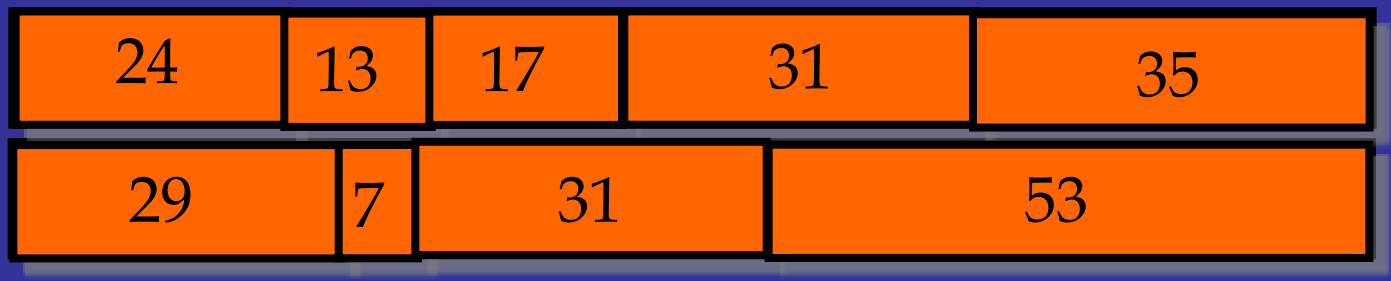


„Geht‘s nicht noch etwas schneller?“

Gesucht: Eine Verteilung auf zwei Koffer der Größe K

Kofferpacken für die Reise

Gegeben: Eine Menge von n Objekten, jedes mit einer Größe l_i; Gesamtgröße $\sum_{i=1}^{n} l_i = 2K$



Gesucht: Eine Verteilung auf zwei Koffer der Größe K

Eigenschaften des Problems

(1) Wenn es eine Lösung gibt, so lässt sie sich ziemlich schnell *NachPrüfen*.

Die Klasse NP

(2) Wenn es *keine* Lösung gibt, lässt sich das anscheinend nur schlecht *beweisen*.

Probleme, für die man pfiffig „schnell“ eine Lösung *Pfinden* kann:

Die Klasse P

(Echt pfundig - gepfundes Pfressen für Algorithmiker!)

Wo ist mein Schlüssel??

Ein wiedergefundener Schlüssel lässt sich relativ leicht identifizieren.

Wo ist mein Schlüssel??

Ein wiedergefundener Schlüssel lässt sich relativ leicht identifizieren.

Aber wie weist man nach, dass ein Schlüssel NICHT in einem Raum ist?

Wo ist mein Schlüssel??

Alltagserfahrungen:

Verlieren geht schneller als Wiederpfinden.

Ja sagen geht schneller als Nein begründen.

Fragen stellen geht schneller als Fragen beantworten.

Karl Popper (1902-1994)

Universalaussagen lassen sich nicht verifizieren, sondern nur falsifizieren.

Ein komplexitätstheoretischer Glaubenssatz

$$P \neq NP$$

Also: NachPrüfen allein ist noch lange nicht Pfindig!

(Oder Pfiffig)

Schlechte Nachrichten

Satz von Cook (1973):

Es gibt schwerste Probleme in NP, sogenannte „NP-vollständige Probleme“

Noch schlechtere Nachrichten

Ein polynomieller Algorithmus für

Rundreiseproblem

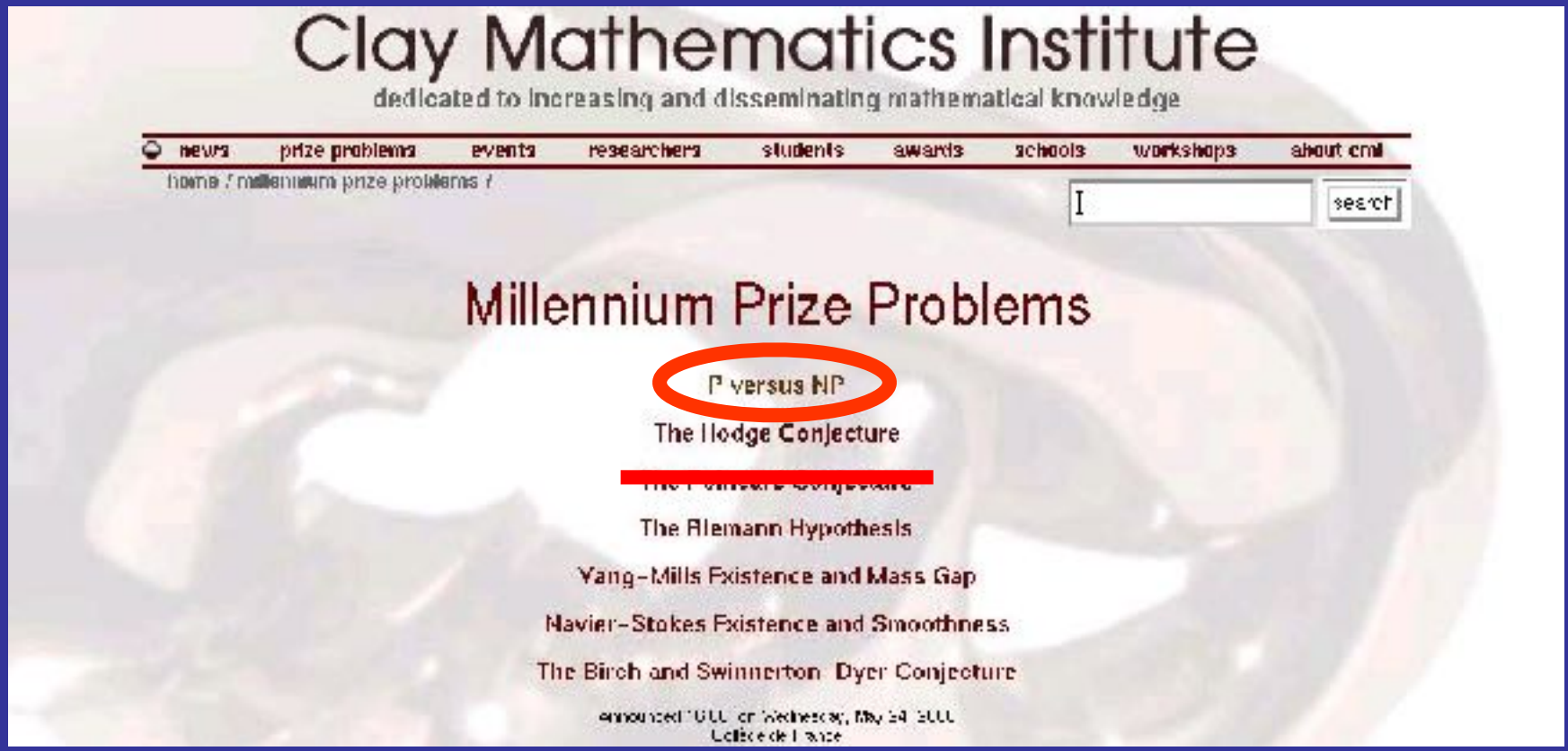
Partitionsproblem

die meisten interessanten Optimierungsprobleme

bedeutet auch schon

$$P = NP$$

Ein Milleniumproblem



Preisgeld: 1.000.000 $

Was tun bei NP-Vollständigkeit?

Idealer Algorithmus:

Liefert

1. immer
2. schnell
3. ein optimales Ergebnis.

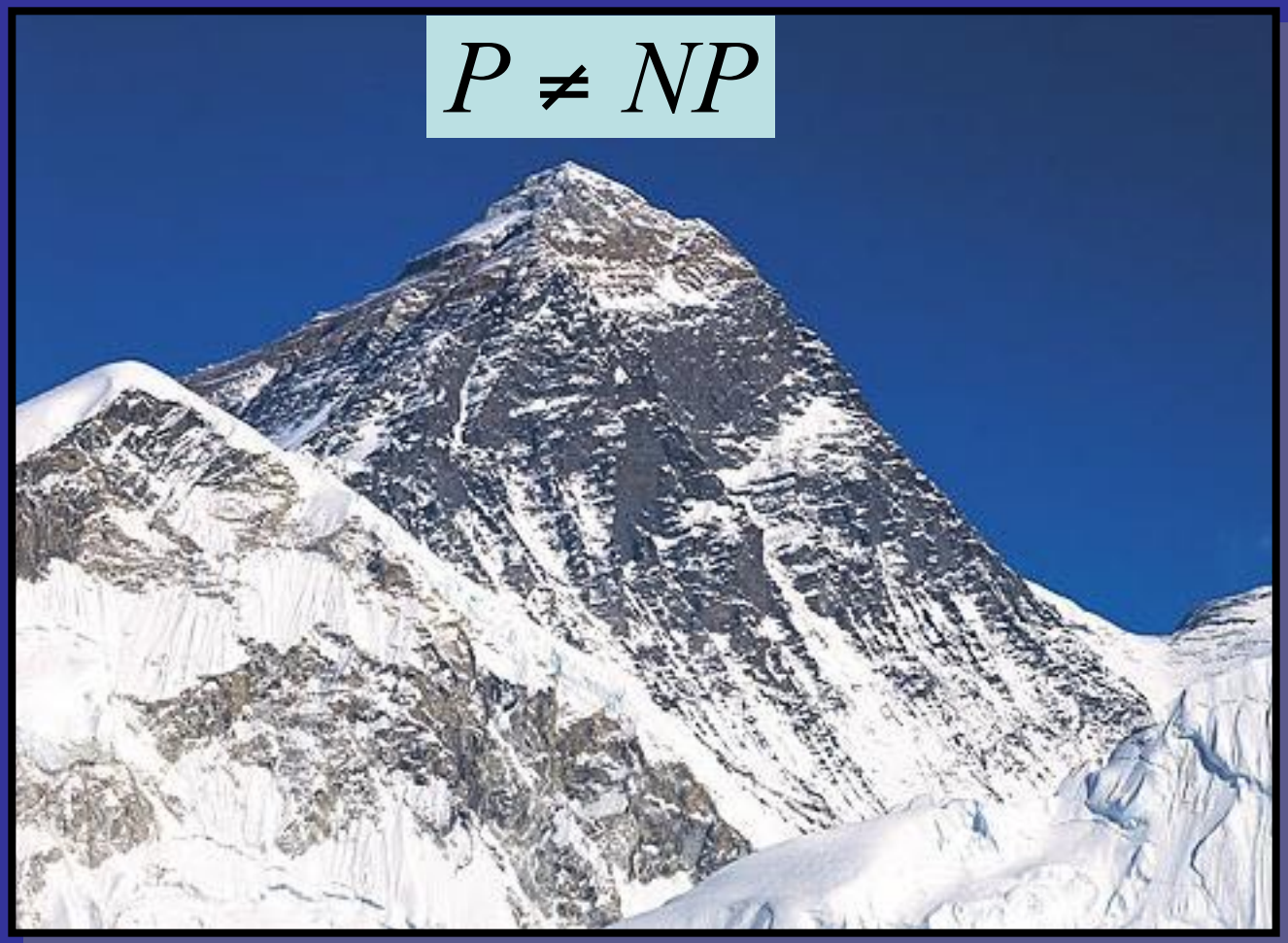
Strategien in schwierigen Situationen:

(A) Mit dem Schicksal hadern + diskutieren
(B) Auf Glück vertrauen
(C) Hart arbeiten
(D) Erwartungen zurückschrauben

Algorithmische Strategien:

(A) Komplexitätsanalyse: Gibt es Fälle, die nicht NP-vollständig sind?
(B) Heuristiken: Raten und hoffen
(C) Algorithmen-Tuning, um konkrete Instanzen optimal lösen zu können.
(D) „Gut“ statt „optimal“: Approximationsalgorithmen

Eine Idealroute zum Gipfel



Es gibt noch viele Gipfel…

Danke fürs Zuhören!