

Methoden und Anwendungen der Optimierung (MAO)

Kapitel 1: Grundlagen

Univ.-Prof. Dr. Michael Schneider
Christian Schröder

Deutsche Post Chair – Optimization of Distribution Networks (DPO)
RWTH Aachen University

schroeder@dpo.rwth-aachen.de

WS 2017/18



Deutsche Post
Chair - Optimization of
Distribution Networks

RWTHAACHEN
UNIVERSITY

Gesamtgliederung

- 1 Grundlagen**
- 2 Greedy Algorithmen**
- 3 Performancemessung**
- 4 Lokale Suche**
- 5 Metaheuristiken: Einführung, Design, Vergleich und Bewertung**

Agenda

1 Grundlagen

- Operations Research und kombinatorische Optimierung
- Algorithmen, Aufwand und Komplexitätsklassen
- Heuristiken

Agenda

1 Grundlagen

- Operations Research und kombinatorische Optimierung
- Algorithmen, Aufwand und Komplexitätsklassen
- Heuristiken

Operations Research

Operations Research (OR) (dt. "Unternehmensforschung", in den USA auch "Management Science") untersucht Wertschöpfungs- und Geschäftsprozesse in Unternehmen mit dem Ziel

- optimale Entscheidungen vorzubereiten und durchzuführen mittels quantitativer Modelle und Methoden.

ORler analysieren die "Operations" der Wertschöpfungs- und Geschäftsprozesse.

Operations Research

Operations Research (OR) (dt. "Unternehmensforschung", in den USA auch "Management Science") untersucht Wertschöpfungs- und Geschäftsprozesse in Unternehmen mit dem Ziel

- optimale Entscheidungen vorzubereiten und durchzuführen mittels quantitativer Modelle und Methoden.

ORler analysieren die "Operations" der Wertschöpfungs- und Geschäftsprozesse.

USA (INFORMS):

- "OR – the Science of Better"
- d.h. Verbesserungspotenziale aufdecken und erschließen
- Verbesserung als Ersatz für "optimale Entscheidungen"
- In a nutshell, operations research (OR) is the discipline of applying advanced analytical methods to help make better decisions.

Operations Research

Quantitative Modelle und Methoden

- sind **formale** und **mathematische** Modelle und Methoden (z.B. der Mathematik oder Informatik)
- gehen mit **quantifizierten Größen** um
- auch: **experimentelle** Methoden, bei denen gemessen, verglichen wird

Operations Research

Quantitative Modelle und Methoden

- sind **formale** und **mathematische** Modelle und Methoden (z.B. der Mathematik oder Informatik)
- gehen mit **quantifizierten Größen** um
- auch: **experimentelle** Methoden, bei denen gemessen, verglichen wird

Modell vs. Methode

- **Modell** (modellieren ~ darstellen, abbilden)
- **Methode, Verfahren, Algorithmus** (~ rechnen, → Computer)

OR Anwendungen

Beispiel Fluggesellschaften:



- **Demand Planning:** "Auf welchen Strecken sollen welche Dienstleistungen/Produkte angeboten werden?"
- **Umlaufplanung:** "Welches Flugzeug bedient welches Flugsegment und wird wo/wann gewartet?"
- **Crew Scheduling:** Dienstpläne; "Wer erledigt wann, welche Aufgaben?"
- **Disruption Management:** "Wie reagieren auf Störungen, Abweichungen und Ausfälle?"
- **Revenue Management:** Pricing, Kapazitätssteuerung ("Welche Buchungsklasse ist wie lange buchbar?") und Überbuchungssteuerung
- ...

OR Anwendungen

Beispiel KEP (Kurier, Express, Post):



- **Netzwerk-Design:** "Was ist die physische Struktur des logistischen Netzes?"; "Welche Standorte übernehmen welche Aufgaben?"; "Auf welchen Routen gelangen Sendungen vom Kunden zum Empfänger?"
- **Transportplanung:** "Welches Fahrzeug wird wann, wo und wozu eingesetzt?"
- **Standortplanung:** "Wo sollen Briefkästen stehen?", "Wie viele, wie große Depots sollen wo genutzt werden?"
- **Letzte Meile:** Bezirkseinteilung und Routing der Boten/Zusteller durch die Straßen ihres Bezirks

OR Anwendungen

Beispiel Produktionsplanung:



- Nachfrageprognose: "Wie viel kann wann von einem Produkt verkauft werden"
- Aggregierte Produktionsplanung: "Welche Rohstoffe, Vorprodukte, Maschinen, Mitarbeiter (=Produktionsfaktoren) werden wann zur Herstellung welcher Produkte(-gruppen) benötigt?"
- Maschinenbelegungsplanung: "Welcher Auftrag wird wann auf welcher Maschine bearbeitet?"
- Layoutplanung: "Wie sollen räumlich die Produktionsstätten und -anlagen angeordnet werden?"
- Verpackung und Zuschnitt: ...

OR Anwendungen

Beispiel Telekommunikation:



- **Netzwerk-Design:** "Wo werden welche Anlagen und Verbindungen eingerichtet/erweitert/verändert?"
- **Kanalzuweisung:** "Welche Zelle nutzt welche Kanäle/Frequenzen?"
- **Netzwerk-Routing:** "Bei bestehendem Netz, wo entlang werden welche Daten/Gespräche gesendet?"
- ...

OR Anwendungen

Beispiel Timetabling:



- **Kurs- und Raumplanung:**
“Welcher Lehrer/Dozent unterrichtet wann, wo, welchen Kurs?”
- **Prüfungsplanung:** “Welche Prüfung findet wann, wo statt?”
- **Sportligaplanung:** “Wer spielt an welchem Spieltag wo gegen wen?”
- **TV und Radio:** “Welche Werbung wird wann ausgestrahlt?”

Methoden des OR

Quantitative Modelle und Methoden:

■ Optimierung

- exakt (math. Optimierung) vs. heuristisch (Heuristiken und Metaheuristiken, künstliche Intelligenz)
- kontinuierlich vs. diskret
- linear vs. nicht-linear

Methoden des OR

Quantitative Modelle und Methoden:

- Optimierung
 - exakt (math. Optimierung) vs. heuristisch (Heuristiken und Metaheuristiken, künstliche Intelligenz)
 - kontinuierlich vs. diskret
 - linear vs. nicht-linear
- Warteschlangenmodelle und -theorie
- Prognosemodelle und -verfahren
 - vgl. Vorlesung Operations Management

Methoden des OR

Quantitative Modelle und Methoden:

- Optimierung
 - exakt (math. Optimierung) vs. heuristisch (Heuristiken und Metaheuristiken, künstliche Intelligenz)
 - kontinuierlich vs. diskret
 - linear vs. nicht-linear
- Warteschlangenmodelle und -theorie
- Prognosemodelle und -verfahren
 - vgl. Vorlesung Operations Management
- Simulation
- Multi-Criteria Decision Analysis/Making (MCDA, MCDM)
- Statistik und Wahrscheinlichkeitstheorie
- ...

Optimierung

Was ist “optimieren” und was ist “optimal”?

Auswahl einer Alternative (=Entscheidung, Aktion, Lösung) aus einer Menge von mindestens zwei Alternativen.

Optimierung

Was ist “optimieren” und was ist “optimal”?

Auswahl einer Alternative (=Entscheidung, Aktion, Lösung) aus einer Menge von mindestens zwei Alternativen.

Die Menge der möglichen **Alternativen** kann **gegeben** sein:

- **explizit:** A_1, A_2, \dots, A_n
 (“Wähle eine Zahl 1, 2, 3, …, 10”) oder
- **implizit:** mit Hilfe von Entscheidungsvariablen und Restriktionen
 (=Nebenbedingungen)
 (“Wähle eine ganze Zahl x mit $-100 \leq x \leq 1000$ ”)

Optimierung

Was ist “optimieren” und was ist “optimal”?

Auswahl einer Alternative (=Entscheidung, Aktion, Lösung) aus einer Menge von mindestens zwei Alternativen.

Die Menge der möglichen **Alternativen** kann **gegeben** sein:

- **explizit:** A_1, A_2, \dots, A_n
("Wähle eine Zahl 1, 2, 3, ..., 10") oder
- **implizit:** mit Hilfe von Entscheidungsvariablen und Restriktionen
("Nebenbedingungen")
("Wähle eine ganze Zahl x mit $-100 \leq x \leq 1000$ ")

Die möglichen **Alternativen** sind **bewertet** gemäß:

- **zulässig** oder **unzulässig**,
- **optimal**, d.h. **zulässig** und **bestmöglich**
(hinsichtlich einer reellen Bewertungsfunktion/Zielfunktion)

Optimierung

Optimierungsmodell mit Entscheidungsvariablen

Gegeben sind:



- $X \subset \mathbb{R}^n$: Menge der **zulässigen Lösungen**
(gebildet mit Hilfe von Nebenbedingungen)
- $z : X \rightarrow \mathbb{R}^1, x \mapsto z(x)$: **Zielfunktion**
- mit Optimierungsrichtung "min" oder "max"
 - Minimierung: z.B. Kosten, Projektdauer
 - Maximierung: z.B. Gewinn, Auslastung, Servicequalität

Dann heißt $(X, z(x), \text{min})$ bzw. $(X, z(x), \text{max})$
Optimierungsproblem.

Optimierung

Beispiel: Ein Spediteur fährt täglich eine sog. Rahmentour $A \rightarrow B \rightarrow C \rightarrow D$.
Ihm liegen 5 Anfragen für FTL-Transporte auf bestimmten Teilstrecken vor:

1. $A \rightarrow B$ mit Profit 1
2. $A \rightarrow C$ mit Profit 5
3. $B \rightarrow C$ mit Profit 3
4. $B \rightarrow D$ mit Profit 4
5. $C \rightarrow D$ mit Profit 2

Welche Transportaufträge soll er annehmen?

Optimierung

Beispiel: Ein Spediteur fährt täglich eine sog. Rahmentour $A \rightarrow B \rightarrow C \rightarrow D$.
Ihm liegen 5 Anfragen für FTL-Transporte auf bestimmten Teilstrecken vor:

1. $A \rightarrow B$ mit Profit 1
2. $A \rightarrow C$ mit Profit 5
3. $B \rightarrow C$ mit Profit 3
4. $B \rightarrow D$ mit Profit 4
5. $C \rightarrow D$ mit Profit 2

Welche Transportaufträge soll er annehmen?

Explizit: $X = \{\emptyset, 1, 2, 3, 4, 5, 1\&3, 1\&4, 1\&5, 2\&5, 3\&5, 1\&3\&5\}$

Optimierung

Beispiel: Ein Spediteur fährt täglich eine sog. Rahmentour $A \rightarrow B \rightarrow C \rightarrow D$.
Ihm liegen 5 Anfragen für FTL-Transporte auf bestimmten Teilstrecken vor:

1. $A \rightarrow B$ mit Profit 1
2. $A \rightarrow C$ mit Profit 5
3. $B \rightarrow C$ mit Profit 3
4. $B \rightarrow D$ mit Profit 4
5. $C \rightarrow D$ mit Profit 2

Welche Transportaufträge soll er annehmen?

Explizit: $X = \{\emptyset, 1, 2, 3, 4, 5, 1\&3, 1\&4, 1\&5, 2\&5, 3\&5, 1\&3\&5\}$

Implizit:

$$\max 1 \cdot x_1 + 5 \cdot x_2 + 3 \cdot x_3 + 4 \cdot x_4 + 2 \cdot x_5$$

so dass $x_1, x_2, x_3, x_4, x_5 \in \{0, 1\}$ (Auftrag annehmen: nein/ja)

$x_1 + x_2 \leq 1$ (nur ein Auftrag auf Abschnitt $A \rightarrow B$)

$x_2 + x_3 + x_4 \leq 1$ (nur ein Auftrag auf Abschnitt $B \rightarrow C$)

$x_4 + x_5 \leq 1$ (nur ein Auftrag auf Abschnitt $C \rightarrow D$)

Optimierung

Beispiel: Ein Spediteur fährt täglich eine sog. Rahmentour $A \rightarrow B \rightarrow C \rightarrow D$.
Ihm liegen 5 Anfragen für FTL-Transporte auf bestimmten Teilstrecken vor:

1. $A \rightarrow B$ mit Profit 1
2. $A \rightarrow C$ mit Profit 5
3. $B \rightarrow C$ mit Profit 3
4. $B \rightarrow D$ mit Profit 4
5. $C \rightarrow D$ mit Profit 2

Welche Transportaufträge soll er annehmen?

Explizit: $X = \{\emptyset, 1, 2, 3, 4, 5, 1\&3, 1\&4, 1\&5, 2\&5, 3\&5, 1\&3\&5\}$

Implizit:

$$\max 1 \cdot x_1 + 5 \cdot x_2 + 3 \cdot x_3 + 4 \cdot x_4 + 2 \cdot x_5$$

so dass $x_1, x_2, x_3, x_4, x_5 \in \{0, 1\}$ (Auftrag annehmen: nein/ja)

$x_1 + x_2 \leq 1$ (nur ein Auftrag auf Abschnitt $A \rightarrow B$)

$x_2 + x_3 + x_4 \leq 1$ (nur ein Auftrag auf Abschnitt $B \rightarrow C$)

$x_4 + x_5 \leq 1$ (nur ein Auftrag auf Abschnitt $C \rightarrow D$)

Lösung?

Optimierung

Schreibweise für Optimierungsprobleme

$$z^* = \min_{x \in X} z(x) \quad \left(\text{ bzw. } z^* = \max_{x \in X} z(x) \right)$$



bedeutet: bestimme eine(!) zulässige Lösung $x^* \in X$ mit
 $z(x^*) \leq z(x)$ für alle $x \in X$ (bzw. bei Maximierung $z(x^*) \geq z(x)$
für alle $x \in X$).
 x^* heißt **optimale Entscheidung/Alternative/Lösung**.

Optimierung

Schreibweise für Optimierungsprobleme

$$z^* = \min_{x \in X} z(x) \quad \left(\text{ bzw. } z^* = \max_{x \in X} z(x) \right)$$

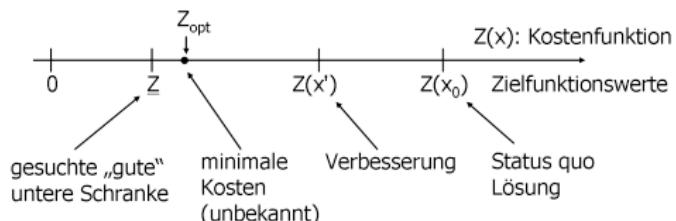


bedeutet: bestimme eine(!) zulässige Lösung $x^* \in X$ mit $z(x^*) \leq z(x)$ für alle $x \in X$ (bzw. bei Maximierung $z(x^*) \geq z(x)$ für alle $x \in X$).
 x^* heißt **optimale Entscheidung/Alternative/Lösung**.

Bemerkungen:

- 1 Man beachte die Priorität der Restriktionen gegenüber der Zielfunktion, d.h. nur für zulässige $x \in X$ wird die Zielfunktion untersucht. **Notwendig für Optimalität ist Zulässigkeit!**
- 2 Abstrahiert man von den konkreten Daten/Zahlen, die die Zielfunktion und die Nebenbedingungen beschreiben, so erhält man eine **Klasse/einen Typ von Optimierungsproblem**, kurz: ein **Problem**. Die konkrete Ausprägung des Problems heißt **Instanz** des Problems.

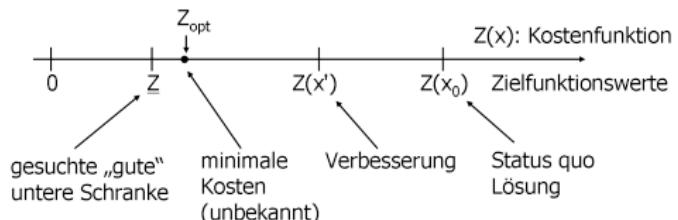
Optimierungsbegriff der “Praxis”



Bekannt sei eine aktuelle Lösung x_0 (Status Quo) für ein Minimierungsproblem und der zugehörige Zielfunktionswert $z(x_0)$.

beachte: Oft sind in der Praxis weder x_0 noch $z(x_0)$ also z.B. die Kosten der gegenwärtigen Lösung (genau) bekannt!

Optimierungsbegriff der “Praxis”

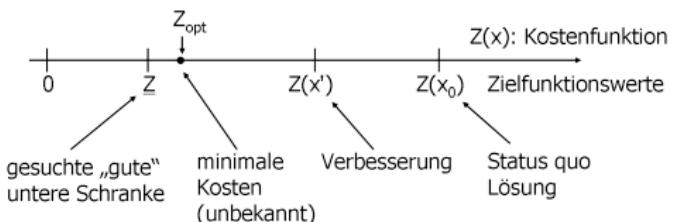


Bekannt sei eine aktuelle Lösung x_0 (Status Quo) für ein Minimierungsproblem und der zugehörige Zielfunktionswert $z(x_0)$.

beachte: Oft sind in der Praxis weder x_0 noch $z(x_0)$ also z.B. die Kosten der gegenwärtigen Lösung (genau) bekannt!

Gesucht ist ein zulässiges $x' \in X$ mit $z(x') < z(x_0)$, also eine **verbessernde Lösung**, eine Lösung besser als der Status Quo.

Optimierungsbegriff der "Praxis"



Bekannt sei eine aktuelle Lösung x_0 (Status Quo) für ein Minimierungsproblem und der zugehörige Zielfunktionswert $z(x_0)$.

beachte: Oft sind in der Praxis weder x_0 noch $z(x_0)$ also z.B. die Kosten der gegenwärtigen Lösung (genau) bekannt!

Gesucht ist ein zulässiges $x' \in X$ mit $z(x') < z(x_0)$, also eine **verbessernde Lösung**, eine Lösung besser als der Status Quo.

⇒ das Finden einer solchen (guten) Lösung x'
nennt man in der Praxis "Optimierung"

Optimierungsbegriff der “Praxis”

Nachteil: Man weiß nicht,

- wie weit man vom tatsächlichen Optimum entfernt ist,
- auf welches Potential man verzichtet, wenn man sich mit x' und $z(x')$ zufrieden gibt.

Optimierungsbegriff der “Praxis”

Nachteil: Man weiß nicht,

- wie weit man vom tatsächlichen Optimum entfernt ist,
- auf welches Potential man verzichtet, wenn man sich mit x' und $z(x')$ zufrieden gibt.

Minimale Kosten $z_{opt} = z(x^*)$ seien unbekannt.

Optimierungsbegriff der “Praxis”

Nachteil: Man weiß nicht,

- wie weit man vom tatsächlichen Optimum entfernt ist,
- auf welches Potential man verzichtet, wenn man sich mit x' und $z(x')$ zufrieden gibt.

Minimale Kosten $z_{opt} = z(x^*)$ seien unbekannt.

Vorgehen:

- Man sucht eine untere Schranke \underline{z} für die Zielfunktion (Kosten),
- d.h. $\underline{z} \leq z(x)$ für alle $x \in X$
- insbesondere also auch $\underline{z} \leq z(x')$ und $\underline{z} \leq z_{opt}$

Optimierungsbegriff der “Praxis”

Nachteil: Man weiß nicht,

- wie weit man vom tatsächlichen Optimum entfernt ist,
- auf welches Potential man verzichtet, wenn man sich mit x' und $z(x')$ zufrieden gibt.

Minimale Kosten $z_{opt} = z(x^*)$ seien unbekannt.

Vorgehen:

- Man sucht eine untere Schranke \underline{z} für die Zielfunktion (Kosten),
- d.h. $\underline{z} \leq z(x)$ für alle $x \in X$
- insbesondere also auch $\underline{z} \leq z(x')$ und $\underline{z} \leq z_{opt}$

Kennt man \underline{z} , dann gibt $z(x') - \underline{z}$ den **maximalen Abstand zum unbekannten Kostenminimum** an: noch vorhandenes maximales Kostensenkungspotential.

Optimierung – State of the Art

State of the Art: Die meisten interessanten Problemstellungen sind “praktisch” nicht beweisbar optimal lösbar (\rightarrow Komplexitätsklasse \mathcal{NP}).

Daher:

- ① gute untere Schranken bestimmen mit $z_{opt} - z$ “klein”
- ② mit Heuristiken gute zulässige Lösungen $x' \in X$ bestimmen (liefert obere Schranke) mit $z(x') - z_{opt}$ “klein”

Optimierung – State of the Art

State of the Art: Die meisten interessanten Problemstellungen sind “praktisch” nicht beweisbar optimal lösbar
(→Komplexitätsklasse \mathcal{NP}).

Daher:

- ① gute untere Schranken bestimmen mit $z_{opt} - \underline{z}$ “klein”
- ② mit Heuristiken gute zulässige Lösungen $x' \in X$ bestimmen
(liefert obere Schranke) mit $z(x') - z_{opt}$ “klein”

Dann lässt sich der maximale relative Fehler abschätzen:

$$\frac{z(x') - z_{opt}}{z(x')} \cdot 100\% \leq \frac{z(x') - \underline{z}}{z(x')} \cdot 100\%$$

Simulation

Simulation (Definition)

Simulation ist die Durchführung von Experimenten anhand von Modellen.

Simulation

Simulation (Definition)

Simulation ist die Durchführung von Experimenten anhand von Modellen.

Logistikplanung



Innerbetrieblich: Analyse von Materialflüssen und Produktionsprozessen

Distribution: Analyse von Transport-, Umschlag-, und Lagerungsprozessen, ...

Verkehrsfluss-simulation



Straßenbau;
Gestaltung von
Fahrspuren;
Ampelsteuerung

Flugsimulator



Allgemein:
Schulung von
Mitarbeitern und
des Managements

Simulation

“Simulation ist keine Optimierung!”



- Simulation zeigt nur die Konsequenzen bestimmter Entscheidungen (=Szenarios, Alternativen, Parameter) auf
- Simulation macht keinen Entscheidungsvorschlag
- Simulation gestattet nur den Vergleich von (zuvor gewählten) Alternativen

Warteschlangentheorie

(Warteschlangentheorie) (Definition)

Die *Warteschlangentheorie* ist die wissenschaftliche Disziplin, die sich mit der mathematischen Analyse von Systemen, in denen Aufträge (=User,Clients) von Bedienstationen (=Server) bearbeitet werden, beschäftigt.

Warteschlangentheorie

(Warteschlangentheorie) (Definition)

Die *Warteschlangentheorie* ist die wissenschaftliche Disziplin, die sich mit der mathematischen Analyse von Systemen, in denen Aufträge (=User,Clients) von Bedienstationen (=Server) bearbeitet werden, beschäftigt.

Typische Fragen:

- Was ist die durchschnittliche **Wartezeit** oder **Verweilzeit** von Aufträgen im System?
- Was ist die durchschnittliche **Länge der Warteschlange** und wie viele **Aufträge** sind durchschnittlich im System?
- Was ist die durchschnittliche **Auslastung der Bedienstation(en)**?

Beispiele Warteschlangentheorie

Call Center



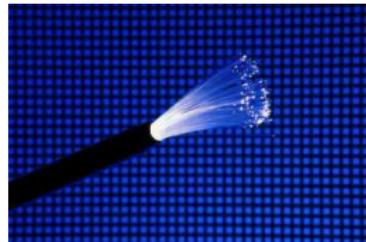
Telefonanruf ist
Auftrag;
Agent ist Server

Produktion



Fertigungsauftrag wird
von Maschine oder
Mitarbeiter bearbeitet

IT und TK



Datenpaket ist
Auftrag
Leitung/Kanal,
Router, Switch etc.
sind Server

Lineare gemischt-ganzzahlige Programmierungsprobleme

Lineare gemischt-ganzzahlige Programmierungsprobleme

(engl.: (linear) mixed integer programming problems MIP) lassen sich allgemein durch die Angabe zweier Mengen von Entscheidungsvariablen (EV), den

- **reellen** Variablen und
- **ganzzahligen** Variablen

sowie

- linearen Restriktionen und
- einer linearen Zielfunktion

definieren.

Lineare gemischt-ganzzahlige Programmierungsprobleme

Lineare gemischt-ganzzahlige Programmierungsprobleme

(engl.: (linear) mixed integer programming problems MIP) lassen sich allgemein durch die Angabe zweier Mengen von Entscheidungsvariablen (EV), den

- **reellen** Variablen und
- **ganzzahligen** Variablen

sowie

- linearen Restriktionen und
- einer linearen Zielfunktion

definieren. Die reellen Variablen können beliebige Werte in \mathbb{R} annehmen. Aus mathematischen Gründen beschränkt man sich i.Allg. auf Werte aus der Menge der rationalen Zahlen \mathbb{Q} .

Lineare gemischt-ganzzahlige Programmierungsprobleme

Lineare gemischt-ganzzahlige Programmierungsprobleme

(engl.: (linear) mixed integer programming problems MIP) lassen sich allgemein durch die Angabe zweier Mengen von Entscheidungsvariablen (EV), den

- **reellen** Variablen und
- **ganzzahligen** Variablen

sowie

- linearen Restriktionen und
- einer linearen Zielfunktion

definieren. Die reellen Variablen können beliebige Werte in \mathbb{R} annehmen. Aus mathematischen Gründen beschränkt man sich i.Allg. auf Werte aus der Menge der rationalen Zahlen \mathbb{Q} . Der Wertebereich der ganzzahligen Variablen ist auf die Menge der ganzen Zahlen $\mathbb{Z} = \{\dots, -2, -1, 0, 1, 2, 3, \dots\}$ beschränkt. Die nicht-negativen natürlichen Zahlen seien mit $\mathbb{Z}_+ = \{0, 1, 2, 3, \dots\}$ und die nicht-negativen reellen Zahlen mit \mathbb{R}_+ bezeichnet.

Lineare gemischt-ganzzahlige Programmierungsprobleme

Ganzzahlige und binäre Entscheidungsvariablen werden aus folgenden Gründen zur Modellierung benötigt:

- 1 Die Größen, die sie beschreiben, kommen in der Realität ausschließlich ganzzahlig vor und eine Approximation durch eine reelle Größe stellt eine nicht-zulässige Vereinfachung dar.
- 2 Binäre Variablen beschreiben oftmals das Vorhandensein oder Nicht-Vorhandensein einer Größe oder Eigenschaft (vorhanden/nicht vorhanden; ja/nein; an/aus etc.). Graduelle Abstufungen sind in vielen Anwendungsfällen nicht möglich.

Lineare gemischt-ganzzahlige Programmierungsprobleme

Wir gehen davon aus, dass wir ein Modell formuliert haben mit

- n reellen Variablen; jede Einheit der reellen Entscheidungsvariablen x_i bringe einen Gewinn von c_i
- p ganzzahligen Variablen; jede Einheit der ganzzahligen Entscheidungsvariablen y_j bringe einen Gewinn von h_j

Lineare gemischt-ganzzahlige Programmierungsprobleme

Wir gehen davon aus, dass wir ein Modell formuliert haben mit

- n reellen Variablen; jede Einheit der reellen Entscheidungsvariablen x_i bringe einen Gewinn von c_i
- p ganzzahligen Variablen; jede Einheit der ganzzahligen Entscheidungsvariablen y_j bringe einen Gewinn von h_j

Ferner werden m Restriktionen als Ungleichungen oder Gleichungen mit *rechter Seite* b_k betrachtet. Die Koeffizienten der Variablen x_i und y_j in der k -ten Restriktion seien jeweils a_{ki} und g_{kj} . Das Modell lässt sich folgendermaßen formal definieren:

$$(1) \quad z_{MIP} = \max \sum_{i=1}^n c_i x_i + \sum_{j=1}^p h_j y_j$$

$$(2) \quad \text{so dass } \sum_{i=1}^n a_{ki} x_i + \sum_{j=1}^p g_{kj} y_j \leq b_k \quad \text{für } k = 1, \dots, m$$

$$(3) \quad x_i \in \mathbb{R} \text{ für } i = 1, \dots, n$$

$$(4) \quad y_j \in \mathbb{Z} \text{ für } j = 1, \dots, p$$

Lineare gemischt-ganzzahlige Programmierungsprobleme

Oft fordert man, dass die Entscheidungsvariablen *nicht-negativ* sein sollen. Dann ersetzt man die Restriktionen (3) und (4) durch

$$(5) \quad x_i \in \mathbb{R}_+ \quad \text{für } i = 1, \dots, n$$

$$(6) \quad y_j \in \mathbb{Z}_+ \quad \text{für } j = 1, \dots, p.$$

Lineare gemischt-ganzzahlige Programmierungsprobleme

Es ist üblich, die etwas umständliche Schreibweise mit Koeffizienten und Summen durch eine Vektor-Schreibweise zu ersetzen. Dazu definiert man die Spaltenvektoren

- der **Entscheidungsvariablen** $x = (x_1, \dots, x_n)^\top$ und
 $y = (y_1, \dots, y_p)^\top$
- der **Zielfunktionskoeffizienten** $c = (c_1, \dots, c_n)^\top$ und
 $h = (h_1, \dots, h_p)^\top$
- und den **Kapazitätsvektor** oder sog. "rechte Seite"
 $b = (b_1, \dots, b_m)^\top$

Ferner benötigt man zwei **Koeffizientenmatrizen** die $(m \times n)$ -Matrix A und die $(m \times p)$ -Matrix G . Dann ist Problem (1), (2), (5) und (6) äquivalent zu

$$(1) \quad z_{MIP} = \max c^T x + h^T y$$

$$(2) \quad \text{so dass} \quad Ax + Gy \leq b$$

$$(5) \quad x \geq 0$$

$$(6) \quad y \geq 0 \text{ und ganzzahlig}$$

Lineare gemischt-ganzzahlige Programmierungsprobleme

Bemerkungen:

- Die Relationen \leq und \geq werden dabei komponentenweise gelesen. Dies gilt ebenfalls für die Nichtnegativitäts- und Ganzzahligkeitsbedingungen (3) und (6), in denen die rechte Seite einen Nullvektor darstellt.
- Im Folgenden wird vereinbart, dass 0 und 1 als Vektoren mit einer passenden Dimension gelesen werden, wenn die Modelle in Vektor-Schreibweise formuliert werden.
- Eine *Instanz* des Problems MIP wird eindeutig durch die Daten (A, G, b, c, h) definiert, also durch die Koeffizientenmatrizen, die Zielfunktionsvektoren und die rechte Seite.

Lineare gemischt-ganzzahlige Programmierungsprobleme

Die Menge

$$X = \{(x, y) : x \in \mathbb{R}^n, y \in \mathbb{Z}^p, x \geq 0, y \geq 0, Ax + Gy \leq b\}$$

heißt **Zulässigkeitsbereich** der Instanz des Problems MIP.

- Ein Vektor $(x, y) \in X$ heißt *zulässige Lösung* oder einfach *Lösung*.
- Ist die Menge X leer, so ist die Instanz *unzulässig*.

Lineare gemischt-ganzzahlige Programmierungsprobleme

Hat das Modell ausschließlich **ganzzahlige** Variablen, so erhält man ein **ganzzahliges Programmierungsproblem** (engl.: integer program, IP). Das Modell lautet:

$$(1) \quad z_{IP} = \max h^T y$$

$$(2) \quad \text{so dass} \quad Gy \leq b$$

$$(3) \quad y \geq 0 \text{ und ganzzahlig}$$

Lineare gemischt-ganzzahlige Programmierungsprobleme

Hat das Modell ausschließlich **ganzzahlige** Variablen, so erhält man ein **ganzzahliges Programmierungsproblem** (engl.: integer program, IP). Das Modell lautet:

$$(1) \quad z_{IP} = \max h^T y$$

$$(2) \quad \text{so dass} \quad Gy \leq b$$

$$(3) \quad y \geq 0 \text{ und ganzzahlig}$$

Sehr oft werden die ganzzahligen Variablen zur Modellierung logischer Zusammenhänge verwendet. Dann nehmen sie nur die **binären** Werte 0 und 1 an. In diesen Fällen spricht man von **binären Programmierungsproblemen** (engl.: binary program, BP) und ersetzt sinngemäß die Ganzzahligkeitsbedingung (3) durch

$$(3) \quad y \in \{0, 1\}^P.$$

Kombinatorische Optimierungsprobleme

Instanz eines kombinatorischen Optimierungsproblems (Definition)

Eine *Instanz eines kombinatorischen Optimierungsproblems* ist gegeben durch *einen endlichen zulässigen Bereich X* und *eine Zielfunktion $c : X \rightarrow \mathbb{R}$, $x \mapsto c(x)$* :

$$z = \min c(x) \quad \text{so dass} \quad x \in X$$

Kombinatorische Optimierungsprobleme

Instanz eines kombinatorischen Optimierungsproblems (Definition)

Eine *Instanz eines kombinatorischen Optimierungsproblems* ist gegeben durch *einen endlichen zulässigen Bereich X* und *eine Zielfunktion $c : X \rightarrow \mathbb{R}, x \mapsto c(x)$* :

$$z = \min c(x) \quad \text{so dass} \quad x \in X$$

kombinatorisches Optimierungsproblem (Definition)

Ein *kombinatorisches Optimierungsproblem Π* ist durch eine Menge von Instanzen $P \in \Pi$ definiert. Jede Instanz $P = (X_P, c_P)$ besteht aus einer endlichen Menge X_P von zulässigen Lösungen und einer Zielfunktion $c_P : X_P \rightarrow \mathbb{R}, x \mapsto c_P(x)$.

Beispiel: Jede Instanz $P = (X_P, c_P)$ des kombinatorischen Optimierungsproblems $\Pi = \text{TSP}$ ist eindeutig durch eine Distanzmatrix $C^P = (c_{ij}^P) \in \mathbb{R}^{n \times n}$ beschrieben. Es ist

$$X_P = \{\text{Permutationen } q \text{ von } \{1, 2, \dots, n\}\} \text{ und}$$

$$c_P(q) = (\sum_{i=1}^{n-1} c_{q(i), q(i+1)}^P) + c_{q(n), q(1)}^P.$$

Übersicht bekannter Optimierungsprobleme

- Matching Probleme
- Set-Packung, Set-Covering und Set-Partitioning
- Traveling-Salesman-Problem (TSP)
- Bin-Packung-Problem (BP)
- Rucksackproblem (KP)

Matching-Probleme

Matchings haben immer etwas mit der Bildung von Paaren zu tun.

Matching-Probleme

Matchings haben immer etwas mit der Bildung von Paaren zu tun.

Aus den Knoten eines ungerichteten bewerteten Graphen $G = (V, E, c_{ij})$ können Paare von adjazenten Knoten gebildet werden. Formal ist diese Paarbildung, d.h. das Matching, durch eine Teilmenge $M \subset E$ beschrieben. Man unterscheidet:

- Matchings M

(jeder Knoten in (V, M) ist mit maximal einem anderen Knoten verbunden)

- perfekte Matchings M

(... genau einem Knoten ...)

sowie Matching-Probleme mit

- Kosten

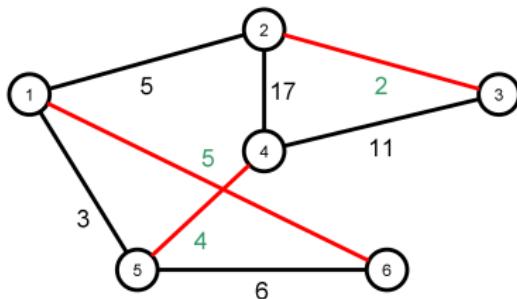
(die Bewertungen c_{ij} auf den Kanten $\{i,j\} \in E$ stellen Kosten dar)

- Profiten

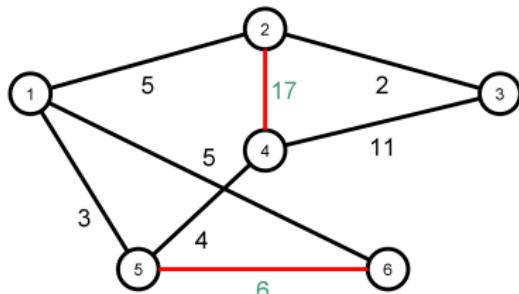
(... Profite ...)

Matching-Probleme

Perfektes Matching mit
minimalen Kosten



Matching mit maximalem Profit



Matching-Probleme

Gegeben: Bewerteter ungerichteter Graph $G = (V, E, c_{ij})$

Gesucht: (perfektes) Matching $M \subset E$ mit minimalen
Kosten/maximalem Profit

Matching-Probleme

Gegeben: Bewerteter ungerichteter Graph $G = (V, E, c_{ij})$

Gesucht: (perfektes) Matching $M \subset E$ mit minimalen Kosten/maximalem Profit

Modelle:

Perfektes Matching mit minimalen Kosten

$$\min \sum_{\{i,j\} \in E} c_{ij} x_{ij}$$

$$\text{s.d. } \sum_{i: \{i,j\} \in E} x_{ij} = 1 \quad \text{für alle } j \in V$$
$$x_{ij} \in \{0, 1\} \quad \text{für alle } \{i,j\} \in E$$

Matching mit maximalem Profit

$$\max \sum_{\{i,j\} \in E} c_{ij} x_{ij}$$

$$\text{s.d. } \sum_{i: \{i,j\} \in E} x_{ij} \leq 1 \quad \text{für alle } j \in V$$
$$x_{ij} \in \{0, 1\} \quad \text{für alle } \{i,j\} \in E$$

Bemerkungen:

- die Binärvariable x_{ij} gibt an, ob die Kante $\{i,j\} \in E$ zum Matching gehört oder nicht, d.h. $M := \{\{i,j\} \in E : x_{ij} = 1\}$
- die Nebenbedingungen sagen, dass für jeden Knoten $j \in V$ nur maximal/genau eine inzidente Kante gewählt werden darf

Set-Covering-, Set-Partitioning- und Set-Packing-Probleme

Set-Packing-, Set-Covering- und Set-Partitioning-Probleme lassen sich allgemein mit Hilfe von **Mengen** (engl.: sets) beschreiben.

Set-Covering-, Set-Partitioning- und Set-Packing-Probleme

Set-Packing-, Set-Covering- und Set-Partitioning-Probleme lassen sich allgemein mit Hilfe von **Mengen** (engl.: sets) beschreiben.

Gegeben:

- eine m -elementige Grundmenge $M = \{1, \dots, m\}$
- n Teilmengen $M_1, M_2, \dots, M_n \subset M$
- Kosten/Profite der Teilmengen c_1, c_2, \dots, c_n

Set-Covering-, Set-Partitioning- und Set-Packing-Probleme

Set-Packing-, Set-Covering- und Set-Partitioning-Probleme lassen sich allgemein mit Hilfe von **Mengen** (engl.: sets) beschreiben.

Gegeben:

- eine m -elementige Grundmenge $M = \{1, \dots, m\}$
- n Teilmengen $M_1, M_2, \dots, M_n \subset M$
- Kosten/Profite der Teilmengen c_1, c_2, \dots, c_n

Gesucht:

Auswahl von Teilmengen...

- 1 mit maximalem Profit bzw. minimalen Kosten,
- 2 so dass die Grundmenge M gepackt, überdeckt oder partitioniert wird.

Eine Auswahl von Teilmengen, d.h. eine Lösung, ist durch die Indexmenge $I \subseteq \{1, \dots, n\}$ der darin enthaltenen Teilmengen eindeutig festgelegt.

Set-Covering-, Set-Partitioning- und Set-Packing-Probleme

Beispiel

Stehen beispielsweise $n = 10$ Teilmengen M_1, M_2, \dots, M_{10} zur Auswahl, so kennzeichnet $I = \{1, 4, 5, 9\}$ eine Lösung, die die Teilmengen M_1, M_4, M_5 und M_9 enthält.

Set-Covering-, Set-Partitioning- und Set-Packing-Probleme

Beispiel

Stehen beispielsweise $n = 10$ Teilmengen M_1, M_2, \dots, M_{10} zur Auswahl, so kennzeichnet $I = \{1, 4, 5, 9\}$ eine Lösung, die die Teilmengen M_1, M_4, M_5 und M_9 enthält.

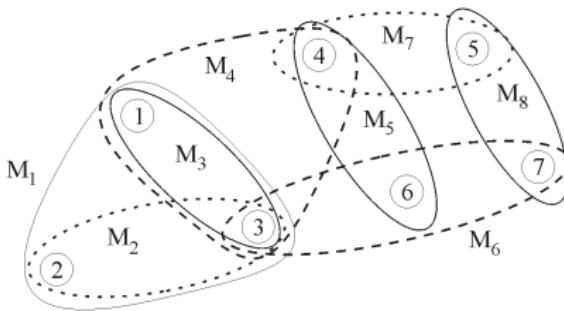
Man sagt, eine Lösung I ist

- 1 ein **Set-Packing**, wenn $M_j \cap M_k = \emptyset$ für alle $j, k \in I$ mit $j \neq k$ gilt,
- 2 ein **Set-Covering**, wenn $\bigcup_{j \in I} M_j = M$ gilt und
- 3 ein **Set-Partitioning**, wenn sie sowohl ein Set-Packing als auch ein Set-Covering ist.

Set-Covering-, Set-Partitioning- und Set-Packing-Probleme

Beispiel

Grundmenge $M = \{1, \dots, 7\}$. $n = 8$ Teilmengen M_1, M_2, \dots, M_8 gegeben, wobei $M_1 = \{1, 2, 3\}$, $M_2 = \{2, 3\}$ usw.



Dann ist $I = \{2, 5, 8\}$ ein Set-Packing, $I = \{1, 6, 7\}$ ein Set-Covering und $I = \{1, 5, 8\}$ ein Set-Partitioning.

Set-Covering-, Set-Partitioning- und Set-Packing-Probleme

Beispiel (Forts.)

Die verallgemeinerte Inzidenzmatrix für das Beispiel ist durch die folgende Matrix B gegeben:

	Teilmengen	M_1	M_2	M_3	M_4	M_5	M_6	M_7	M_8
Elemente									
	1	1	0	1	1	0	0	0	0
	2	1	1	0	0	0	0	0	0
$B =$	3	1	1	1	1	0	1	0	0
	4	0	0	0	1	1	0	1	0
	5	0	0	0	0	0	0	1	1
	6	0	0	0	0	1	1	0	0
	7	0	0	0	0	0	1	0	1

Die j -te Spalte enthält den charakteristischen Vektor der Menge M_j !



Set-Covering-, Set-Partitioning- und Set-Packing-Probleme

Optimierungsprobleme entstehen durch Zuordnung von Kosten bzw. Profiten zu den Teilmengen:

- 1 Set-Packing-Problem: Finde ein profitmaximales Packing
- 2 Set-Covering-Problem: Finde ein kostenminimales Covering
- 3 Set-Partitioning-Problem: Finde ein kostenminimales Partitioning

Set-Covering-, Set-Partitioning- und Set-Packing-Probleme

Optimierungsprobleme entstehen durch Zuordnung von Kosten bzw. Profiten zu den Teilmengen:

- 1 Set-Packing-Problem: Finde ein profitmaximales Packing
- 2 Set-Covering-Problem: Finde ein kostenminimales Covering
- 3 Set-Partitioning-Problem: Finde ein kostenminimales Partitioning

Entscheidungsvariablen:

- $x_1, \dots, x_n \in \{0, 1\}$ für jede Teilmenge
- $x_j = 1$, wenn die Teilmenge M_j in der Lösung ist; $x_j = 0$ sonst

Set-Covering-, Set-Partitioning- und Set-Packing-Probleme

Optimierungsprobleme entstehen durch Zuordnung von Kosten bzw. Profiten zu den Teilmengen:

- 1 Set-Packing-Problem: Finde ein profitmaximales Packing
- 2 Set-Covering-Problem: Finde ein kostenminimales Covering
- 3 Set-Partitioning-Problem: Finde ein kostenminimales Partitioning

Entscheidungsvariablen:

- $x_1, \dots, x_n \in \{0, 1\}$ für jede Teilmenge
- $x_j = 1$, wenn die Teilmenge M_j in der Lösung ist; $x_j = 0$ sonst

Modelle:

$$\begin{array}{lll} z_{SPK} & = & \max c^T x \\ & \text{so dass} & \\ & Bx \leq \mathbf{1} & \\ & x \in \{0, 1\}^n & \end{array} \quad \begin{array}{lll} z_{SC} & = & \min c^T x \\ & \text{so dass} & \\ & Bx \geq \mathbf{1} & \\ & x \in \{0, 1\}^n & \end{array} \quad \begin{array}{lll} z_{SP} & = & \min c^T x \\ & \text{so dass} & \\ & Bx = \mathbf{1} & \\ & x \in \{0, 1\}^n & \end{array}$$

Set-Covering-, Set-Partitioning- und Set-Packing-Probleme

Anwendung: "Aufstellen von Besuchsplänen"

- Vertreter sollen eine Menge von Kunden M besuchen.
- Für die Vertreter gibt es eine Menge von möglichen Besuchsplänen entsprechend Teilmengen $M_j \subset M$

Nun kann man – je nach Anforderung des Unternehmens – Besuchspläne so zusammenstellen, dass jeder Kunde

- 1 höchstens einmal,
- 2 mindestens einmal oder
- 3 genau einmal

besucht wird. Wir wollen dabei annehmen, dass jeder Vertreter jeden Besuchsplan durchführen kann und eine ausreichende Zahl von Vertretern zur Verfügung steht. Je nach Anforderung erhält man ein Set-Packung-Problem, Set-Covering-Problem oder Set-Partitioning-Problem.

Set-Covering-, Set-Partitioning- und Set-Packing-Probleme

Diese Probleme treten sehr häufig in unterschiedlichen Varianten in der Logistik auf.

- **Speditionen:** kostenminimales Zusammenstellen von Touren in einer Region;

Achtung: In vielen Anwendungen wächst die Anzahl der möglichen Teilmengen (=Teillösungen) extrem schnell an!

Set-Covering-, Set-Partitioning- und Set-Packing-Probleme

Diese Probleme treten sehr häufig in unterschiedlichen Varianten in der Logistik auf.

- **Speditionen:** kostenminimales Zusammenstellen von Touren in einer Region;
 - eine Tour besucht eine Teilmenge der Kunden

Achtung: In vielen Anwendungen wächst die Anzahl der möglichen Teilmengen (=Teillösungen) extrem schnell an!

Set-Covering-, Set-Partitioning- und Set-Packing-Probleme

Diese Probleme treten sehr häufig in unterschiedlichen Varianten in der Logistik auf.

- **Speditionen:** kostenminimales Zusammenstellen von Touren in einer Region;
 - eine Tour besucht eine Teilmenge der Kunden
 - Kosten hängen von der Reihenfolge ab, in der die Kunden besucht werden, und können erst durch das Lösen eines weiteren Optimierungsproblems bestimmt werden (\rightarrow TSP o.ä.)

Achtung: In vielen Anwendungen wächst die Anzahl der möglichen Teilmengen (=Teillösungen) extrem schnell an!

Set-Covering-, Set-Partitioning- und Set-Packing-Probleme

Diese Probleme treten sehr häufig in unterschiedlichen Varianten in der Logistik auf.

- **Speditionen:** kostenminimales Zusammenstellen von Touren in einer Region;
 - eine Tour besucht eine Teilmenge der Kunden
 - Kosten hängen von der Reihenfolge ab, in der die Kunden besucht werden, und können erst durch das Lösen eines weiteren Optimierungsproblems bestimmt werden (\rightarrow TSP o.ä.)
- **Fluggesellschaften:** kostenminimales Abdecken einer Menge von Flugverbindungen durch Crews

Achtung: In vielen Anwendungen wächst die Anzahl der möglichen Teilmengen (=Teillösungen) extrem schnell an!

Set-Covering-, Set-Partitioning- und Set-Packing-Probleme

Diese Probleme treten sehr häufig in unterschiedlichen Varianten in der Logistik auf.

- **Speditionen:** kostenminimales Zusammenstellen von Touren in einer Region;
 - eine Tour besucht eine Teilmenge der Kunden
 - Kosten hängen von der Reihenfolge ab, in der die Kunden besucht werden, und können erst durch das Lösen eines weiteren Optimierungsproblems bestimmt werden (\rightarrow TSP o.ä.)
- **Fluggesellschaften:** kostenminimales Abdecken einer Menge von Flugverbindungen durch Crews
- **Produktions- und Dienstleistungsgewerbe:** kostenminimales Abdecken eines Schichtplans durch Mitarbeiter

Achtung: In vielen Anwendungen wächst die Anzahl der möglichen Teilmengen (=Teillösungen) extrem schnell an!

Traveling Salesman Problem (TSP)

- am meisten untersuchtes kombinatorisches Optimierungsproblem
- leicht zu formulieren aber schwer zu lösen (\mathcal{NP} -schweres Problem)

Traveling Salesman Problem (TSP)

- am meisten untersuchtes kombinatorisches Optimierungsproblem
- leicht zu formulieren aber schwer zu lösen (\mathcal{NP} -schweres Problem)
- Handlungsreisender befindet sich an einem Ort und möchte weitere Orte aufsuchen
- Abstände zwischen den Orten sind bekannt

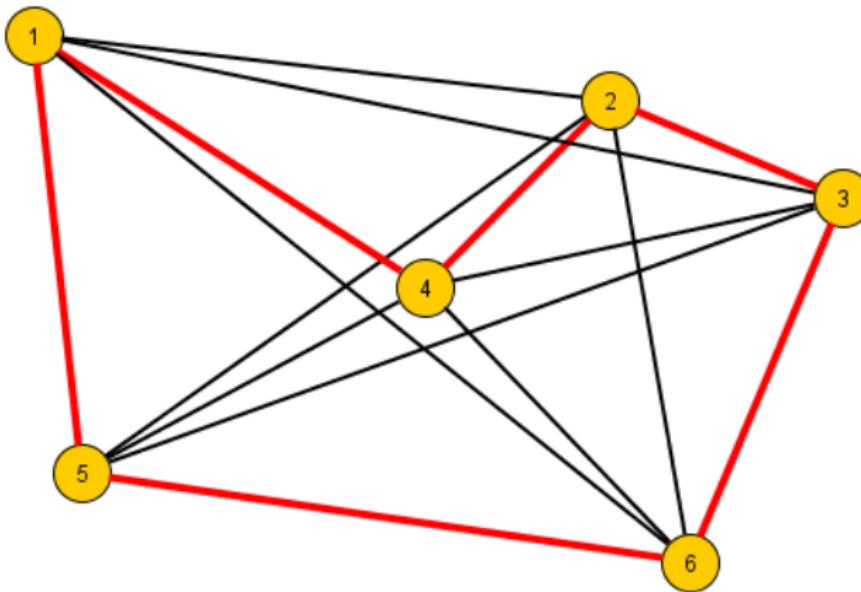
Traveling Salesman Problem (TSP)

- am meisten untersuchtes kombinatorisches Optimierungsproblem
- leicht zu formulieren aber schwer zu lösen (\mathcal{NP} -schweres Problem)
- Handlungsreisender befindet sich an einem Ort und möchte weitere Orte aufsuchen
- Abstände zwischen den Orten sind bekannt
- In welcher **Reihenfolge** muss der Handlungsreisende die **Orte besuchen**, so dass er **alle Orte genau einmal besucht**, er zum **Ursprungsort zurückkehrt** und dabei eine möglichst kurze Strecke zurücklegt?

Beispiel TSP – Real



Beispiel TSP – Abstrakt



Anwendungsgebiete TSP

Anwendungsgebiete des TSP (in der Logistik) sind z. B.:

- Bestimmung kostenminimaler Touren für Boten oder Fahrzeuge

Anwendungsgebiete TSP

Anwendungsgebiete des TSP (in der Logistik) sind z. B.:

- Bestimmung kostenminimaler Touren für Boten oder Fahrzeuge
- Bestimmung optimaler Bedienreihenfolgen auf Maschinen

Anwendungsgebiete TSP

Anwendungsgebiete des TSP (in der Logistik) sind z. B.:

- Bestimmung kostenminimaler Touren für Boten oder Fahrzeuge
- Bestimmung optimaler Bedienreihenfolgen auf Maschinen
- Varianten von Briefträgerproblemen können als TSP gelöst werden

Anwendungsgebiete TSP

Anwendungsgebiete des TSP (in der Logistik) sind z. B.:

- Bestimmung kostenminimaler Touren für Boten oder Fahrzeuge
- Bestimmung optimaler Bedienreihenfolgen auf Maschinen
- Varianten von Briefträgerproblemen können als TSP gelöst werden
- Teilproblem im Vehicle Routing Problem (VRP)

Anwendungsgebiete TSP

Anwendungsgebiete des TSP (in der Logistik) sind z. B.:

- Bestimmung kostenminimaler Touren für Boten oder Fahrzeuge
- Bestimmung optimaler Bedienreihenfolgen auf Maschinen
- Varianten von Briefträgerproblemen können als TSP gelöst werden
- Teilproblem im Vehicle Routing Problem (VRP)

Das TSP tritt in der Praxis jedoch selten in seiner „reinen“ Form auf: oft sind **weitere Restriktionen** zu beachten, z. B.:

- Vorrangsbeziehungen (sog. Präzedenzen)
- Fahrzeiten und Zeitfenster
- zeitabhängige Fahrzeiten etc.

Anwendungsgebiete TSP

Anwendungsgebiete des TSP (in der Logistik) sind z. B.:

- Bestimmung kostenminimaler Touren für Boten oder Fahrzeuge
- Bestimmung optimaler Bedienreihenfolgen auf Maschinen
- Varianten von Briefträgerproblemen können als TSP gelöst werden
- Teilproblem im Vehicle Routing Problem (VRP)

Das TSP tritt in der Praxis jedoch selten in seiner „reinen“ Form auf: oft sind **weitere Restriktionen** zu beachten, z. B.:

- Vorrangsbeziehungen (sog. Präzedenzen)
- Fahrzeiten und Zeitfenster
- zeitabhängige Fahrzeiten etc.

Wir betrachten hier das **TSP ohne weitere Nebenbedingungen** und bezeichnen dieses Problem als **klassisches TSP**.

Formale Definition TSP

Symmetrischer Fall (STSP):

- $D = (d_{ij})_{1 \leq i,j \leq n}$ symmetrische Distanzmatrix
- bzw. ein vollständiger, ungerichteter, bewerteter Graph (V, E, d)

Formale Definition TSP

Symmetrischer Fall (STSP):

- $D = (d_{ij})_{1 \leq i,j \leq n}$ symmetrische Distanzmatrix
- bzw. ein vollständiger, ungerichteter, bewerteter Graph (V, E, d)

Asymmetrischer Fall (ATSP):

- $D = (d_{ij})_{1 \leq i,j \leq n}$ beliebige Distanzmatrix
- bzw. ein vollständiger, bewerteter Digraph $G = (V, A, d)$

Formale Definition TSP

Symmetrischer Fall (STSP):

- $D = (d_{ij})_{1 \leq i,j \leq n}$ symmetrische Distanzmatrix
- bzw. ein vollständiger, ungerichteter, bewerteter Graph (V, E, d)

Asymmetrischer Fall (ATSP):

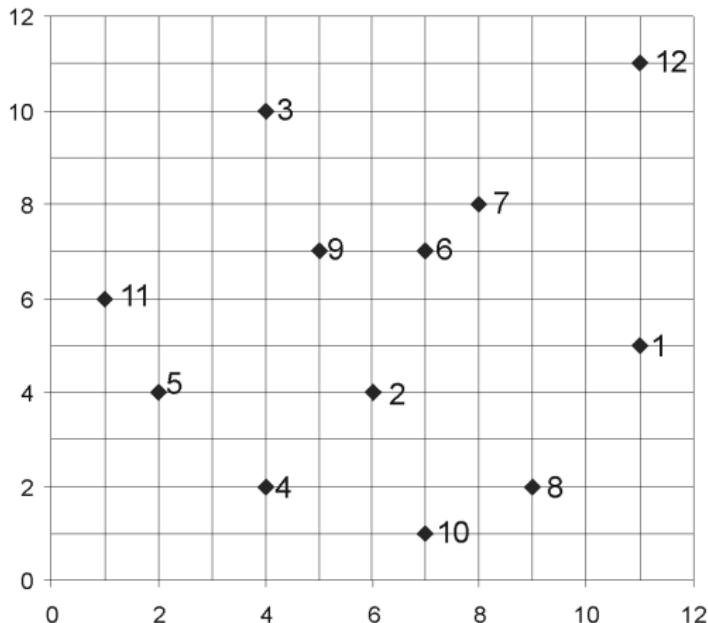
- $D = (d_{ij})_{1 \leq i,j \leq n}$ beliebige Distanzmatrix
- bzw. ein vollständiger, bewerteter Digraph $G = (V, A, d)$

Gesucht:

- eine kostenminimale Rundreise durch alle Städte
- = d.h. eine kostenminimale (Hamilton-)Tour
- = d.h. eine kostenminimale geschlossene Kantenfolge, die alle Knoten (=Städte) genau einmal besucht

Beispiel TSP

Beispiel 12 Städte TSP



Beispiel TSP

Beispiel (Forts.)

Die **euklidische Distanz** zwischen zwei Orten (x_i, y_i) und (x_j, y_j) ist definiert als

$$d_{ij} = \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2}.$$

Durch Runden der Werte auf eine Nachkommastelle erhält man folgende Distanzmatrix:

$$D = \begin{pmatrix} \text{---} & 5.1 & 8.6 & 7.6 & 9.1 & 4.5 & 4.2 & 3.6 & 6.3 & 5.7 & 10.0 & 6.0 \\ 5.1 & \text{---} & 6.3 & 2.8 & 4.0 & 3.2 & 4.5 & 3.6 & 3.2 & 3.2 & 5.4 & 8.6 \\ 8.6 & 6.3 & \text{---} & 8.0 & 6.3 & 4.2 & 4.5 & 9.4 & 3.2 & 9.5 & 5.0 & 7.1 \\ 7.6 & 2.8 & 8.0 & \text{---} & 2.8 & 5.8 & 7.2 & 5.0 & 5.1 & 3.2 & 5.0 & 11.4 \\ 9.1 & 4.0 & 6.3 & 2.8 & \text{---} & 5.8 & 7.2 & 7.3 & 4.2 & 5.8 & 2.2 & 11.4 \\ 4.5 & 3.2 & 4.2 & 5.8 & 5.8 & \text{---} & 1.4 & 5.4 & 2.0 & 6.0 & 6.1 & 5.7 \\ 4.2 & 4.5 & 4.5 & 7.2 & 7.2 & 1.4 & \text{---} & 6.1 & 3.2 & 7.1 & 7.3 & 4.2 \\ 3.6 & 3.6 & 9.4 & 5.0 & 7.3 & 5.4 & 6.1 & \text{---} & 6.4 & 2.2 & 8.9 & 9.2 \\ 6.3 & 3.2 & 3.2 & 5.1 & 4.2 & 2.0 & 3.2 & 6.4 & \text{---} & 6.3 & 4.1 & 7.2 \\ 5.7 & 3.2 & 9.5 & 3.2 & 5.8 & 6.0 & 7.1 & 2.2 & 6.3 & \text{---} & 7.8 & 10.8 \\ 10.0 & 5.4 & 5.0 & 5.0 & 2.2 & 6.1 & 7.3 & 8.9 & 4.1 & 7.8 & \text{---} & 11.2 \\ 6.0 & 8.6 & 7.1 & 11.4 & 11.4 & 5.7 & 4.2 & 9.2 & 7.2 & 10.8 & 11.2 & \text{---} \end{pmatrix}$$

Beispiel TSP

Beispiel (Forts.)

Die **Manhattan-Distanz** zwischen zwei Orten (x_i, y_i) und (x_j, y_j) ist definiert als

$$d_{ij} = |x_i - x_j| + |y_i - y_j|.$$

Die resultierende (symmetrische) Distanzmatrix lautet:

$$D = \begin{pmatrix} \text{---} & 6 & 12 & 10 & 10 & 6 & 6 & 5 & 8 & 8 & 11 & 6 \\ 6 & \text{---} & 8 & 4 & 4 & 4 & 6 & 5 & 4 & 4 & 7 & 12 \\ 12 & 8 & \text{---} & 8 & 8 & 6 & 6 & 13 & 4 & 12 & 7 & 8 \\ 10 & 4 & 8 & \text{---} & 4 & 8 & 10 & 5 & 6 & 4 & 7 & 16 \\ 10 & 4 & 8 & 4 & \text{---} & 8 & 10 & 9 & 6 & 8 & 3 & 16 \\ 6 & 4 & 6 & 8 & 8 & \text{---} & 2 & 7 & 2 & 6 & 7 & 8 \\ 6 & 6 & 6 & 10 & 10 & 2 & \text{---} & 7 & 4 & 8 & 9 & 6 \\ 5 & 5 & 13 & 5 & 9 & 7 & 7 & \text{---} & 9 & 3 & 12 & 11 \\ 8 & 4 & 4 & 6 & 6 & 2 & 4 & 9 & \text{---} & 8 & 5 & 10 \\ 8 & 4 & 12 & 4 & 8 & 6 & 8 & 3 & 8 & \text{---} & 11 & 14 \\ 11 & 7 & 7 & 7 & 3 & 7 & 9 & 12 & 5 & 11 & \text{---} & 15 \\ 6 & 12 & 8 & 16 & 16 & 8 & 6 & 11 & 10 & 14 & 15 & \text{---} \end{pmatrix}$$

Modellierung TSP

Entscheidungsvariablen beim ATSP:

- $x_{ij} \in \{0, 1\}$ für alle Bögen $(i, j) \in A$; also $|V|(|V| - 1)$ EV
- mit $x_{ij} = 1$, falls j direkt nach i besucht wird, und 0 sonst

Erweitertes Zuordnungsmodell für das ATSP:

$$(1) \quad z_{ATSP} = \min \sum_{(i,j) \in A} d_{ij} x_{ij}$$

$$(2) \text{ so dass } \sum_{(i,j) \in \delta^+(i)} x_{ij} = 1 \quad \text{für alle } i \in V$$

$$(3) \quad \sum_{(i,j) \in \delta^-(j)} x_{ij} = 1 \quad \text{für alle } j \in V$$

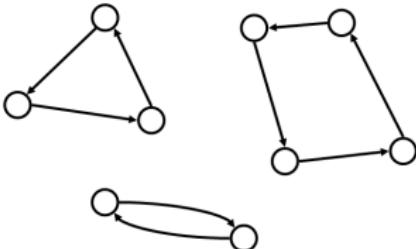
$$(4) \quad \sum_{(i,j) \in A(S)} x_{ij} \leq |S| - 1 \quad \text{für alle } S \subset V, 2 \leq |S| < |V|$$

$$(5) \quad x_{ij} \in \{0, 1\} \quad \text{für alle } (i, j) \in A$$

Modellierung TSP

Erweitertes Zuordnungsmodell für das ATSP (Forts.):

- (1) Minimiere die Gesamtlänge der Rundtour
- (2) Jeder Knoten (Stadt) besitzt genau einen Nachfolger;
 $\delta^+(i)$ ist die Menge aller Bögen, die aus i herausführen
- (3) Jeder Knoten (Stadt) besitzt genau einen Vorgänger;
 $\delta^-(j)$ ist die Menge aller Bögen, die in j hineinführen
- (4) Subtour-Eliminationsbedingungen (SEB); es werden Kreise durch jede Teilmenge S der Knoten verhindert;
 $A(S)$ ist die Menge der Bögen mit beiden Endpunkten in S



- (5) Binäre EV

Modellierung TSP

Entscheidungsvariablen beim STSP:

- $x_{ij} \in \{0, 1\}$ für alle Kanten $\{i, j\} \in E$; $|V|(|V| - 1)/2$ EV
- mit $x_{ij} = 1$, falls i und j direkt verbunden, und 0 sonst

2-Matching-Modell für das STSP:

$$(1)_{STSP} = \min \sum_{\{i,j\} \in E} d_{ij} x_{ij}$$

$$\text{s@ dass } \sum_{\{i,j\} \in \delta(i)} x_{ij} = 2 \quad \text{für alle } i \in V$$

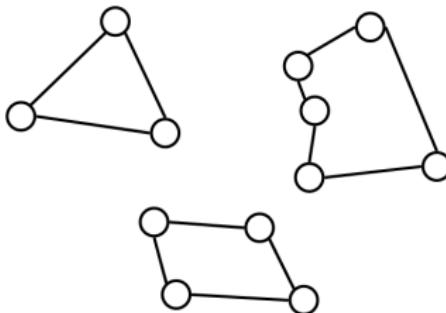
$$(3) \quad \sum_{\{i,j\} \in E(S)} x_{ij} \leq |S| - 1 \quad \text{für alle } S \subset V, 3 \leq |S| \leq |V| - 2$$

$$(4) \quad x_{ij} \in \{0, 1\} \quad \text{für alle } \{i, j\} \in E$$

Modellierung TSP

2-Matching-Modell für das STSP (Forts.):

- (1) Minimiere die Gesamtlänge der Rundtour
- (2) Jeder Knoten (Stadt) ist mit genau zwei Knoten verbunden;
 $\delta(i)$ ist die Menge aller Kanten, die mit i inzidieren
- (3) Subtour-Eliminationsbedingungen (SEB); es werden Kreise durch jede Teilmenge S der Knoten verhindert;
 $E(S)$ ist die Menge der Kanten mit beiden Endpunkten in S



- (4) Binäre EV

TSP – Lösung durch vollständige Enumeration

- Warum kann man nicht einfach alle möglichen Touren aufzählen und ihre Längen vergleichen?

TSP – Lösung durch vollständige Enumeration

- Warum kann man nicht einfach alle möglichen Touren aufzählen und ihre Längen vergleichen?
- Die Anzahl der Touren eines STSP mit n Städten ist $(n - 1)!/2$. Warum?

TSP – Lösung durch vollständige Enumeration

- Warum kann man nicht einfach alle möglichen Touren aufzählen und ihre Längen vergleichen?
- Die Anzahl der Touren eines STSP mit n Städten ist $(n - 1)!/2$. Warum?

n	3	4	5	6	7	8	9	10
$\frac{(n-1)!}{2}$	1	3	12	60	360	2520	20160	181440
n		15		20		30		40
$\frac{(n-1)!}{2}$	4,4E+10	6,1E+16		4,4E+30		1,0E+46		3,0E+62

	Operationen pro	ms	s	min	h
1 MHz	C64	1000	1000000	60000000	36000000000
10 MHz	PC 80286	10000	10000000	600000000	360000000000
600 MHz	PC Pentium III	600000	600000000	360000000000	2,16E+12
2,8 GHz	PC Pentium 4	2800000	2800000000	1,68E+11	1,008E+13

	Operationen pro	Tag	Woche	Jahr
1 MHz	C64	86400000000	6,048E+11	3,1536E+13
10 MHz	PC 80286	8,64E+11	6,048E+12	3,1536E+14
600 MHz	PC Pentium III	5,184E+13	3,6288E+14	1,89216E+16
2,8 GHz	PC Pentium 4	2,4192E+14	1,69344E+15	8,83008E+16

Ausblick: Schwierigkeit von Optimierungsproblemen

Folgt aus einer hohen Anzahl möglicher Lösungen die Schwierigkeit des Problems?

Ausblick: Schwierigkeit von Optimierungsproblemen

Folgt aus einer hohen Anzahl möglicher Lösungen die Schwierigkeit des Problems?

Anzahl Lösungen vs. „Schwierigkeit“



Die Tatsache, dass es für ein Optimierungsproblem sehr viele zulässige Lösungen gibt (z.B. exponentiell viele), ist **kein** Argument für die Schwierigkeit eines Problems.

Ausblick: Schwierigkeit von Optimierungsproblemen

Folgt aus einer hohen Anzahl möglicher Lösungen die Schwierigkeit des Problems?

Anzahl Lösungen vs. „Schwierigkeit“



Die Tatsache, dass es für ein Optimierungsproblem sehr viele zulässige Lösungen gibt (z.B. exponentiell viele), ist **kein** Argument für die Schwierigkeit eines Problems.

■ Ausblick:

- „Schwierigkeit eines Problems“ mittels Laufzeitkomplexität von Algorithmen beschreiben
- Klassifikation zur Beschreibung der Schwierigkeit eines Optimierungsproblems

Das Rucksackproblem

Rucksackprobleme (engl.: knapsack problems) treten immer dann auf, wenn man eine Ressource mit begrenzter Kapazität betrachtet, die es möglichst effizient zu nutzen gilt. Der Name röhrt daher, dass eine Person die Ressource, den Rucksack, mit verschiedenen Gegenständen bepacken möchte.

Das Rucksackproblem

Rucksackprobleme (engl.: knapsack problems) treten immer dann auf, wenn man eine Ressource mit begrenzter Kapazität betrachtet, die es möglichst effizient zu nutzen gilt. Der Name röhrt daher, dass eine Person die Ressource, den Rucksack, mit verschiedenen Gegenständen bepacken möchte.

- Jeder Gegenstand hat ein Gewicht und einen Profit (der Nutzen, den die Mitnahme mit sich bringt).
- Die Gegenstände können nicht zerteilt werden.
- Aufgrund der unterschiedlichen Eigenschaften der Gegenstände ist der spezifische Gewinn pro kg verschieden.

Das Rucksackproblem

Rucksackprobleme (engl.: knapsack problems) treten immer dann auf, wenn man eine Ressource mit begrenzter Kapazität betrachtet, die es möglichst effizient zu nutzen gilt. Der Name röhrt daher, dass eine Person die Ressource, den Rucksack, mit verschiedenen Gegenständen bepacken möchte.

- Jeder Gegenstand hat ein Gewicht und einen Profit (der Nutzen, den die Mitnahme mit sich bringt).
- Die Gegenstände können nicht zerteilt werden.
- Aufgrund der unterschiedlichen Eigenschaften der Gegenstände ist der spezifische Gewinn pro kg verschieden.

Es stellt sich nun die Frage, welche Gegenstände in den Rucksack gepackt werden sollen, so dass man möglichst viel Gewinn macht. Das Rucksackproblem ist also ein Auswahlproblem.

Formale Definition Rucksackproblem

Gegeben:

- Profite $p_1, p_2, \dots, p_n \geq 0$
- Gewichte $w_1, w_2, \dots, w_n \geq 0$
- Kapazität $C > 0$

Formale Definition Rucksackproblem

Gegeben:

- Profite $p_1, p_2, \dots, p_n \geq 0$
- Gewichte $w_1, w_2, \dots, w_n \geq 0$
- Kapazität $C > 0$

Gesucht: Auswahl, d.h. eine Teilmenge I von $\{1, 2, \dots, n\}$, mit maximalem Profit und Gewicht nicht größer als die Kapazität:

Formale Definition Rucksackproblem

Gegeben:

- Profite $p_1, p_2, \dots, p_n \geq 0$
- Gewichte $w_1, w_2, \dots, w_n \geq 0$
- Kapazität $C > 0$

Gesucht: Auswahl, d.h. eine Teilmenge I von $\{1, 2, \dots, n\}$, mit maximalem Profit und Gewicht nicht größer als die Kapazität:

$$\max_{I \subseteq \{1, 2, \dots, n\}} \sum_{i \in I} p_i$$

mit

$$\sum_{i \in I} w_i \leq C.$$

Beispiel Rucksackproblem

- $n = 6$ Gegenstände und Kapazität $C = 190$
- Gegenstände:

i	1	2	3	4	5	6
Profit p_i	50	50	64	46	50	5
Gewicht w_i	56	59	80	64	75	17

aus: (Martello und Toth, 1990)

- Optimale Lösung

Beispiel Rucksackproblem

- $n = 6$ Gegenstände und Kapazität $C = 190$
- Gegenstände:

i	1	2	3	4	5	6	
Profit p_i	50	50	64	46	50	5	aus: (Martello und Toth, 1990)
Gewicht w_i	56	59	80	64	75	17	

- Optimale Lösung ist $I = \{1, 2, 5\}$ (oder in der Schreibweise mit Inzidenzvektor $x^\top = (1, 1, 0, 0, 1, 0)$) mit ZF-Wert $z = p^\top x = 150$.

Modellierung Rucksackproblem

Modell: Entscheidungsvariable $x_j \in \{0, 1\}$ für $j = 1, 2, \dots, n$ mit der folgenden Bedeutung: $x_j = 1$ heißt, dass der Gegenstand j ausgewählt (im Rucksack) ist; $x_j = 0$ heißt, dass der Gegenstand j nicht gewählt ist.

Modellierung Rucksackproblem

Modell: Entscheidungsvariable $x_j \in \{0, 1\}$ für $j = 1, 2, \dots, n$ mit der folgenden Bedeutung: $x_j = 1$ heißt, dass der Gegenstand j ausgewählt (im Rucksack) ist; $x_j = 0$ heißt, dass der Gegenstand j nicht gewählt ist.

$$(1) \quad z_{KP} = \max \sum_{j=1}^n p_j x_j$$

$$(2) \quad \text{so dass} \quad \sum_{j=1}^n w_j x_j \leq C$$

$$(3) \quad x_j \in \{0, 1\} \quad \text{für alle } j = 1, \dots, n.$$

Das Rucksackproblem

Anwendung:

- 1 **Investitionsrechnung:** Gegeben ist ein Budget C . Es besteht die Möglichkeit, in n Projekte $j = 1, 2, \dots, n$ zu investieren. Jedes Projekt kann entweder komplett durchgeführt werden oder gar nicht. Das j te Projekt erfordert eine Investition w_j und verspricht einen erwarteten Gewinn p_j .

Das Rucksackproblem

Anwendung:

- 1 Investitionsrechnung:** Gegeben ist ein Budget C . Es besteht die Möglichkeit, in n Projekte $j = 1, 2, \dots, n$ zu investieren. Jedes Projekt kann entweder komplett durchgeführt werden oder gar nicht. Das j te Projekt erfordert eine Investition w_j und verspricht einen erwarteten Gewinn p_j .
- 2 Beladung eines Flugzeugs:** Es liegen eine Reihe von möglichen Transportaufträgen für einen Flug vor. Jeder Transportauftrag $i = 1, 2, \dots, n$ ist definiert über ein zu transportierendes Gewicht w_i und einen Gewinn p_i , der durch den Transport realisiert wird. Bei beschränkter Flugzeugkapazität C (d.h. $C < \sum_{i=1}^n w_i$) muss eine Auswahl von Transportaufträgen getätigt werden.

Das Bin Packing Problem

Das Bin Packing Problem ist der Prototyp eines Packproblems:
Eine gegebene Menge von Gegenständen n soll in eine minimale Anzahl gleichgroßer Behälter (engl.: bins) gepackt werden.

Das Bin Packing Problem

Das Bin Packing Problem ist der Prototyp eines Packproblems:
Eine gegebene Menge von Gegenständen n soll in eine minimale Anzahl gleichgroßer Behälter (engl.: bins) gepackt werden.

Gegeben:

- Kapazität $C > 0$
- Gewichte w_1, w_2, \dots, w_n mit $0 < w_i \leq C$

Das Bin Packing Problem

Das Bin Packing Problem ist der Prototyp eines Packproblems:
Eine gegebene Menge von Gegenständen n soll in eine minimale Anzahl gleichgroßer Behälter (engl.: bins) gepackt werden.

Gegeben:

- Kapazität $C > 0$
- Gewichte w_1, w_2, \dots, w_n mit $0 < w_i \leq C$

Gesucht: Partitionierung $P_1 \cup P_2 \cup \dots \cup P_k = \{1, 2, \dots, n\}$
(d.h. auch $P_i \cap P_j = \emptyset$ für $i \neq j \in \{1, 2, \dots, n\}$) mit $k \in \mathbb{N}$
minimal und

$$\sum_{i \in P_j} w_i \leq C \quad \text{für alle } j \in \{1, 2, \dots, k\}.$$

Modellierung Bin Packing Problem

Modellierung Bin Packing Problem

$$(1) \quad z_{BP} = \min \sum_{j=1}^n y_j$$

$$(2) \quad \text{so dass } \sum_{i=1}^n w_i x_{ij} \leq C y_j \quad \text{für alle } j = 1, \dots, n$$

$$(3) \quad \sum_{j=1}^n x_{ij} = 1 \quad \text{für alle } i = 1, \dots, n$$

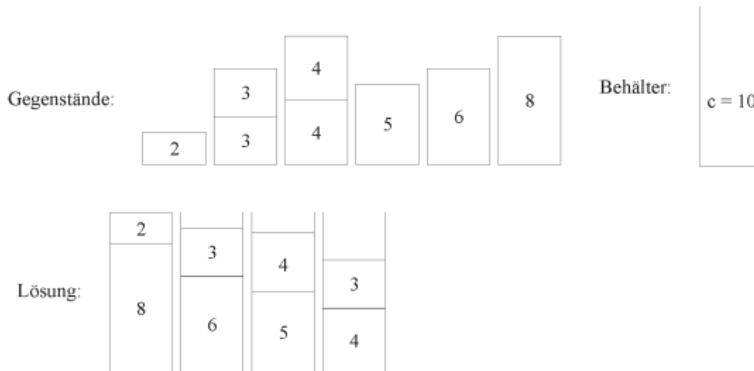
$$(4) \quad y_j \in \{0, 1\} \quad \text{für alle } j = 1, \dots, n$$

$$(5) \quad x_{ij} \in \{0, 1\} \quad \text{für alle } i, j = 1, \dots, n$$

- (1) Minimiere Anzahl der benutzten Behälter/Bins
- (2) Nur benutzte Behälter können bis zur Kapazität C gefüllt werden
- (3) Jeder Gegenstand wird eindeutig einem Behälter zugeordnet
- (4-5) Binäre EV y_j und x_{ij}

Beispiel Bin Packing Problem

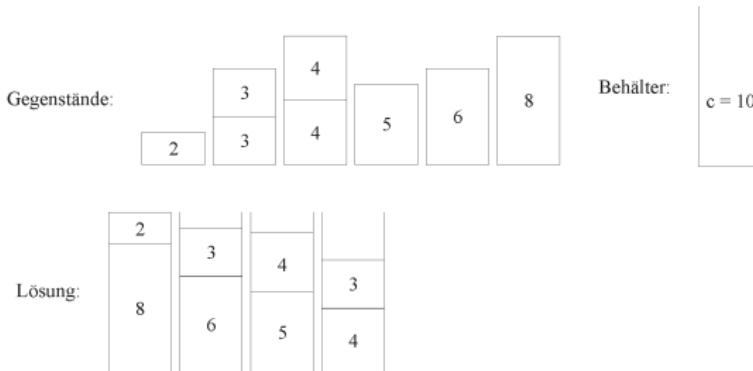
- Bin-Packing-Problem mit $n = 8$ Gegenständen und Behältern der Kapazität $C = 10$



- Die abgebildete Lösung ist optimal

Beispiel Bin Packing Problem

- Bin-Packing-Problem mit $n = 8$ Gegenständen und Behältern der Kapazität $C = 10$



- Die abgebildete Lösung ist **optimal**, denn die insgesamt benötigte Kapazität der Gegenstände ist gleich 35 und die Kapazität eines Behälters ist gleich 10.

Agenda

1 Grundlagen

- Operations Research und kombinatorische Optimierung
- Algorithmen, Aufwand und Komplexitätsklassen
- Heuristiken

Heuristiken

Heuristik (Definition)

Heuristiken sind Algorithmen, die ein gegebenes Optimierungsproblem mit akzeptablem Aufwand möglichst gut zu lösen versuchen.

Heuristiken

Heuristik (Definition)

Heuristiken sind Algorithmen, die ein gegebenes Optimierungsproblem mit akzeptablem Aufwand möglichst gut zu lösen versuchen.

Dazu muss geklärt werden:

- 1 Was sind Optimierungsprobleme und was sind Algorithmen?

Heuristiken

Heuristik (Definition)

Heuristiken sind Algorithmen, die ein gegebenes Optimierungsproblem mit akzeptablem Aufwand möglichst gut zu lösen versuchen.

Dazu muss geklärt werden:

- 1 Was sind **Optimierungsprobleme** und was sind Algorithmen?
- 2 Auf welche Optimierungsprobleme kann und sollte man **Heuristiken** anwenden?

Heuristiken

Heuristik (Definition)

Heuristiken sind Algorithmen, die ein gegebenes Optimierungsproblem mit akzeptablem Aufwand möglichst gut zu lösen versuchen.

Dazu muss geklärt werden:

- 1 Was sind **Optimierungsprobleme** und was sind Algorithmen?
- 2 Auf welche Optimierungsprobleme kann und sollte man **Heuristiken anwenden**?
- 3 Wie kann man die **Qualität** der bestimmten Lösungen analysieren?

Heuristiken

Heuristik (Definition)

Heuristiken sind Algorithmen, die ein gegebenes Optimierungsproblem mit akzeptablem Aufwand möglichst gut zu lösen versuchen.

Dazu muss geklärt werden:

- 1 Was sind **Optimierungsprobleme** und was sind Algorithmen?
- 2 Auf welche Optimierungsprobleme kann und sollte man **Heuristiken anwenden**?
- 3 Wie kann man die **Qualität** der bestimmten Lösungen analysieren?
- 4 Was ist **akzeptabler Aufwand**?

Heuristiken

Heuristik (Definition)

Heuristiken sind Algorithmen, die ein gegebenes Optimierungsproblem mit akzeptablem Aufwand möglichst gut zu lösen versuchen.

Dazu muss geklärt werden:

- 1 Was sind **Optimierungsprobleme** und was sind Algorithmen?
- 2 Auf welche Optimierungsprobleme kann und sollte man **Heuristiken anwenden**?
- 3 Wie kann man die **Qualität** der bestimmten Lösungen analysieren?
- 4 Was ist **akzeptabler Aufwand**?
- 5 Für manche Optimierungsprobleme ist bereits das Finden einer **zulässigen Lösung** ein *schwieriges Problem* (was heißt das?). Daher: „.... zu lösen versuchen.“

Heuristiken

Heuristik (Definition)

Heuristiken sind Algorithmen, die ein gegebenes Optimierungsproblem mit akzeptablem Aufwand möglichst gut zu lösen versuchen.

Dazu muss geklärt werden:

- 1 Was sind **Optimierungsprobleme** und was sind Algorithmen?
- 2 Auf welche Optimierungsprobleme kann und sollte man **Heuristiken anwenden**?
- 3 Wie kann man die **Qualität** der bestimmten Lösungen analysieren?
- 4 Was ist **akzeptabler Aufwand**?
- 5 Für manche Optimierungsprobleme ist bereits das Finden einer **zulässigen Lösung** ein *schwieriges Problem* (was heißt das?). Daher: „.... zu lösen versuchen.“
- 6 Was sind allgemein nützliche Strategien („Werkzeugkiste“) für die Konstruktion heuristischer Optimierungsverfahren?

Heuristiken

Kennzeichen von Heuristiken:

- 1 Ausschluss potentieller Lösungen: Lösungsraum nicht komplett durchsucht, Ausschluss gesuchter Lösungen möglich

Heuristiken

Kennzeichen von Heuristiken:

- 1 Ausschluss potentieller Lösungen: Lösungsraum nicht komplett durchsucht, Ausschluss gesuchter Lösungen möglich
- 2 Fehlende Lösungsgarantie: kein Beweis für Konvergenz gegen bestimmte Lösung

Heuristiken

Kennzeichen von Heuristiken:

- 1 Ausschluss potentieller Lösungen: Lösungsraum nicht komplett durchsucht, Ausschluss gesuchter Lösungen möglich
- 2 Fehlende Lösungsgarantie: kein Beweis für Konvergenz gegen bestimmte Lösung
- 3 Nicht-willkürliche Lösungssuche: gezielte Suche nach intelligenten Regeln

Heuristiken

Kennzeichen von Heuristiken:

- 1 Ausschluss potentieller Lösungen: Lösungsraum nicht komplett durchsucht, Ausschluss gesuchter Lösungen möglich
- 2 Fehlende Lösungsgarantie: kein Beweis für Konvergenz gegen bestimmte Lösung
- 3 Nicht-willkürliche Lösungssuche: gezielte Suche nach intelligenten Regeln
- 4 Künstliche Stopptregel: aufgrund fehlender Lösungsgarantie nötig

Heuristiken

Kennzeichen von Heuristiken:

- 1 Ausschluss potentieller Lösungen: Lösungsraum nicht komplett durchsucht, Ausschluss gesuchter Lösungen möglich
- 2 Fehlende Lösungsgarantie: kein Beweis für Konvergenz gegen bestimmte Lösung
- 3 Nicht-willkürliche Lösungssuche: gezielte Suche nach intelligenten Regeln
- 4 Künstliche Stopptregel: aufgrund fehlender Lösungsgarantie nötig
 - Eigenschaften 1. bis 3. sind notwendig.
 - Eigenschaft 4. möglich.

Algorithmen: Definition und Vorkommen

Algorithmus (Definition)

Ein *Algorithmus* ist eine genau definierte Verarbeitungsvorschrift zur Lösung eines Problems oder einer bestimmten Art von Problemen. Typischerweise wird ein Algorithmus durch eine endliche *Folge von Anweisungen* beschrieben, die nacheinander ausgeführt und oft in festgelegter Weise wiederholt werden.

Algorithmen: Definition und Vorkommen

Algorithmus (Definition)

Ein *Algorithmus* ist eine genau definierte Verarbeitungsvorschrift zur Lösung eines Problems oder einer bestimmten Art von Problemen. Typischerweise wird ein Algorithmus durch eine endliche *Folge von Anweisungen* beschrieben, die nacheinander ausgeführt und oft in festgelegter Weise wiederholt werden.

Im täglichen Leben kommen Algorithmen häufig vor:

- Reparatur- und Bedienungsanleitungen
- Hilfen zum Ausfüllen von Formularen
- Am weitesten verbreitet sind Algorithmen in der Informatik:
Dort steuern sie, in der Form von Computerprogrammen oder elektronischen Schaltkreisen, Rechner und Maschinen.

Algorithmen: Eigenschaften

■ Ein Algorithmus ist

- eindeutig: die einzelnen Schritte und ihre Abfolge sind unmissverständlich beschrieben
- allgemein: es wird nicht nur eine Probleminstanz, sondern alle Instanzen eines Problems gelöst
- ausführbar: der „Prozessor“ muss die Einzelschritte abarbeiten können
- endlich: seine Beschreibung besteht aus einem Text endlicher Länge ($1 + \frac{1}{2} + \frac{1}{4} + \frac{1}{8} + \dots$ ist kein Algorithmus!)

Algorithmen: Eigenschaften

- Ein Algorithmus ist
 - eindeutig: die einzelnen Schritte und ihre Abfolge sind unmissverständlich beschrieben
 - allgemein: es wird nicht nur eine Probleminstanz, sondern alle Instanzen eines Problems gelöst
 - ausführbar: der „Prozessor“ muss die Einzelschritte abarbeiten können
 - endlich: seine Beschreibung besteht aus einem Text endlicher Länge ($1 + \frac{1}{2} + \frac{1}{4} + \frac{1}{8} + \dots$ ist kein Algorithmus!)
- Die Theoretische Informatik beschäftigt sich u.a. mit der Frage, welche Problemstellungen algorithmisch, d.h. mittels genau definierter Handlungsanweisungen, gelöst werden können und wie rechenaufwendig Lösungen gegebener Probleme mindestens sind.

Algorithmen: weitere Definitionen I

In (Knuth 1973) finden wir folgende Definition: „Ein Algorithmus muss nach endlich vielen Schritten enden. Jeder Schritt eines Algorithmus muss exakt beschrieben sein; die in ihm verlangten Aktionen müssen präzise formuliert und in jedem Falle eindeutig interpretierbar sein. Ein Algorithmus hat keine, eine oder mehrere Eingangsgrößen, d.h. Größen, die von ihm benutzt und deren Werte vor Beginn seiner Ausführung festgelegt werden müssen. Ein Algorithmus hat eine oder mehrere Ergebnisgrößen, d.h. Größen, deren Werte in Abhängigkeit von den Eingangsgrößen während der Ausführung des Algorithmus berechnet werden. Ein Algorithmus muss so geartet sein, dass die in ihm verlangten Aktionen im Prinzip von einem Menschen in endlicher Zeit mit Papier und Bleistift ausgeführt werden können.“

Algorithmen: weitere Definitionen II

Aho, Hopcroft und Ullman (1975) geben folgende Definition an:

„Ein Algorithmus ist eine endliche Folge von Instruktionen, die alle eindeutig interpretierbar und mit endlichem Aufwand in endlicher Zeit ausführbar sind. Algorithmen enthalten Instruktionen zur Formulierung von (beliebig vielen) Wiederholungen anderer Instruktionen. Unabhängig von den Werten der Eingangsgrößen endet ein Algorithmus stets nach endlich vielen Instruktionsschritten. Ein Programm ist dann ein Algorithmus, wenn für alle möglichen Eingabewerte sichergestellt ist, dass keine Instruktion unendlich oft wiederholt wird.“

Exakte Algorithmen

exakter Algorithmus (Definition)

Ein *exakter/optimierender* Algorithmus (exaktes/optimierendes Verfahren) A zur Lösung eines Optimierungsproblems Π bestimmt für jede Instanz $P = (X, c) \in \Pi$ ein Optimum $x^* \in X$.

Exakte Algorithmen

exakter Algorithmus (Definition)

Ein *exakter/optimierender* Algorithmus (exaktes/optimierendes Verfahren) A zur Lösung eines Optimierungsproblems Π bestimmt für jede Instanz $P = (X, c) \in \Pi$ ein Optimum $x^* \in X$.

- Aufwand hierzu betrage $time_A(P)$ Rechenschritte und $space_A(P)$ für den verwendeten Speicher.

Exakte Algorithmen

exakter Algorithmus (Definition)

Ein *exakter/optimierender* Algorithmus (exaktes/optimierendes Verfahren) A zur Lösung eines Optimierungsproblems Π bestimmt für jede Instanz $P = (X, c) \in \Pi$ ein Optimum $x^* \in X$.

- Aufwand hierzu betrage $time_A(P)$ Rechenschritte und $space_A(P)$ für den verwendeten Speicher.
- Man möchte den Aufwand in der Regel abhängig von der Größe n der Instanz P messen.

Laufzeitkomplexität

Laufzeitkomplexität (Definition)

Ein Algorithmus A hat *Laufzeitkomplexität* $\mathcal{O}(f(n))$, falls es Konstanten k_1 und k_2 gibt, so dass

$$\text{time}_A(P) \leq k_1 + k_2 f(n)$$

für **alle** Instanzen $P = (X, c)$ mit Größe $|P| = n$ gilt.

Laufzeitkomplexität

Laufzeitkomplexität (Definition)

Ein Algorithmus A hat *Laufzeitkomplexität* $\mathcal{O}(f(n))$, falls es Konstanten k_1 und k_2 gibt, so dass

$$\text{time}_A(P) \leq k_1 + k_2 f(n)$$

für alle Instanzen $P = (X, c)$ mit Größe $|P| = n$ gilt.

Beispiel: Angenommen, die Anzahl der Rechenschritte $\text{time}_A(n)$ eines Algorithmus variiert zwischen $7n^4 + 3n + 17$ und $7n^{5-1/n} + 4n + 22$. Dann gilt:

Laufzeitkomplexität

Laufzeitkomplexität (Definition)

Ein Algorithmus A hat *Laufzeitkomplexität* $\mathcal{O}(f(n))$, falls es Konstanten k_1 und k_2 gibt, so dass

$$\text{time}_A(P) \leq k_1 + k_2 f(n)$$

für alle Instanzen $P = (X, c)$ mit Größe $|P| = n$ gilt.

Beispiel: Angenommen, die Anzahl der Rechenschritte $\text{time}_A(n)$ eines Algorithmus variiert zwischen $7n^4 + 3n + 17$ und $7n^{5-1/n} + 4n + 22$. Dann gilt: $\text{time}_A(n) = \mathcal{O}(n^5)$.

Laufzeitkomplexität

Laufzeitkomplexität (Definition)

Ein Algorithmus A hat *Laufzeitkomplexität* $\mathcal{O}(f(n))$, falls es Konstanten k_1 und k_2 gibt, so dass

$$\text{time}_A(P) \leq k_1 + k_2 f(n)$$

für alle Instanzen $P = (X, c)$ mit Größe $|P| = n$ gilt.

Beispiel: Angenommen, die Anzahl der Rechenschritte $\text{time}_A(n)$ eines Algorithmus variiert zwischen $7n^4 + 3n + 17$ und $7n^{5-1/n} + 4n + 22$. Dann gilt: $\text{time}_A(n) = \mathcal{O}(n^5)$.

Laufzeitkomplexität des Nearest Neighbor Verfahrens für TSP?

Effiziente Algorithmen

Effizienter Algorithmus (Definition)

Ein Algorithmus A zur Lösung eines kombinatorischen Optimierungsproblems Π mit Laufzeitkomplexität $time_A(P) = \mathcal{O}(f(n))$ für ein Polynom f heißt *effizienter Algorithmus*.

Ist die Laufzeitkomplexität nicht durch irgendein Polynom abschätzbar, so heißt A *nicht-effizienter Algorithmus*.

Effiziente Algorithmen

Effizienter Algorithmus (Definition)

Ein Algorithmus A zur Lösung eines kombinatorischen Optimierungsproblems Π mit Laufzeitkomplexität $time_A(P) = \mathcal{O}(f(n))$ für ein Polynom f heißt *effizienter Algorithmus*.

Ist die Laufzeitkomplexität nicht durch irgendein Polynom abschätzbar, so heißt A *nicht-effizienter Algorithmus*.

Beispiele:

- $time_A(n) = 2n^2 + 17n + 99$

Effiziente Algorithmen

Effizienter Algorithmus (Definition)

Ein Algorithmus A zur Lösung eines kombinatorischen Optimierungsproblems Π mit Laufzeitkomplexität $time_A(P) = \mathcal{O}(f(n))$ für ein Polynom f heißt *effizienter Algorithmus*.

Ist die Laufzeitkomplexität nicht durch irgendein Polynom abschätzbar, so heißt A *nicht-effizienter Algorithmus*.

Beispiele:

- $time_A(n) = 2n^2 + 17n + 99 = \mathcal{O}(n^2) \rightarrow$ effizienter Algorithmus

Effiziente Algorithmen

Effizienter Algorithmus (Definition)

Ein Algorithmus A zur Lösung eines kombinatorischen Optimierungsproblems Π mit Laufzeitkomplexität $time_A(P) = \mathcal{O}(f(n))$ für ein Polynom f heißt *effizienter Algorithmus*.

Ist die Laufzeitkomplexität nicht durch irgendein Polynom abschätzbar, so heißt A *nicht-effizienter Algorithmus*.

Beispiele:

- $time_A(n) = 2n^2 + 17n + 99 = \mathcal{O}(n^2) \rightarrow$ effizienter Algorithmus
- $time_A(n) = 7n^{99} + 99n^7$

Effiziente Algorithmen

Effizienter Algorithmus (Definition)

Ein Algorithmus A zur Lösung eines kombinatorischen Optimierungsproblems Π mit Laufzeitkomplexität $time_A(P) = \mathcal{O}(f(n))$ für ein Polynom f heißt *effizienter Algorithmus*.

Ist die Laufzeitkomplexität nicht durch irgendein Polynom abschätzbar, so heißt A *nicht-effizienter Algorithmus*.

Beispiele:

- $time_A(n) = 2n^2 + 17n + 99 = \mathcal{O}(n^2) \rightarrow$ effizienter Algorithmus
- $time_A(n) = 7n^{99} + 99n^7$
- $time_A(n) = n + 2$

Effiziente Algorithmen

Effizienter Algorithmus (Definition)

Ein Algorithmus A zur Lösung eines kombinatorischen Optimierungsproblems Π mit Laufzeitkomplexität $time_A(P) = \mathcal{O}(f(n))$ für ein Polynom f heißt *effizienter Algorithmus*.

Ist die Laufzeitkomplexität nicht durch irgendein Polynom abschätzbar, so heißt A *nicht-effizienter Algorithmus*.

Beispiele:

- $time_A(n) = 2n^2 + 17n + 99 = \mathcal{O}(n^2) \rightarrow$ effizienter Algorithmus
- $time_A(n) = 7n^{99} + 99n^7$
- $time_A(n) = n + 2$
- $time_A(n) = 2^n + 3n^2 + 88$

Effiziente Algorithmen

Effizienter Algorithmus (Definition)

Ein Algorithmus A zur Lösung eines kombinatorischen Optimierungsproblems Π mit Laufzeitkomplexität $time_A(P) = \mathcal{O}(f(n))$ für ein Polynom f heißt *effizienter Algorithmus*.

Ist die Laufzeitkomplexität nicht durch irgendein Polynom abschätzbar, so heißt A *nicht-effizienter Algorithmus*.

Beispiele:

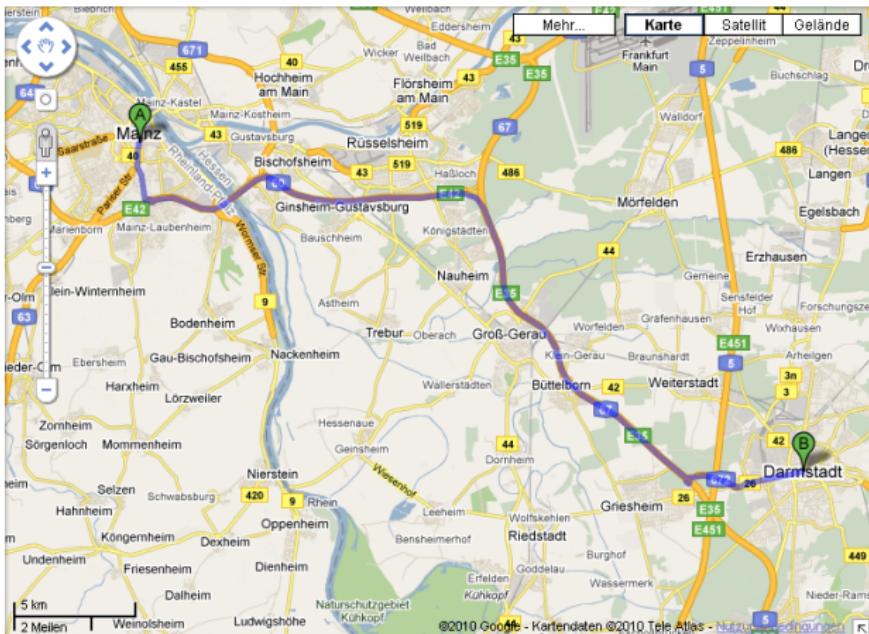
- $time_A(n) = 2n^2 + 17n + 99 = \mathcal{O}(n^2) \rightarrow$ effizienter Algorithmus
- $time_A(n) = 7n^{99} + 99n^7$
- $time_A(n) = n + 2$
- $time_A(n) = 2^n + 3n^2 + 88$
- $time_A(n) = n^{1+\log(n)}$

Exkurs: Die Schwierigkeit von Optimierungsproblemen am Beispiel der Transportplanung

■ Lernziele

- Intuitives Verständnis der Schwierigkeit von Optimierungsproblemen
- Unterscheidung effizient lösbarer und schwieriger Probleme

Problem 1: Shortest Path Problem



„Wie komme ich am schnellsten von Punkt A zu Punkt B?“

Shortest Path Problem – Anwendungen

- Software oder Websites zur Reise- und Fahrtenplanung (sog. Routenplaner)
- Geographischen Informationssysteme (GIS) → Distanzmatrizen als Basisdaten für weiterführende Planungsprobleme der Transportlogistik
- Teilprobleme in vielen anderen Optimierungsproblemen (z. B. in der Tourenplanung)

Shortest Path Problem – Anzahl Lösungen

- Anzahl Wege von Quelle s zu Senke t in vollständigem ungerichteten Graphen mit n Knoten:

$$p(n) = \sum_{k=0}^{n-2} \binom{n-2}{k} \cdot k! = (n-2)! \cdot \sum_{k=0}^{n-2} \frac{1}{(n-2-k)!}$$

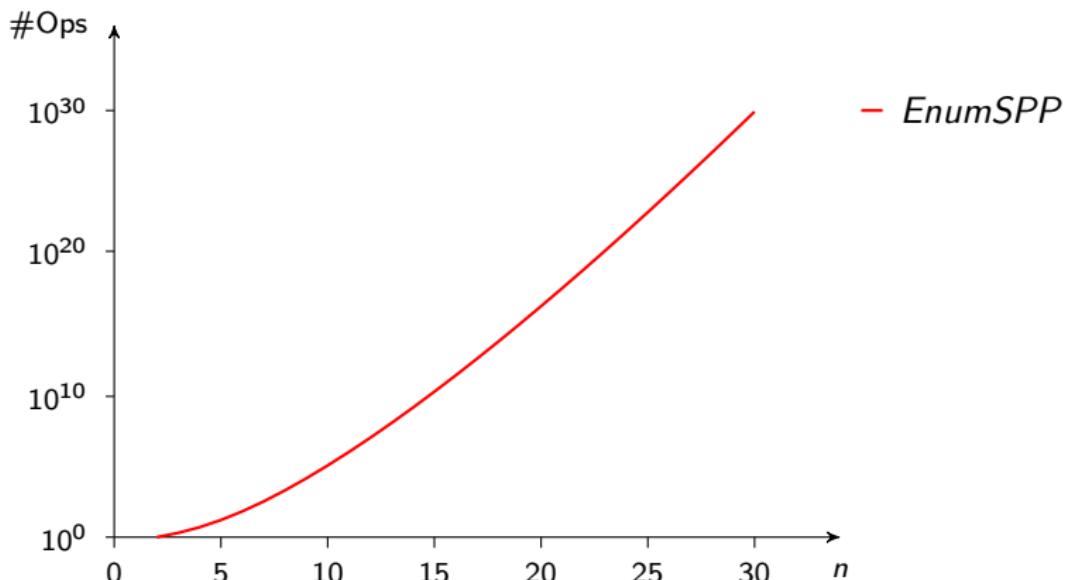
Shortest Path Problem – Anzahl Lösungen

- Anzahl Wege von Quelle s zu Senke t in vollständigem ungerichteten Graphen mit n Knoten:

$$p(n) = \sum_{k=0}^{n-2} \binom{n-2}{k} \cdot k! = (n-2)! \cdot \sum_{k=0}^{n-2} \frac{1}{(n-2-k)!}$$

- Moderner Desktop-Rechner
 - Operationen pro Sekunde: $16 \cdot 10^9$
 - Annahme: Generierung und Bewertung einer Lösung mit einem Rechenschritt
 - Anzahl Rechenschritte (#Ops) gleich Anzahl möglicher Lösungen

Rechenoperationen in Abhängigkeit der Knotenanzahl



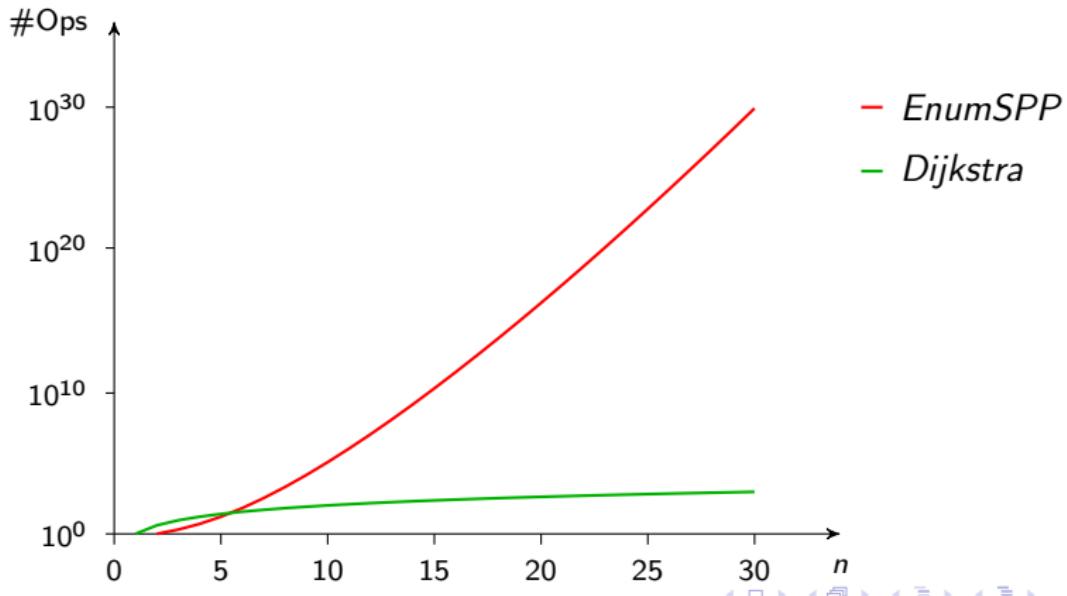
- Rechenzeit: $t_{CPU}(30) = 1\,642\,512\,618\,252$ Jahre

Shortest Path Problem – Schwierigkeit

- Effizient lösbar: z. B. mit dem Dijkstra Algorithmus
- Anzahl benötigter Rechenschritte: n^2

Shortest Path Problem – Schwierigkeit

- Effizient lösbar: z. B. mit dem Dijkstra Algorithmus
- Anzahl benötigter Rechenschritte: n^2



Problem 2: Traveling Salesman Problem (TSP)



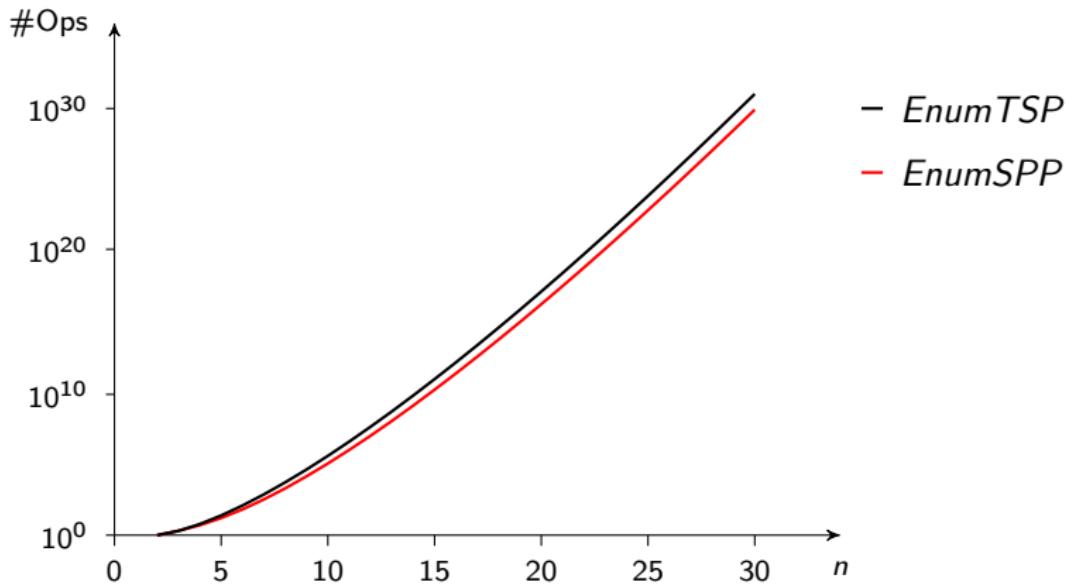
„In welcher Reihenfolge muss der Handlungsreisende die Orte besuchen, so dass er alle Orte genau einmal besucht, er zum Ursprungsort zurückkehrt und dabei eine möglichst kurze Strecke zurücklegt?“

TSP – Schwierigkeit I

- Anzahl der Touren eines TSP mit n Städten ist $(n - 1)!$

TSP – Schwierigkeit I

- Anzahl der Touren eines TSP mit n Städten ist $(n - 1)!$

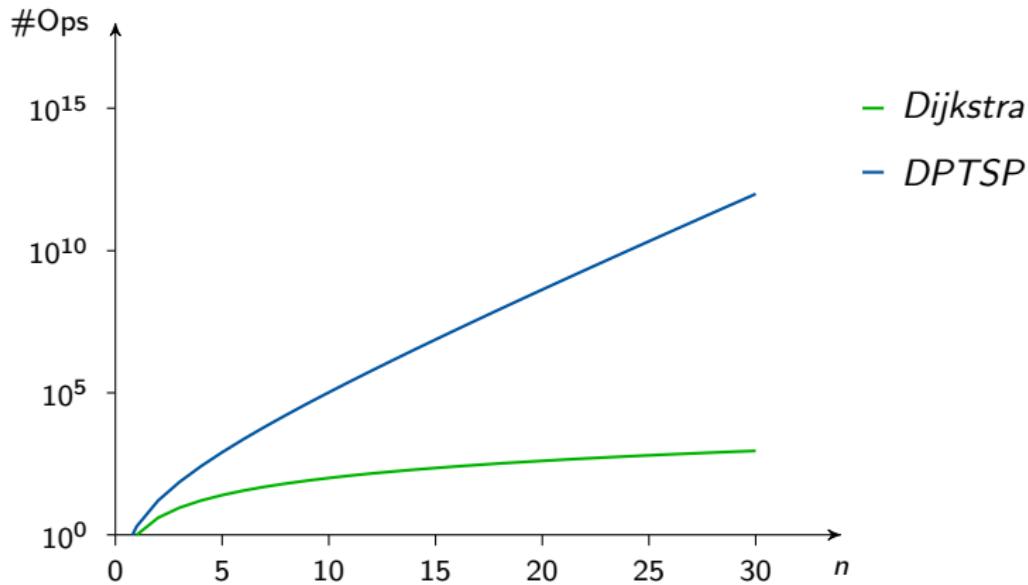


TSP – Schwierigkeit II

- Der beste bekannte exakte Algorithmus benötigt $n^2 \cdot 2^n$ Schritte

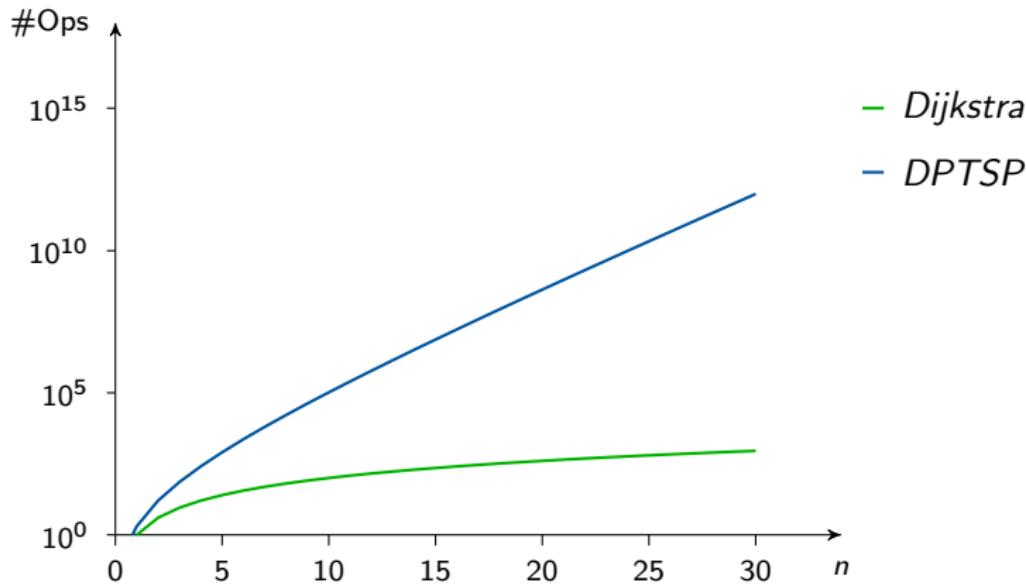
TSP – Schwierigkeit II

- Der beste bekannte exakte Algorithmus benötigt $n^2 \cdot 2^n$ Schritte



TSP – Schwierigkeit II

- Der beste bekannte exakte Algorithmus benötigt $n^2 \cdot 2^n$ Schritte



→ Das TSP ist ein schwieriges Problem!

Schwierigkeit von Optimierungsproblemen

- Große Anzahl zulässiger Lösungen eines Optimierungsproblems (z. B. exponentiell viele) → **kein** Argument für Schwierigkeit

Schwierigkeit von Optimierungsproblemen

- Große Anzahl zulässiger Lösungen eines Optimierungsproblems (z. B. exponentiell viele) → **kein** Argument für Schwierigkeit
- Schwierigkeit eines Optimierungsproblems:
 - bestimmt durch Laufzeitkomplexität (Anzahl benötigter Rechenschritte in Abhängigkeit der Problemgröße) des besten bekannten exakten Algorithmus
 - polynomiell beschränktes Wachstum: effizient lösbare Probleme
 - nicht polynomiell beschränktes Wachstum: schwierige Probleme

Schwierigkeit von Optimierungsproblemen

- Große Anzahl zulässiger Lösungen eines Optimierungsproblems (z. B. exponentiell viele) → **kein** Argument für Schwierigkeit
- Schwierigkeit eines Optimierungsproblems:
 - bestimmt durch Laufzeitkomplexität (Anzahl benötigter Rechenschritte in Abhängigkeit der Problemgröße) des besten bekannten exakten Algorithmus
 - polynomiell beschränktes Wachstum: effizient lösbare Probleme
 - nicht polynomiell beschränktes Wachstum: schwierige Probleme
- Sehr viele „interessanten Probleme“ im Operations Management sind schwierig! Ende Exkurs

Komplexitätsklassen \mathcal{P} und \mathcal{NP}

Klasse \mathcal{P} (Definition)

Die Klasse \mathcal{P} besteht aus allen kombinatorischen Optimierungsproblemen Π , für die es einen *deterministischen*^a effizienten Algorithmus A gibt.

^aDas zugrunde liegende Rechnermodell ist das einer deterministischen Turing-Maschine.

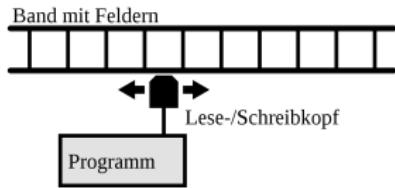
Komplexitätsklassen \mathcal{P} und \mathcal{NP}

Klasse \mathcal{P} (Definition)

Die Klasse \mathcal{P} besteht aus allen kombinatorischen Optimierungsproblemen Π , für die es einen *deterministischen*^a effizienten Algorithmus A gibt.

^aDas zugrunde liegende Rechnermodell ist das einer deterministischen Turing-Maschine.

Turing Maschine:



[http://de.wikipedia.org/
wiki/Turingmaschine](http://de.wikipedia.org/wiki/Turingmaschine)

[http://www.matheprisma.
uni-wuppertal.de/Module/
Turing/](http://www.matheprisma.uni-wuppertal.de/Module/Turing/)

Komplexitätsklassen \mathcal{P} und \mathcal{NP}

Im Gegensatz dazu kann ein **nicht-deterministischer Computer (Turing-Maschine)** an bestimmten Stellen des Programms „raten“, an welcher Stelle die Bearbeitung fortgesetzt werden soll.

Komplexitätsklassen \mathcal{P} und \mathcal{NP}

Im Gegensatz dazu kann ein **nicht-deterministischer Computer (Turing-Maschine)** an bestimmten Stellen des Programms „raten“, an welcher Stelle die Bearbeitung fortgesetzt werden soll.

Ein nicht-deterministischer Algorithmus A zur Lösung eines kombinatorischen Optimierungsproblems arbeitet also in zwei Stufen:

- 1 Raten einer Lösung x
- 2 Feststellen, ob die geratene Lösung x eine zulässige Lösung ist, also $x \in X$ gilt

Komplexitätsklassen \mathcal{P} und \mathcal{NP}

Klasse \mathcal{NP} (Definition)

Die Klasse \mathcal{NP} besteht aus allen kombinatorischen Optimierungsproblemen Π , für die es einen *nicht-deterministischen*^a Algorithmus B gibt, so dass

$$\text{time}_B(P) \leq f(n)$$

für ein Polynom f und alle Instanzen P mit Größe $|P| = n$ gilt.

^aDas zugrunde liegende Rechnermodell ist das einer nicht-deterministischen Turing-Maschine.

Komplexitätsklassen \mathcal{P} und \mathcal{NP}

Klasse \mathcal{NP} (Definition)

Die Klasse \mathcal{NP} besteht aus allen kombinatorischen Optimierungsproblemen Π , für die es einen *nicht-deterministischen*^a Algorithmus B gibt, so dass

$$\text{time}_B(P) \leq f(n)$$

für ein Polynom f und alle Instanzen P mit Größe $|P| = n$ gilt.

^aDas zugrunde liegende Rechnermodell ist das einer nicht-deterministischen Turing-Maschine.

Klar ist: $\mathcal{P} \subseteq \mathcal{NP}$

Komplexitätsklassen \mathcal{P} und \mathcal{NP}

Klasse \mathcal{NP} (Definition)

Die Klasse \mathcal{NP} besteht aus allen kombinatorischen Optimierungsproblemen Π , für die es einen *nicht-deterministischen*^a Algorithmus B gibt, so dass

$$\text{time}_B(P) \leq f(n)$$

für ein Polynom f und alle Instanzen P mit Größe $|P| = n$ gilt.

^aDas zugrunde liegende Rechnermodell ist das einer nicht-deterministischen Turing-Maschine.

Klar ist: $\mathcal{P} \subseteq \mathcal{NP}$

Allgemein vermutet man: $\mathcal{P} \neq \mathcal{NP}$

Komplexitätsklassen \mathcal{P} und \mathcal{NP}

Klasse \mathcal{NP} (Definition)

Die Klasse \mathcal{NP} besteht aus allen kombinatorischen Optimierungsproblemen Π , für die es einen *nicht-deterministischen*^a Algorithmus B gibt, so dass

$$\text{time}_B(P) \leq f(n)$$

für ein Polynom f und alle Instanzen P mit Größe $|P| = n$ gilt.

^aDas zugrunde liegende Rechnermodell ist das einer nicht-deterministischen Turing-Maschine.

Klar ist: $\mathcal{P} \subseteq \mathcal{NP}$

Allgemein vermutet man: $\mathcal{P} \neq \mathcal{NP}$

Folgerung?

Komplexitätsklassen \mathcal{P} und \mathcal{NP}

Klasse \mathcal{NP} (Definition)

Die Klasse \mathcal{NP} besteht aus allen kombinatorischen Optimierungsproblemen Π , für die es einen *nicht-deterministischen*^a Algorithmus B gibt, so dass

$$\text{time}_B(P) \leq f(n)$$

für ein Polynom f und alle Instanzen P mit Größe $|P| = n$ gilt.

^aDas zugrunde liegende Rechnermodell ist das einer nicht-deterministischen Turing-Maschine.

Klar ist: $\mathcal{P} \subseteq \mathcal{NP}$

Allgemein vermutet man: $\mathcal{P} \neq \mathcal{NP}$

Folgerung? Falls die Vermutung richtig ist, gibt es „schwierige“ Optimierungsprobleme, welche in polynomialer Zeit mit einem nicht-deterministischen, jedoch nicht mit einem deterministischen Algorithmus gelöst werden können.

Komplexitätsklassen \mathcal{P} und \mathcal{NP}

Einige (teilweise in der Voranstaltung Logistikplanung bereits behandelte) Optimierungsprobleme können effizient gelöst werden, für die meisten „interessanten Probleme“ ist kein deterministischer effizienter Algorithmus bekannt:

Zur Klasse \mathcal{P} gehören:

Komplexitätsklassen \mathcal{P} und \mathcal{NP}

Einige (teilweise in der Voranstaltung Logistikplanung bereits behandelte) Optimierungsprobleme können effizient gelöst werden, für die meisten „interessanten Probleme“ ist kein deterministischer effizienter Algorithmus bekannt:

Zur Klasse \mathcal{P} gehören:

- Minimal spannender Baum
- Kürzeste-Wege-Problem
- Lineare Optimierungsprobleme,
LP (Achtung: Der Simplex-Algorithmus ist kein polynomieller Algorithmus! Aber:
Interior Point Methoden garantieren polynomielle Laufzeit)

Komplexitätsklassen \mathcal{P} und \mathcal{NP}

Einige (teilweise in der Voranstaltung Logistikplanung bereits behandelte) Optimierungsprobleme können effizient gelöst werden, für die meisten „interessanten Probleme“ ist kein deterministischer effizienter Algorithmus bekannt:

Zur Klasse \mathcal{P} gehören:

- Minimal spannender Baum
- Kürzeste-Wege-Problem
- Lineare Optimierungsprobleme,
LP (Achtung: Der Simplex-Algorithmus ist kein polynomieller Algorithmus! Aber: Interior Point Methoden garantieren polynomielle Laufzeit)

Zur Klasse \mathcal{NP} gehören:

Komplexitätsklassen \mathcal{P} und \mathcal{NP}

Einige (teilweise in der Voranstaltung Logistikplanung bereits behandelte) Optimierungsprobleme können effizient gelöst werden, für die meisten „interessanten Probleme“ ist kein deterministischer effizienter Algorithmus bekannt:

Zur Klasse \mathcal{P} gehören:

- Minimal spannender Baum
- Kürzeste-Wege-Problem
- Lineare Optimierungsprobleme,
LP (Achtung: Der Simplex-Algorithmus ist kein polynomieller Algorithmus! Aber: Interior Point Methoden garantieren polynomielle Laufzeit)

Zur Klasse \mathcal{NP} gehören:

- binäres und ganzzahliges Rucksackproblem
- TSP
- Bin Packing Problem
- ... u.v.m.

Komplexitätsklassen \mathcal{P} und \mathcal{NP}

Da es bisher nicht gelungen ist $\mathcal{P} \neq \mathcal{NP}$ zu zeigen, d.h. nachzuweisen, dass es schwierige Probleme $\Pi \in \mathcal{NP} \setminus \mathcal{P}$ gibt, möchte man zumindest eine Menge von besonders schwierigen Problemen in \mathcal{NP} identifizieren.

Komplexitätsklassen \mathcal{P} und \mathcal{NP}

Da es bisher nicht gelungen ist $\mathcal{P} \neq \mathcal{NP}$ zu zeigen, d.h. nachzuweisen, dass es schwierige Probleme $\Pi \in \mathcal{NP} \setminus \mathcal{P}$ gibt, möchte man zumindest eine Menge von besonders schwierigen Problemen in \mathcal{NP} identifizieren.

Dazu dienen die folgenden Konzepte:

- polynomiale **Reduzierbarkeit** eines Problems auf ein anderes
- Menge der **\mathcal{NP} -schweren** Probleme
- Menge der **\mathcal{NP} -vollständigen** Probleme

Polynomiale Reduzierbarkeit

Gegeben seien zwei Probleme Π_1 und Π_2 .

Polynomiale Reduzierbarkeit

Gegeben seien zwei Probleme Π_1 und Π_2 .

Es sei f eine Funktion, die Instanzen von Π_2 in Instanzen von Π_1 transformiert.

Polynomiale Reduzierbarkeit

Gegeben seien zwei Probleme Π_1 und Π_2 .

Es sei f eine Funktion, die Instanzen von Π_2 in Instanzen von Π_1 transformiert.

Es sei g eine Funktion, die Lösungen von Π_1 in Lösungen von Π_2 transformiert.

Polynomiale Reduzierbarkeit

Gegeben seien zwei Probleme Π_1 und Π_2 .

Es sei f eine Funktion, die Instanzen von Π_2 in Instanzen von Π_1 transformiert.

Es sei g eine Funktion, die Lösungen von Π_1 in Lösungen von Π_2 transformiert.

Dann liefert jeder Algorithmus A_1 zur Lösung von Π_1 mit f und g auch einen Algorithmus $g \circ A_1 \circ f$ zur Lösung von Π_2 .

Polynomiale Reduzierbarkeit

Gegeben seien zwei Probleme Π_1 und Π_2 .

Es sei f eine Funktion, die Instanzen von Π_2 in Instanzen von Π_1 transformiert.

Es sei g eine Funktion, die Lösungen von Π_1 in Lösungen von Π_2 transformiert.

Dann liefert jeder Algorithmus A_1 zur Lösung von Π_1 mit f und g auch einen Algorithmus $g \circ A_1 \circ f$ zur Lösung von Π_2 .

polynomiale Reduzierbarkeit (Definition)

Das Problem Π_2 heißt *polynomial reduzierbar* auf das Problem Π_1 , wenn es polynomial berechenbare Funktionen f und g gibt, wobei

- (i) f Instanzen von Π_2 in Instanzen von Π_1 transformiert und
- (ii) g Lösungen von Π_1 in Lösungen von Π_2 transformiert.

\mathcal{NP} -schwere und \mathcal{NP} -vollständige Probleme

\mathcal{NP} -schweres Problem (Definition)

Ein Problem Π heißt \mathcal{NP} -schwer, falls sich **jedes** Problem $\Pi' \in \mathcal{NP}$ polynomial auf das Problem Π reduzieren lässt.

\mathcal{NP} -schwere und \mathcal{NP} -vollständige Probleme

\mathcal{NP} -schweres Problem (Definition)

Ein Problem Π heißt \mathcal{NP} -schwer, falls sich **jedes** Problem $\Pi' \in \mathcal{NP}$ polynomial auf das Problem Π reduzieren lässt.

Man beachte, dass Π irgendein Problem sein kann, insbesondere muss Π selbst nicht unbedingt zu \mathcal{NP} gehören. Dennoch stellt sich die Frage, wie man überhaupt zeigen kann, dass sich jedes (erdenkliche) Problem in \mathcal{NP} auf Π reduzieren lässt.

\mathcal{NP} -schwere und \mathcal{NP} -vollständige Probleme

\mathcal{NP} -schweres Problem (Definition)

Ein Problem Π heißt \mathcal{NP} -schwer, falls sich **jedes** Problem $\Pi' \in \mathcal{NP}$ polynomial auf das Problem Π reduzieren lässt.

Man beachte, dass Π irgendein Problem sein kann, insbesondere muss Π selbst nicht unbedingt zu \mathcal{NP} gehören. Dennoch stellt sich die Frage, wie man überhaupt zeigen kann, dass sich jedes (erdenkliche) Problem in \mathcal{NP} auf Π reduzieren lässt.

\mathcal{NP} -vollständiges Problem (Definition)

Ein Problem Π heißt \mathcal{NP} -vollständig, falls Π \mathcal{NP} -schwer ist und $\Pi \in \mathcal{NP}$ gilt.

\mathcal{NP} -schwere und \mathcal{NP} -vollständige Probleme

\mathcal{NP} -schweres Problem (Definition)

Ein Problem Π heißt \mathcal{NP} -schwer, falls sich **jedes** Problem $\Pi' \in \mathcal{NP}$ polynomial auf das Problem Π reduzieren lässt.

Man beachte, dass Π irgendein Problem sein kann, insbesondere muss Π selbst nicht unbedingt zu \mathcal{NP} gehören. Dennoch stellt sich die Frage, wie man überhaupt zeigen kann, dass sich jedes (erdenkliche) Problem in \mathcal{NP} auf Π reduzieren lässt.

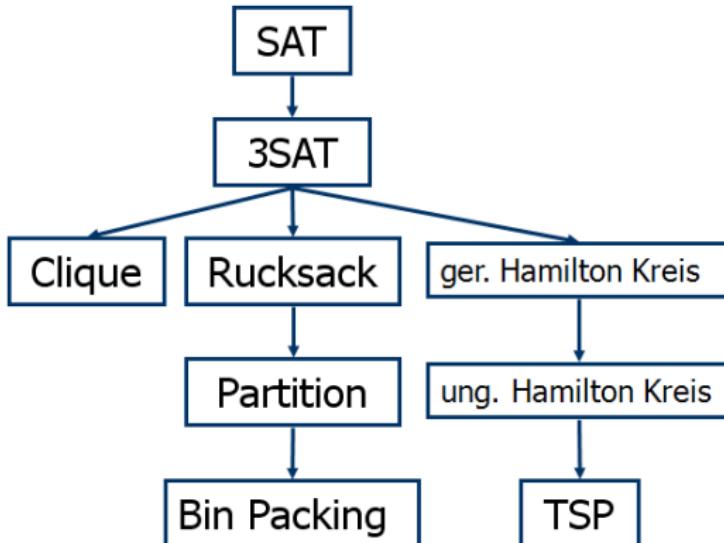
\mathcal{NP} -vollständiges Problem (Definition)

Ein Problem Π heißt \mathcal{NP} -vollständig, falls Π \mathcal{NP} -schwer ist und $\Pi \in \mathcal{NP}$ gilt.

Berühmtes Resultat von S.A. Cook (1971): Das Problem **SAT** (Erfüllbarkeitsproblem der Aussagenlogik) ist ein Entscheidungsproblem, welches \mathcal{NP} -vollständig ist.

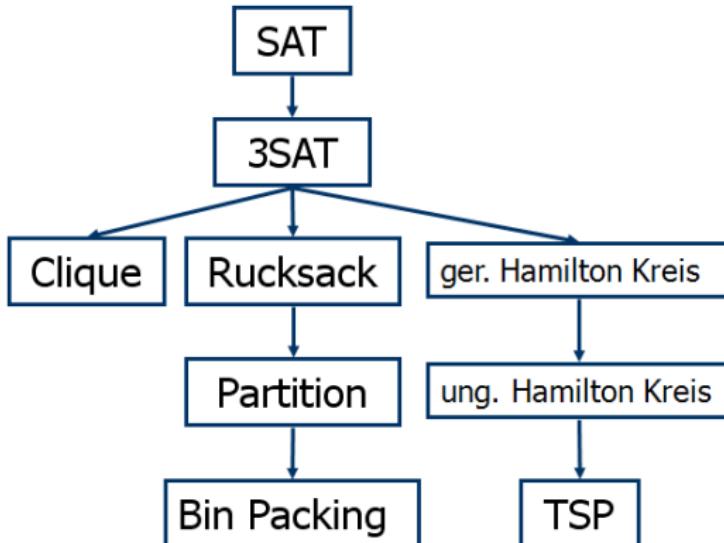
\mathcal{NP} -schwere und \mathcal{NP} -vollständige Probleme

\mathcal{NP} -vollständige Entscheidungsprobleme, bewiesen durch Reduktion:



\mathcal{NP} -schwere und \mathcal{NP} -vollständige Probleme

\mathcal{NP} -vollständige Entscheidungsprobleme, bewiesen durch Reduktion:



Siehe auch: A compendium of \mathcal{NP} optimization problems

<http://www.nada.kth.se/~viggo/problemList/compendium.html>

Agenda

1 Grundlagen

- Operations Research und kombinatorische Optimierung
- Algorithmen, Aufwand und Komplexitätsklassen
- Heuristiken

Heuristiken

Konsequenz aus $P \neq NP$ -Vermutung: praxisrelevante Problemstellungen sind u.U. zu „schwierig“, als dass sie durch einen exakten Optimierungsalgorithmus mit akzeptablem Aufwand gelöst werden könnten.

Heuristiken

Konsequenz aus $P \neq NP$ -Vermutung: praxisrelevante Problemstellungen sind u.U. zu „schwierig“, als dass sie durch einen exakten Optimierungsalgorithmus mit akzeptablem Aufwand gelöst werden könnten. [Wieso unter Umständen?](#)

Heuristiken

Konsequenz aus $P \neq NP$ -Vermutung: praxisrelevante Problemstellungen sind u.U. zu „schwierig“, als dass sie durch einen exakten Optimierungsalgorithmus mit akzeptablem Aufwand gelöst werden könnten. [Wieso unter Umständen?](#)

Wünschenswert wären Algorithmen, die zugleich alle drei folgenden Prinzipien erfüllen:

Heuristiken

Konsequenz aus $P \neq NP$ -Vermutung: praxisrelevante Problemstellungen sind u.U. zu „schwierig“, als dass sie durch einen exakten Optimierungsalgorithmus mit akzeptablem Aufwand gelöst werden könnten. [Wieso unter Umständen?](#)

Wünschenswert wären Algorithmen, die zugleich alle drei folgenden Prinzipien erfüllen:

Allgemeinheit: Der Algorithmus soll alle Instanzen des Problems Π lösen

Heuristiken

Konsequenz aus $\mathcal{P} \neq \mathcal{NP}$ -Vermutung: praxisrelevante Problemstellungen sind u.U. zu „schwierig“, als dass sie durch einen exakten Optimierungsalgorithmus mit akzeptablem Aufwand gelöst werden könnten. [Wieso unter Umständen?](#)

Wünschenswert wären Algorithmen, die zugleich alle drei folgenden Prinzipien erfüllen:

Allgemeinheit: Der Algorithmus soll alle Instanzen des Problems Π lösen

Effizienz: Der Algorithmus soll eine polynomiale Laufzeit haben

Heuristiken

Konsequenz aus $\mathcal{P} \neq \mathcal{NP}$ -Vermutung: praxisrelevante Problemstellungen sind u.U. zu „schwierig“, als dass sie durch einen exakten Optimierungsalgorithmus mit akzeptablem Aufwand gelöst werden könnten. [Wieso unter Umständen?](#)

Wünschenswert wären Algorithmen, die zugleich alle drei folgenden Prinzipien erfüllen:

Allgemeinheit: Der Algorithmus soll alle Instanzen des Problems Π lösen

Effizienz: Der Algorithmus soll eine polynomiale Laufzeit haben

Optimalität: Der Algorithmus soll exakt sein, d.h. eine optimale Lösung bestimmen

Heuristiken

Konsequenz aus $\mathcal{P} \neq \mathcal{NP}$ -Vermutung: praxisrelevante Problemstellungen sind u.U. zu „schwierig“, als dass sie durch einen exakten Optimierungsalgorithmus mit akzeptablem Aufwand gelöst werden könnten. [Wieso unter Umständen?](#)

Wünschenswert wären Algorithmen, die zugleich alle drei folgenden Prinzipien erfüllen:

Allgemeinheit: Der Algorithmus soll alle Instanzen des Problems Π lösen

Effizienz: Der Algorithmus soll eine polynomiale Laufzeit haben

Optimalität: Der Algorithmus soll exakt sein, d.h. eine optimale Lösung bestimmen

Existenz?

Heuristiken

Konsequenz aus $\mathcal{P} \neq \mathcal{NP}$ -Vermutung: praxisrelevante Problemstellungen sind u.U. zu „schwierig“, als dass sie durch einen exakten Optimierungsalgorithmus mit akzeptablem Aufwand gelöst werden könnten. [Wieso unter Umständen?](#)

Wünschenswert wären Algorithmen, die zugleich alle drei folgenden Prinzipien erfüllen:

Allgemeinheit: Der Algorithmus soll alle Instanzen des Problems Π lösen

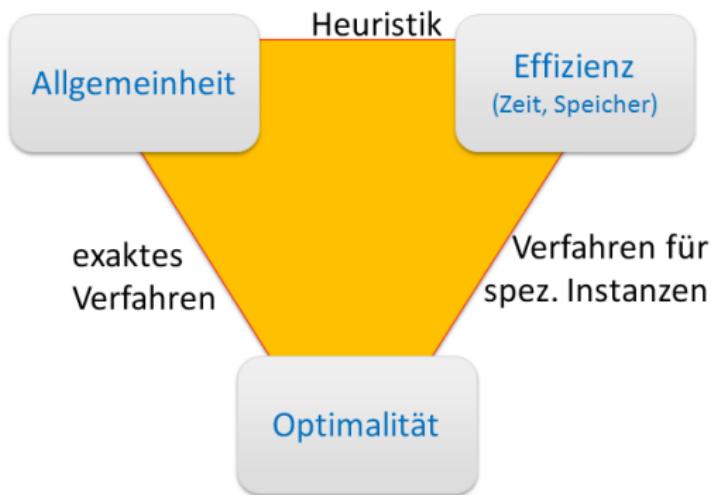
Effizienz: Der Algorithmus soll eine polynomiale Laufzeit haben

Optimalität: Der Algorithmus soll exakt sein, d.h. eine optimale Lösung bestimmen

Existenz? Falls $\mathcal{P} \neq \mathcal{NP}$, gibt es solche Algorithmen nicht!

Heuristiken

- Weicht man von einem der drei Prinzipien ab, erhält man:



Heuristiken

Wiederholung...

Heuristik (Definition)

Heuristiken sind Algorithmen, die ein gegebenes Optimierungsproblem mit akzeptablem Aufwand möglichst gut zu lösen versuchen.

Die Begriffe „gut lösen“, „Güte von Lösungen“ oder „Lösungsqualität“ werden später in der Veranstaltung präzisiert.

Begriff Heuristik

Zum Wort Heuristik und seiner Geschichte:

Das Wort Heuristik wurde aus dem griechischen Verb „heuriskein“ hergeleitet und bedeutet *finden* oder *entdecken*. Es wird erzählt, dass Archimedes laut „Heureka“ (Ich habe es herausgefunden!) gerufen hat, als er entdeckte, dass der Schmied den König mit der Krone betrogen hatte.

Spätere Generationen wandelten das Wort „Heureka“ zu Eureka. In der Geschichte der Künstlichen Intelligenz (KI) durchging die technische Bedeutung des Wortes mehrere Wechsel. Im Jahre 1957 schrieb Georg Polya sein bekanntes Buch „How to Solve It“, welches heuristische Methoden benutzt, die Problemlösungsverfahren und besonders mathematische Beweise finden.

Begriff Heuristik

Einige Leute betrachten die Heuristik als das Gegenteil der Algorithmik. Zum Beispiel haben 1963 Novell, Shaw und Simon festgelegt: „Ein Prozess, welcher ein gegebenes Problem lösen kann, aber keine Lösung garantiert, wird als Heuristik bezeichnet“.

Begriff Heuristik

Einige Leute betrachten die Heuristik als das Gegenteil der Algorithmik. Zum Beispiel haben 1963 Novell, Shaw und Simon festgelegt: „Ein Prozess, welcher ein gegebenes Problem lösen kann, aber keine Lösung garantiert, wird als Heuristik bezeichnet“.

Ein heuristischer Algorithmus hat allerdings nichts mit Zufall oder Nichtdeterminismus zu tun. Er terminiert genau dann, wenn dieser ein Resultat erreicht. In einigen Fällen gibt es keine Garantie, wie lange ein Algorithmus benötigt, um ein Ergebnis zu liefern, und in manchen Fällen ist die Qualität der Lösung nicht garantiert. Nichtsdestoweniger ist es wichtig, zwischen „nicht algorithmisch“ und „nicht-präzise“ zu unterscheiden.

Heuristiken

Gründe für den Einsatz von Heuristiken:

- 1 die zur Verfügung stehende **Zeit** zum exakten Lösen des Problems reicht nicht aus

Heuristiken

Gründe für den Einsatz von Heuristiken:

- 1 die zur Verfügung stehende **Zeit** zum exakten Lösen des Problems reicht nicht aus
- 2 der zur Verfügung stehende **Speicherplatz** zum exakten Lösen des Problems reicht nicht aus

Heuristiken

Gründe für den Einsatz von Heuristiken:

- 1 die zur Verfügung stehende **Zeit** zum exakten Lösen des Problems reicht nicht aus
- 2 der zur Verfügung stehende **Speicherplatz** zum exakten Lösen des Problems reicht nicht aus
- 3 aus praktischer Sicht ferner:
 - Heuristiken sind u.U. **einfacher zu verstehen** als exakte Verfahren
 - die Entwicklung von Heuristiken kann **kostengünstiger** sein, sowohl im Hinblick auf die Implementierung als auch bzgl. Rechenleistung
 - Heuristiken können als **verständliche Lehrmittel** eingesetzt werden, um Anwendern das Potential und die Reichweite von Entscheidungen zu verdeutlichen

Klassifikation von Heuristiken nach der Stoppregel

Während optimierende Algorithmen dann terminieren, wenn der Nachweis der Optimalität der besten bekannten Lösung erfolgt ist, benötigt man bei Heuristiken eine andere – künstliche – Stopp-Regel zur Terminierung der Heuristik.

Klassifikation von Heuristiken nach der Stoppregel

Während optimierende Algorithmen dann terminieren, wenn der Nachweis der Optimalität der besten bekannten Lösung erfolgt ist, benötigt man bei Heuristiken eine andere – künstliche – Stopp-Regel zur Terminierung der Heuristik. In einem engen Zusammenhang dazu steht die

Unterscheidung von:

Eröffnungsverfahren generieren eine (möglichst gute) zulässige Lösung.

Sie terminieren, wenn die zulässige Lösung berechnet wurde. **Synonym:** Konstruktionsverfahren

Klassifikation von Heuristiken nach der Stoppregel

Während optimierende Algorithmen dann terminieren, wenn der Nachweis der Optimalität der besten bekannten Lösung erfolgt ist, benötigt man bei Heuristiken eine andere – künstliche – Stopp-Regel zur Terminierung der Heuristik. In einem engen Zusammenhang dazu steht die

Unterscheidung von:

Eröffnungsverfahren generieren eine (möglichst gute) zulässige Lösung.

Sie terminieren, wenn die zulässige Lösung berechnet wurde. **Synonym:** Konstruktionsverfahren

Verbesserungsverfahren gehen von einer zulässigen Lösung aus und verbessern diese solange, bis ein Stopp-Kriterium erfüllt ist. Typische Stopp-Kriterien sind das Auffinden eines lokalen Minimums und die Durchführung einer vorgegebenen Anzahl an Iterationen.

Klassifikation von Heuristiken nach der Stoppregel

Während optimierende Algorithmen dann terminieren, wenn der Nachweis der Optimalität der besten bekannten Lösung erfolgt ist, benötigt man bei Heuristiken eine andere – künstliche – Stopp-Regel zur Terminierung der Heuristik. In einem engen Zusammenhang dazu steht die

Unterscheidung von:

Eröffnungsverfahren generieren eine (möglichst gute) zulässige Lösung.

Sie terminieren, wenn die zulässige Lösung berechnet wurde. **Synonym:** Konstruktionsverfahren

Verbesserungsverfahren gehen von einer zulässigen Lösung aus und verbessern diese solange, bis ein Stopp-Kriterium erfüllt ist. Typische Stopp-Kriterien sind das Auffinden eines lokalen Minimums und die Durchführung einer vorgegebenen Anzahl an Iterationen.

Zusammengesetzte Verfahren kombinieren ein Eröffnungs- und ein Verbesserungsverfahren. **Synonym:** 2-Stufen-Verfahren.

Zur Vertiefung...



- (Grünert und Irnich, 2005), Abschnitt 3.8, S. 182–190
- (Zimmermann, 2005), Abschnitt 6-6.3

- [Grünert und Irnich 2005] Grünert, T. ; Irnich, S.: *Optimierung im Transport Band I: Grundlagen*. Aachen : Shaker Verlag, 2005
- [Martello und Toth 1990] Martello, S. ; Toth, P.: *Knapsack Problems: Algorithms and Computer Implementations*. New York : Wiley, 1990 (Wiley Interscience Series in Discrete Mathematics and Optimization)
- [Zimmermann 2005] Zimmermann, H.J.: *Operations Research Methoden und Modelle*. Wiesbaden : Vieweg, 2005