Moderne Heuristische Optimierungsverfahren:
Meta-Heuristiken

Dr. Peter Merz

pmerz@informatik.uni-tuebingen.de

Wilhelm-Schickard-Institut für
Informatik - WSI-RA
Sand 1, Raum A 316

Lerninhalte

- **Einführung in Optimierungsprobleme**
- **Lösungsverfahren für kombinatorische und nichtlineare Optimierungsprobleme**
- **Lokale Suchverfahren und deren Vor- und Nachteile**
- **Moderne Ansätze (Meta-Heuristiken) und deren Leistungsfähigkeit**
- **Methoden zur systematischen empirischen Untersuchung von (Meta-)Heuristiken**
- **Anwendungsmöglichkeiten von (Meta-)Heuristiken**

Gründe & „Driving Forces“

- **Kombinatorische/nichtlineare Optimierungsprobleme treten in sehr vielen Anwendungsbereichen auf**
- **Es gibt keine generelle/universelle Lösungsmethode**
- **Neue algorithmische Ideen**
  - Naturinspirierte Suchverfahren
  - Hybride Algorithmen
- **Verständnis des Verhaltens der Algorithmen notwendig**
- **Übertragung der Verfahren auf neue Anwendungsgebiete durch schnelle Hardwareentwicklung möglich**

Inhalte der Vorlesung (1)

Vorläufiger Inhalt:

- **Einleitung**
  - Optimierungsprobleme
  - Exakte Verfahren vs. Heuristiken
  - Klassifikation von Heuristiken
- **Konstruktionsheuristiken**
  - Problemabhängige Heuristiken
  - Greedy-Strategien
- **Nachbarschaftssuche**
  - Lokale Suche
  - Simulated Annealing
  - Tabu Search

Inhalte der Vorlesung (2)

- **Fitnesslandschaften**
  - Modell und Definition
  - Effektivität von Heuristiken
- **Populationsbasierte Heuristiken**
  - Evolutionäre Algorithmen
  - Partikel-Schwärme
  - Populationsbasiertes Inkrementelles Lernen
  - Ameisenkolonien
  - Scatter Search, ...
- **Anwendungsgebiete**
  - Optimierung in der Bioinformatik und Bildverarbeitung

Literatur zur Vorlesung

- **Folien zur Vorlesung im Netz als PDF.**
- **Weitere Literatur:**

  D. Corne, M. Dorigo und F. Glover: *New Ideas in Optimization*, McGraw Hill, 1999.

  Z. Michalewicz und D. B. Fogel: *How to Solve It: Modern Heuristics*, Springer-Verlag, 1999.

  C. Reeves: *Modern Heuristic Techniques for Combinatorial Problems*, McGraw Hill, 1995.

  E.H.L. Aarts, J.K. Lenstra (editors): *Local search in Combinatorial Optimization*, John Wiley & Sons, 1997.

Optimierungsprobleme

- **Was ist ein Optimierungsproblem?**

  - Betriebswirtschaftlich:

    Entscheidungssituation, in der ein vorgegebener Nutzen in kostenminimaler Weise zu realisieren ist, wobei aus mehreren Alternativen eine nach speziellen Kriterien ausgewählt wird.

  - Mathematisch:

    Zu einer Funktion soll ein Eingabewert gefunden werden, so dass die Funktion einen minimalen Wert annimmt, wobei i. A. eine Beschränkung für die Eingabewerte existiert.

Arten von Optimierungsproblemen (1)

- **Lineare Programmierung (LP):**
  - Lineare Zielfunktion, lineare Nebenbedingungen
  - Effizient lösbar durch Simplexmethode, *Interior Point* Methode

$$\min f(x) = c^T x$$
$$\text{mit } Ax \geq b$$
$$x \geq 0$$
$$x \in R^n$$

- **Nichtlineare Programmierung (NLP):**
  - Minimierungsfunktion ist mindestens quadratisch
  - Varianten: linear beschränkte Optimierung, quadratische Programmierung
  - Allgemein *nicht* effizient lösbar

Arten von Optimierungsproblemen (2)

- **Integer Programmierung (IP)**
  - Wie LP/NLP jedoch: $\boldsymbol{x} \in \boldsymbol{Z}^n$
  - Spezialfall: 0/1 IP mit bool'schen Variablen
  - Linear (ILP) oder nicht linear (IP)
- **Mixed Integer Programmierung (MIP)**
  - Mischung von reellwertigen und ganzen Zahlen
  - MILP oder MIP
- **Kombinatorische Optimierung**
  - Mathematisch gleichbedeutend mit IP/ILP
  - Permutations-, Reihenfolge-Probleme, Zuordnungs-Probleme, Finden von Teilmengen

Kombinatorische Optimierungsprobleme

- **Definition:**
  - Ein KOP ist ein Maximierungs- oder Minimierungsproblem und besteht aus:
    1. Einer Menge D_P von Instanzen,
    2. Einer endlichen Menge $S_P(I)$ von (möglichen) Lösungen für jede Instanz $I \in D_P$, und
    3. Einer Funktion m_P, die jeder Lösung $x \in S_P(I)$ zu jeder Instanz $I \in D_P$, einen positiven, reellwertigen Lösungswert $m_P(x,I)$ zuordnet.
  - Optimale Lösung:

    $x^* \in S_P(I)$ mit $m_P(x,I) \leq m_P(x^*,I) \quad \forall x \in S_P(I)$ (Maximierung)

    $x^* \in S_P(I)$ mit $m_P(x^*,I) \leq m_P(x,I) \quad \forall x \in S_P(I)$ (Minimierung)

Problem des Handlungsreisenden

- **Traveling Salesman Problem (TSP):**
  - Gesucht: eine Rundreise durch *n* Städte, jede Stadt darf nur einmal besucht werden.
  - Ziel: Die zurückgelegte Wegstrecke soll minimal sein.

$$L(p) = \sum_{i=1}^{n-1} d_{p(i),p(i+1)} + d_{p(n),p(1)}$$

  - Lösung: Reihenfolge der Städte: $\{\pi(1),\pi(2),...,\pi(n)\}$
    $\rightarrow$ eine Permutation der Menge $\{1,2,...,n\}$.
  - Die Distanzmatrix $(d_{i,j})$ gibt die Entfernungen der Städte an.

Problem des Handlungsreisenden (1)

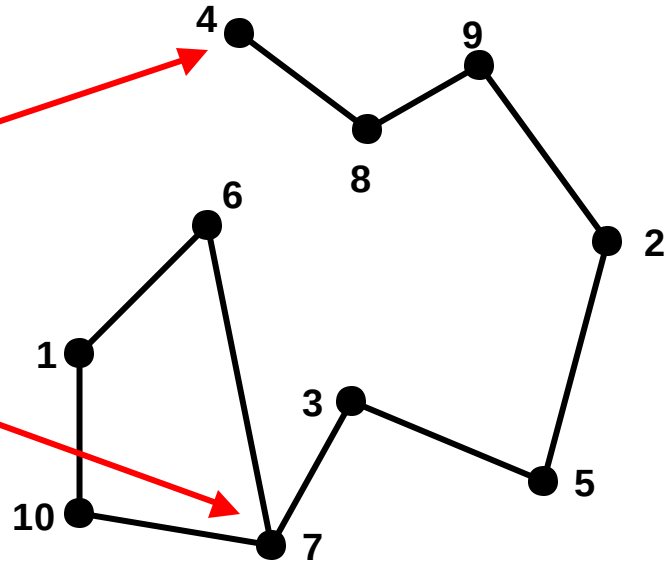
- **Traveling Salesman Problem als ILP:**

$$\min L(x) = \sum_{i=1}^{n} d_{ij}\, x_{ij}$$

$$\text{mit} \quad \sum_{i=1, i\neq j}^{n} x_{ij} = 1 \quad \forall\, j \in \{1,\ldots,n\}$$

$$\sum_{i=j, j\neq i}^{n} x_{ij} = 1 \quad \forall\, i \in \{1,\ldots,n\}$$

$$\sum_{i\in Q} \sum_{j\in V-Q} x_{ij} \geq 1 \quad \forall\, Q \subset \{1,\ldots,n\}$$



Constraints verletzt!

Problem des Handlungsreisenden (2)

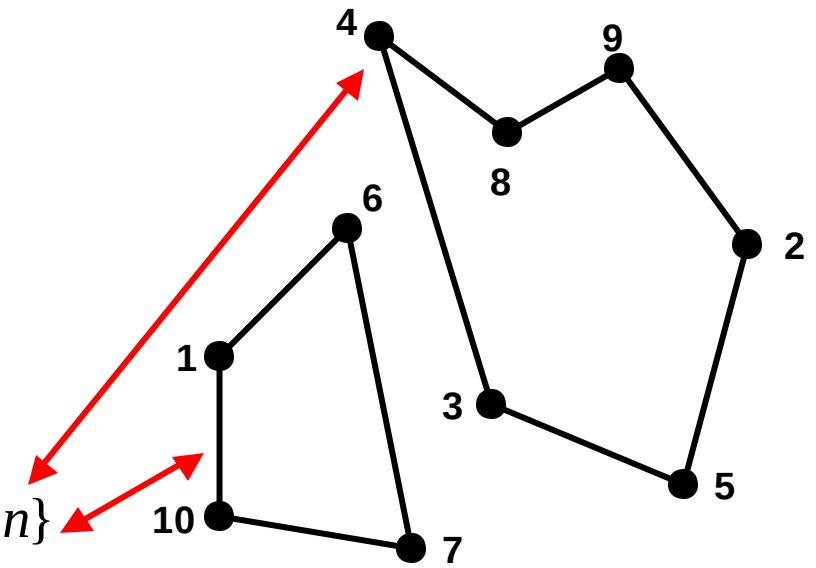
- **Traveling Salesman Problem als ILP:**

$$\min L(x) = \sum_{i=1}^{n} d_{ij}\, x_{ij}$$

$$\text{mit} \quad \sum_{i=1, i\neq j}^{n} x_{ij} = 1 \quad \forall\, j \in \{1,\ldots,n\}$$

$$\sum_{i=j, j\neq i}^{n} x_{ij} = 1 \quad \forall\, i \in \{1,\ldots,n\}$$

$$\sum_{i\in Q} \sum_{j\in V-Q} x_{ij} \geq 1 \quad \forall\, Q \subset \{1,\ldots,n\}$$



Constraints verletzt!

Problem des Handlungsreisenden (3)

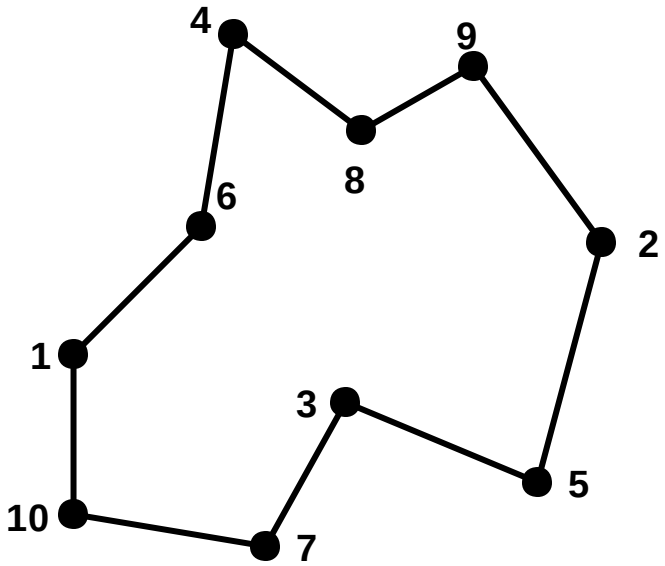
- **Traveling Salesman Problem als ILP:**

$$\min L(x) = \sum_{i=1}^{n} d_{ij}\, x_{ij}$$

$$\text{mit} \quad \sum_{i=1, i\neq j}^{n} x_{ij} = 1 \quad \forall\, j \in \{1,\ldots,n\}$$

$$\sum_{i=j, j\neq i}^{n} x_{ij} = 1 \quad \forall\, i \in \{1,\ldots,n\}$$

$$\sum_{i\in Q} \sum_{j\in V-Q} x_{ij} \geq 1 \quad \forall\, Q \subset \{1,\ldots,n\}$$



Alle Constraints erfüllt!

Graph-Partitionierungsproblem

- **Graph Partitioning Problem (GPP):**
  - Gesucht: Zerlegung eines Graphen *G=(V,E)* in *k* gleich große Teilmengen.
  - Ziel: Die Anzahl der Kanten zwischen den Teilmengen soll minimal sein.

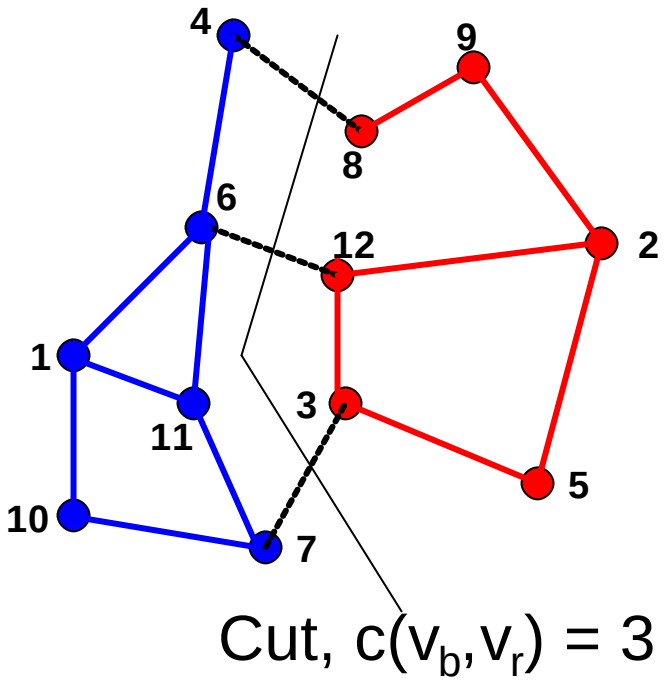
$$c(V_1,...,V_k) = \left| e(V_1,...,V_k) \right|$$

$$e(V_1,...,V_k) = \{(i,j) \in E \mid \exists l \in \{1,...,k\} : i \in V_l \wedge i \notin V_l\}$$

  - Allgemeinerer Fall: gewichtete Kanten
    → Minimierung der Summe der Kantengewichte von *e*
  - Spezialfall k = 2: Graph Bipartitioning (GBP)

Graph-Bipartitionierungsproblem

- **Graph Bipartitioning Problem (GBP):**
  - Zerlegung eines Graphen in k=2 gleich große Teilmengen.



Quadratisches Zuweisungsproblem

- **Quadratic Assignment Problem (QAP):**
  - Gesucht: Zuordnung von *n* Objekten zu *n* Positionen
  - Gegeben:
    - Fluß f_{ij} von Objekt *i* nach Objekt *j*
    - Entfernung d_{ij} von Position *i* und Position *j*
  - Ziel: Minimierung des Produktes aus Fluß und Entfernung

$$C(p) = \sum_{i=1}^{n} \sum_{j=1}^{n} d_{i,j} f_{p(i),p(j)}$$

  - Lösung: Permutation p, $p(i)=j$ bedeutet eine Zuweisung von Objekt *j* zu Position *i*

Binäre quadratische Optimierung

- **Binary Quadratic Programming (BQP):**
  - Gesucht: binärer Vektor, der eine quadratische Funktion f(x) maximiert.
  - Gegeben: *n* x *n* Matrix $Q=(q_{ij})$.

$$f(x) = x^t Q x = \sum_{i=1}^{n} \sum_{j=1}^{n} q_{ij} x_i x_j, \quad x_i \in \{0,1\}$$

  - Spezialfälle: Maximum Clique, Maximum Independent Set, Maximum Cut Problem

Transportoptimierung (1)

- **Vehicle Routing Problem (VRP):**
  - Gegeben: *n* Kunden, *m* LKW und 1 Depot.
  - Gesucht: *m* Touren um alle *n* Kunden zu bedienen.
  - Ziel: Minimierung der zurückgelegten Wegstrecke:

$$L(p) = \sum_{j=1}^{m} \sum_{i=1}^{l_j - 1} \left(d_{p_j(i), p_j(i+1)} + d_{p_j(l_j), 0} + d_{0, p_j(1)} \right)$$

  - l_j : Anzahl Kunden für LKW *j*
  - π_j : Besuchsreihenfolge für LKW *j*

Transportoptimierung (2)

- **Capacitated Vehicle Routing Problem (CVRP):**
  - Wie VRP nur mit zusätzlichen Restriktionen für die Beladung der LKW: Das Gesamtgewicht darf die Kapazität C_j der einzelnen LKW nicht überschreiten:

$$\sum_{i=1}^{l_j - 1} w_{p_j(i)} \leq C_j \quad \forall j = 1,\ldots,m$$

  - W_k*:* Gewicht der Ware für Kunde *k*
  - C_j*:* Maximales Beladungsgewicht

- *Time constrained VRP*: Zeitlimit pro Kunde, Zeitlimit pro Route
- *VRP with time windows*: Für jeden Kunden gibt es ein Zeitfenster zur Belieferung.

Rucksackproblem

- **Multidimensional Knapsack Problem (MKP):**
  - Gegeben: *m* Rucksäcke und *n* Gegenstände
  - Ziel: Profitmaximierung unter Berücksichtigung des Gewichtes

$$P(x) = \sum_{i=1}^{n} c_i x_i \quad mit \quad \sum_{i=1}^{n} w_{ij} x_i \leq W_j$$

  - c_i : Profit zu Gegenstand *i*
  - w_{ij} : Gewicht von Gegenstand *i* in Rucksack *j*
  - W_j : Maximalgewicht von Rucksack *j*

Klassifizierung von KOP

- **Arten nach Aufgabenstellung, zum Beispiel:**
  - Zuweisung
  - Reihenfolgebestimmung
  - Partitionierung
  - Teilmengenbestimmung
- **Art der Nebenbedingungen:**
  - Keine *Constraints* (BQP)
  - Implizite *Constraints* (TSP,QAP)
  - Explizite *Constraints* (MKP)
  - Implizite und explizite *Constraints* (VRP)

Kombinatorische Optimierung

- **Kombinatorische Optimierung heißt:**
  - Endliche Anzahl von möglichen Lösungen
  - Schnell wachsende Zahl der Lösungen, idR. exponentiell mit der Problemgröße
- **Problemgröße:**
  - Eigenschaft der Datenmenge einer Instanz bzw. Lösung
  - Bsp. TSP:
    - Anzahl *n* der Städte
    - Länge der Permutation π zur Beschreibung der Besuchsreigenfolge
    - Der Lösungsraum *S* hat die Größe $|S|=(n-1)!/2$

Komplexität

- **Wachstum des Suchraums:**

n	10	100	1000	10000
n^2	100	10000	10^6	10^8
n^3	10^3	10^6	10^9	10^{12}
2^n	1024	10^{30}	10^{301}	10^{3010}
$n!$	10^6	10^{157}	10^{2567}	10^{35659}

- **Vergleich:**
  - Anzahl der Atome im Universum ca. 10^{80}
  - Alter des Universums ca. 5×10^7 s

Komplexitätstheorie (1)

- **Theorie der *NP*-Vollständigkeit**
  - Klassifikation der Probleme aufgrund der Schwierigkeit ihrer algorithmischen Lösung
  - Klassifikation aufgrund des besten bekannten Algorithmus für ein Problem
  - Klassifikation über asymptotisches Verhalten der Algorithmen
- **Beschränkungen der *NP*-Vollständigkeit**
  - Nur Zeitkomplexität
  - Worst-Case Analyse der Algorithmen
  - Nur Entscheidungsprobleme (Probleme die „Ja“ oder „Nein“ liefern)

Komplexitätstheorie (2)

- **Zeitkomplexität**
  - Wird gemessen an der Anzahl elementarer Operationen
  - Formalisierung mittels $\mathcal{O}(\bullet)$-Notation:
    - Sei $f,g : \mathbb{N} \rightarrow \mathbb{N}$
    - Dann $f(n) \in \mathcal{O}(g(n))$, falls positive natürliche Zahlen c und n_0 existieren, so dass für alle $n \geq n_0$ gilt: $f(n) \leq c \cdot g(n)$
  - Ein Algorithmus ist polynomial, falls seine Zeitkomplexität in $\mathcal{O}(p(n))$ ist, wobei $p(n)$ ein Polynom ist.
  - Ein Algorithmus ist exponentiell, wenn Zeitkomplexität nicht polynomial beschränkbar ist.
  - Polynomial beschränkt / polynomiell $\leftrightarrow$ effizient
  - Exponentiell $\leftrightarrow$ ineffizient

NP-Vollständigkeit (1)

- **Grundlegende Klassen:**
  - Klasse *P*:

    Entscheidungsprobleme, die mit einem polynomiellen Algorithmus gelöst werden können
  - Klasse *NP*:

    Entscheidungsprobleme, die mit einem nichtdeterministischen polynimiellen Algorithmus gelöst werden können
  - Nichtdeterministischer Algorithmus:
    - In der ersten Phase wird eine Lösung geraten (nichtdet. Turingmaschine)
    - In der zweiten Phase wird deterministisch in polynomieller Zeit die Lösung verifiziert
  - $P \subseteq NP$, aber: $P = NP$?
  - Allgemein anerkannte Annahme: $P \neq NP$!

NP-Vollständigkeit (2)

- **Polynomielle Reduzierbarkeit:**

  Ein Problem Π ist *polynomiell reduzierbar* auf ein Problem Π‘, falls ein poynomieller Algorithmus existiert, der jede Instanz von Π in eine Instanz von Π‘ transformiert und für jede Instanz von Π ein „Ja“ ausgegeben wird, gdw. „Ja“ für die entsprechende Instanz von Π‘ ausgegeben wird.

- ***NP*-Vollständigkeit:**
  - Ein Problem P ist *NP*-vollständig, genau dann wenn
    1. Π ist in *NP* und
    2. Für alle Π‘ in *NP* gilt, das Π‘ polynomiell reduzierbar auf Π ist.

NP-Vollständigkeit (3)

- **Anmerkungen:**
  - Klasse der *NP*-vollständigen Probleme umfasst die schwersten in *NP*
  - Wird ein polynimal beschränkter Algorithmus für ein Problem aus *NP* gefunden, gilt *P=NP!*
  - Es ist unwahrscheinlich, dass für ein *NP*-vollständiges Problem ein polynomieller Algorithmus gefunden wird!
- **Beweis der *NP*-Vollständigkeit:**
  1. Zeige, dass Problem in *NP* liegt
  2. Wähle bekanntes, *NP*-vollständiges Problem und konstruiere Transformation auf das neue Problem
  3. Zeige, dass Transformation polynomial beschränkt ist

NP-harte Optimierungsprobleme

- **Erweiterung auf Optimierungsprobleme:**
  - Offensichtlich nicht leichter als die Entscheidungsvariante
  - Optimierungsvariante nicht in *NP*
  - Optimierungsproblem löst das Entscheidungsproblem
  - Neuer Begriff: *NP*-hart
- ***NP*-harte Probleme:**

  Ein Problem ist *NP*-hart, genau dann wenn für alle Π' in *NP* gilt, dass Π' polynomiell reduzierbar auf Π ist.
- **Viele kombinatorische Optimierungsprobleme sind *NP*-hart!**

Exakte Verfahren

- **Vollständige Enumeration:**
  - Möglich, da endlich großer Suchraum
  - Nicht praktikabel, da exponentielle Laufzeit
- **Implizite Enumeration:**
  - Suchraum wird durch einen Suchbaum eingeteilt
  - Effizienzsteigerung durch Elimination von Teilbäumen:
    - Branch & Bound
    - Branch & Cut

Branch & Bound

- **Suchbaumverfahren für Optimierungsprobleme:** Zusätzliche Berücksichtigung von Lösungskosten
- **Branching:** Verzweigung innerhalb des Suchbaums und Betrachtung von (disjunkten) Teilproblemen
- **Bounding:** Verwendung oberer und unterer Schranken für Zielfunktionswerte
  - **Obere Schranke:** Beste gefundene Lösung (Minimierung)
  - **Untere Schranke:** Günstigste Vervollständigung einer partiellen Lösung / (Diskrete) Relaxation des Problems
  - Falls untere Schranke größer als obere Schranke, kann Teilbaum eliminiert werden.

Branch & Cut

- **Ähnlich *Branch & Bound*, aber:**
  - Verwendung der LP-Relaxation des Problems für untere Schranken: ILP $\rightarrow$ LP
  - Verwendung leistungsfähiger Heuristiken für obere Schranken
  - Iterativer Ansatz:
    Stellt die aktuelle Lösung des LP-Problems nicht das Optimum dar, wird eine neue Gleichung eingefügt, um die Lösung zu entfernen (*cut*) und das neue LP wird mit Simplex-Algorithmus oder anderen LP-Algorithmen gelöst.
  - Schwierigkeit: Das Finden zulässiger Cuts
- **Relaxation:** Entfernung von Constraints, um das Problem leichter lösbar zu machen

Exakte Lösung des TSP (1)

- **Historie der Rekorde bei der exakten Lösung von TSP-Instanzen mit *Branch & Cut:***

Jahr	Forschergruppe	n
1954	G. Dantzig, R. Fulkerson und S. Johnson	49 Städte
1971	M. Held und R.M. Karp	64 Städte
1975	P.M. Camerini, L. Fratta und F. Maffioli	100 Städte
1977	M. Grötschel	120 Städte
1980	H. Crowder und M.W. Padberg	318 Städte
1987	M. Padberg und G. Rinaldi	532 Städte
1987	M. Grötschel und O. Holland	666 Städte
1987	M. Padberg und G. Rinaldi	2,392 Städte
1994	D. Applegate, R. Bixby, V. Chvátal und W. Cook	7,397 Städte
1998	D. Applegate, R. Bixby, V. Chvátal und W. Cook	13,509 Städte
2001	D. Applegate, R. Bixby, V. Chvátal und W. Cook	15,112 Städte

Exakte Lösung des TSP (2)

- **Aktueller Rekord:**
  - **n=15112 Städte**
  - **ergibt Suchraumgröße $|S|>10^{56592}$**
  - **Netzwerk von 110 Prozessoren von der Rice University und der Princeton University**
  - **Rechenzeit: 22.6 Jahre, skaliert auf Compaq EV6 Alpha Prozessor mit 500 MHz**

Exakte Verfahren vs. Heuristiken (1)

- **Generell:** Exakte Verfahren nur bei kleiner Problemgröße anwendbar
- **Branch & Cut:**
  - Sehr hohe Rechenzeit
  - Bei TSP sehr leistungsfähig
  - Aber nicht leicht übertragbar auf andere Probleme
  - Theorie für das Finden von *Cuts* notwendig
  - Gute Verfahren werden für obere Schranken benötigt

➠ **Alternativen zu exakten Verfahren sind nötig!**

Exakte Verfahren vs. Heuristiken (2)

- **Heuristik:**
  - **Griechisch *heuriskein*: Finden, entdecken**
  - Eine Technik zur ***Suche*** von guten (nahezu optimalen) Lösungen für ein Optimierungsproblem in möglichst kurzer Zeit
  - Ohne Gültigkeit oder Optimalität zu garantieren!
  - In vielen Fällen wird nicht mal eine Aussage getroffen, wie nahe die gefundene Lösung am Optimum liegt.