Informatik-Propädeutikum

Dozentin: Dr. Claudia Ermel

Betreuer: Sepp Hartung, André Nichterlein, Clemens Hoffmann
Sekretariat: Christlinde Thielcke (TEL 509b)

TU Berlin
Institut für Softwaretechnik und Theoretische Informatik
Prof. Niedermeier
Fachgruppe Algorithmik und Komplexitätstheorie
`http://www.akt.tu-berlin.de`

Wintersemester 2013/2014

Gliederung

Heuristiken

Nach Gerd Gigerenzer:
Welche Stadt hat mehr Einwohner: San Antonio oder San Diego?

Heureka! (Ich hab's gefunden!)

Auch eine Heuristik...

Umgangssprachlich ist oft auch von „Daumenregeln“ oder „Faustregeln“ die Rede.
Vgl. auch „Intuition“ und „Bauchentscheidungen“.

Was sind Heuristiken?

Wikipedia: Heuristik (altgriechisch heurisko, ich finde, zu heuriskein (auf)finden, entdecken) bezeichnet die Kunst, mit begrenztem Wissen und wenig Zeit zu guten Lösungen zu kommen.

Heuristiken kommen in allen möglichen Wissenschaftsgebieten vor:

- Wirtschaftswissenschaften: Zur Entscheidungsfindung.
- Philosophie: Gewinnung von Erkenntnis über eine Sache durch Übertragung von Wissen von einer anderen, ähnlichen Sache.
- Psychologie: Einfache Regeln für komplexe Situationen; Objekterkennung.
- Chemie: Verständnis und Klassifikation gewisser chemischer Reaktionen; Vorhersage.
- Mathematik: Z.B. Nullstellenraten bei Polynomen; Raten einer Anfangslösung in Optimierungsproblemen.
- Informatik: Künstliche Intelligenz; Intelligente Suche;...

Heuristiken aus Informatiksicht

Zwei zentrale Heuristik-Ansätze:

- Sukzessiver Aufbau einer Lösung beginnend mit „leerer“ Lösung.
- Iteratives Verbessern einer Anfangslösung (z.B. einer trivialen, noch schlechten Lösung).

Motivation: Berechnungsschwere (wie z.B. NP-vollständige) Probleme haben hohe Worst-Case-Komplexität bei ihrer Lösung. Kann man sie trotzdem (oft) gut und schnell in der Praxis lösen?

Heuristiken verzichten in der Regel auf mindestens eine der zwei folgenden Eigenschaften eines Lösungsalgorithmus:

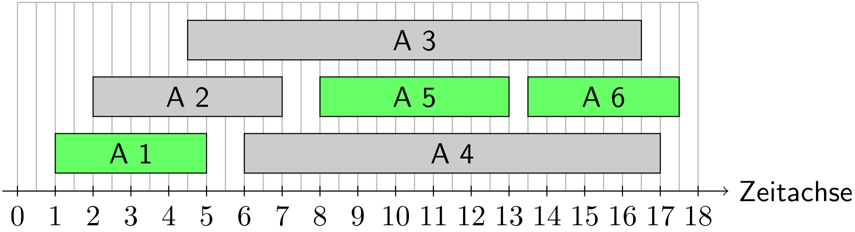
- Beweisbare Optimalität der gefundenen Lösung.
- Beweisbar schnelle Laufzeit des Algorithmus für *jede* Eingabe.

Bemerkung: Durch Einsatz von Heuristiken und ihrer empirischen Bewertung wird Informatik auch zur experimentellen Wissenschaft!

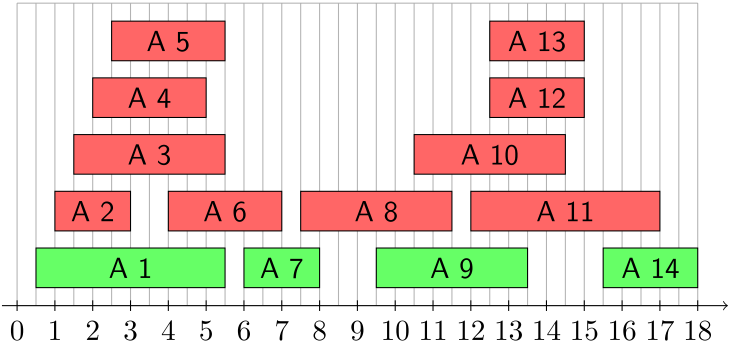
Viele heuristische Verfahren der Informatik sind Alltags- oder Naturprinzipien nachempfunden.

Einstiegsbeispiel Interval Scheduling (Ablaufplanung)

Eingabe: Eine Menge von „Jobs“, jeder mit einer Start- und Endzeit.
Aufgabe: Arbeite so viele Jobs wie möglich ab, wobei immer nur eine Aufgabe auf einmal abgearbeitet wird.



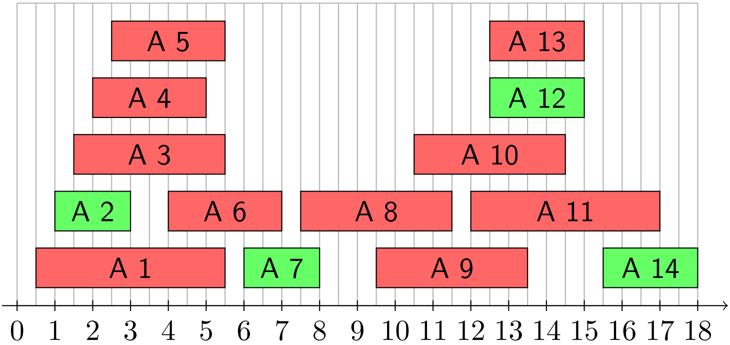
Greedy-Heuristiken für Interval Scheduling I



Größere Instanz: Wie finde ich eine beste Lösung?

Strategie 1: Nimm Job mit frühester Startzeit.
$\rightsquigarrow$ Vier Jobs können bearbeitet werden.

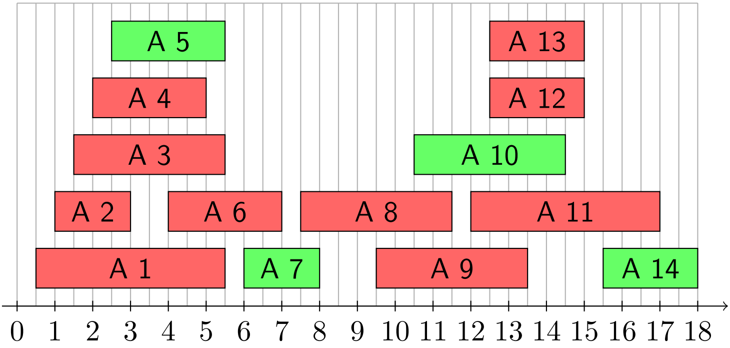
Greedy-Heuristiken für Interval Scheduling II



Größere Instanz: Wie finde ich eine beste Lösung?

Strategie 2: Nimm Job mit kürzester Länge.
$\rightsquigarrow$ Vier Jobs können bearbeitet werden.

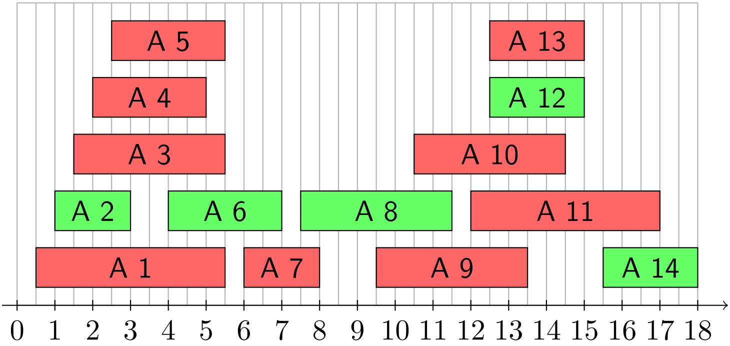
Greedy-Heuristiken für Interval Scheduling III



Größere Instanz: Wie finde ich die eine Lösung?

Strategie 3: Nimm' Job mit wenigsten Überschneidungen.
$\rightsquigarrow$ Vier Jobs können bearbeitet werden.

Greedy-Heuristiken für Interval Scheduling IV



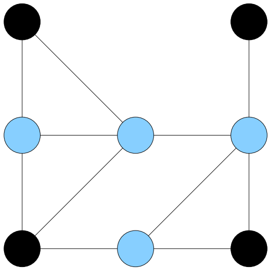
Größere Instanz: Wie finde ich eine beste Lösung?

Strategie 4: Nimm' Job mit frühester Endzeit.
$\rightsquigarrow$ Fünf Jobs können bearbeitet werden.
Mitteilung: Strategie 4 liefert beweisbar immer eine optimale Lösung.

Einstiegsbeispiel Vertex Cover

Vertex Cover,
formuliert als Optimierungs- statt als Entscheidungsproblem:

Finde kleinstmögliche Knotenmenge in Graph, sodass jede Kante mindestens einen dieser Knoten als Endpunkt hat.



Zwei einfache Greedy-Heuristiken (effizient implementierbar)

1. Solange es noch eine Kante gibt, die an keinem Lösungsknoten anliegt, wähle irgendeine und nimm *beide* Endpunkte in die Lösungsmenge.
2. Nimm jeweils Knoten mit der höchsten Anzahl anliegender Kanten.

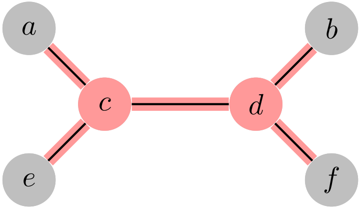
Erinnerung: Die Entscheidungsvariante von Vertex Cover ist NP-vollständig, d.h. es gibt (wahrscheinlich) keinen effizienten Algorithmus zum Finden einer kleinstmöglichen Knotenmenge…

Analyse der Greedy-Heuristik 1 für Vertex Cover

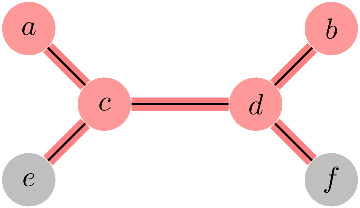
Greedy-Heuristik 1: Solange es eine Kante gibt, die an keinem Lösungsknoten anliegt, wähle irgendeine und nimm' beide Endpunkte in die Lösungsmenge.

Beobachtung: „Faktor-2-Approximation“.

Guter Fall:



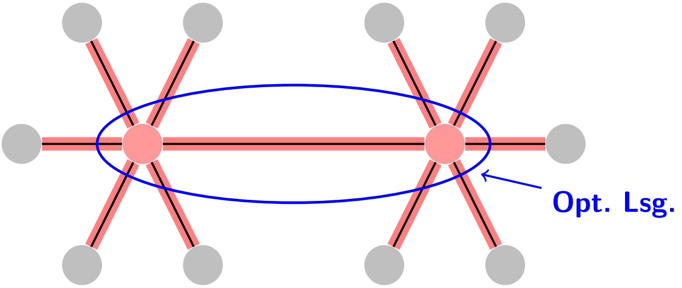
Schlechtester Fall:



Analyse der Greedy-Heuristik 2 für Vertex Cover I

Greedy-Heuristik 2: Nimm' jeweils Knoten mit der höchsten Anzahl anliegender Kanten.

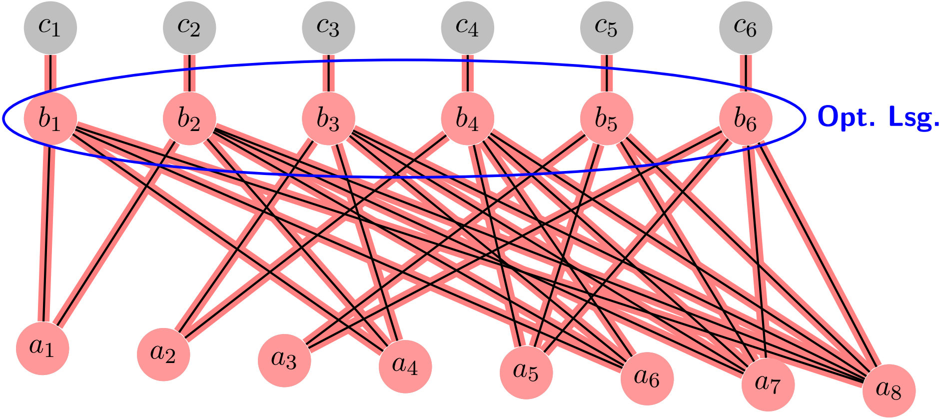
Guter Fall:



Analyse der Greedy-Heuristik 2 für Vertex Cover II

Greedy-Heuristik 2: Nimm jeweils Knoten mit der höchsten Anzahl anliegender Kanten.

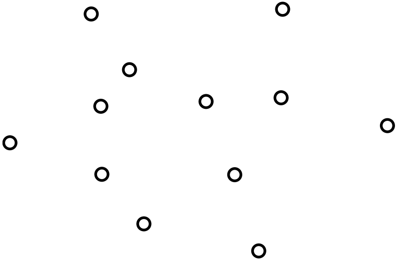
Beobachtung: Bei n Knoten „Faktor-$\ln n$-Approximation“.



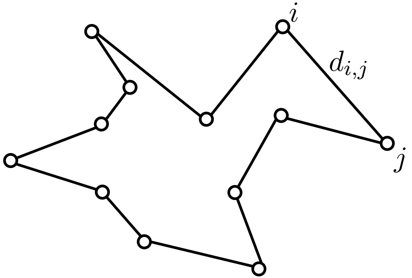
Traveling Sales Person (TSP) als fortlaufendes Beispiel

Eingabe: n Punkte mit paarweisen Abständen $d_{i,j}$.
Aufgabe: Finde eine kürzeste Rundtour, die alle Punkte genau einmal besucht.

Formal: Finde eine Permutation π (Rundtour) der Zahlen $1, 2, \ldots, n$, sodass $\sum_{i=1}^{n-1} d_{\pi(i),\pi(i+1)} + d_{\pi(n),\pi(1)}$ minimal ist.



Eingabe



Rundtour

Anwendungen des TSPs

- Routenplanung (Logistik)
- DNS-Sequenzierung (Bioinformatik)
- Layoutfindung für integrierte Schaltkreise (Hardware-Entwurf)
- Steuerung von Robotern in der Industrie (z. B. kürzeste Rundtour durch alle Lötstellen)
- ...

Erinnerung: Die Entscheidungsvariante des TSP ist NP-vollständig, d. h. es gibt (wahrscheinlich) keinen effizienten Algorithmus zum Finden optimaler Rundtouren.

Greedy-Heuristiken für das TSP I

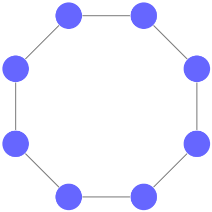
Annahme in folgenden Beispielen: Euklidsche Abstände.

Nächster-Nachbar-Heuristik:

1. Wähle beliebigen Startpunkt. Dieser heiße p.
2. Solange es einen noch nicht besuchten Punkt gibt und p der zuletzt besuchte Punkt ist:
   - Wähle als nächsten zu besuchenden Punkt den unbesuchten Punkt, der p am nächsten liegt.
   - Mache diesen Punkt zum neuen p.
3. Die Tour ergibt sich aus der Abfolge der besuchten Punkte.

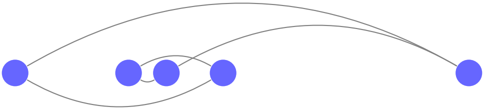
Einfache und effiziente Heuristik!

Aber immer optimal? **Beispiel**:

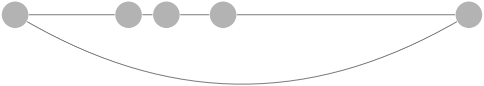


Greedy-Heuristiken für das TSP II

Beispiel:
Nächster-Nachbar-Heuristik findet folgende Rundtour bei Start im mittleren Punkt.



Optimal wäre aber offensichtlich folgende Rundtour:



Greedy-Heuristiken für das TSP III

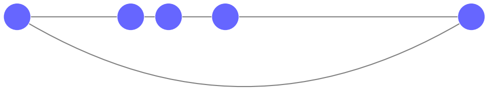
Eine zweite einfache Heuristik:

Annahme: Rundtour durch n Punkte wird durch einen Kantenzug beschrieben. Trivialer Kantenzug besteht aus einem Punkt.

Engstes-Paar-Heuristik:

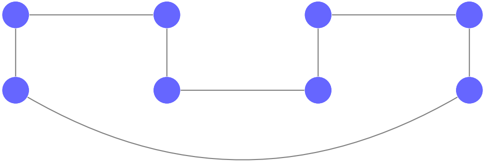
1. Solange es mehr als einen Kantenzug gibt
   - Verbinde zwei Endpunkte s und t zweier *verschiedener* Kantenzüge, sodass $d_{s,t}$ minimal unter allen möglichen Punktepaaren ist.
2. Verbinde die zwei Endpunkte des „Gesamtpfades".

Beispiel:



Greedy-Heuristiken für das TSP IV

Wie gut ist die Engstes-Paar-Heuristik?
Beispiel: Durch die Heuristik gefundene Lösung:



Optimal aber wäre:

Greedy-Heuristiken für das TSP V

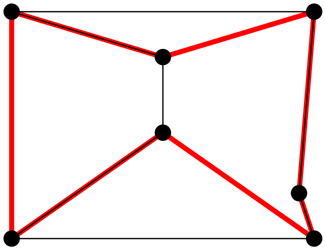
Inkrementelles-Einfügen-Heuristik:

Baue Punkt für Punkt die Rundtour auf, indem man

1. für jeden noch nicht besuchten Punkt den minimalen Abstand zu zwei benachbarten Punkten der bisherigen Teiltour bestimmt, und
2. den Punkt mit dem größten minimalen Abstand einfügt („furthest point insertion“).

Intuition: „Erst Grobtour skizzieren, dann Details klären.“

Nicht optimal! Denn z.B. rot markierte Tour ist kürzer.



Euklidisches TSP I

Wichtiger und „angenehmerer" Spezialfall[7] des TSP:
Alle Punkte liegen in der (Euklidischen) Ebene und ihr paarweiser Abstand ergibt sich aus den Entfernungen, d.h. für je drei beliebige Punkte i, j, und k gilt inbesondere die **Dreiecksungleichung**:

$$d_{i,j} \leq d_{i,k} + d_{k,j}.$$

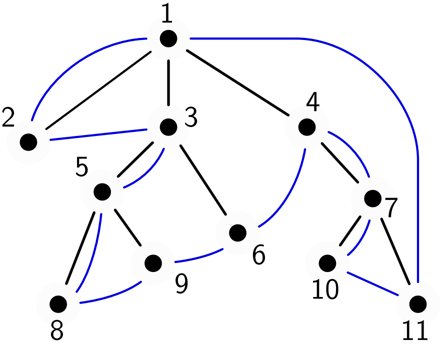
Faktor-2-Approximation mit Hilfe kostenminimaler Spannbäume:
Kostenminimaler Spannbaum = Baum welcher alle Eingabepunkte verbindet und die *Summe* der „Kantenkosten" (d.h. die jeweilige Entfernung zwischen den zwei Kantenendpunkten) ist kleinstmöglich.

Hinweis: Kostenminimale Spannbäume lassen sich effizient berechnen (vgl. Algorithmen von Boruvka, Kruskal, Prim).

[7] Zugehöriges Entscheidungsproblem bleibt aber weiterhin NP-vollständig.

Euklidisches TSP II

1. Berechne kostenminimalen Spannbaum.
2. Laufe den Spannbaum gemäß Tiefensuche ab.
3. Mache Abkürzungen falls Knoten doppelt besucht werden.



Tiefensuche: 1–2–1–3–5–8–5–9–5–3–6–3–1–4–7–10–7–11–7–4–1

Beobachtung: Manche Punkte treten doppelt auf → Abkürzungen!

⇝ Rundtour: 1–2–3–5–8–9–6–4–7–10–11–1

Euklidisches TSP III

Obige Spannbaumheuristik liefert Faktor-2-Approximation, d.h. die gefundene Tour ist höchstens doppelt so lang wie eine optimale:

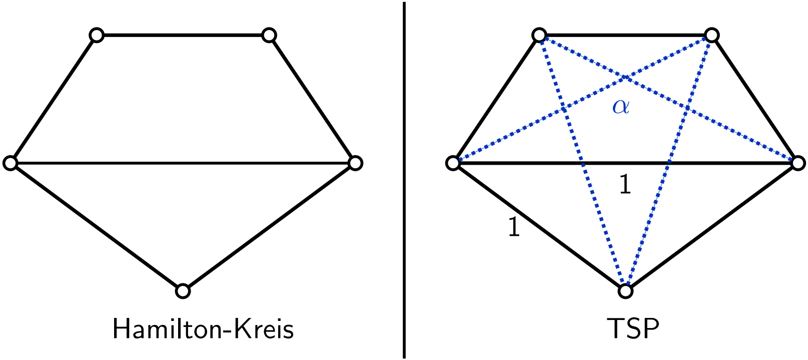
1. Kosten eines Spannbaumes sind höchstens so groß wie die Länge einer optimalen Rundtour! (Warum?)
2. Beim Ablaufen des Spannbaumes wird jede Kante höchstens zweimal überquert (vgl. Tiefensuche), deshalb liefert dies eine Tour die höchstens doppelt so lang ist wie eine optimale Rundtour.
3. Dank Dreiecksungleichung verlängert Abkürzen (sic!) die Tour nicht!

Begründung für Punkt 1: Eine optimale Rundtour ist ein Kreis. Entfernt man eine beliebige Kante so erhält man einen Spannbaum. Wir haben jedoch einen kostenminimalen Spannbaum gewählt...

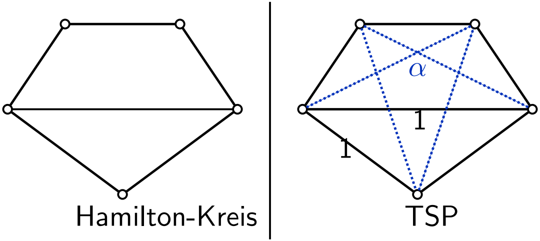
Nichtapproximierbarkeit des TSPs I

Die Approximierbarkeit in polynomieller Laufzeit des allgemeinen TSPs ist dramatisch schlechter als die des Euklidischen TSPs:

Beweis durch „Reduktion" vom NP-vollständigen Hamilton-Kreis-Problem auf TSP:
Erzeuge aus einer Hamilton-Kreis-Instanz $G = (V, E)$ eine TSP-Instanz durch Wahl von $d_{i,j} = 1$ falls $\{v_i, v_j\} \in E$ und $d_{i,j} = \alpha > 1$ sonst.



Nichtapproximierbarkeit des TSPs II

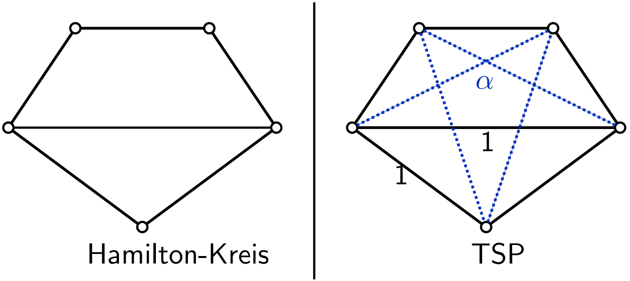


Behauptung: Die konstruierte TSP-Instanz hat eine Rundtour der Länge höchstens $n := |V|$ genau dann, wenn G einen Hamilton-Kreis hat. (Warum?)

Begründung: Ein Kreis enthält immer genau n Kanten, d. h. hat Mindestlänge n. Da $\alpha > 1$, entspricht jede Rundtour der Länge n einem Hamilton-Kreis im (Original-) Graphen und umgekehrt.

Beobachtung: Für entsprechende Wahl von α (z. B. $\alpha = n^2$) kann das Verhältnis Länge Rundtour mit „α-Kante" zu Rundtour ohne „α-Kante" nicht durch eine Konstante beschränkt werden.

Nichtapproximierbarkeit des TSPs III



Konsequenz aus obiger Reduktion: Könnten wir für das TSP einen effizienten Approximationsalgorithmus mit Approximationsfaktor kleiner als α angeben, so könnten wir das Hamilton-Kreis Problem effizient lösen.

Unter der Hypothese P $\neq$ NP folgern wir damit die Nichtapproximierbarkeit des allgemeinen TSPs.

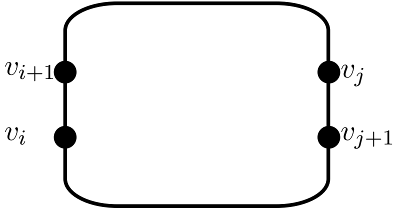
Bemerkung: Die Kosten für Flugtickets erfüllen z.B. in der Regel nicht die Dreiecksungleichung.

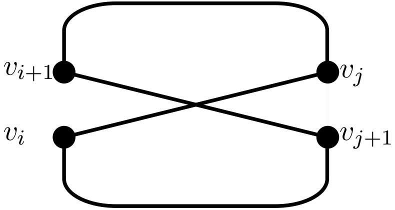
TSP und Lokale Suche

Lokale-Suche-Heuristik: Verbessere schrittweise eine gegebene Rundtour durch „kleine lokale Änderungen“.

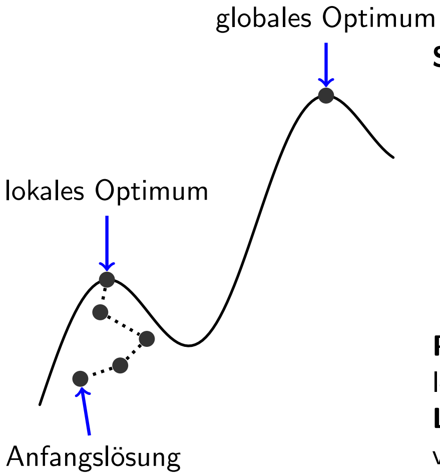
Idee: Suche innerhalb der k-Opt-Nachbarschaft nach einer kürzeren Rundtour.

***k*-Opt:** Lösche bis zu $k \in \mathbb{N}$ Kanten innerhalb der Rundtour und füge stattdessen k neue Kanten ein, um wieder eine Rundtour zu erzeugen.





Allgemeines Prinzip der Lokalen Suche



Strategie:

1. Erzeuge Anfangslösung L
2. Solange es innerhalb der „lokalen Umgebung" von L eine Verbesserung gibt:
   $L \leftarrow$ lok. verbesserte Lsg.

Problem: Algorithmus kann in lokalem Optimum hängenbleiben.
Lösung: Wiederhole Alg. für verschiedene Anfangslösungen oder **Simulated Annealing**...

TSP und Simulated Annealing, Metropolis-Algorithmus

„Simuliertes Abkühlen“:
Idee analog zur lokalen Suche, nur dass mit einer gewissen Wahrscheinlichkeit auch schlechtere Lösungen übernommen werden. Diese Wahrscheinlichkeit nimmt im Laufe des Algorithmus ab und geht gegen Null.

Simulated Annealing [Metropolis et al. 1953]

Erzeuge Anfangslösung L
$s \leftarrow 0$ ▷ Anzahl Schritte
while $s \leq$ max. Anzahl Schritte
    $s \leftarrow s + 1$
    Erzeuge aus L eine zufällig veränderte (bzw. mutierte) Lösung L'
    if L' besser als L oder $\mathrm{e}^{-s} \geq random[0, 1]$ **then:**
        $L \leftarrow L'$

Fazit zum TSP

Bei der algorithmischen Lösung von TSP-Instanzen gilt es diverse Randbedingungen zu berücksichtigen:

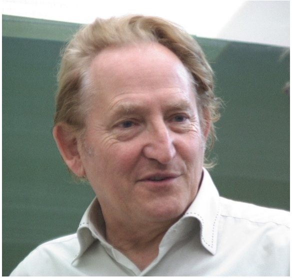
- Der Spezialfall, dass alle Punkte paarweise gleichen Abstand haben, entspricht dem Hamilton-Kreis-Problem.
- Erfüllt die Eingabe die Dreiecksungleichung, so lässt sich das TSP (beweisbar) gut approximativ lösen.
- Ist die paarweise Abstandsfunktion asymmetrisch (d.h. $d_{i,j} \neq d_{j,i}$), so ist das TSP in der Regel viel schwerer zu handhaben als im symmetrischen Fall (d.h. $d_{i,j} = d_{j,i}$).
- Es gibt viele gute Heuristiken, die oft nahe am Optimum liegende Lösungen schnell liefern.

Genetische Algorithmen I

Teilthema der Evolutionären Algorithmen...:

Inspiriert durch biologische Evolutionstheorie wurden in den 1960ern und 1970ern **evolutionäre Algorithmen** eingeführt. Maßgeblicher Einfluss (i.w. unabhängig voneinander) durch:

- Lawrence J. Fogel (Univ. of California, LA; 1928-2007) : evolutionäres Programmieren.
- John H. Holland (U. Michigan; 1929-): genetische Algorithmen.
- **Ingo Rechenberg (TU Berlin)** & Hans-Paul Schwefel (TU Dortmund; 1940-): Verfahren zur Evolutionsstrategie.



Ingo Rechenberg, 1934-

Genetische Algorithmen II

Idee: Nicht einzelne Lösungen verändern, sondern eine **Population** von Lösungen verändern.
Veränderungen durch **Mutationen** und **Kreuzungen**: Repräsentation der Lösungen als Bitfolgen.

Mutation: Zufälliges Stören/Flippen einzelner Bits.

$$a_i = 110010100101$$
$$a_i^* = 1100\mathbf{0}010010\mathbf{0}$$

Kreuzung (engl. *cross-over*): Aufteilen am Cross-Over-Punkt und Kreuzen der Teile.

$$a_i = 110010|100101 \qquad a_j = 010101|010110$$
$$\Rightarrow a_i^* = 110010|010110 \qquad a_j^* = 010101|100101$$

Genetische Algorithmen III

Nötig: **Fitnessfunktion** zur Bewertung von Lösungen.

Schema

```
Erzeuge Anfangspopulation {L_1, L_2, ..., L_r}
Repeat
    Erzeuge Mutationen von {L_1, ..., L_r}
    Erzeuge Kreuzungen von {L_1, ..., L_r}
    Behalte die r fittesten (besten) Lösungen
UNTIL Fitness verbessert sich nicht mehr
```

Effiziente heuristische Suche: Der A*-Algorithmus

Der A*-Algorithmus dient zur effizienten Suche in Graphen: Wie finde ich einen optimalen Weg von A nach B?
Optimal kann bedeuten: kürzester, schnellster, billigster, etc.

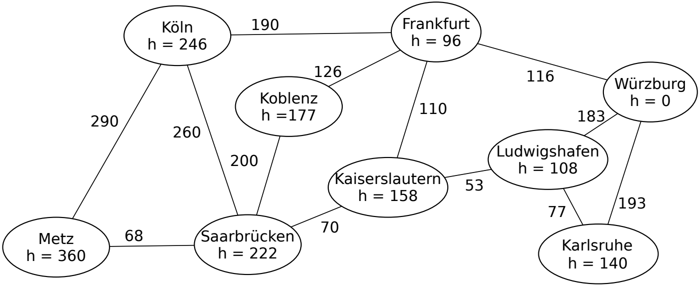
Idee: Benutze Zusatzinformationen als untere Schranke für „Entfernung“ eines Knotens zum Ziel um in der „richtigen Richtung“ zu suchen.

Erklärung am Beispiel: Berechnen des kürzesten Weges von Knoten A nach Knoten B.
Jeder Knoten x speichert als untere Schranke $h(x)$ die Entfernung (Luftlinie) zum Ziel.

Algorithmus:

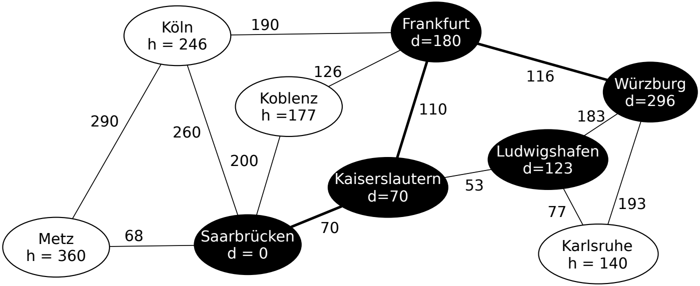
1 Merke alle besuchten Knoten mit berechneter Distanz zu A.
2 Starte in Knoten A.
3 **Repeat**
4 Besuche den Nachbarn x eines bereits besuchten Knotens, sodass (Distanz x zu A) $+$ $(h(x))$ minimiert wird.
5 **UNTIL** B erreicht

Der A*-Algorithmus: Beispiel I



Gegeben ist obiger Ausschnitt einer Karte.
Zahlen in den Knoten geben die Luftliniendistanz (untere Schranke) zu Würzburg an. Zahlen auf den Kanten sind tatsächliche Entfernungen entlang der jeweiligen Kante.
Gesucht: Kürzeste Strecke von Saarbrücken nach Würzburg.

Der A*-Algorithmus: Beispiel II



Starte in Saarbrücken.

Besuche Kaiserslautern: Distanz + untere Schranke = 70 + 158 = 228.

Besuche Ludwigshafen: Distanz + untere Schranke = 123 + 108 = 231.

Besuche Frankfurt: Distanz + untere Schranke = 180 + 96 = 276.

Besuche Würzburg: Distanz + untere Schranke = 296 + 0 = 296.

Kürzester Weg: Saarbrücken → Kaiserslautern → Frankfurt → Würzburg.