Greedy- und Priority Algorithmen

Was ist ein Greedy-Algorithmus? (1/2)

Greedy-Algorithmen bestimmen eine Lösung iterativ, wobei

- jedesmal eine Entscheidung getroffen wird, die „lokal“ am vielversprechendsten ist.
- Getroffene Entscheidungen werden nicht zurückgenommen.

- „Definition“ ist nicht präzise.
- Wie soll man zeigen, dass ein Optimerungsproblem durch Greedy-Algorithmen nicht scharf approximiert werden kann?

Was ist ein Greedy-Algorithmus? (2/2)

Wir benötigen den Begriff eines **Datenelements**.

- Beispiele sind Jobs mit Startzeiten, Verarbeitungszeiten, Fristen und Strafen für Scheduling Probleme oder
- Kanten und ihre Gewichte, bzw Knoten, ihre Gewichte und Nachbarn für Graph Probleme.

- Eine Eingabe besteht aus einer Menge von *tatsächlichen Datenelementen*.
- Bestimme in jedem Schritt eine vollständige Ordnung auf *allen möglichen Datenelementen* ohne Kenntnis der Eingabe.
- Der Algorithmus erhält das Datenelement der Eingabe mit höchster Priorität und trifft eine nicht revidierbare Entscheidung.

Ein solcher Algorithmus heisst **Priority-Algorithmus**.
Der Algorithmus heißt **nicht-adaptiv**, wenn nur eine einzige Ordnung bestimmt wird und ansonsten **adaptiv**.

Intervall Scheduling

Beispiel: INTERVALL SCHEDULING

- Aufgaben $A_1, \ldots, A_n$ sind auf einem Prozessor auszuführen.
  Aufgabe A_i hat den Startpunkt s_i und den Endpunkt e_i.
- Führe möglichst viele Aufgaben aus, ohne dass die Zeitintervalle verschiedener Aufgaben überlappen.

- Sortiere die Aufgabe nach aufsteigendem Endpunkt („frühester Endpunkt zuerst“).
  - Diese Entscheidung ist optimal, da die Ausführung jeder anderen Aufgabe höchstens weitere Aufgaben ausschließt.
- Wir erhalten einen nicht-adaptiven Priority-Algorithmus.
  - Ein Datenelement ist eine Aufgabe, repräsentiert durch ihren Start- und Endpunkt.
  - Der Priority-Algorithmus ordnet Aufgaben nach steigenden Endpunkten.

Die Berechnung des Huffman Code

HUFFMAN CODE

- Eine Datei D ist aus Buchstaben eines Alphabets Σ aufgebaut.
- Für jeden Buchstaben $a \in \Sigma$ ist $H(a)$ die **Häufigkeit** von a.
- Bestimme einen **Präfixcode**, der D optimal komprimiert.

Huffman's Algorithmus baut einen binären **(Kodier-)Baum** B

- Bestimme die beiden Buchstaben a, b geringster Häufigkeit und mache sie zu Kindern eines neuen Knoten $v_{a,b}$.
- a und b werden entfernt und $v_{a,b}$ wird als neuer Buchstabe mit Häufigkeit $H(a) + H(b)$ hinzugenommen.
- Wiederhole solange, bis nur noch ein Buchstabe vorhanden ist.

- Die Blätter von B sind mit Buchstaben und die Kanten mit 0 (Kante zum linken Kind) und 1 (Kante zum rechten Kind) markiert.
- Der Binärcode des Buchstabens a ist die Folge der Kantenbeschriftungen auf dem Weg von der Wurzel zum Blatt von a.

Huffman's Algorithmus ist ein Priority Algorithmus

Das Datenelement von Buchstaben a besteht aus a und der Häufigkeit von a.

- Huffman's Algorithmus sortiert Buchstaben nach aufsteigenden Häufigkeiten.
- Die Entscheidungen des Algorithmus bestehen im Einfügen von Kanten zwischen dem Elternknoten $v_{a,b}$ und den Kinderknoten a und b.
- Getroffene Entscheidungen werden nicht zurückgenommen.

Huffman's Algorithmus ist ebenfalls nicht-adaptiv.

Kürzeste Wege

KÜRZESTE WEGE

- Sei $G = (V, E)$ ein gerichteter Graph mit Quelle $s \in V$ und Kantengewichtung Länge : $E \to \mathbb{R}_{\geqslant 0}$.
- Bestimme **kürzeste Wege von s nach v** für alle Knoten $v \in V$.

- Setze $S = \{s\}$. In jedem Schritt berechnet Dijkstra's Algorithmus ein Array D mit

$$D[v] \quad = \quad \text{Länge eines kürzesten Weges von } s \text{ nach } v, \text{ der bis auf } v \text{ nur in } S \text{ verläuft.}$$

- Wenn $v \in V \setminus S$ den kleinsten D-Wert besitzt, dann
  - füge v zu S hinzu
  - und aktualisiere die D-Werte der Nachbarn von v.

Die Aktualisierung garantiert, dass D weiterhin die Länge „kürzester S-Wege“ wiedergibt.

Dijkstra's Algorithmus ist ein Priority-Algorithmus

Das Datenelement von Knoten v ist:

v mitsamt seinen Nachbarn und den Distanzen zu den Nachbarn.

- Zu Anfang:
  - In der ersten Ordnung erhalten alle möglichen Datenelemente von s die höchste Priorität.
  - Der Algorithmus erhält das (tatsächliche) Datenelement von s und initialisiert das Array D.
- Nach jedem weiteren Schritt:
  - Dijkstra's Algorithmus bestimmt den Knoten $v \in V \setminus S$ mit kleinstem D-Wert
  - und berechnet eine Ordnung, die allen möglichen Datenelementen von v die größte Priorität gibt.

Dijkstra's Algorithmus ist ein **adaptiver** Priority-Algorithmus.

Die Berechnung minimaler Spannbäume

MINIMALER SPANNBAUM

- G ist ein ungerichteter zusammenhängender Graph, dessen Kanten Gewichte erhalten.
- Bestimme einen **leichtesten Spannbaum** für G.

Kruskal's Algorithmus

- sortiert die Kanten nach aufsteigendem Gewicht.
- Beginnend mit der leichtesten Kante e:

  Füge Kante e ein, wenn kein Kreis geschlossen wird.

Kruskal's Algorithmus ist ein nicht-adaptiver Priority Algorithmus:

Die Datenelemente sind die Kanten und ihre Gewichte.

Matroide

Monotone Teilmengensysteme

- $(\mathcal{P}, X)$ ist ein **monotones Teilmengensystem**, falls
  - $\mathcal{P}$ eine Menge von Teilmengen von X ist und
  - aus $Y_1 \in \mathcal{P}$ und $Y_2 \subseteq Y_1$ stets $Y_2 \in \mathcal{P}$ folgt.

  Wir nennen die Mengen $Y \in \mathcal{P}$ **unabhängig**.
- Im **Optimierungsproblem** für ein monotones Teilmengensystem erhält jedes Element $x \in X$ ein Gewicht $w_x \geqslant 0$. Das Ziel ist die Bestimmung einer **schwersten unabhängigen Menge**.

- MINIMLER SPANNBAUM wie auch INDEPEDENT SET sind Optimierungsprobleme für monotone Teilmengensysteme.
- Das erste Problem ist einfach, das zweite sehr schwer. Warum?

Der Greedy Algorithmus für Matroide

Was **sollte** der Greedy-Algorithmus für ein monotones Teilmengensystem $(\mathcal{P}, X)$ machen?

- Anfänglich sei $Y = \emptyset$.
- Füge das schwerste Element x zu Y hinzu, so dass $Y \cup \{x\}$ unabhängig bleibt.
- Gib Y aus, wenn Y nicht erweitert werden kann.

Wir nennen $(\mathcal{P}, X)$ ein **Matroid**, wenn der Greedy-Algorithmus für **alle** Gewichtungen eine schwerste unabhängige Menge bestimmt.

Das Graph-Matroid

Das Graph-Matroid

Sei $G = (V, E)$ ein ungerichteter, zusammenhängender Graph.

- Wir sagen, dass eine Menge $E' \subseteq E$ unabhängig ist, wenn die Kanten in E' einen Wald definieren.
- Definiere $\mathcal{P}(E)$ als die Menge aller unabhängigen Mengen.
- $(\mathcal{P}(E), E)$ ist ein monotones Teilmengensystem und wird das **Graph-Matroid** genannt.

Der Greedy-Algorithmus versucht, einen schwersten Spannbaume zu berechnen. Betrachte deshalb die neuen Kantengewichte

$$w'_e := \max\{w_f \mid f \in E\} - w_e.$$

- Kruskal's Algorithmus für die Gewichtung w_e berechnet leichteste Spannbäume.
- Der Greedy Algorithmus für die Gewichtung w'_e berechnet denselben Baum wie Kruskals's Algorithmus für w_e.

Weitere monotone Teilmengensysteme

Der Greedy-Algorithmus berechnet einen schwersten Spannbaum:

Das Graph-Matroid $(\mathcal{P}(E), E)$ ist tatsächlich ein Matroid.

- Sei $G = (V, E)$ ein ungerichteter Graph.
  - Nenne $V' \subseteq V$ unabhängig, wenn keine zwei Knoten in V' durch eine Kante verbunden sind.
  - $\mathcal{Q}(V)$ ist die Klasse aller unabhängigen Mengen.
  - Ist $(\mathcal{Q}(V), V)$ ein Matroid?
- Sei $V = \{v_1, \ldots, v_m\}$ eine Menge von Vektoren in einem Vektorraum.
  - Nenne $V' \subseteq V$ unabhängig, wenn V' eine Menge von linear unabhängigen Vektoren ist.
  - $\mathcal{R}(V)$ ist die Klasse aller unabhängigen Mengen.
  - Ist $(\mathcal{R}(V), V)$ ein Matroid?

    Ja! $(\mathcal{R}(V), V)$ wird **Matrix-Matroid** genannt.

Charakterisierungen von Matroiden

Für ein monotones Teilmengensystem $(\mathcal{P}, X)$ sind äquivalent:

(a) $(\mathcal{P}, X)$ ist ein Matroid.

(b) Die **Ergänzungseigenschaft** gilt: Für je zwei unabhängige Mengen Y_1, Y_2 mit $|Y_1| = |Y_2| - 1$ gibt es ein $x \in Y_2 \setminus Y_1$, so dass $Y_1 \cup \{x\}$ unabhängig ist.

(c) Die **Maximalitätseigenschaft** gilt: Alle nicht vergrößerbaren unabhängigen Teilmengen einer beliebigen Menge $Z \subseteq X$ besitzen dieselbe Größe.

- Warum ist das Graph-Matroid ein Matroid?
  - Eigenschaft (c) gilt: Für jede Teilmenge Z von Kanten haben alle nicht-vergrößerbaren Spannwälder dieselbe **Kantenzahl**.
- Warum ist das Matrix-Matroid ein Matroid?
  - Eigenschaft (c) gilt: Die **Dimension** des von einer Teilmenge Z aufgespannten Raums stimmt überein mit der Mächtigkeit irgendeiner nicht-vergrößerbaren unabhängigen Teilmenge von Z.

Matroid-Eigenschaft $\Rightarrow$ Ergänzungseigenschaft

Sei $(\mathcal{P}, X)$ ein Matroid.

- Y_1 und Y_2 seien unabhängig mit $p := |Y_1| = |Y_2| - 1$, aber $Y_1 \cup \{x\} \notin \mathcal{P}$ für jedes $x \in Y_2 \setminus Y_1$.
- Wir falsifizieren den Greedy-Algorithmus mit den Gewichten

$$w_x := \begin{cases} p+2 & \text{falls } x \in Y_1 \\ p+1 & \text{falls } x \in Y_2 \setminus Y_1 \\ 0 & \text{sonst.} \end{cases}$$

  - Greedy wählt p Elemente aus Y_1 und erreicht das Gesamtgewicht $|Y_1| \cdot (p+2) = p \cdot (p+2) = p^2 + 2p$,
  - während die unabhängige Menge Y_2 das größere Gewicht von mindestens $(p+1)^2 = p^2 + 2p + 1$ besitzt.

Ergänzungseigenschaft $\Rightarrow$ Maximalitätseigenschaft

- Wenn die Maximalitätseigenschaft für $Z \subseteq X$ nicht gilt, dann gibt es zwei größte, unabhängige Teilmengen $Y_1, Y_2 \subseteq Z$ mit

$$|Y_1| < |Y_2|.$$

- Sei $Y_2^* \subseteq Y_2$ eine beliebige Teilmenge mit der Eigenschaft

$$|Y_1| = |Y_2^*| - 1.$$

- Mit der Ergänzungseigenschaft:
  - Wir können Y_1 um ein Element aus $Y_2^* \subseteq Y_2$ vergrößern.
  - Y_1 ist nicht größtmöglich — Widerspruch.

Maximalitätseigenschaft $\Rightarrow$ Matroideigenschaft (1/2)

- Eine beliebige Gewichtung $w_x \geqslant 0$ für $x \in X$ ist gegeben.
  - Der Greedy-Algorithmus bestimmt eine nicht-vergrößerbare unabhängige Menge Y.
  - Y^* sei eine schwerste unabhängige Menge.
- Maximalitätseigenschaft: $|Y| = |Y^*|$ mit $Z = X$.
- Nummeriere die Elemente von Y und Y^* gemäß absteigendem Gewicht:

$$Y = \{y_1, \ldots, y_m\}, \quad Y^* = \{y_1^*, y_2^*, \ldots, y_m^*\}.$$

Zeige: $w_{y_i} \geqslant w_{y_i^*}$ für $i = 1, \ldots m$.

Maximalitätseigenschaft $\Rightarrow$ Matroideigenschaft (2/2)

Warum gilt stets $w_{y_i} \geqslant w_{y_i^*}$?

- $i = 1$: Stimmt, denn Greedy wählt das schwerste Element.
- $i - 1 \Rightarrow i$: Wähle

$$Z^* := \{x \in X \mid w_x \geqslant w_{y_i^*}\}.$$

  - Wenn $w_{y_i} < w_{y_i^*}$, dann ist $\{y_1, \ldots, y_{i-1}\}$ eine nicht vergrößerbare unabhängige Teilmenge von Z^*,
  - aber $\{y_1^*, y_2^*, \ldots, y_i^*\} \subseteq Z^*$ ist um ein Element größer.
  - Wir haben einen Widerspruch zur Maximalitätseigenschaft erhalten.

Matroid-Scheduling

MATROID SCHEDULING

- Die Aufgaben $A_1, \ldots, A_n$ sind auf einem Prozessor auszuführen.
- Aufgabe A_i kann in einem Schritt ausgeführt werden. Wenn aber keine Ausführung zur Frist f_i gelingt, dann fällt die Strafe b_i an.
- Maximiere die Summe der „Nicht-Strafen“.

- Sei $X := \{A_1, \ldots A_n\}$ und $\mathcal{P}$ bestehe aus allen ohne Fristüberschreitung ausführbaren Teilmengen von Aufgaben.

  Wenn $Y \in \mathcal{P}$, dann gibt es zu jeden Zeitpunkt t höchstens t Aufgaben in Y mit Frist $f_i \leqslant t$.
- Zeige, dass $(\mathcal{P}, X)$ die Ergänzungseigenschaft erfüllt.
  - Dann ist $(\mathcal{P}, X)$ ein Matroid und der Greedy-Algorithmus löst das Scheduling Problem optimal.
  - Entschuldigung, der Greedy Algorithmus bestimmt doch nur eine schwerste Menge fristgerecht ausführbarer Aufgaben!? Und was sind denn überhaupt die Gewichte?

Wie erhalten wir ein Scheduling der Aufgaben?

Das Gewicht von $\mathbf{A_i}$ ist $\mathbf{b_i}$: Der Greedy-Algorithmus maximiert die Nicht-Strafe und minimiert damit die Strafe.

Aber der Greedy-Algorithmus bestimmt nur eine Teilmenge T von fristgerecht ausführbaren Aufgaben. Wie erhält man ein Scheduling?

- Führe die Aufgaben in T gemäß aufsteigender Frist aus.
- Alle nicht in T enthaltenen Aufgaben werden am Ende in beliebiger Reihenfolge ausgeführt.

Wird die Ergänzungseigenschaft erfüllt?

Die Mengen $Y_1, Y_2 \in \mathcal{P}$ mit $|Y_1| = |Y_2| - 1$ seien beliebig.

- Gibt es $A \in Y_2$, so dass $Y_1 \cup \{A\}$ ohne Fristüberschreitung ausführbar ist?
- Wähle $t \geqslant 0$ maximal mit

$$|\{A_j \in Y_2 \mid f_j \leqslant t\}| \leqslant |\{A_j \in Y_1 \mid f_j \leqslant t\}|.$$

  Ein solches maximales t existiert, da $|Y_2| > |Y_1|$.
- Es gibt also eine Aufgabe $A \in Y_2 \setminus Y_1$ mit Frist $t + 1$. Aber t ist maximal, also

$$\forall s \geqslant t + 1 : \quad |\{A_j \in Y_1 \mid f_j \leqslant s\}| < |\{A_j \in Y_2 \mid f_j \leqslant s\}| \leqslant s.$$

- Die Aufgaben in $Y_1 \cup \{A\}$ sind fristgerecht ausführbar:
  - Schiebe Aufgabe A zum Zeitpunkt $t + 1$ in die fristgerechte Ausführung der Aufgaben in Y_1 ein.
  - Verschiebe alle nach Zeit t ausgeführten Aufgaben um 1 Schritt.

k-Matroide

Sei $k \in \mathbb{R}$

Ein monotones Teilmengensystem $(\mathcal{P}, X)$ heißt ein

$$\textbf{k} - \textbf{Matroid},$$

wenn der Greedy-Algorithmus für monotone Teilmengensysteme **k-approximative** Lösungen für **jede** Gewichtung bestimmt.

- 1-Matroide sind Matroide.
- k muss keine natürliche Zahl sein.
- Lassen sich k-Matroide wieder über die Ergänzungs- und Maximalitätseigenschaft charakterisieren?

Für ein monotones Teilmengensystem $(\mathcal{P}, X)$ sind äquivalent:

(a) $(\mathcal{P}, X)$ ist ein **k-Matroid**.

(b) Die **k-Ergänzungseigenschaft** gilt: Für je zwei unabhängige Mengen Y_1, Y_2 mit $|Y_2| > k \cdot |Y_1|$ gibt es ein $x \in Y_2 \setminus Y_1$, so dass $Y_1 \cup \{x\}$ unabhängig ist.

(c) Die **k-Maximalitätseigenschaft** gilt: Für jede beliebige Teilmenge $Z \subseteq X$ und je zwei nicht vergrößerbare unabhängige Teilmengen $Y_1, Y_2 \subseteq Z$ gilt $|Y_2| \leqslant k \cdot |Y_1|$.

Die Argumentation ist ähnlich wie für die Charakterisierungen von Matroiden.

Eine hinreichende Bedingung (1/2)

Ein monotones Teilmengensystem $(\mathcal{P}, X)$ heißt **k-erweiterbar**, wenn für je zwei unabhängige Mengen $Y_1 \subset Y_2$ und für jedes Element $x \notin Y_2$ gilt:

> Wenn $Y_1 \cup \{x\}$ unabhängig ist, dann gibt es eine Teilmenge $T \subseteq Y_2 \setminus Y_1$ von höchstens k Elementen, so dass $(Y_2 \setminus T) \cup \{x\}$ unabhängig ist.

Nach Herausnahme von bis zu k Elementen kann x auch zur größeren Menge Y_2 hinzugefügt werden.

Eine hinreichende Bedingung (2/2)

Wenn das monotone Teilmengensystem $(\mathcal{P}, X)$ **k-erweiterbar** ist, dann ist $(\mathcal{P}, X)$ ein **k-Matroid**.

- Sei $(\mathcal{P}, X)$ k-erweiterbar. Verifiziere die k-Ergänzungseigenschaft:

  Für unabhängige Mengen Y_1 und Y_2 mit $|Y_2| > k \cdot |Y_1|$ bestimme $x \in Y_2 \setminus Y_1$, so dass $Y_1 \cup \{x\}$ unabhängig ist.

- O.B.d.A gilt $Y_1 \not\subset Y_2$. Wähle ein beliebiges $u \in Y_1 \setminus Y_2$ und betrachte die unabhängigen Mengen $Z_1 := Y_1 \cap Y_2$ und $Z_2 := Y_2$.
  - $(\mathcal{P}, X)$ ist k-erweiterbar. Also gibt es eine Teilmenge $T \subseteq Z_2 \setminus Z_1 = Y_2 \setminus Y_1$ mit $|T| \leqslant k$, so dass $Y_2' = (Y_2 \setminus T) \cup \{u\}$ unabhängig ist.
  - Wenn Y_1 noch keine Teilmenge von Y_2' ist, dann wiederhole das Vorgehen mit den Mengen Y_1 und Y_2'.
  - Nur Elemente, die nicht zu Y_1 gehören, werden aus Y_2 entfernt:

    Nach höchstens $|Y_1 \setminus Y_2| \leqslant |Y_1|$ Schritten erhalten wir eine unabhängige Menge Y_2' mit $Y_1 \subseteq Y_2'$.
  - Aber $|Y_2| > k \cdot |Y_1|$ und Y_1 ist eine echte Untermenge von Y_2'.

Der Durchschnitt von k Matroiden

Der Durchschnitt von k Matroiden (1/2)

Ein Durchschnitt von k Matroiden ist k-erweiterbar und deshalb ein k-Matroid.

k = 1: Warum ist ein Matroid 1-erweiterbar?

Für unabhängige Mengen $Y_1 \subset Y_2$ und $x \notin Y_2$ sei $Y_1 \cup \{x\}$ unabhängig.

Finde $y \in Y_2 \setminus Y_1$, so dass $Y_2 \setminus \{y\} \cup \{x\}$ unabhängig ist.

- Ergänzungseigenschaft für Matroide: Vergrößere die unabhängige Menge $Y_1 \cup \{x\}$ mit Elementen aus $Y_2 \setminus Y_1$.
- Wiederhole, bis wir eine unabhängige Teilmenge Y_1' mit $Y_1 \cup \{x\} = Y_1' \subseteq Y_2 \cup \{x\}$ und $|Y_1'| = |Y_2|$ erhalten.
- Da $x \notin Y_2$, besteht $Y_2 \setminus Y_1'$ aus genau einem Element y.

  Aber dann ist $Y_2 \setminus \{y\} \cup \{x\} = Y_1'$ unabhängig und das war zu zeigen.

Der Durchschnitt von k Matroiden (2/2)

Ein Durchschnitt von k Matroiden ist also ein Durchschnitt von k vielen 1-erweiterbaren monotonen Mengensystemen.

- Warum ist der Durchschnitt von k vielen 1-erweiterbaren monotonen Mengensystemen k-erweiterbar?
  - Für Teilmengen $Y_1 \subset Y_2$ und $x \notin Y_2$ sei also $Y_1 \cup \{x\}$ unabhängig.
  - Dann gibt es im i.ten monotonen Mengensystem ein Element $y_i \in Y_2 \setminus Y_1$, so dass $Y_2 \setminus \{y_i\} \cup \{x\}$ im i.ten System unabhängig ist.
  - Aber dann ist $Y_2 \setminus \{y_1, \ldots, y_k\} \cup \{x\}$ eine unabhängige Menge des Durchschnitts.
- Und das war zu zeigen.

Das f-Matching Problem

f-MATCHING (1/2)

- $G = (V, E)$ sei ein ungerichteter Graph, dessen Kanten nicht-negative Gewichte besitzen.
- Sei $f : V \to \mathbb{N}$ eine Funktion. Ein f-Matching ist eine Kantenmenge $M \subseteq E$, so dass die Anzahl mit v inzidenter Kanten durch $f(v)$ für jeden Knoten v beschränkt ist.
- Bestimme ein schwerstes f-Matching.

Unser Mengensystem besteht also aus der Grundmenge E und allen Teilmengen von E, die f-Matchings entsprechen.

Zeige, dass das Mengensystem 2-erweiterbar ist: Der Greedy Algorithmus ist also 2-approximativ.

Warum sind f-Matchings 2-erweiterbar?

- Betrachte zwei f-Matchings $Y_1 \cup \{e\}, Y_2$ mit $Y_1 \subset Y_2$, wobei Kante $e = \{u, v\}$ nicht in Y_2 liege.
- Wenn $Y_2 \cup \{e\}$ ein f-Matching ist, dann sind wir fertig.
- Ist $Y_2 \cup \{e\}$ kein f-Matching,
  - dann besitzt Y_2 bereits $f(u)$ Kanten, die inzident mit u, oder $f(v)$ Kanten, die inzident mit v sind.
  - Entferne eine mit u und eine mit v inzidente Kante aus Y_2 und setze e ein!

MAX-TSP

MAX-TSP (1/2)

- Ein vollständiger gerichteter Graph mit nicht-negativen Kantengewichten ist gegeben.
- Bestimme eine Rundreise **maximaler** Länge. (Eine Rundreise besucht jeden Knoten genau einmal.)

- Die Grundmenge besteht aus allen Kanten. Eine Teilmenge der Kanten ist unabhängig, wenn
  - ihre Kanten Knoten-disjunkte Pfade oder eine Rundreise bilden.
- Zeige, dass dieses Mengensystem 3-erweiterbar ist.
- Der Greedy-Algorithmus berechnet also eine 3-approximative Lösung. Der beste von effizienten Algorithmen erreichbare Approximationsfaktor ist $8/5$.

Warum sind Knoten-disjunkte Pfade 3-erweiterbar?

- $Y_1 \cup \{e\}$ und Y_2 seien zwei unabhängige Mengen mit $Y_1 \subset Y_2$, wobei die Kante $e = (u, v)$ nicht in der Menge Y_2 liege.
- Entferne zuerst, falls vorhanden, die eine u verlassende und die eine in in v eintreffende Kante aus Y_2.
- Füge e zu Y_2 hinzu und jeder Knoten hat höchstens eine eintreffende und eine ausgehende Kante.

  Wenn die Hinzunahme von e unzulässig ist, dann muss es genau einen Kreis geben, der keine Rundreise ist, aber e benutzt.
- Aber dann hat dieser Kreis eine Kante e', die nicht zur Menge Y_1 gehört.

  Entfernen die Kante e', der Kreis ist gebrochen, und wir haben eine unabhängige Menge erhalten.

Sequenzierung

Sequenzierung

Bestimme die Reihenfolge der Basenpaare für einen Strang eines DNA-Moleküls.

- Direkte Sequenzierung:
  - Schneide iterativ kurze Präfixe von ungefähr 500 Basenpaaren aus der DNA Sequenz und
  - sequenziere die kleinen Fragmente.

    Extrem zeitaufwändiger Prozess, der nicht für die Sequenzierung langer Sequenzen geeignet ist.
- Shotgun Sequenzierung:
  - zerschneide die Sequenz parallel, an fast zufällig verteilten Positionen und sequenziere die entstehenden Fragmente.
  - Leider geht die Reihenfolge der Fragmente verloren.
  - Führe den Prozess für genügend viele Kopien der Sequenz aus.

    Bestimme die korrekte Reihenfolge algorithmisch.

Shotgun Sequenzierung und Fragment Assmenbly

(1) Führe das Shotgun-Verfahren durch.
(2) Bestimme *Reads* für jedes Fragment, also die Folge der ungefähr 500 Basenpaare an einem Ende des Fragments.
(3) Fragment-Assembly: Rekonstruiere die DNA-Sequenz aus den Reads.

Die Aufgabenstellung der Informatik:

Fragment Assembly

- Es gibt viele „Superstrings“, also Strings, die jeden Read als Teilstring besitzen.
- Sollten wir den kürzesten Superstring wählen?

Was so alles passieren kann

(1) Fehler beim Sequenzieren der Fragmente: Einige Basen werden falsch angegeben.

(2) Fehler beim Kopieren: Zum Beispiel können sich Fragmente einer Wirt-DNA mit den Fragmenten der DNA-Sequenz vermischen.

(3) Fragmente von beiden Strängen des Moleküls tauchen auf.

(4) Überdeckungslücken: Einige Regionen der DNA-Sequenz werden möglicherweise nicht von Fragmenten überdeckt.

(5) Lange *Repeats* tauchen auf:

- Repeats sind wiederholt in der DNA-Sequenz auftretende Teilstrings.
- Beispielsweise gibt es ein 300 Basenpaare langes Segment, das mit 5-15 prozentiger Variation mehr als 1 Million Mal im menschlichen Genom vorkommt:

  Datenmüll pur!

Shortest Common Superstring

SHORTEST COMMON SUPERSTRING

- Eine Menge U von Strings ist gegeben.
- Suche einen String S minimaler Länge, der alle Strings in U als Teilstrings enthält.

- Beachte, dass sich kürzeste Superstrings nicht unbedingt als Lösung des Fragment Assembly Problems aufdrängen.

  Lange Repeats!
- SHORTEST COMMON SUPERSTRING ist deshalb eine sehr optimistische Abstraktion.

  In praktischen Anwendungen werden Methoden für SHORTEST COMMON SUPERSTRING mit anderen Verfahren kombiniert, um Fehlertoleranz zu erreichen.

Overlaps

Wir können annehmen, dass kein String in U Teilstring eines anderen Strings in U ist.

- $\text{overlap}(u, v)$ die Länge des längsten Suffix von u, der auch Präfix von v und
- $\text{präfix}(u, v) = |u| - \text{overlap}(u, v)$ ist die Länge des nicht überdeckten Präfixes von u.
- Die Menge U bestehe aus k Strings. Für eine Bijektion $\pi : \{1, \ldots, k\} \to U$ erhalten wir den Superstring

$$S(\pi) = \text{präfix}(\pi(1), \pi(2)) \cdots \text{präfix}(\pi(k-1), \pi(k)) \cdot \pi(k).$$

- Wie lang ist $S(\pi)$?

$$|S(\pi)| = \sum_{u \in U} |u| - \sum_{i=1}^{k-1} \text{overlap}(\pi(i), \pi(i+1)).$$

Overlaps und Perioden

Es ist $|S(\pi)| = \sum_{u \in U} |u| - \sum_{i=1}^{k-1} \text{overlap}(\pi(i), \pi(i+1))$.

- Wir müssen also eine Permutation π mit einer möglichst großen Overlap-Summe bestimmen.
- Wie groß können Overlaps werden?

  Das hängt sehr stark von den Perioden der Strings ab!

- Fasse einen String v als zyklischen String auf:

  Klebe die ersten und letzten Buchstaben von v aneinander.
- String u hat die Periode v, falls wir u vollständig um v wickeln können.
- v ist die minimale Periode von u, falls u die Periode v hat und falls $|v|$ die minimale Länge einer Periode von u ist.

Das Overlap Lemma

Das Overlap Lemma

- Für koperiodische Strings u_1, u_2, also Strings mit gleicher Periode, kann overlap(u_1, u_2) sehr gross werden.
- Und für aperiodische Strings, also Strings mit verschiedenen minimalen Perioden?

Das Overlap-Lemma (ohne Beweis)

Für aperiodische Strings u_1 und u_2 mit Periodenlängen p_1, p_2 ist

$$\text{overlap}(u_1, u_2) < p_1 + p_2.$$

- Ein Beispiel:
  - $u_1 = ab^k ab^k$ und $u_2 = b^k abb^k ab$ haben die Periodenlängen $k+1$, bzw. $k+2$.
  - $\max\{k+1, k+2\} < \text{overlap}(u_1, u_2) = 2k+1 < (k+1)+(k+2)$.

Das zyklische Überdeckungsproblem

Zyklische Überdeckungen

(a) Für einen String v sei v^∞ der unendlich oft mit sich selbst konkatenierte String v.

(b) Ein String v ist eine zyklische Überdeckung eines Strings u, wenn u ein Teilstring des Strings v^∞ ist.

(c) U und V sind Mengen von Strings:

- V ist eine zyklische Überdeckung von U, wenn jedes $u \in U$ von mindestens einem String $v \in V$ zyklisch überdeckt wird.
- $\sum_{v \in V} |v|$ ist die Länge der Zyklus-Überdeckung V.

(d) ZYKLUS ÜBERDECKUNG: Bestimme eine Zyklus-Überdeckung minimaler Länge für eine Menge U von Strings.

Und wenn wir minimale Zyklusüberdeckungen effizient bestimmen können?

Bestimmung kurzer Superstrings

(1) Eine Menge U von Strings ist gegeben.

Kein String aus U ist Teilstring eines anderen Strings aus U.

(2) Bestimme eine minimale Zyklus-Überdeckung V von U.

(3) Betrachte alle Zyklen $v \in V$ nacheinander.

- U_v besteht aus allen von v zyklisch überdeckten Strings aus U.
- Das erste Vorkommen der Strings aus U_v in v^∞ definiert eine Ordnung auf den Strings aus U_v.

Bestimme den durch diese Ordnung induzierten Superstring S_v.

Kommentar: Wir erhalten S_v, wenn wir den Zyklus v „aufbrechen“, also die Strings in U_v gemäß ihrer Ordnung maximal übereinander schieben, aber den letzten String nicht über den ersten schieben.

(4) Gib die Konkatenation $\Pi_{v \in V} S_v$ der Superstrings als Resultat aus.

Die Analyse (1/2)

- Sei V eine minimale Zyklusüberdeckung von U.
  - Für $v \in V$ sei u_v der letzte von v zyklisch überdeckte String.
  - Wie lang ist S_v?

$$|S_v| \leqslant |v| + |u_v|.$$

- Wir berechnen einen Superstring der Länge L mit

$$L \leqslant \sum_{v \in V} (|v| + |u_v|).$$

- Sei L^* die Länge eines kürzesten Superstrings S^* auf den Strings u_v für alle $v \in V$.

  Die Strings $u_1, \ldots, u_r$ mögen in dieser Reihenfolge in S^* vorkommen. Dann ist

$$L^* = \sum_{i=1}^{r} |u_i| - \sum_{i=1}^{r-1} \text{overlap}(u_i, u_{i+1}) > \sum_{i=1}^{r} |u_i| - \sum_{i=1}^{r-1} (|v_i| + |v_{i+1}|)$$

Die Analyse (2/2)

Wir wissen:

(*) $L \leqslant \sum_{v \in V}(|v| + |u_v|)$

(*) und $L^* > \sum_{i=1}^{r} |u_i| - \sum_{i=1}^{r-1}(|v_i| + |v_{i+1}|) \geqslant \sum_{i=1}^{r} |u_i| - 2\sum_{v \in V} |v|$.

Als Konsequenz:

$$L \leqslant \sum_{v \in V} |v| + \sum_{i=1}^{r} |u_i| < L^* + 3 \cdot \sum_{v \in V} |v| \leqslant 4 \cdot L_{\text{opt}},$$

denn die Länge L_{opt} eines kürzesten Superstrings ist mindestens so groß wie L^* und die Länge einer kürzesten Zyklusüberdeckung.

Unser Ansatz berechnet eine 4-approximative Lösung für SHORTEST COMMON SUPERSTRING.

Wie berechnet man eine minimale Zyklusüberdeckung?

Die Monge Ungleichung

Übungsaufgabe

u_1, u_2, v_1, v_2 sind vier beliebige Strings mit

$$\text{overlap}(u_1, v_1) = \max\{\text{overlap}(u_i, v_j) \mid 1 \leqslant i, j \leqslant 2\}.$$

Dann gilt die *Monge-Ungleichung* (Gaspard Monge 1746-1818):

$$\text{overlap}(u_1, v_1) + \text{overlap}(u_2, v_2) \geqslant \text{overlap}(u_1, v_2) + \text{overlap}(u_2, v_1).$$

Bestimmung einer minimalen Zyklusüberdeckung

(1) Die Eingabe besteht aus einer Menge U von Strings.

(2) Bestimme den gerichteten Graph $G = (U, U \times U)$ mit den Kantengewichten $\text{overlap}(u_1, u_2)$.

(3) Sortiere die Kanten in E nach ihrem Gewicht. Setze $E^* = \emptyset$.

(4) Wiederhole, solange $E \neq \emptyset$:

- **(4a)** Bestimme die Kante $e = (u_1, u_2) \in E$ mit größtem Overlap. Füge e zu E^* hinzu.
- **(4b)** Entferne alle den Knoten u_1 verlassenden und alle den Knoten u_2 erreichenden Kanten.

(5) Berechne eine Zyklus-Überdeckung aus den durch E^* definierten Kreisen.

Der Greedy Superstring Algorithmus

Der Greedy Superstring Algorithmus (1/2)

Der wahrscheinlich im Allgemeinen bessere Algorithmus:

(1) Gegeben ist eine Menge U von Strings, wobei keine zwei Strings in U Teilstrings voneinander sind.

(2) Wiederhole, solange bis $|U| = 1$.

(2a) Bestimme zwei Strings $\mathbf{u} \neq \mathbf{v} \in U$ mit $\mathrm{overlap}(u, v)$ größtmöglich.

(2b) Ersetze U durch $U \setminus \{u, v\} \cup \{w\}$, wobei $w = \mathrm{praefix}(u, v) \cdot v$.

Kommentar: Schiebe v weitestgehend über u, und wir erhalten w.

(3) Gib den verbleibenden String in U als Superstring aus.

Offene Frage

Was ist der Approximationsfaktor des Greedy Algorithmus?

Was ist bekannt?

- Der Approximationsfaktor ist höchstens vier,
- aber mindestens zwei:

$$\begin{aligned} u_1 &= c(ab)^n, \\ u_2 &= (ba)^n, \\ u_3 &= (ab)^n d. \end{aligned}$$

- Die optimale Reihenfolge ist u_1, u_2, u_3 mit Länge $2n + 4$,
- Greedy wählt die Reihenfolge u_1, u_3, u_2 mit Länge $4n + 2$.

Fazit

- Wir haben eine minimale Zyklusüberdeckung mit einem nicht-adaptiven Priority Algorithmus exakt bestimmt.
- Der Greedy Superstring Algorithmus ist fast identisch, darf aber keine Kreise schließen.

Eine der wichtigsten Probleme im Gebiet der String Algorithmen:

Bestimme den Approximationsfaktor des Greedy Superstring Algorithmus.

Das Travelling Salesman Problem

TSP

- Greedy-Algorithmen sind zu schwach, um schwierige Optimierungsprobleme exakt zu lösen.
- Aber ihr Einsatzgebiet als Heuristiken für die approximative Lösung ist extrem breit.
- Wir betrachten exemplarisch das metrische Travelling Salesman Problem (TSP).

- n Orte $v_1, \ldots, v_n$ sowie Distanzen $d(v_i, v_j) \geqslant 0$ zwischen je zwei verschiedenen Orten sind gegeben.
- Eine Rundreise entspricht einer Permutation $v_{\pi(1)}, v_{\pi(2)}, \ldots, v_{\pi(n)}$ der Orte:

  Wenn $v_{\pi(i)}$ besucht wird, dann wird $v_{\pi(i+1)}$ für $i < n$, bzw v_1 für $i = n$ als nächster Ort besucht.
- Bestimme eine Rundreise minimaler Länge.

Wie schwierig ist TSP?

Das Sprachenproblem HAMILTONSCHER KREIS

Gibt es einen Hamiltonschen Kreis, also einen Kreis, der alle Knoten eines ungerichteten Graphen $G = (V, E)$ genau einmal besucht?

ist $\mathcal{NP}$-vollständig.

- Sei $G = (\{1, \ldots, n\}, E)$ eine Eingabe für HAMILTONSCHER KREIS. Erfinde die Instanz $T(G)$ für *TSP*:
  - Die Orte entsprechen den Knoten von V.
  - Für alle Knoten $u, v \in V$: Wenn $\{u, v\}$ eine Kante in G ist, dann setzen wir $d(u, v) = 1$ und ansonsten $d(u, v) = D$.
- G besitzt genau dann einen Hamiltonschen Kreis, wenn $T(G)$ eine Rundreise der Länge höchstens n besitzt.
- Besitzt G hingegen keinen Hamiltonschen Kreis, dann hat jede Rundreise mindestens die Länge $n - 1 + D$.

Das metrische TSP

Das metrische TSP

Es gelte P $\neq$ NP.

- Setze $D = n \cdot 2^n$ und der Approximationsfaktor effizienter Algorithmen ist mindestens $\frac{n-1+D}{n} \geqslant 2^n$.
- Es gibt keinen effizienten 2^n-approximativen Algorithmus für *TSP*!

- TSP ist viel zu schwierig.
- Wir betrachten das einfachere und glücklicherweise interessantere **metrische TSP**:

  Wir verlangen, dass die Distanzen zwischen den Orten (oder Punkten) durch eine Metrik gegeben sind: d ist genau dann eine Metrik, wenn für alle Punkte u, v, w gilt:

  - $d(u, u) = 0$,
  - $d(u, v) = d(v, u)$ und
  - $d(u, v) + d(v, w) \leqslant d(u, w)$.

Die Spannbaum Heuristik

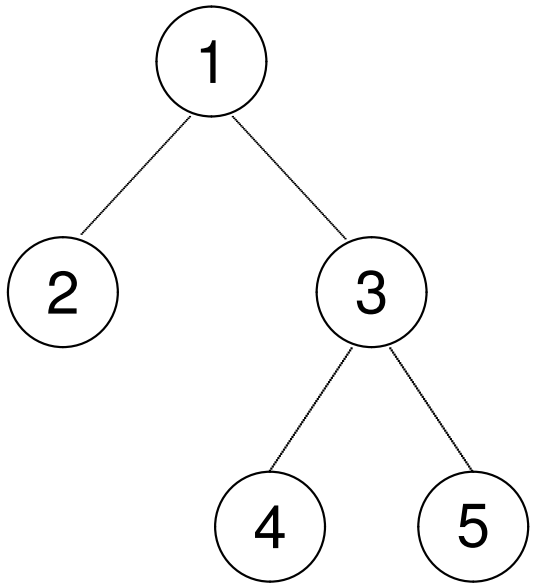
- G ist der vollständige Graph mit allen Orten als Knoten (und Kanten zwischen je zwei Knoten).
- Die Kante $\{u, v\}$ erhält das Gewicht $d(u, v)$.

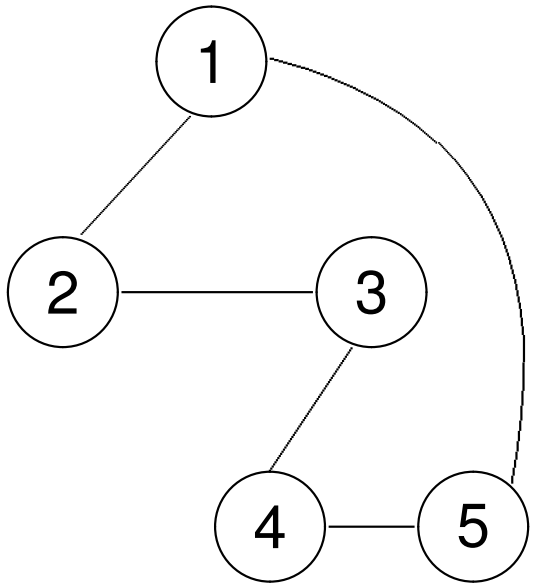
- Die Spannbaum Heuristik:
  - Berechne einen minimalen Spannbaum B.
  - Besuche alle Orte gemäß einem **Präorderdurchlauf** von B.
- Wie lang ist die Rundreise?
  - Irgendein Rücksprung während der Präorder-Traversierung ist nicht länger als die Länge aller beteiligten Kanten $\Rightarrow$ Die Traversierung ist also höchstens doppelt so lang wie das Gewicht von B.
  - Andererseits ist jede Rundreise mindestens so lang wie das Gewicht von B.

Die Spannbaum Heuristik ist 2-approximativ.

Die Spannbaum Heuristik: Ein Beispiel

Durchlaufe den minimalen Spannbaum B in Präorder-Reihenfolge.





Wie lang ist unsere Rundreise höchstens?

$$d_{1,2} + (d_{2,1} + d_{1,3}) + d_{3,4} + (d_{4,3} + d_{3,5}) + (d_{5,3} + d_{3,1}).$$

Alle Kanten des Spannbaums treten genau zweimal auf. Es ist

$$\begin{aligned} \text{Gewicht}(B) &\leqslant \text{minimale Länge einer Rundreise} \\ &\leqslant \text{Länge unserer Rundreise} \leqslant 2\,\text{Gewicht}(B). \end{aligned}$$

Der Algorithmus von Christofides

Der Algorithmus von Christofides (1/2)

Sei G ein ungerichteter, zusammenhängender Graph.

G besitzt eine Euler-Tour –einen Kreis, der jede Kante genau einmal durchläuft– $\Leftrightarrow$ Jeder Knoten besitzt eine gerade Anzahl von Nachbarn.

$\Rightarrow$
- Wenn es eine Euler-Tour gibt, betritt man auf einer solchen Tour jeden Knoten so oft, wie man ihn verlässt.
- Da jede Kante auf der Tour genau einmal besucht wird, besitzt jeder Knoten eine gerade Anzahl von Nachbarn.

$\Leftarrow$
- Jeder Knoten besitzt eine gerade Anzahl von Nachbarn: Der Graph kann in kantendisjunkte Kreise zerlegt werden.
- Verschmelze zwei Kreise mit einem gemeinsamen Knoten zu einem Kreis.
- G ist zusammenhängend: Alle Kreise lassen sich zu einer Euler-Tour verschmelzen.

Der Algorithmus von Christofides (2/2)

(1) Die Eingabe besteht aus einer Menge $V = \{1, \ldots, n\}$ von n Punkten und der Metrik d.

(2) Setze $G := (V, \{\{v, w\} \mid v, w \in V, v \neq w\})$ und weise der Kante $e = \{v, w\}$ die Distanz $d_e = d_{v,w}$ zu.

(3) Berechne einen **minimalen Spannbaum** B für G.

(4) Sei U die Menge der Knoten von B mit einer ungeraden Anzahl von Nachbarn in B. Berechne ein **minimales Matching** M der Größe $\frac{1}{2} \cdot |U|$ auf den Knoten in U.

(5) Füge die Kanten in M zu den Kanten in B hinzu und konstruiere eine **Euler-Tour** E.

(6) Berechne aus E eine Rundreise durch Abkürzungen.

Der Algorithmus von Christofides: Die Analyse

- Jeder ungerichtete Graph $G = (V, E)$ besitzt eine gerade Anzahl von Knoten mit einer ungeraden Anzahl von Nachbarn.
  - In $\sum_{v \in V} |\{w \mid \{v, w\} \in E\}|$ wird jede Kante zweimal gezählt.
  - Die Menge U hat eine gerade Anzahl von Elementen.
-
  - L_{opt} ist die Länge einer kürzesten Rundreise,
  - MS_{opt} ist die Länge eines minimalen Spannbaumes und
  - Match das Gewicht eines minimalen Matchings auf U.
- Es ist $\text{MS}_{\text{opt}} \leqslant L_{\text{opt}}$.
- $\text{Match} \leqslant \frac{1}{2} \cdot L_{\text{opt}}$:
  - Die Länge einer kürzesten Rundreise auf den Punkten in U ist höchstens L_{opt}.
  - Eine Rundreise auf den gerade vielen Punkten aus U zerlegt sich in zwei perfekte Matchings.
- Die Länge der Rundreise ist höchstens $\text{MS}_{\text{opt}} + \text{Match} \leqslant \frac{3}{2} \cdot L_{\text{opt}}$.

Der Algorithmus von Christofides: Eine Zusammenfassung

Der Algorithmus von Christofides ist ein $\frac{3}{2}$-approximativer Algorithmus.

- Was war die Idee?
  - Verfolge weiterhin die Spannbaum Heuristik.
  - Ersetze aber einen Präorderdurchlauf durch eine Eulertour.
    - ★ Dazu muss gewährleistet sein, dass jeder Knoten des Spannbaums geraden Grad hat.
    - ★ Füge ein leichtestes Matching auf den Knoten mit ungeradem Grad zum Spannbaum hinzu.
- Das metrische TSP besitzt eine Fülle weiterer Heuristiken.

Nearest Neighbor, Nearest und Farthest Insertion

Nearest Neighbor

Wenn **Nearest Neighbor** den Punkt v erreicht hat, dann wird als Nächster der Punkt w besucht, wobei w unter allen noch nicht besuchten Knoten eine minimale Distanz zu v hat.

- Der Approximationsfaktor von Nearest-Neighbor ist unbeschränkt,
- aber durch $O(\log_2 n)$ für n Punkte beschränkt.

Nearest und Farthest Insertion

- **Nearest Insertion** beginnt mit einem Paar nächstliegender Punkte.
- Wähle einen zur bisher gebauten partiellen Rundreise nächstliegenden Punkt.
- Füge den Punkt so hinzu, dass die Länge der Rundreise geringstmöglich anwächst.

- **Farthest Insertion** beginnt mit einem Paar am weitesten voneinander entfernter Punkte.
- Farthest Insertion verhält sich wie Nearest Insertion, aber
- wählt den von der bisher gebauten partiellen Rundreise am weitesten entfernten Punkt.

- **Nearest Insertion** ist 2-approximativ,
- der Approximationsfaktor von **Farthest Insertion** ist unbekannt, aber durch $O(\log_2 n)$ beschränkt.

Eine experimentelle Analyse

Bis zu 100.000 Punkte werden zufällig aus dem Quadrat $[0, 1]^2$ ausgewählt.

- Der **Algorithmus von Christofides** ist der Gewinner, allerdings mit geringem Abstand zu **Farthest Insertion**.
- **Nearest-Insertion** wird deutlich geschlagen und endet auf Platz drei.
- Der klare Verlierer ist die **Spannbaum Heuristik**, die bis zu 40% über der optimalen Länge einer Rundreise liegt.
- Der **Christofides Algorithmus** und **Farthest Insertion** sind ungefähr 10% über dem Wert einer optimalen Rundreise, während **Nearest Insertion** um bis zu 25% schwächer ist.