Hauptseminar im Fachbereich Komplexitätstheorie und Effiziente Algorithmen

Thema: *kürzeste überdeckende Wörter*

Gliederung:

Gliederung:

- **Einführung**

Gliederung:

- **Einführung**
- **Faktor 4 Approximation**

Gliederung:

- **Einführung**
- **Faktor 4 Approximation**
- **Kompression**

Gliederung:

- **Einführung**
- **Faktor 4 Approximation**
- **Kompression**
- **Faktor 3 Approximation**

Gliederung:

- Einführung
- Faktor 4 Approximation
- Kompression
- Faktor 3 Approximation
- Zusammenfassung

Gliederung:

- **Einführung**
- Faktor 4 Approximation
- Kompression
- Faktor 3 Approximation
- Zusammenfassung

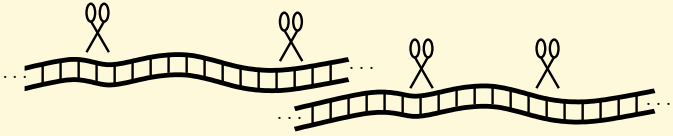
Aufgabe: Bestimme die Sequenz eines DNA-Strangs.

Aufgabe: Bestimme die Sequenz eines DNA-Strangs.

- Viele DNA-Stränge für die selbe Sequenz gegeben

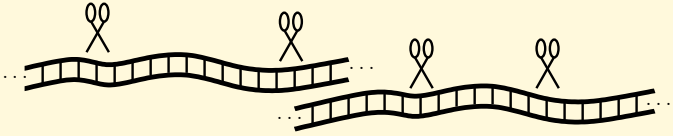
Aufgabe: Bestimme die Sequenz eines DNA-Strangs.

- Viele DNA-Stränge für die selbe Sequenz gegeben
- Auftrennen aller Stränge an unbekannten Stellen



Aufgabe: Bestimme die Sequenz eines DNA-Strangs.

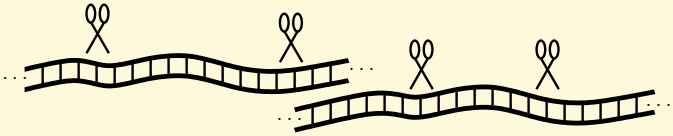
- Viele DNA-Stränge für die selbe Sequenz gegeben
- Auftrennen aller Stränge an unbekannten Stellen



- Sequenzieren aller Fragmente

Aufgabe: Bestimme die Sequenz eines DNA-Strangs.

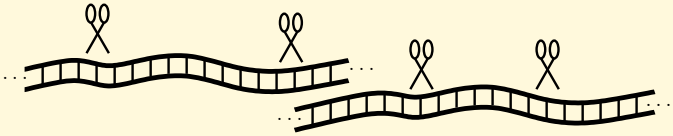
- Viele DNA-Stränge für die selbe Sequenz gegeben
- Auftrennen aller Stränge an unbekannten Stellen



- Sequenzieren aller Fragmente

Aufgabe: Bestimme die Sequenz eines DNA-Strangs.

- Viele DNA-Stränge für die selbe Sequenz gegeben
- Auftrennen aller Stränge an unbekannten Stellen

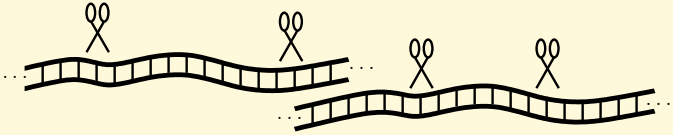


- Sequenzieren aller Fragmente

- Zusammenfügen der Fragmente zu einer Kette, die alle Fragmente enthält und möglichst kurz ist

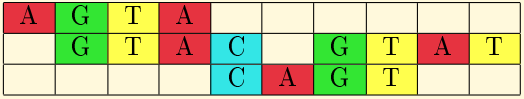
Aufgabe: Bestimme die Sequenz eines DNA-Strangs.

- Viele DNA-Stränge für die selbe Sequenz gegeben
- Auftrennen aller Stränge an unbekannten Stellen



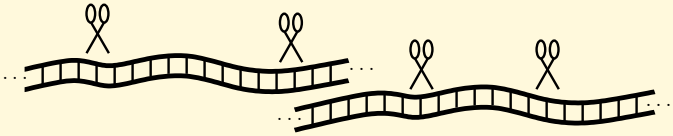
- Sequenzieren aller Fragmente

- Zusammenfügen der Fragmente zu einer Kette, die alle Fragmente enthält und möglichst kurz ist



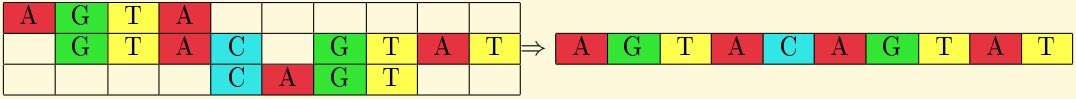
Aufgabe: Bestimme die Sequenz eines DNA-Strangs.

- Viele DNA-Stränge für die selbe Sequenz gegeben
- Auftrennen aller Stränge an unbekannten Stellen



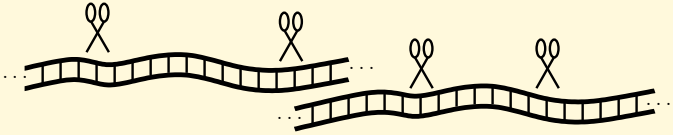
- Sequenzieren aller Fragmente

- Zusammenfügen der Fragmente zu einer Kette, die alle Fragmente enthält und möglichst kurz ist



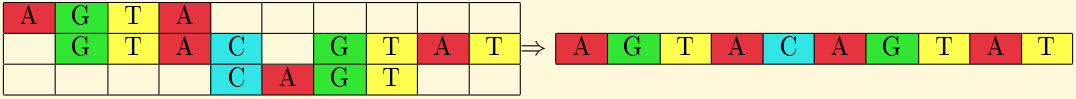
Aufgabe: Bestimme die Sequenz eines DNA-Strangs.

- Viele DNA-Stränge für die selbe Sequenz gegeben
- Auftrennen aller Stränge an unbekannten Stellen



- Sequenzieren aller Fragmente

- Zusammenfügen der Fragmente zu einer Kette, die alle Fragmente enthält und möglichst kurz ist



- Kette ist gesuchte DNA Sequenz

Formaler:

Formaler:

- Gegeben, Menge von Wörtern $S = \{s_1, \ldots, s_n\} \subseteq \Sigma^+$

Formaler:

- Gegeben, Menge von Wörtern $S = \{s_1, \ldots, s_n\} \subseteq \Sigma^+$
- Wort s heisst *überdeckendes Wort* für S (engl. *superstring*), falls s alle s_i als Teilwörter enthält

Formaler:

- Gegeben, Menge von Wörtern $S = \{s_1, \ldots, s_n\} \subseteq \Sigma^+$
- Wort s heisst *überdeckendes Wort* für S (engl. *superstring*), falls s alle s_i als Teilwörter enthält
- Ist $|s|$ minimal, so heisst s ein *kürzestes überdeckendes Wort* (engl. *shortest superstring*)

Formaler:

- Gegeben, Menge von Wörtern $S = \{s_1, \ldots, s_n\} \subseteq \Sigma^+$
- Wort s heisst *überdeckendes Wort* für S (engl. *superstring*), falls s alle s_i als Teilwörter enthält
- Ist $|s|$ minimal, so heisst s ein *kürzestes überdeckendes Wort* (engl. *shortest superstring*)
- Problem bezeichnen wir mit SSP

Beispiel:

Beispiel:

- S={abra,brac,cabr,brad,bra}

Beispiel:

- S={abra,brac,cabr,brad,bra}
- bra ist Teilwort von abra $\Rightarrow$ bra entfernen

Beispiel:

- S={abra,brac,cabr,brad,bra}
- bra ist Teilwort von abra $\Rightarrow$ bra entfernen
- abrabraccabrbrad ist *naiver Superstring* von S mit Länge 16.

Beispiel:

- S={abra,brac,cabr,brad,bra}
- bra ist Teilwort von abra $\Rightarrow$ bra entfernen
- abrabraccabrbrad ist *naiver Superstring* von S mit Länge 16.
- abracabrad ist ein kürzester Superstring mit Länge 10.

Wie kann man zu gegebenen S ein kürzestes s finden?

Wie kann man zu gegebenen S ein kürzestes s finden?

Satz 1. *SSP ist NP-vollständig.*

Wie kann man zu gegebenen S ein kürzestes s finden?

Satz 1. *SSP ist NP-vollständig.*

- SSP ist NP-Vollständig

- SSP ist NP-Vollständig $\Rightarrow$ falls $P \neq NP$ gilt, existiert kein effizienter Algorithmus für SSP

- SSP ist NP-Vollständig $\Rightarrow$ falls $P \neq NP$ gilt, existiert kein effizienter Algorithmus für SSP
- Mitteles *Approximationsalgorithmen* lassen sich jedoch angenäherte Lösungen finden

- SSP ist NP-Vollständig $\Rightarrow$ falls $P \neq NP$ gilt, existiert kein effizienter Algorithmus für SSP
- Mitteles *Approximationsalgorithmen* lassen sich jedoch angenäherte Lösungen finden

Gliederung:

- **Einführung**
- **Faktor 4 Approximation**
- Kompression
- Faktor 3 Approximation
- Zusammenfassung

- OPT bezeichnet Länge eines kürzesten Superstring einer Menge S

- OPT bezeichnet Länge eines kürzesten Superstring einer Menge S
- Suchen nun Verfahren mit Ausgabe s nicht länger als $k \cdot \mathrm{OPT}$, d.h. $|s| \leq k \cdot \mathrm{OPT}$

- OPT bezeichnet Länge eines kürzesten Superstring einer Menge S
- Suchen nun Verfahren mit Ausgabe s nicht länger als $k \cdot \mathrm{OPT}$, d.h. $|s| \leq k \cdot \mathrm{OPT}$
- Problem: Wissen über OPT nichts, wie kann man dann Länge der Ausgabe so garantieren?

- Lösung: Finden einer möglichst guten unteren Schranke OPT_l für OPT, über die man eine solche Aussage machen kann

- Lösung: Finden einer möglichst guten unteren Schranke OPT_l für OPT, über die man eine solche Aussage machen kann
- Denn: Aus $\mathrm{OPT}_\mathrm{l} \leq \mathrm{OPT}$ und $|s| \leq k \cdot \mathrm{OPT}_\mathrm{l}$ folgt,

- Lösung: Finden einer möglichst guten unteren Schranke OPT_l für OPT, über die man eine solche Aussage machen kann
- Denn: Aus $\mathrm{OPT}_\mathrm{l} \leq \mathrm{OPT}$ und $|s| \leq k \cdot \mathrm{OPT}_\mathrm{l}$ folgt, dass $|s| \leq k \cdot \mathrm{OPT}$

Bezeichnungen

Bezeichnungen

- $overlap(s_i, s_j)$ ... Überlappung von s_i mit s_j

Bezeichnungen

- $overlap(s_i, s_j)$ ... Überlappung von s_i mit s_j
- $prefix(s_i, s_j)$ ... Präfix von s_i, der bei Entfernung der Überlappung mit s_j entsteht

Bezeichnungen

- $overlap(s_i, s_j)$ ... Überlappung von s_i mit s_j
- $prefix(s_i, s_j)$ ... Präfix von s_i, der bei Entfernung der Überlappung mit s_j entsteht
- Bsp: $overlap$(abra,brad)=bra und $prefix$(abra,brad)=a

Bezeichnungen

- $overlap(s_i, s_j)$ ... Überlappung von s_i mit s_j
- $prefix(s_i, s_j)$ ... Präfix von s_i, der bei Entfernung der Überlappung mit s_j entsteht
- Bsp: $overlap$(abra,brad)=bra und $prefix$(abra,brad)=a

Definition (Präfixgraph)

Bezeichnungen

- $overlap(s_i, s_j)$ ... Überlappung von s_i mit s_j
- $prefix(s_i, s_j)$ ... Präfix von s_i, der bei Entfernung der Überlappung mit s_j entsteht
- Bsp: $overlap$(abra,brad)=bra und $prefix$(abra,brad)=a

Definition (Präfixgraph)

- Gegeben Menge von Wörtern $X = \{x_1, \ldots, x_n\}$

Bezeichnungen

- $overlap(s_i, s_j)$ ... Überlappung von s_i mit s_j
- $prefix(s_i, s_j)$ ... Präfix von s_i, der bei Entfernung der Überlappung mit s_j entsteht
- Bsp: $overlap$(abra,brad)=bra und $prefix$(abra,brad)=a

Definition (Präfixgraph)

- Gegeben Menge von Wörtern $X = \{x_1, \ldots, x_n\}$
- *Präfixgraph* $I(X)$ ist gerichter Graph mit n Knoten entsprechend x_i

Bezeichnungen

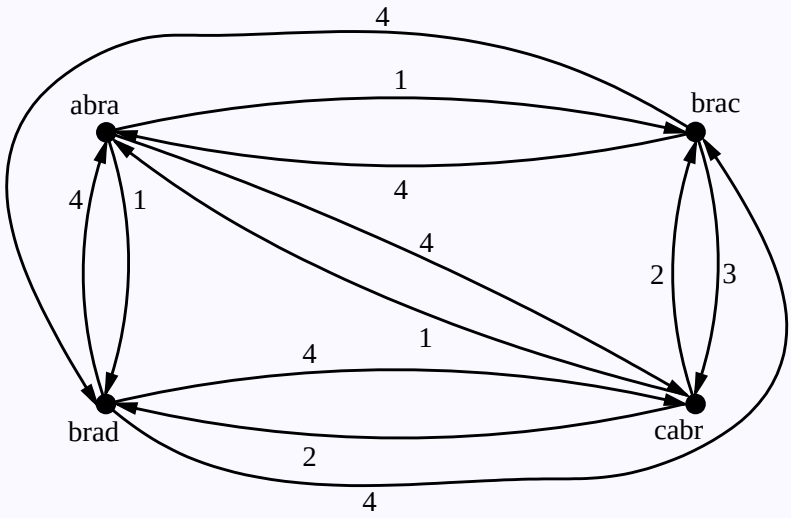
- $overlap(s_i, s_j)$ ... Überlappung von s_i mit s_j
- $prefix(s_i, s_j)$ ... Präfix von s_i, der bei Entfernung der Überlappung mit s_j entsteht
- Bsp: $overlap$(abra,brad)=bra und $prefix$(abra,brad)=a

Definition (Präfixgraph)

- Gegeben Menge von Wörtern $X = \{x_1, \ldots, x_n\}$
- *Präfixgraph* $I(X)$ ist gerichter Graph mit n Knoten entsprechend x_i
- Für alle $1 \leq i, j \leq n$ und $i \neq j$ hat Kante (v_i, v_j) Kosten von $|prefix(x_i, x_j)|$
- Für alle $1 \leq i \leq n$ hat Kante (v_i, v_i) Kosten von $|x_i|$.

Beispiel für Präfixgraphen, wenn S={abra,brac,cabr,brad}

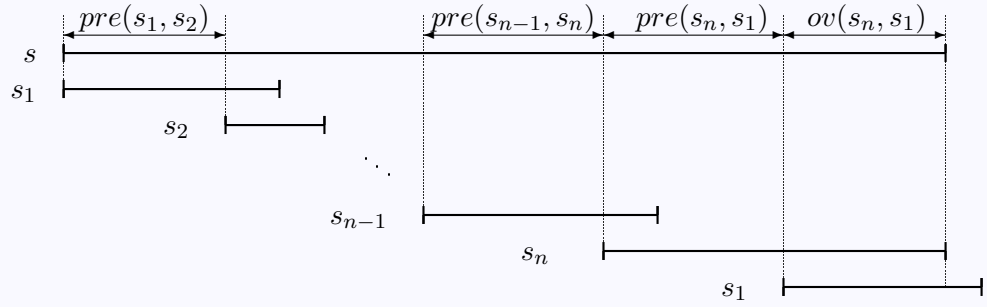
Beispiel für Präfixgraphen, wenn S={abra,brac,cabr,brad}



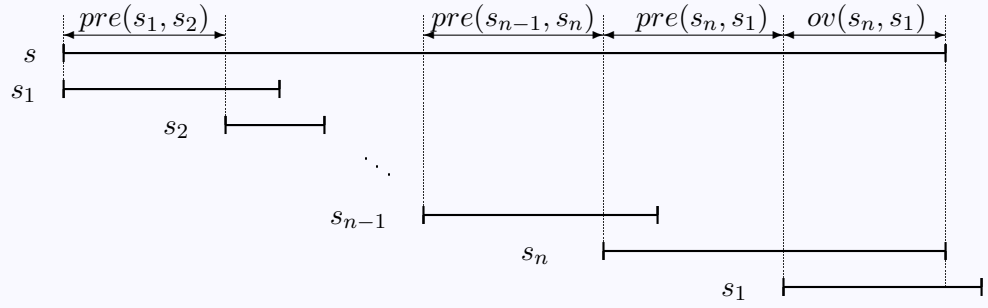
Betrachten die Struktur von einem kürzesten Superstring s.

Betrachten die Struktur von einem kürzesten Superstring s. Annahme, dass $s_1, \ldots, s_n$ in dieser Anordung in s vorkommen.

Betrachten die Struktur von einem kürzesten Superstring s. Annahme, dass $s_1, \ldots, s_n$ in dieser Anordung in s vorkommen.



Betrachten die Struktur von einem kürzesten Superstring s. Annahme, dass $s_1, \ldots, s_n$ in dieser Anordung in s vorkommen.



$$\text{OPT} = |s| = |prefix(s_1, s_2)| + |prefix(s_2, s_3)| + \ldots +$$

$$|prefix(s_n, s_1)| + |overlap(s_n, s_1)|$$

$$\mathrm{OPT} = |prefix(s_1, s_2)| + \ldots + |prefix(s_n, s_1)| + |overlap(s_n, s_1)|$$

$\mathrm{OPT} = |\mathit{prefix}(s_1, s_2)| + \ldots + |\mathit{prefix}(s_n, s_1)| + |\mathit{overlap}(s_n, s_1)|$

Ähnlichkeit mit Minimum TSP Tour

$$1 \to 2 \to \ldots \to n \to 1$$

auf Präfixgraphen von S. Denn

$$\text{OPT} = |prefix(s_1, s_2)| + \ldots + |prefix(s_n, s_1)| + |overlap(s_n, s_1)|$$

Ähnlichkeit mit Minimum TSP Tour

$$1 \to 2 \to \ldots \to n \to 1$$

auf Präfixgraphen von S. Denn

$$\text{TSP} = |prefix(s_1, s_2)| + \ldots + |prefix(s_n, s_1)|$$

$$\text{OPT} = |\mathit{prefix}(s_1, s_2)| + \ldots + |\mathit{prefix}(s_n, s_1)| + |\mathit{overlap}(s_n, s_1)|$$

Ähnlichkeit mit Minimum TSP Tour

$$1 \to 2 \to \ldots \to n \to 1$$

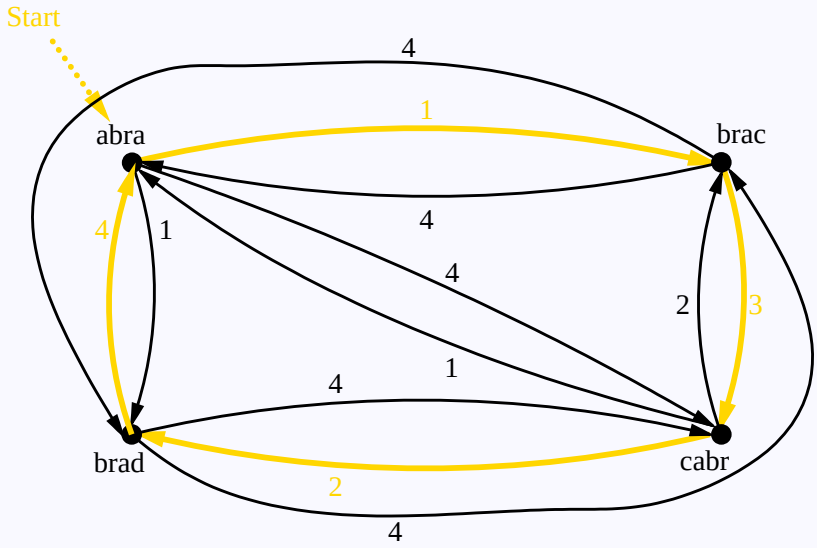
auf Präfixgraphen von S. Denn

$$\text{TSP} = |\mathit{prefix}(s_1, s_2)| + \ldots + |\mathit{prefix}(s_n, s_1)|$$

Also ist $\text{TSP} \leq \text{OPT}$ und damit untere Schranke

Beispiel des Präfixgraphen für S={abra,brac,cabr,brad}

Beispiel des Präfixgraphen für S={abra,brac,cabr,brad}



Start bei abra $\Rightarrow$ Entsprechend der Kanten des Kreises die Wörter maximal überlappend aneinanderhängen $\Rightarrow$ Kürzester Superstring abracabrad

- Problem hier: TSP ist auch NP-Vollständig

- Problem hier: TSP ist auch NP-Vollständig
- Ausweichen auf ein TSP-verwandtes Problem: das Finden einer *Kreisüberdeckung* (engl. *cycle cover*) in Graphen G.

- Problem hier: TSP ist auch NP-Vollständig
- Ausweichen auf ein TSP-verwandtes Problem: das Finden einer *Kreisüberdeckung* (engl. *cycle cover*) in Graphen G.
- Kreisüberdeckung ist eine Menge C aus disjunkten Kreisen, die zusammen alle Knoten von G überdecken

- Problem hier: TSP ist auch NP-Vollständig
- Ausweichen auf ein TSP-verwandtes Problem: das Finden einer *Kreisüberdeckung* (engl. *cycle cover*) in Graphen G.
- Kreisüberdeckung ist eine Menge C aus disjunkten Kreisen, die zusammen alle Knoten von G überdecken
- Für ein $c \in C$ seien $\mathit{weight}(c)$ Kosten des Kreises c, d.h. die Summe der Gewichte aller Kanten aus c

- Problem hier: TSP ist auch NP-Vollständig
- Ausweichen auf ein TSP-verwandtes Problem: das Finden einer *Kreisüberdeckung* (engl. *cycle cover*) in Graphen G.
- Kreisüberdeckung ist eine Menge C aus disjunkten Kreisen, die zusammen alle Knoten von G überdecken
- Für ein $c \in C$ seien $\mathit{weight}(c)$ Kosten des Kreises c, d.h. die Summe der Gewichte aller Kanten aus c
- $\mathit{weight}(C)$ ist Summe der Kosten aller Kreise in C

- Problem hier: TSP ist auch NP-Vollständig
- Ausweichen auf ein TSP-verwandtes Problem: das Finden einer *Kreisüberdeckung* (engl. *cycle cover*) in Graphen G.
- Kreisüberdeckung ist eine Menge C aus disjunkten Kreisen, die zusammen alle Knoten von G überdecken
- Für ein $c \in C$ seien $\mathit{weight}(c)$ Kosten des Kreises c, d.h. die Summe der Gewichte aller Kanten aus c
- $\mathit{weight}(C)$ ist Summe der Kosten aller Kreise in C
- Minimale Kreisüberdeckung bezeichnet mit CC ist Kreisüberdeckung mit $\mathit{weight}(C)$ minimal

- Problem hier: TSP ist auch NP-Vollständig
- Ausweichen auf ein TSP-verwandtes Problem: das Finden einer *Kreisüberdeckung* (engl. *cycle cover*) in Graphen G.
- Kreisüberdeckung ist eine Menge C aus disjunkten Kreisen, die zusammen alle Knoten von G überdecken
- Für ein $c \in C$ seien $\mathit{weight}(c)$ Kosten des Kreises c, d.h. die Summe der Gewichte aller Kanten aus c
- $\mathit{weight}(C)$ ist Summe der Kosten aller Kreise in C
- Minimale Kreisüberdeckung bezeichnet mit CC ist Kreisüberdeckung mit $\mathit{weight}(C)$ minimal

- Eine Minimale TSP Tour $1 \rightarrow 2 \rightarrow \ldots \rightarrow$n$\rightarrow 1$ ist Kreisüberdeckung

- Eine Minimale TSP Tour $1 \rightarrow 2 \rightarrow \ldots \rightarrow$n$\rightarrow 1$ ist Kreisüberdeckung
- Dann können Kosten einer minimalen Kreisüberdeckung nicht höher sein

- Eine Minimale TSP Tour $1 \rightarrow 2 \rightarrow \ldots \rightarrow$n$\rightarrow 1$ ist Kreisüberdeckung
- Dann können Kosten einer minimalen Kreisüberdeckung nicht höher sein
- Also: CC $\leq$ TSP

- Eine Minimale TSP Tour $1 \rightarrow 2 \rightarrow \ldots \rightarrow$n$\rightarrow 1$ ist Kreisüberdeckung
- Dann können Kosten einer minimalen Kreisüberdeckung nicht höher sein
- Also: $\mathrm{CC} \leq \mathrm{TSP} \leq \mathrm{OPT}$

- Eine Minimale TSP Tour $1 \rightarrow 2 \rightarrow \ldots \rightarrow n \rightarrow 1$ ist Kreisüberdeckung
- Dann können Kosten einer minimalen Kreisüberdeckung nicht höher sein
- Also: $CC \leq TSP \leq OPT$
- CC in Präfixgraphen kann in Polynomialzeit gefunden werden

Eingabe: $G = (V, E)$ mit $V = \{v_1, \ldots, v_n\}$

Eingabe: $G = (V, E)$ mit $V = \{v_1, \ldots, v_n\}$

1. Konstruiere Graphen H mit Knotenmengen $U = \{u_1, \ldots, u_n\}$ und $W = \{w_1, \ldots, w_n\}$

Eingabe: $G = (V, E)$ mit $V = \{v_1, \ldots, v_n\}$

1. Konstruiere Graphen H mit Knotenmengen $U = \{u_1, \ldots, u_n\}$ und $W = \{w_1, \ldots, w_n\}$
2. Füge für $1 \leq i, j \leq n$ Kanten (u_i, w_j) mit Kosten $\mathit{weight}((v_i, v_j))$ hinzu

Eingabe: $G = (V, E)$ mit $V = \{v_1, \ldots, v_n\}$

1. Konstruiere Graphen H mit Knotenmengen $U = \{u_1, \ldots, u_n\}$ und $W = \{w_1, \ldots, w_n\}$
2. Füge für $1 \leq i, j \leq n$ Kanten (u_i, w_j) mit Kosten $\mathit{weight}((v_i, v_j))$ hinzu
3. Finde perfektes Matching M minimaler Kosten in H

Eingabe: $G = (V, E)$ mit $V = \{v_1, \ldots, v_n\}$

1. Konstruiere Graphen H mit Knotenmengen $U = \{u_1, \ldots, u_n\}$ und $W = \{w_1, \ldots, w_n\}$
2. Füge für $1 \leq i, j \leq n$ Kanten (u_i, w_j) mit Kosten $\mathit{weight}((v_i, v_j))$ hinzu
3. Finde perfektes Matching M minimaler Kosten in H
4. Ersetze alle Kanten $(u_i, w_j) \in M$ in M durch (v_i, v_j)

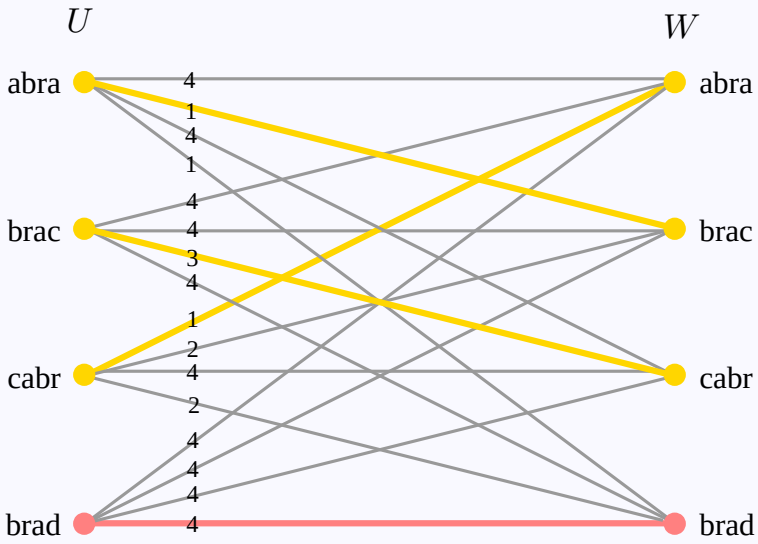
Eingabe: $G = (V, E)$ mit $V = \{v_1, \ldots, v_n\}$

1. Konstruiere Graphen H mit Knotenmengen $U = \{u_1, \ldots, u_n\}$ und $W = \{w_1, \ldots, w_n\}$
2. Füge für $1 \leq i, j \leq n$ Kanten (u_i, w_j) mit Kosten $\mathit{weight}((v_i, v_j))$ hinzu
3. Finde perfektes Matching M minimaler Kosten in H
4. Ersetze alle Kanten $(u_i, w_j) \in M$ in M durch (v_i, v_j)
5. Gib alle Kreise aus M zurück

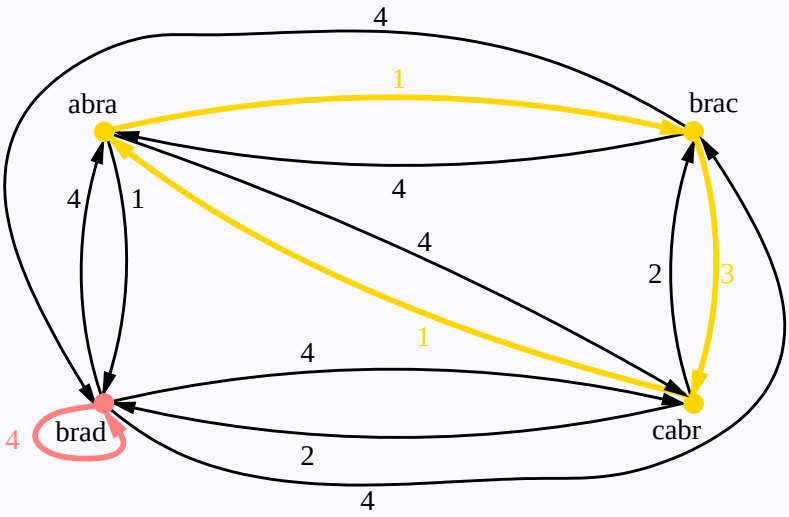
Eingabe: $G = (V, E)$ mit $V = \{v_1, \ldots, v_n\}$

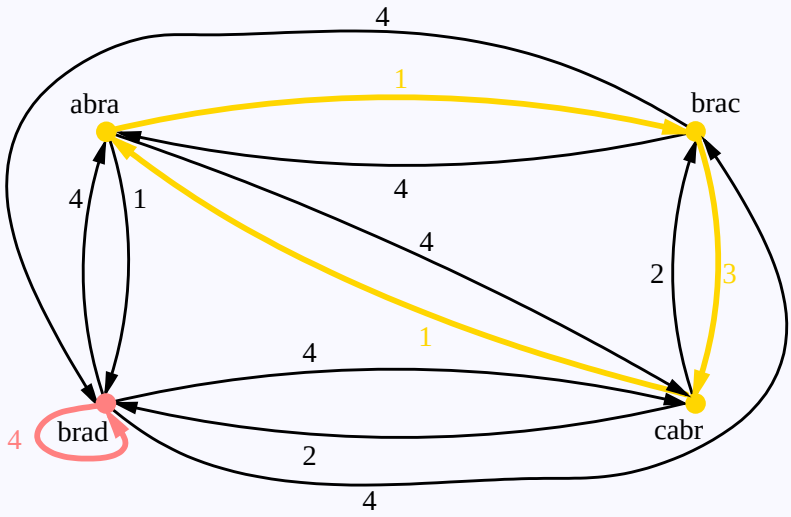
1. Konstruiere Graphen H mit Knotenmengen $U = \{u_1, \ldots, u_n\}$ und $W = \{w_1, \ldots, w_n\}$
2. Füge für $1 \leq i, j \leq n$ Kanten (u_i, w_j) mit Kosten $\mathit{weight}((v_i, v_j))$ hinzu
3. Finde perfektes Matching M minimaler Kosten in H
4. Ersetze alle Kanten $(u_i, w_j) \in M$ in M durch (v_i, v_j)
5. Gib alle Kreise aus M zurück

Da H bipartit ist, sind alle Kanten in M nach Schritt 3 tatsächlich von der Form (u_i, w_j).



Graph H für Beispiel. Ein perfektes Matching minimaler Kosten (=9) ist hervorgehoben


4
1
abra
brac
4
4
1
4
2
3
1
4
4
brad
cabr
2
4



Präfixgraph für Beispiel. CC ist hervorgehoben.

Sei $c = (i_1 \rightarrow \ldots \rightarrow i_l \rightarrow i_1)$ ein Kreis im Präfixgraphen

Sei $c = (i_1 \to \ldots \to i_l \to i_1)$ ein Kreis im Präfixgraphen

- $\alpha(c) = \mathit{prefix}(s_{i_1}, s_{i_2}) \circ \ldots \circ \mathit{prefix}(s_{i_{l-1}}, s_{i_l}) \circ \mathit{prefix}(s_{i_l}, s_{i_1})$

Sei $c = (i_1 \to \ldots \to i_l \to i_1)$ ein Kreis im Präfixgraphen

- $\alpha(c) = \mathit{prefix}(s_{i_1}, s_{i_2}) \circ \ldots \circ \mathit{prefix}(s_{i_{l-1}}, s_{i_l}) \circ \mathit{prefix}(s_{i_l}, s_{i_1})$
- $\sigma(c) = \alpha(c) \circ s_{i_1}$

Sei $c = (i_1 \to \ldots \to i_l \to i_1)$ ein Kreis im Präfixgraphen

- $\alpha(c) = \mathit{prefix}(s_{i_1}, s_{i_2}) \circ \ldots \circ \mathit{prefix}(s_{i_{l-1}}, s_{i_l}) \circ \mathit{prefix}(s_{i_l}, s_{i_1})$
- $\sigma(c) = \alpha(c) \circ s_{i_1}$

Es gilt:

- $|\alpha(c)| = \mathit{weight}(c)$, $|\sigma(c)| = \mathit{weight}(c) + |s_{i_1}|$

Sei $c = (i_1 \to \ldots \to i_l \to i_1)$ ein Kreis im Präfixgraphen

- $\alpha(c) = \mathit{prefix}(s_{i_1}, s_{i_2}) \circ \ldots \circ \mathit{prefix}(s_{i_{l-1}}, s_{i_l}) \circ \mathit{prefix}(s_{i_l}, s_{i_1})$
- $\sigma(c) = \alpha(c) \circ s_{i_1}$

Es gilt:

- $|\alpha(c)| = \mathit{weight}(c)$, $|\sigma(c)| = \mathit{weight}(c) + |s_{i_1}|$
- $\sigma(c)$ überdeckt Wörter $s_{i_1}, \ldots, s_{i_l}$

Sei $c = (i_1 \to \ldots \to i_l \to i_1)$ ein Kreis im Präfixgraphen

- $\alpha(c) = \mathit{prefix}(s_{i_1}, s_{i_2}) \circ \ldots \circ \mathit{prefix}(s_{i_{l-1}}, s_{i_l}) \circ \mathit{prefix}(s_{i_l}, s_{i_1})$
- $\sigma(c) = \alpha(c) \circ s_{i_1}$

Es gilt:

- $|\alpha(c)| = \mathit{weight}(c)$, $|\sigma(c)| = \mathit{weight}(c) + |s_{i_1}|$
- $\sigma(c)$ überdeckt Wörter $s_{i_1}, \ldots, s_{i_l}$
- $s_{i_1}, \ldots, s_{i_l}$ sind Teilwörter von $\alpha(c)^\infty$

Sei $c = (i_1 \to \ldots \to i_l \to i_1)$ ein Kreis im Präfixgraphen

- $\alpha(c) = \mathit{prefix}(s_{i_1}, s_{i_2}) \circ \ldots \circ \mathit{prefix}(s_{i_{l-1}}, s_{i_l}) \circ \mathit{prefix}(s_{i_l}, s_{i_1})$
- $\sigma(c) = \alpha(c) \circ s_{i_1}$

Es gilt:

- $|\alpha(c)| = \mathit{weight}(c)$, $|\sigma(c)| = \mathit{weight}(c) + |s_{i_1}|$
- $\sigma(c)$ überdeckt Wörter $s_{i_1}, \ldots, s_{i_l}$
- $s_{i_1}, \ldots, s_{i_l}$ sind Teilwörter von $\alpha(c)^\infty$
- s_{i_1} kann ein beliebiges Wort aus c sein, s_{i_1} ist *repräsentatives* Wort von c

Algorithmus 1.(Shortest Superstrings Faktor 4)

Algorithmus 1.(Shortest Superstrings Faktor 4)

- Erstelle Präfixgraphen $I(S)$.

Algorithmus 1.(Shortest Superstrings Faktor 4)

- Erstelle Präfixgraphen $I(S)$.
- Finde eine Kreisüberdeckung minimaler Kosten des Präfixgraphen, $C = \{c_1, \ldots, c_k\}$.

Algorithmus 1.(Shortest Superstrings Faktor 4)

- Erstelle Präfixgraphen $I(S)$.
- Finde eine Kreisüberdeckung minimaler Kosten des Präfixgraphen, $C = \{c_1, \ldots, c_k\}$.
- Gebe $\sigma(c_1) \circ \ldots \circ \sigma(c_k)$ zurück.

Algorithmus 1.(Shortest Superstrings Faktor 4)

- Erstelle Präfixgraphen $I(S)$.
- Finde eine Kreisüberdeckung minimaler Kosten des Präfixgraphen, $C = \{c_1, \ldots, c_k\}$.
- Gebe $\sigma(c_1) \circ \ldots \circ \sigma(c_k)$ zurück.

Beobachtung

Algorithmus 1.(Shortest Superstrings Faktor 4)

- Erstelle Präfixgraphen $I(S)$.
- Finde eine Kreisüberdeckung minimaler Kosten des Präfixgraphen, $C = \{c_1, \ldots, c_k\}$.
- Gebe $\sigma(c_1) \circ \ldots \circ \sigma(c_k)$ zurück.

Beobachtung

- Falls in jedem Kreis c_i ein repräsentatives Wort r_i mit $|r_i| \leq \mathit{weight}(c_i)$, dann Ausgabelänge $\leq 2 \cdot \mathrm{OPT}$

Algorithmus 1.(Shortest Superstrings Faktor 4)

- Erstelle Präfixgraphen $I(S)$.
- Finde eine Kreisüberdeckung minimaler Kosten des Präfixgraphen, $C = \{c_1, \ldots, c_k\}$.
- Gebe $\sigma(c_1) \circ \ldots \circ \sigma(c_k)$ zurück.

Beobachtung

- Falls in jedem Kreis c_i ein repräsentatives Wort r_i mit $|r_i| \leq \mathit{weight}(c_i)$, dann Ausgabelänge $\leq 2 \cdot \mathrm{OPT}$
- Ansonsten sind Wörter in c_i *periodisch*, da Teilwörter von $\alpha(c_i)^\infty$

Lemma 2. *Falls jedes Wort in $S' \subseteq S$ ein Teilwort von t^∞ ist, so existiert ein Kreis im Präfixgraphen $I(S)$, der alle Knoten, die den Wörten aus S' entsprechen, überdeckt. Die Kosten des Kreises sind höchstens $|t|$.*

Lemma 2. *Falls jedes Wort in $S' \subseteq S$ ein Teilwort von t^∞ ist, so existiert ein Kreis im Präfixgraphen $I(S)$, der alle Knoten, die den Wörten aus S' entsprechen, überdeckt. Die Kosten des Kreises sind höchstens $|t|$.*

Beweis.

Lemma 2. *Falls jedes Wort in $S' \subseteq S$ ein Teilwort von t^∞ ist, so existiert ein Kreis im Präfixgraphen $I(S)$, der alle Knoten, die den Wörten aus S' entsprechen, überdeckt. Die Kosten des Kreises sind höchstens $|t|$.*

Beweis.

- Betrachte $\forall s' \in S'$ Startpositionen des ersten Vorkommens in t^∞.

Lemma 2. *Falls jedes Wort in $S' \subseteq S$ ein Teilwort von t^∞ ist, so existiert ein Kreis im Präfixgraphen $I(S)$, der alle Knoten, die den Wörten aus S' entsprechen, überdeckt. Die Kosten des Kreises sind höchstens $|t|$.*

Beweis.

- Betrachte $\forall s' \in S'$ Startpositionen des ersten Vorkommens in t^∞.
- Positionen sind alle verschieden und liegen in ersten Kopie von t.

Lemma 2. *Falls jedes Wort in $S' \subseteq S$ ein Teilwort von t^∞ ist, so existiert ein Kreis im Präfixgraphen $I(S)$, der alle Knoten, die den Wörten aus S' entsprechen, überdeckt. Die Kosten des Kreises sind höchstens $|t|$.*

Beweis.

- Betrachte $\forall s' \in S'$ Startpositionen des ersten Vorkommens in t^∞.
- Positionen sind alle verschieden und liegen in ersten Kopie von t.
- Betrachte Kreis im Präfixgraphen, der in Folge des Vorkommens der Wörter s' in t, durch entspechenden Knoten geht.

Lemma 2. *Falls jedes Wort in $S' \subseteq S$ ein Teilwort von t^∞ ist, so existiert ein Kreis im Präfixgraphen $I(S)$, der alle Knoten, die den Wörten aus S' entsprechen, überdeckt. Die Kosten des Kreises sind höchstens $|t|$.*

Beweis.

- Betrachte $\forall s' \in S'$ Startpositionen des ersten Vorkommens in t^∞.
- Positionen sind alle verschieden und liegen in ersten Kopie von t.
- Betrachte Kreis im Präfixgraphen, der in Folge des Vorkommens der Wörter s' in t, durch entspechenden Knoten geht.
- Klar, Kosten des Kreises sind nicht höher als $|t|$.

□

Lemma 3. *Sind c und c' zwei Kreise aus C und sind r und r' zwei repräsentative Wörter dieser Kreise. Es gilt:*

$$|overlap(r, r')| < weight(c) + weight(c')$$

Lemma 3. *Sind c und c' zwei Kreise aus C und sind r und r' zwei repräsentative Wörter dieser Kreise. Es gilt:*

$$|overlap(r, r')| < weight(c) + weight(c')$$

Beweis.

Lemma 3. *Sind c und c' zwei Kreise aus C und sind r und r' zwei repräsentative Wörter dieser Kreise. Es gilt:*

$$|overlap(r, r')| < weight(c) + weight(c')$$

Beweis.

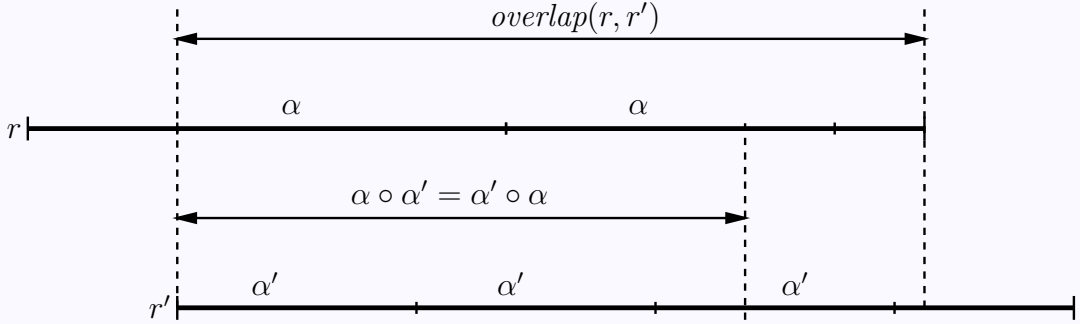
- Annahme, es gelte $|overlap(r, r')| \geq weight(c) + weight(c')$.

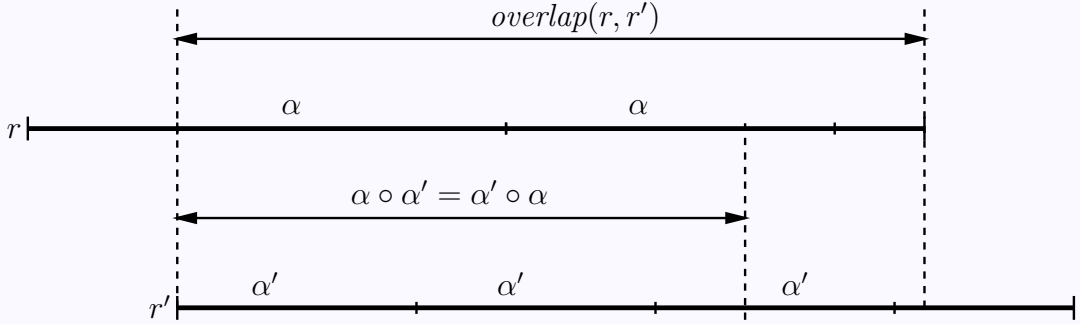
Lemma 3. *Sind c und c' zwei Kreise aus C und sind r und r' zwei repräsentative Wörter dieser Kreise. Es gilt:*

$$|overlap(r, r')| < weight(c) + weight(c')$$

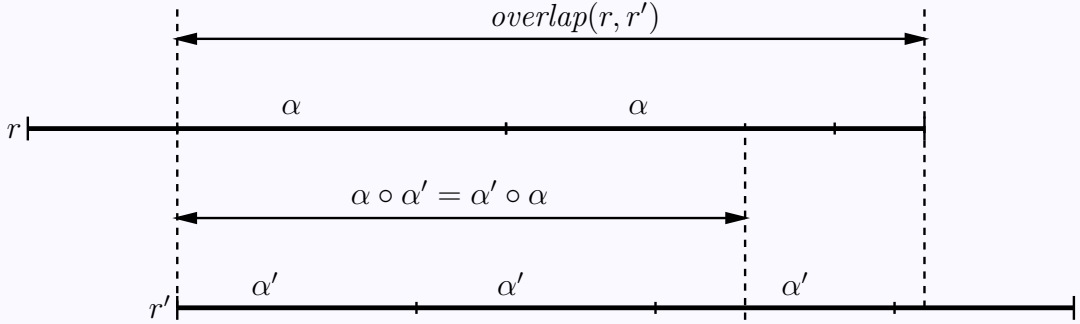
Beweis.

- Annahme, es gelte $|overlap(r, r')| \geq weight(c) + weight(c')$.
- Sind α und α' Präfixe von $overlap(r, r')$ der Länge $weight(c)$ bzw. $weight(c')$.

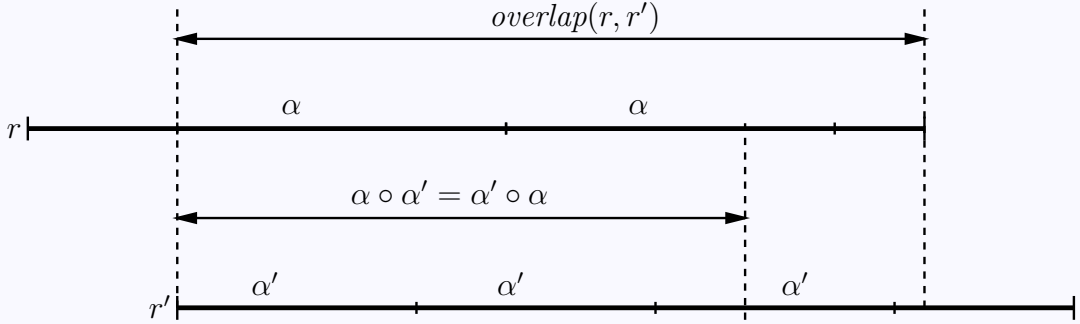

$overlap(r, r')$
r
α
α
$\alpha \circ \alpha' = \alpha' \circ \alpha$
r'
α'
α'
α'



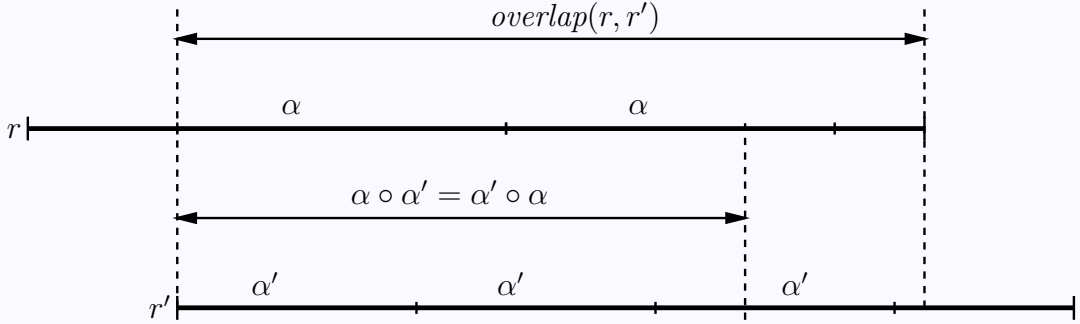
- $overlap(r, r')$ ist Präfix von α^∞ und Präfix von $(\alpha')^\infty$



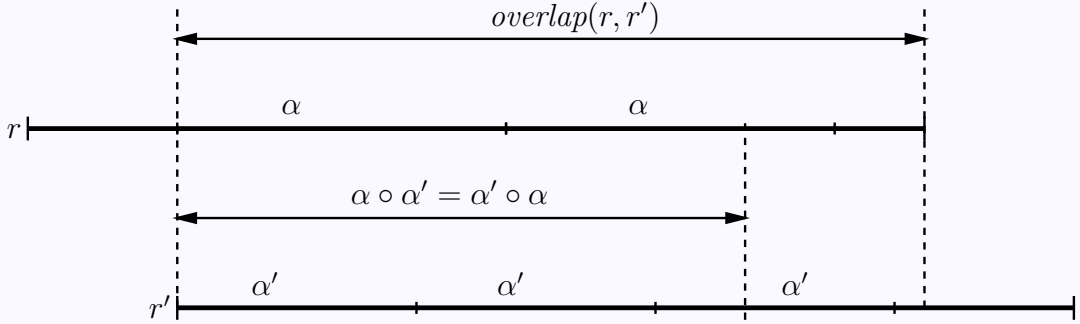
- $overlap(r, r')$ ist Präfix von α^∞ und Präfix von $(\alpha')^\infty$
- α ist Präfix von $(\alpha')^\infty$ und α' Präfix von α^∞



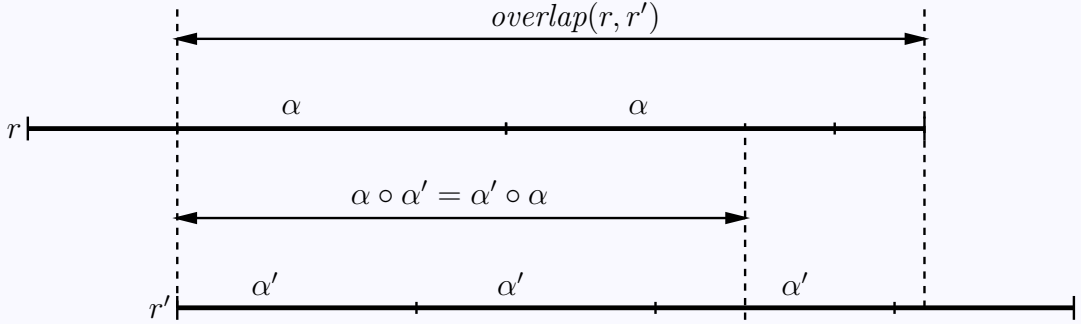
- $overlap(r, r')$ ist Präfix von α^∞ und Präfix von $(\alpha')^\infty$
- α ist Präfix von $(\alpha')^\infty$ und α' Präfix von α^∞
- Es folgt $\alpha \circ \alpha' = \alpha' \circ \ \alpha$



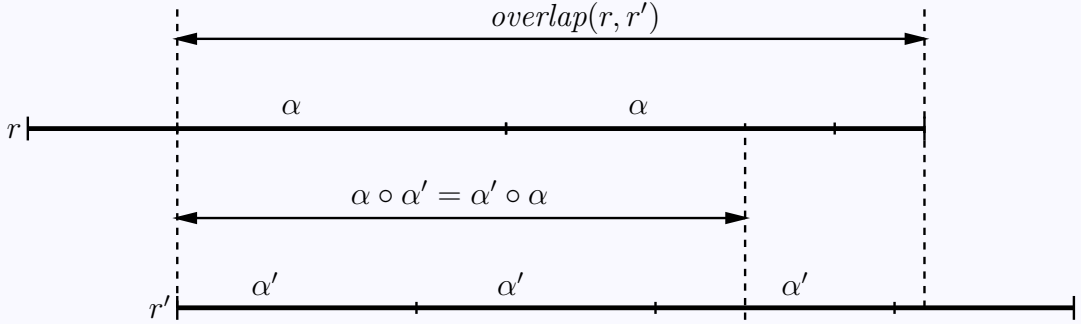
- $overlap(r, r')$ ist Präfix von α^∞ und Präfix von $(\alpha')^\infty$
- α ist Präfix von $(\alpha')^\infty$ und α' Präfix von α^∞
- Es folgt $\alpha \circ \alpha' = \alpha' \circ \ \alpha$und dann $\alpha^\infty = (\alpha')^\infty$



- $overlap(r, r')$ ist Präfix von α^∞ und Präfix von $(\alpha')^\infty$
- α ist Präfix von $(\alpha')^\infty$ und α' Präfix von α^∞
- Es folgt $\alpha \circ \alpha' = \alpha' \circ \alpha$und dann $\alpha^\infty = (\alpha')^\infty$
- Alle Wörter in c' Teilwörter von $(\alpha')^\infty = \alpha^\infty$.



- $overlap(r, r')$ ist Präfix von α^∞ und Präfix von $(\alpha')^\infty$
- α ist Präfix von $(\alpha')^\infty$ und α' Präfix von α^∞
- Es folgt $\alpha \circ \alpha' = \alpha' \circ \alpha$und dann $\alpha^\infty = (\alpha')^\infty$
- Alle Wörter in c' Teilwörter von $(\alpha')^\infty = \alpha^\infty$. Nach Lemma 2 existiert Kreis mit Kosten von höchstens $|\alpha| = weight(c)$, der alle Wörter in c und c' überdeckt.



- $overlap(r, r')$ ist Präfix von α^∞ und Präfix von $(\alpha')^\infty$
- α ist Präfix von $(\alpha')^\infty$ und α' Präfix von α^∞
- Es folgt $\alpha \circ \alpha' = \alpha' \circ \alpha$und dann $\alpha^\infty = (\alpha')^\infty$
- Alle Wörter in c' Teilwörter von $(\alpha')^\infty = \alpha^\infty$. Nach Lemma 2 existiert Kreis mit Kosten von höchstens $|\alpha| = weight(c)$, der alle Wörter in c und c' überdeckt. Widerspruch zu C ist Kreisüberdeckung minimaler Kosten.

Satz 4. *Algorithmus 1 erreicht Approximationfaktor von 4 für SSP.*

Satz 4. *Algorithmus 1 erreicht Approximationfaktor von 4 für SSP.*

Gliederung:

- **Einführung**
- **Faktor 4 Approximation**
- **Kompression**
- Faktor 3 Approximation
- Zusammenfassung

- Sei X Menge von Wörtern. $\|X\|$ ist Summe der Längen aller Wörter aus X.

- Sei X Menge von Wörtern. $\|X\|$ ist Summe der Längen aller Wörter aus X.
- *Kompression* eines Superstrings s für die Menge S ist $\|S\| - |s|$.

- Sei X Menge von Wörtern. $\|X\|$ ist Summe der Längen aller Wörter aus X.
- *Kompression* eines Superstrings s für die Menge S ist $\|S\| - |s|$.
- Offensichtlich maximale Kompression, wenn s shortest Superstring.

- Sei X Menge von Wörtern. $\|X\|$ ist Summe der Längen aller Wörter aus X.
- *Kompression* eines Superstrings s für die Menge S ist $\|S\| - |s|$.
- Offensichtlich maximale Kompression, wenn s shortest Superstring.
- Es gibt Algorithmen, die wenigstens halbe maximale Kompressionen erzielen.

- Sei X Menge von Wörtern. $\|X\|$ ist Summe der Längen aller Wörter aus X.
- *Kompression* eines Superstrings s für die Menge S ist $\|S\| - |s|$.
- Offensichtlich maximale Kompression, wenn s shortest Superstring.
- Es gibt Algorithmen, die wenigstens halbe maximale Kompressionen erzielen.

Definition (Überlappungsgraph)

- Sei X Menge von Wörtern. $\|X\|$ ist Summe der Längen aller Wörter aus X.
- *Kompression* eines Superstrings s für die Menge S ist $\|S\| - |s|$.
- Offensichtlich maximale Kompression, wenn s shortest Superstring.
- Es gibt Algorithmen, die wenigstens halbe maximale Kompressionen erzielen.

Definition (Überlappungsgraph)

- Gegeben Menge von Wörtern $X = \{x_1, \ldots, x_n\}$

- Sei X Menge von Wörtern. $\|X\|$ ist Summe der Längen aller Wörter aus X.
- *Kompression* eines Superstrings s für die Menge S ist $\|S\| - |s|$.
- Offensichtlich maximale Kompression, wenn s shortest Superstring.
- Es gibt Algorithmen, die wenigstens halbe maximale Kompressionen erzielen.

Definition (Überlappungsgraph)

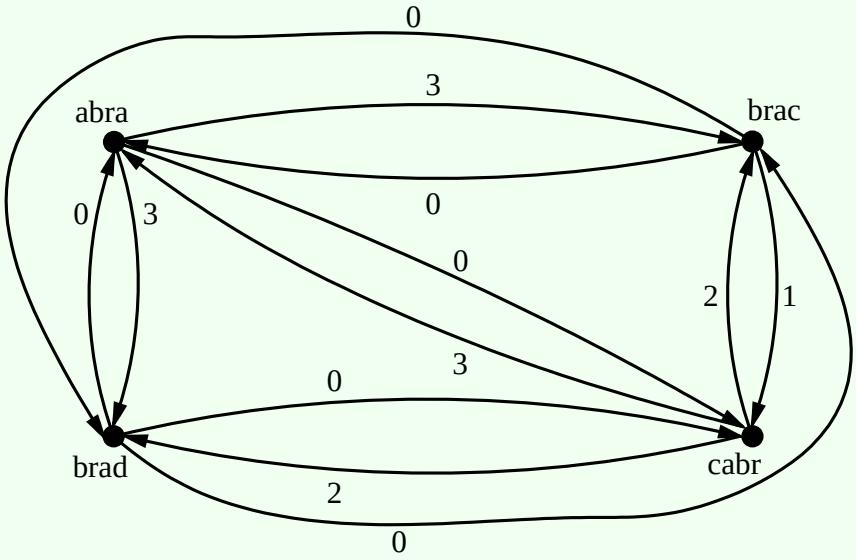
- Gegeben Menge von Wörtern $X = \{x_1, \ldots, x_n\}$
- *Überlappungs* $H(X)$ ist gerichter Graph mit n Knoten entsprechend x_i

- Sei X Menge von Wörtern. $\|X\|$ ist Summe der Längen aller Wörter aus X.
- *Kompression* eines Superstrings s für die Menge S ist $\|S\| - |s|$.
- Offensichtlich maximale Kompression, wenn s shortest Superstring.
- Es gibt Algorithmen, die wenigstens halbe maximale Kompressionen erzielen.

Definition (Überlappungsgraph)

- Gegeben Menge von Wörtern $X = \{x_1, \ldots, x_n\}$
- *Überlappungs* $H(X)$ ist gerichter Graph mit n Knoten entsprechend x_i
- Für alle $1 \leq i, j \leq n$ und $i \neq j$ hat Kante (v_i, v_j) Kosten von $|overlap(x_i, x_j)|$

Überlappungsgraph für Beispiel S ={abra,brac,cabr,brad}



- Annahme, dass $s_1, \ldots, s_n$ in dieser Anordung in s vorkommen

- Annahme, dass $s_1, \ldots, s_n$ in dieser Anordung in s vorkommen
- Maximale Kompression ist bestimmt durch

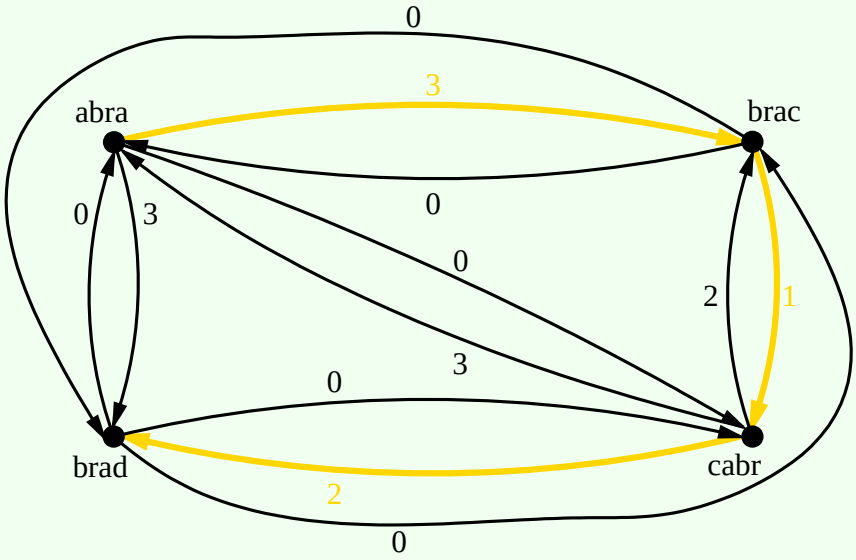
$$\sum_{i=1}^{n-1} |overlap(s_i, s_{i+1})|$$

- Annahme, dass $s_1, \ldots, s_n$ in dieser Anordung in s vorkommen
- Maximale Kompression ist bestimmt durch

$$\sum_{i=1}^{n-1} |overlap(s_i, s_{i+1})|$$

- Entspricht dem Maximum TSP Pfad auf Überlappungsgraph

Überlappungsgraph für S ={abra,brac,cabr,brad}. Maximum TSP Pfad ist hervorgeoben.



- Maximum TSP Pfad wird durch Maximum TSP Tour begrenzt

- Maximum TSP Pfad wird durch Maximum TSP Tour begrenzt
- Maximum TSP Tour durch Kreisüberdeckung maximaler Kosten (ähnlich bestimmbar wie oben)

- Maximum TSP Pfad wird durch Maximum TSP Tour begrenzt
- Maximum TSP Tour durch Kreisüberdeckung maximaler Kosten (ähnlich bestimmbar wie oben)

Algorithmus 2.(Kompressionsalgorithmus)

- Finde Kreisüberdeckung C maximaler Kosten in $H(S)$

- Maximum TSP Pfad wird durch Maximum TSP Tour begrenzt
- Maximum TSP Tour durch Kreisüberdeckung maximaler Kosten (ähnlich bestimmbar wie oben)

Algorithmus 2.(Kompressionsalgorithmus)

- Finde Kreisüberdeckung C maximaler Kosten in $H(S)$
- Entferne aus jedem $c \in C$ Kante minimaler Kosten (jedes c ist nun ein Pfad)

- Maximum TSP Pfad wird durch Maximum TSP Tour begrenzt
- Maximum TSP Tour durch Kreisüberdeckung maximaler Kosten (ähnlich bestimmbar wie oben)

Algorithmus 2.(Kompressionsalgorithmus)

- Finde Kreisüberdeckung C maximaler Kosten in $H(S)$
- Entferne aus jedem $c \in C$ Kante minimaler Kosten (jedes c ist nun ein Pfad)
- Überlappe Wörter $s_1, \ldots, s_n$ entsprechend Kanten der Pfade

- Maximum TSP Pfad wird durch Maximum TSP Tour begrenzt
- Maximum TSP Tour durch Kreisüberdeckung maximaler Kosten (ähnlich bestimmbar wie oben)

Algorithmus 2.(Kompressionsalgorithmus)

- Finde Kreisüberdeckung C maximaler Kosten in $H(S)$
- Entferne aus jedem $c \in C$ Kante minimaler Kosten (jedes c ist nun ein Pfad)
- Überlappe Wörter $s_1, \ldots, s_n$ entsprechend Kanten der Pfade
- Gebe resultierende Wörter aneinandergehängt zurück

- Algorithmus erzielt wenigstens halbe der maximalen Kompression

- Algorithmus erzielt wenigstens halbe der maximalen Kompression(approximiert unser ursprüngliches Problem aber nicht mit Faktor 2!)

- Algorithmus erzielt wenigstens halbe der maximalen Kompression(approximiert unser ursprüngliches Problem aber nicht mit Faktor 2!)
- Weiterer Algorithmus mit dieser Eigenschaft (wenigstens halbe Kompression) ist *Greedy Superstring Algorithmus*

- Algorithmus erzielt wenigstens halbe der maximalen Kompression(approximiert unser ursprüngliches Problem aber nicht mit Faktor 2!)
- Weiterer Algorithmus mit dieser Eigenschaft (wenigstens halbe Kompression) ist *Greedy Superstring Algorithmus*
- Aus S werden iterativ zwei Wörter zusammengefügt, die maximale Überlappung haben bis S genau ein Wort enthält.

Gliederung:

- Einführung
- Faktor 4 Approximation
- Kompression
- **Faktor 3 Approximation**
- Zusammenfassung

- In Algorithmus 1 werden $\sigma(c_i)$ einfach aneinander gehängt

- In Algorithmus 1 werden $\sigma(c_i)$ einfach aneinander gehängt
- Klar, jeder Superstring der $\sigma(c_i)$ ist auch Superstring aller Wörter aus S

- In Algorithmus 1 werden $\sigma(c_i)$ einfach aneinander gehängt
- Klar, jeder Superstring der $\sigma(c_i)$ ist auch Superstring aller Wörter aus S

Algorithmus 3.(Shorstest Superstring Faktor 3)

- In Algorithmus 1 werden $\sigma(c_i)$ einfach aneinander gehängt
- Klar, jeder Superstring der $\sigma(c_i)$ ist auch Superstring aller Wörter aus S

Algorithmus 3.(Shorstest Superstring Faktor 3)

- Finde Kreisüberdeckung minimaler Kosten in $I(S)$, $C = \{c_1, \ldots, c_k\}$.

- In Algorithmus 1 werden $\sigma(c_i)$ einfach aneinander gehängt
- Klar, jeder Superstring der $\sigma(c_i)$ ist auch Superstring aller Wörter aus S

Algorithmus 3.(Shorstest Superstring Faktor 3)

- Finde Kreisüberdeckung minimaler Kosten in $I(S)$, $C = \{c_1, \ldots, c_k\}$.
- Führe Kompressionsalgorithmus auf $\{\sigma(c_1), \ldots, \sigma(c_k)\}$ aus. Das Resultat sei τ.

- In Algorithmus 1 werden $\sigma(c_i)$ einfach aneinander gehängt
- Klar, jeder Superstring der $\sigma(c_i)$ ist auch Superstring aller Wörter aus S

Algorithmus 3.(Shorstest Superstring Faktor 3)

- Finde Kreisüberdeckung minimaler Kosten in $I(S)$, $C = \{c_1, \ldots, c_k\}$.
- Führe Kompressionsalgorithmus auf $\{\sigma(c_1), \ldots, \sigma(c_k)\}$ aus. Das Resultat sei τ.
- Gebe τ zurück

OPT_σ sei Länge des kürzesten überdeckenden Wortes von $S_\sigma = \{\sigma(c_1), \ldots, \sigma(c_k)\}$

OPT_σ sei Länge des kürzesten überdeckenden Wortes von $S_\sigma = \{\sigma(c_1), \ldots, \sigma(c_k)\}$

Lemma 5. $|\tau| \leq \mathrm{OPT}_\sigma + \mathit{weight}(C)$.

OPT_σ sei Länge des kürzesten überdeckenden Wortes von $S_\sigma = \{\sigma(c_1), \ldots, \sigma(c_k)\}$

Lemma 5. $|\tau| \leq \mathrm{OPT}_\sigma + \mathit{weight}(C)$.

Lemma 6. $\mathrm{OPT}_\sigma \leq \mathrm{OPT} + \mathit{weight}(C)$.

OPT_σ sei Länge des kürzesten überdeckenden Wortes von $S_\sigma = \{\sigma(c_1), \ldots, \sigma(c_k)\}$

Lemma 5. $|\tau| \leq \mathrm{OPT}_\sigma + \mathit{weight}(C)$.

Lemma 6. $\mathrm{OPT}_\sigma \leq \mathrm{OPT} + \mathit{weight}(C)$.

Aus beiden Lemmas folgt:

OPT_σ sei Länge des kürzesten überdeckenden Wortes von $S_\sigma = \{\sigma(c_1), \ldots, \sigma(c_k)\}$

Lemma 5. $|\tau| \leq \mathrm{OPT}_\sigma + \mathit{weight}(C)$.

Lemma 6. $\mathrm{OPT}_\sigma \leq \mathrm{OPT} + \mathit{weight}(C)$.

Aus beiden Lemmas folgt:

Satz 7. *Algorithmus 3 erzielt einen Approximationsfaktor von 3 für das Shortest Superstring Problem.*

Gliederung:

- Einführung
- Faktor 4 Approximation
- Kompression
- Faktor 3 Approximation
- Zusammenfassung

- SSP ist NP-Vollständig

- SSP ist NP-Vollständig
- Es existieren Algorithmen, die SSP mit kontanten Faktor approximieren

- SSP ist NP-Vollständig
- Es existieren Algorithmen, die SSP mit kontanten Faktor approximieren
- Extistieren Polynomialzeitalgorithmen die SSP beliebig gut approximieren könnnen?

- SSP ist NP-Vollständig
- Es existieren Algorithmen, die SSP mit kontanten Faktor approximieren
- Extistieren Polynomialzeitalgorithmen die SSP beliebig gut approximieren könnnen?
- Falls $P \neq NP$ gilt, dann nicht, da SSP auch MAX SNP-hart ist

- SSP ist NP-Vollständig
- Es existieren Algorithmen, die SSP mit kontanten Faktor approximieren
- Extistieren Polynomialzeitalgorithmen die SSP beliebig gut approximieren könnnen?
- Falls $P \neq NP$ gilt, dann nicht, da SSP auch MAX SNP-hart ist
- Jedoch existieren schon Algorithmen mit besserern Approximationsfaktoren als 3

- SSP ist NP-Vollständig
- Es existieren Algorithmen, die SSP mit kontanten Faktor approximieren
- Extistieren Polynomialzeitalgorithmen die SSP beliebig gut approximieren könnnen?
- Falls $P \neq NP$ gilt, dann nicht, da SSP auch MAX SNP-hart ist
- Jedoch existieren schon Algorithmen mit besserern Approximationsfaktoren als 3

Einige bisherige Verbesserungen zur Approximation kürzester Superstrings

Einige bisherige Verbesserungen zur Approximation kürzester Superstrings

Autoren	Jahr	Faktor
Blum, Jiang u.a.	1991	3
S. Teng, F. Yao	1993	$2\frac{8}{9}$
A. Czumaj u.a.	1994	$2\frac{5}{6}$
R. Kosaraju, J. Park, C. Stein	1994	$2\frac{50}{63}$
C. Armen, C. Stein	1995	$2\frac{3}{4}$
C. Armen, C. Stein	1996	$2\frac{2}{3}$
D. Breslauer, T. Jiang, Z. Jiang	1997	2,596
E. Z. Sweedyk	1999	$2\frac{1}{2}$

- Besondere Stellung hat der Greedy Superstring Algorithmus.

- Besondere Stellung hat der Greedy Superstring Algorithmus.
- Es wird vermutet, dass der Approximationsfaktor bei 2 liegt, da noch keine Eingabe gefunden wurde, bei der ein längerer Superstring erzeugt wurde.

- Besondere Stellung hat der Greedy Superstring Algorithmus.
- Es wird vermutet, dass der Approximationsfaktor bei 2 liegt, da noch keine Eingabe gefunden wurde, bei der ein längerer Superstring erzeugt wurde.
- Der Beweis fehlt allerdings (lediglich 4OPT konnte bisher formal bewiesen werden).

- Besondere Stellung hat der Greedy Superstring Algorithmus.
- Es wird vermutet, dass der Approximationsfaktor bei 2 liegt, da noch keine Eingabe gefunden wurde, bei der ein längerer Superstring erzeugt wurde.
- Der Beweis fehlt allerdings (lediglich 4OPT konnte bisher formal bewiesen werden).
- Approximationsfaktor für den Greedy Superstring Algorithmus ist *offenes Problem*.

Literatur:

- Blum, Jiang, Li, Tromp und Yannakakis. *Linear Approximation of Shortest Superstrings. 1994.*
- V. Vazirani. *Approximation Algorithms. 2001.*

Gliederung:

- Einführung
- Faktor 4 Approximation
- Kompression
- Faktor 3 Approximation
- Zusammenfassung