Unter der Annahme **P $\neq$ NP** gibt es für NP-schwere Probleme keine polynomiellen Algorithmen.

Unter der Annahme **P $\neq$ NP** gibt es für NP-schwere Probleme keine polynomiellen Algorithmen.

Unter der **Exponentialzeithypothese** gilt für viele NP-schwere Probleme, dass exponentielle Rechenzeit benötigt wird.

Unter der Annahme **P $\neq$ NP** gibt es für NP-schwere Probleme keine polynomiellen Algorithmen.

Unter der **Exponentialzeithypothese** gilt für viele NP-schwere Probleme, dass exponentielle Rechenzeit benötigt wird.

Einfacher Algorithmus durch **vollständige Aufzählung** aller Lösungen.

Unter der Annahme **P $\neq$ NP** gibt es für NP-schwere Probleme keine polynomiellen Algorithmen.

Unter der **Exponentialzeithypothese** gilt für viele NP-schwere Probleme, dass exponentielle Rechenzeit benötigt wird.

Einfacher Algorithmus durch **vollständige Aufzählung** aller Lösungen.

Frage: Gibt es noch effizientere Exponentialzeitalgorithmen?

Traveling Salesman Problem (TSP)

Eingabe: Menge $V = \{v_1, \ldots, v_n\}$ von Knoten

Distanzfunktion $d : V \times V \to \mathbb{R}_{\geq 0}$ mit:

$\forall u, v \in V$:

(1) $d(u, v) = d(v, u) \geq 0$

(2) $d(u, v) = 0 \Leftrightarrow u = v$

Lösungen: alle Permutationen $\pi : \{1, \ldots, n\} \to \{1, \ldots, n\}$

eine solche Permutation nennen wir auch Tour

Zielfunktion: minimiere $\sum_{i=1}^{n-1} d\left(v_{\pi(i)}, v_{\pi(i+1)}\right) + d\left(v_{\pi(n)}, v_{\pi(1)}\right)$

7.1 Held-Karp Algorithmus

Traveling Salesman Problem (TSP)

Eingabe: Menge $V = \{v_1, \ldots, v_n\}$ von Knoten

Distanzfunktion $d : V \times V \to \mathbb{R}_{\geq 0}$ mit:

$\forall u, v \in V$:

(1) $d(u, v) = d(v, u) \geq 0$

(2) $d(u, v) = 0 \Leftrightarrow u = v$

Lösungen: alle Permutationen $\pi : \{1, \ldots, n\} \to \{1, \ldots, n\}$

eine solche Permutation nennen wir auch Tour

Zielfunktion: minimiere $\sum_{i=1}^{n-1} d\big(v_{\pi(i)}, v_{\pi(i+1)}\big) + d\big(v_{\pi(n)}, v_{\pi(1)}\big)$

Anzahl Lösungen: $n!$ Touren

7.1 Held-Karp Algorithmus

Traveling Salesman Problem (TSP)

Eingabe: Menge $V = \{v_1, \dots, v_n\}$ von Knoten

Distanzfunktion $d : V \times V \to \mathbb{R}_{\geq 0}$ mit:

$\forall u, v \in V$:

(1) $d(u, v) = d(v, u) \geq 0$

(2) $d(u, v) = 0 \Leftrightarrow u = v$

Lösungen: alle Permutationen $\pi : \{1, \dots, n\} \to \{1, \dots, n\}$

eine solche Permutation nennen wir auch Tour

Zielfunktion: minimiere $\sum_{i=1}^{n-1} d\left(v_{\pi(i)}, v_{\pi(i+1)}\right) + d\left(v_{\pi(n)}, v_{\pi(1)}\right)$

Anzahl Lösungen: $n!$ Touren

Gibt es einen Algorithmus für TSP mit Laufzeit in $O(2^n \cdot p(n))$ für ein Polynom p?

7.1 Held-Karp Algorithmus

Seien Knoten $V = \{1, \ldots, n\}$ mit $n \geq 3$ und Distanzen $d_{ij} \in \mathbb{R}_{\geq 0}$ gegeben.

7.1 Held-Karp Algorithmus

Seien Knoten $V = \{1, \dots, n\}$ mit $n \geq 3$ und Distanzen $d_{ij} \in \mathbb{R}_{\geq 0}$ gegeben.

Definiere **$D_{V,j}$** als die **Länge eines kürzesten Weges** von Knoten 1 nach Knoten j, der **jeden Knoten aus V** genau **einmal** enthält.

7.1 Held-Karp Algorithmus

Seien Knoten $V = \{1, \ldots, n\}$ mit $n \geq 3$ und Distanzen $d_{ij} \in \mathbb{R}_{\geq 0}$ gegeben.

Definiere **$D_{V,j}$** als die **Länge eines kürzesten Weges** von Knoten 1 nach Knoten j, der **jeden Knoten aus V** genau **einmal** enthält.

Länge der **optimalen Tour** in V ist

$$\min_{j \in V \setminus \{1\}} \left(D_{V,j} + d_{j1} \right)$$

7.1 Held-Karp Algorithmus

Seien Knoten $V = \{1, \dots, n\}$ mit $n \geq 3$ und Distanzen $d_{ij} \in \mathbb{R}_{\geq 0}$ gegeben.

Definiere **$\mathrm{D}_{\mathrm{V,j}}$** als die **Länge eines kürzesten Weges** von Knoten 1 nach Knoten j, der **jeden Knoten aus V** genau **einmal** enthält.

Länge der **optimalen Tour** in V ist

$$\min_{j \in V \setminus \{1\}} \left(D_{V,j} + d_{j1} \right)$$

Idee: Halte Knoten 1 als Startknoten fest und nehme an, dass Kante $(j, 1)$ in der optimalen Tour enthalten ist.

7.1 Held-Karp Algorithmus

Seien Knoten $V = \{1, \ldots, n\}$ mit $n \geq 3$ und Distanzen $d_{ij} \in \mathbb{R}_{\geq 0}$ gegeben.

Definiere **$D_{V,j}$** als die **Länge eines kürzesten Weges** von Knoten 1 nach Knoten j, der **jeden Knoten aus V** genau **einmal** enthält.

Länge der **optimalen Tour** in V ist

$$\min_{j \in V \setminus \{1\}} (D_{V,j} + d_{j1})$$

Idee: Halte Knoten 1 als Startknoten fest und nehme an, dass Kante $(j, 1)$ in der optimalen Tour enthalten ist. Dann formen die anderen Kanten der optimalen Tour einen solchen kürzesten Weg.

7.1 Held-Karp Algorithmus

Seien Knoten $V = \{1, \ldots, n\}$ mit $n \geq 3$ und Distanzen $d_{ij} \in \mathbb{R}_{\geq 0}$ gegeben.

Definiere **$D_{V,j}$** als die **Länge eines kürzesten Weges** von Knoten 1 nach Knoten j, der **jeden Knoten aus V** genau **einmal** enthält.

Länge der **optimalen Tour** in V ist

$$\min_{j \in V \setminus \{1\}} \left(D_{V,j} + d_{j1}\right)$$

Idee: Halte Knoten 1 als Startknoten fest und nehme an, dass Kante $(j, 1)$ in der optimalen Tour enthalten ist. Dann formen die anderen Kanten der optimalen Tour einen solchen kürzesten Weg.

Wie bestimmen wir $D_{V,j}$?

Definition 7.1

Sei $S \subseteq \{1, \ldots, n\}$ mit $1 \in S$. Für jedes $j \in S \setminus \{1\}$ sei $D_{S,j}$ die Länge eines kürzesten Weges von Knoten 1 zu Knoten j, der jeden Knoten aus S genau einmal besucht und keine Knoten aus $V \setminus S$ enthält.

7.1 Held-Karp Algorithmus

Definition 7.1

Sei $S \subseteq \{1, \ldots, n\}$ mit $1 \in S$. Für jedes $j \in S \setminus \{1\}$ sei $D_{S,j}$ die Länge eines kürzesten Weges von Knoten 1 zu Knoten j, der jeden Knoten aus S genau einmal besucht und keine Knoten aus $V \setminus S$ enthält.

Wir können die Werte $D_{S,j}$ rekursiv wie folgt bestimmen.

Lemma 7.2

Sei $S \subseteq \{1, \ldots, n\}$ mit $1 \in S$ und $|S| \geq 3$. Für $j \in S \setminus \{1\}$ gilt

$$D_{S,j} = \min_{k \in S \setminus \{1,j\}} (D_{S \setminus \{j\},k} + d_{kj}).$$

Lemma 7.2

Sei $S \subseteq \{1, \dots, n\}$ mit $1 \in S$ und $|S| \geq 3$. Für $j \in S \setminus \{1\}$ gilt

$$D_{S,j} = \min_{k \in S \setminus \{1,j\}} (D_{S \setminus \{j\},k} + d_{kj}).$$

Beweis:

7.1 Held-Karp Algorithmus

Lemma 7.2

Sei $S \subseteq \{1, \ldots, n\}$ mit $1 \in S$ und $|S| \geq 3$. Für $j \in S \setminus \{1\}$ gilt

$$D_{S,j} = \min_{k \in S \setminus \{1,j\}} (D_{S \setminus \{j\},k} + d_{kj}).$$

Beweis: Per Definition ist $D_{S,j}$ die Länge eines kürzesten 1-j-Weges, der jeden Knoten aus S genau einmal besucht.

Lemma 7.2

Sei $S \subseteq \{1, \ldots, n\}$ mit $1 \in S$ und $|S| \geq 3$. Für $j \in S \setminus \{1\}$ gilt

$$D_{S,j} = \min_{k \in S \setminus \{1,j\}} (D_{S \setminus \{j\},k} + d_{kj}).$$

Beweis: Per Definition ist $D_{S,j}$ die Länge eines kürzesten 1-j-Weges, der jeden Knoten aus S genau einmal besucht. Sei P ein solcher Weg.

Lemma 7.2

Sei $S \subseteq \{1, \ldots, n\}$ mit $1 \in S$ und $|S| \geq 3$. Für $j \in S \setminus \{1\}$ gilt

$$D_{S,j} = \min_{k \in S \setminus \{1,j\}} (D_{S \setminus \{j\},k} + d_{kj}).$$

Beweis: Per Definition ist $\mathbf{D_{S,j}}$ die Länge eines kürzesten 1-j-Weges, der jeden Knoten aus S genau einmal besucht. Sei **P** ein solcher Weg. Sei **P′ der Teilweg von *P*** von Knoten 1 zum **vorletzten Knoten k**.

Lemma 7.2

Sei $S \subseteq \{1, \ldots, n\}$ mit $1 \in S$ und $|S| \geq 3$. Für $j \in S \setminus \{1\}$ gilt

$$D_{S,j} = \min_{k \in S \setminus \{1,j\}} (D_{S \setminus \{j\},k} + d_{kj}).$$

Beweis: Per Definition ist **$D_{S,j}$** die Länge eines kürzesten 1-j-Weges, der jeden Knoten aus S genau einmal besucht. Sei **P** ein solcher Weg. Sei **P' der Teilweg von P** von Knoten 1 zum **vorletzten Knoten k**. Die **Länge von P'** ist **$D_{S \setminus \{j\},k}$**, da P' ein kürzester 1-k-Weg ist, der nur Knoten aus $S \setminus \{j\}$ enthält.

Lemma 7.2

Sei $S \subseteq \{1, \ldots, n\}$ mit $1 \in S$ und $|S| \geq 3$. Für $j \in S \setminus \{1\}$ gilt

$$D_{S,j} = \min_{k \in S \setminus \{1,j\}} (D_{S \setminus \{j\},k} + d_{kj}).$$

Beweis: Per Definition ist **$D_{S,j}$** die Länge eines kürzesten 1-j-Weges, der jeden Knoten aus S genau einmal besucht. Sei **P** ein solcher Weg. Sei **P′ der Teilweg von P** von Knoten 1 zum **vorletzten Knoten k**. Die **Länge von P′** ist **$D_{S \setminus \{j\},k}$**, da P' ein kürzester 1-k-Weg ist, der nur Knoten aus $S \setminus \{j\}$ enthält.

Umgekehrt kann ein Weg, der $D_{S \setminus \{j\},k}$ für ein $k \in S \setminus \{1,j\}$ realisiert, immer mit (k,j) erweitert werden zu einem 1-j-Weg, der jeden Knoten in S genau einmal besucht.

Lemma 7.2

Sei $S \subseteq \{1, \ldots, n\}$ mit $1 \in S$ und $|S| \geq 3$. Für $j \in S \setminus \{1\}$ gilt

$$D_{S,j} = \min_{k \in S \setminus \{1,j\}} (D_{S \setminus \{j\},k} + d_{kj}).$$

Beweis: Per Definition ist $\mathbf{D_{S,j}}$ die Länge eines kürzesten 1-j-Weges, der jeden Knoten aus S genau einmal besucht. Sei **P** ein solcher Weg. Sei **P′ der Teilweg von *P*** von Knoten 1 zum **vorletzten Knoten k**. Die **Länge von P′** ist $\mathbf{D_{S \setminus \{j\},k}}$, da P' ein kürzester 1-k-Weg ist, der nur Knoten aus $S \setminus \{j\}$ enthält.

Umgekehrt kann ein Weg, der $D_{S \setminus \{j\},k}$ für ein $k \in S \setminus \{1, j\}$ realisiert, immer mit (k, j) erweitert werden zu einem 1-j-Weg, der jeden Knoten in S genau einmal besucht. Die Länge des erweiterten Weges ist $D_{S \setminus \{j\},k} + d_{k,j}$. □

7.1 Held-Karp Algorithmus

Held-Karp-Algorithmus

1. **for** ($j = 2$; $j \leq n$; j++) $D_{\{1,j\},j} = d_{1j}$;
2. **for** ($s = 3$; $i \leq n$; s++)
3. **for each** ($S \subseteq V$ mit $|S| = s$ und $1 \in S$)
4. **for each** ($j \in S \setminus \{1\}$)
5. $D_{S,j} = \min_{k \in S \setminus \{1,j\}} (D_{S \setminus \{j\},k} + d_{kj})$;
6. **return** $\min_{j \in V \setminus \{1\}} (D_{V,j} + d_{j1})$;

7.1 Held-Karp Algorithmus

Theorem 7.3

Für eine Eingabe für das TSP mit n Knoten berechnet der Held-Karp-Algorithmus in Laufzeit $O(n^2 2^n)$ die Länge einer kürzesten Tour.

Beweis (Korrektheit):

Theorem 7.3

Für eine Eingabe für das TSP mit n Knoten berechnet der Held-Karp-Algorithmus in Laufzeit $O(n^2 2^n)$ die Länge einer kürzesten Tour.

Beweis (Korrektheit): Wir zeigen, dass $D_{S,j}$ in Zeile 5 korrekt berechnet wird,

Theorem 7.3

Für eine Eingabe für das TSP mit n Knoten berechnet der Held-Karp-Algorithmus in Laufzeit $O(n^2 2^n)$ die Länge einer kürzesten Tour.

Beweis (Korrektheit): Wir zeigen, dass $D_{S,j}$ in Zeile 5 korrekt berechnet wird, via Induktion über alle Mengen $S \subseteq V$ mit $1 \in S$ in aufsteigender Größe.

Theorem 7.3

Für eine Eingabe für das TSP mit n Knoten berechnet der Held-Karp-Algorithmus in Laufzeit $O(n^2 2^n)$ die Länge einer kürzesten Tour.

Beweis (Korrektheit): Wir zeigen, dass $D_{S,j}$ in Zeile 5 korrekt berechnet wird, via Induktion über alle Mengen $S \subseteq V$ mit $1 \in S$ in aufsteigender Größe.

Basisfall bilden die Mengen S mit $|S| = 2$. Es gilt $D_{S,j} = d_{1j}$ für $S = \{1, j\}$, da nur eine Kante möglich ist.

Theorem 7.3

Für eine Eingabe für das TSP mit n Knoten berechnet der Held-Karp-Algorithmus in Laufzeit $O(n^2 2^n)$ die Länge einer kürzesten Tour.

Beweis (Korrektheit): Wir zeigen, dass $D_{S,j}$ in Zeile 5 korrekt berechnet wird, via Induktion über alle Mengen $S \subseteq V$ mit $1 \in S$ in aufsteigender Größe.

Basisfall bilden die Mengen S mit $|S| = 2$. Es gilt $D_{S,j} = d_{1j}$ für $S = \{1, j\}$, da nur eine Kante möglich ist. Korrektheit des Induktionsschrittes folgt aus Lemma 7.2 und $|S \setminus \{j\}| = |S| - 1$.

7.1 Held-Karp Algorithmus

Theorem 7.3

Für eine Eingabe für das TSP mit n Knoten berechnet der Held-Karp-Algorithmus in Laufzeit $O(n^2 2^n)$ die Länge einer kürzesten Tour.

Beweis (Korrektheit): Wir zeigen, dass $D_{S,j}$ in Zeile 5 korrekt berechnet wird, via Induktion über alle Mengen $S \subseteq V$ mit $1 \in S$ in aufsteigender Größe.

Basisfall bilden die Mengen S mit $|S| = 2$. Es gilt $D_{S,j} = d_{1j}$ für $S = \{1, j\}$, da nur eine Kante möglich ist. Korrektheit des Induktionsschrittes folgt aus Lemma 7.2 und $|S \setminus \{j\}| = |S| - 1$.

Am Ende des Algorithmus entspricht der Wert $D_{V,j}$ der Länge eines kürzesten Weges von Knoten 1 zu Knoten j, der jeden Knoten genau einmal enthält.

Theorem 7.3

Für eine Eingabe für das TSP mit n Knoten berechnet der Held-Karp-Algorithmus in Laufzeit $O(n^2 2^n)$ die Länge einer kürzesten Tour.

Beweis (Korrektheit): Wir zeigen, dass $D_{S,j}$ in Zeile 5 korrekt berechnet wird, via Induktion über alle Mengen $S \subseteq V$ mit $1 \in S$ in aufsteigender Größe.

Basisfall bilden die Mengen S mit $|S| = 2$. Es gilt $D_{S,j} = d_{1j}$ für $S = \{1, j\}$, da nur eine Kante möglich ist. Korrektheit des Induktionsschrittes folgt aus Lemma 7.2 und $|S \setminus \{j\}| = |S| - 1$.

Am Ende des Algorithmus entspricht der Wert $D_{V,j}$ der Länge eines kürzesten Weges von Knoten 1 zu Knoten j, der jeden Knoten genau einmal enthält.

Dieser Weg kann mit $(j, 1)$ zu einer Tour der Länge $D_{V,j} + d_{j1}$ erweitert werden. $\square$

Theorem 7.3

Für eine Eingabe für das TSP mit n Knoten berechnet der Held-Karp-Algorithmus in Laufzeit $O(n^2 2^n)$ die Länge einer kürzesten Tour.

Beweis (Laufzeit):

7.1 Held-Karp Algorithmus

Theorem 7.3

Für eine Eingabe für das TSP mit n Knoten berechnet der Held-Karp-Algorithmus in Laufzeit $O(n^2 2^n)$ die Länge einer kürzesten Tour.

Beweis (Laufzeit): Der Held-Karp Algorithmus ist ein dynamisches Programm.

Theorem 7.3

Für eine Eingabe für das TSP mit n Knoten berechnet der Held-Karp-Algorithmus in Laufzeit $O(n^2 2^n)$ die Länge einer kürzesten Tour.

Beweis (Laufzeit): Der Held-Karp Algorithmus ist ein dynamisches Programm.

Die Anzahl der Teilprobleme ist in $O(n2^n)$, da es 2^n Teilmengen von S und $n-1$ Wahlmöglichkeiten für j gibt.

Theorem 7.3

Für eine Eingabe für das TSP mit n Knoten berechnet der Held-Karp-Algorithmus in Laufzeit $O(n^2 2^n)$ die Länge einer kürzesten Tour.

Beweis (Laufzeit): Der Held-Karp Algorithmus ist ein dynamisches Programm.

Die Anzahl der Teilprobleme ist in $O(n2^n)$, da es 2^n Teilmengen von S und $n - 1$ Wahlmöglichkeiten für j gibt.

Jedes Teilproblem, das kein Basisproblem ist, wird durch die Bildung eines Minimums über weniger als n Terme in Zeile 5 gelöst.

Theorem 7.3

Für eine Eingabe für das TSP mit n Knoten berechnet der Held-Karp-Algorithmus in Laufzeit $O(n^2 2^n)$ die Länge einer kürzesten Tour.

Beweis (Laufzeit): Der Held-Karp Algorithmus ist ein dynamisches Programm.

Die Anzahl der Teilprobleme ist in $O(n2^n)$, da es 2^n Teilmengen von S und $n - 1$ Wahlmöglichkeiten für j gibt.

Jedes Teilproblem, das kein Basisproblem ist, wird durch die Bildung eines Minimums über weniger als n Terme in Zeile 5 gelöst.

Damit ergibt sich eine Gesamtlaufzeit von $O(n^2 2^n)$. $\square$

Held-Karp Algorithmus

1. **for** ($j = 2$; $j \leq n$; j++) $D_{\{1,j\},j} = d_{1j}$;
2. **for** ($s = 3$; $i \leq n$; s++)
3. **for each** ($S \subseteq V$ mit $|S| = s$ und $1 \in S$)
4. **for each** ($j \in S \setminus \{1\}$)
5. $D_{S,j} = \min_{k \in S \setminus \{1,j\}} \left(D_{S \setminus \{j\},k} + d_{kj} \right)$;
6. **return** $\min_{j \in V \setminus \{1\}} \left(D_{V,j} + d_{j1} \right)$;

Der Algorithmus kann leicht so abgeändert werden, dass er auch eine optimale Tour berechnet.

7.1 Held-Karp Algorithmus

Held-Karp Algorithmus

1. **for** ($j = 2; j \leq n; j{+}{+}$) $D_{\{1,j\},j} = d_{1j}$;
2. **for** ($s = 3; i \leq n; s{+}{+}$)
3.   **for each** ($S \subseteq V$ mit $|S| = s$ und $1 \in S$)
4.   **for each** ($j \in S \setminus \{1\}$)
5. $D_{S,j} = \min_{k \in S \setminus \{1,j\}} \left(D_{S \setminus \{j\},k} + d_{kj} \right)$;
6. **return** $\min_{j \in V \setminus \{1\}} \left(D_{V,j} + d_{j1} \right)$;

Der Algorithmus kann leicht so abgeändert werden, dass er auch eine optimale Tour berechnet. Speichere dazu in den Zeilen 5 und 6 jeweils noch zusätzlich werden, für welches k bzw. j das Minimum angenommen wird.

7.1 Held-Karp Algorithmus

Held-Karp Algorithmus

1. **for** $(j = 2; j \leq n; j{+}{+})\ D_{\{1,j\},j} = d_{1j};$
2. **for** $(s = 3; i \leq n; s{+}{+})$
3.   **for each** $(S \subseteq V$ mit $|S| = s$ und $1 \in S)$
4.   **for each** $(j \in S \setminus \{1\})$
5. $D_{S,j} = \min_{k \in S \setminus \{1,j\}} (D_{S \setminus \{j\},k} + d_{kj});$
6. **return** $\min_{j \in V \setminus \{1\}} (D_{V,j} + d_{j1});$

Der Algorithmus kann leicht so abgeändert werden, dass er auch eine optimale Tour berechnet. Speichere dazu in den Zeilen 5 und 6 jeweils noch zusätzlich werden, für welches k bzw. j das Minimum angenommen wird. Mit dieser Information kann eine optimale Tour rekonstruiert werden.

Wir betrachten ein weiteres NP-vollständiges Problem.

Vertex-Cover-Problem (VC)

Eingabe: ungerichteter Graph $G = (V, E)$, Zahl $k \in \mathbb{N}$

Frage: Gibt es $V' \subseteq V$ mit $|V'| \leq k$, sodass jede Kante aus E zu mindestens einem Knoten aus V' inzident ist?

Wir betrachten ein weiteres NP-vollständiges Problem.

Vertex-Cover-Problem (VC)

Eingabe: ungerichteter Graph $G = (V, E)$, Zahl $k \in \mathbb{N}$

Frage: Gibt es $V' \subseteq V$ mit $|V'| \leq k$, sodass jede Kante aus E zu mindestens einem Knoten aus V' inzident ist?

Anzahl Lösungskandidaten: $\binom{n}{k} \approx n^k$

Wir betrachten ein weiteres NP-vollständiges Problem.

Vertex-Cover-Problem (VC)

Eingabe: ungerichteter Graph $G = (V, E)$, Zahl $k \in \mathbb{N}$

Frage: Gibt es $V' \subseteq V$ mit $|V'| \leq k$, sodass jede Kante aus E zu mindestens einem Knoten aus V' inzident ist?

Anzahl Lösungskandidaten: $\binom{n}{k} \approx n^k$

Gibt es einen Algorithmus für VC mit Laufzeit $O(2^k \cdot p(|V| + |E|))$ für ein Polynom p?

Sei $\mathbf{e} = (\mathbf{x}, \mathbf{y})$ beliebige Kante in G. Jedes Vertex-Cover V' von G muss entweder x oder y (oder beide) enthalten.

7.2 Parametrisierte Algorithmen

Sei $\mathbf{e} = (\mathbf{x}, \mathbf{y})$ beliebige Kante in G. Jedes Vertex-Cover V' von G muss entweder x oder y (oder beide) enthalten.

Sei $\mathbf{G_x}$ der durch die Knotenmenge $V \setminus \{x\}$ induzierten Teilgraph von G.

Sei $\mathbf{e = (x, y)}$ beliebige Kante in G. Jedes Vertex-Cover V' von G muss entweder x oder y (oder beide) enthalten.

Sei $\mathbf{G_x}$ der durch die Knotenmenge $V \setminus \{x\}$ induzierten Teilgraph von G.

Falls $\mathbf{x \in V'}$, dann ist $V' \setminus \{x\}$ ein Vertex-Cover für G_x.

Sei $\mathbf{e = (x, y)}$ beliebige Kante in G. Jedes Vertex-Cover V' von G muss entweder x oder y (oder beide) enthalten.

Sei $\mathbf{G_x}$ der durch die Knotenmenge $V \setminus \{x\}$ induzierten Teilgraph von G.

Falls $\mathbf{x \in V'}$, dann ist $V' \setminus \{x\}$ ein Vertex-Cover für G_x.

Falls $\mathbf{y \in V'}$, dann gilt analog, dass $V' \setminus \{y\}$ ein Vertex-Cover für G_y ist.

Der folgende Algorithmus VC-BACKTRACKING nutzt diese Fallunterscheidung um die Menge der Lösungen effizient zu durchlaufen.

VC-BACKTRACKING$(G = (V, E), k)$

1. **if** $(|E| = 0)$ **return** true;
2. **if** $(k = 0)$ **return** false;
3. Sei $e = (x, y) \in E$ beliebig.
4. Sei G_x der von $V \setminus \{x\}$ induzierte Teilgraph.
5. Sei G_y der von $V \setminus \{y\}$ induzierte Teilgraph.
6. **if** (VC-BACKTRACKING$(G_x, k - 1)$) **return** true;
7. **if** (VC-BACKTRACKING$(G_y, k - 1)$) **return** true;
8. **return** false;

Theorem 7.4

Der Algorithmus VC-Backtracking entscheidet in Zeit $O(2^k(n+m))$, ob der Graph G ein Vertex Cover der Größe k besitzt.

Beweis (Korrektheit):

Theorem 7.4

Der Algorithmus VC-Backtracking entscheidet in Zeit $O(2^k(n+m))$, ob der Graph G ein Vertex Cover der Größe k besitzt.

Beweis (Korrektheit): Der Graph G besitzt genau dann ein Vertex Cover der Größe k, wenn entweder G_x oder G_y ein Vertex Cover der Größe $k-1$ besitzt.

Theorem 7.4

Der Algorithmus VC-Backtracking entscheidet in Zeit $O(2^k(n+m))$, ob der Graph G ein Vertex Cover der Größe k besitzt.

Beweis (Korrektheit): Der Graph G besitzt genau dann ein Vertex Cover der Größe k, wenn entweder G_x oder G_y ein Vertex Cover der Größe $k-1$ besitzt.

Basisfälle der Rekursion:

Ist E leer, so besitzt der Graph G für jedes $k \geq 0$ ein Vertex Cover der Größe k (Zeile 1).

Theorem 7.4

Der Algorithmus VC-Backtracking entscheidet in Zeit $O(2^k(n+m))$, ob der Graph G ein Vertex Cover der Größe k besitzt.

Beweis (Korrektheit): Der Graph G besitzt genau dann ein Vertex Cover der Größe k, wenn entweder G_x oder G_y ein Vertex Cover der Größe $k-1$ besitzt.

Basisfälle der Rekursion:

Ist E leer, so besitzt der Graph G für jedes $k \geq 0$ ein Vertex Cover der Größe k (Zeile 1).

Ist die Menge E nichtleer, so besitzt der Graph kein Vertex Cover der Größe 0 (Zeile 2). $\square$

Theorem 7.4

Der Algorithmus VC-Backtracking entscheidet in Zeit $O(2^k(n+m))$, ob der Graph G ein Vertex Cover der Größe k besitzt.

Beweis (Laufzeit): Sei G ein Graph mit n Knoten und m Kanten.

Theorem 7.4

Der Algorithmus VC-Backtracking entscheidet in Zeit $O(2^k(n+m))$, ob der Graph G ein Vertex Cover der Größe k besitzt.

Beweis (Laufzeit): Sei G ein Graph mit n Knoten und m Kanten.

Rekursionsbaum hat Tiefe k und jeder Knoten hat 2 Kinder (rekursive Aufrufe).

Theorem 7.4

Der Algorithmus VC-Backtracking entscheidet in Zeit $O(2^k(n+m))$, ob der Graph G ein Vertex Cover der Größe k besitzt.

Beweis (Laufzeit): Sei G ein Graph mit n Knoten und m Kanten.

Rekursionsbaum hat Tiefe k und jeder Knoten hat 2 Kinder (rekursive Aufrufe).

Somit gibt es insgesamt höchstens $\sum_{i=0}^{k} 2^i = O(2^k)$ Aufrufe von VC-Backtracking.

Theorem 7.4

Der Algorithmus VC-Backtracking entscheidet in Zeit $O(2^k(n+m))$, ob der Graph G ein Vertex Cover der Größe k besitzt.

Beweis (Laufzeit): Sei G ein Graph mit n Knoten und m Kanten.

Rekursionsbaum hat Tiefe k und jeder Knoten hat 2 Kinder (rekursive Aufrufe).

Somit gibt es insgesamt höchstens $\sum_{i=0}^{k} 2^i = O(2^k)$ Aufrufe von VC-Backtracking.

Jeder dieser Aufrufe benötigt eine Laufzeit von $O(n+m)$: Ist der Graph G als Adjazenzliste gespeichert, dann können G_x und G_y in $O(n+m)$ Zeit erstellt werden. $\square$

Können wir die Laufzeit noch weiter verbessern?

Können wir die Laufzeit noch weiter verbessern?

Ja, falls der Graph **maximalen Knotengrad** 2 hat, dann lässt sich ein optimales Vertex-Cover in Zeit $O(|V| + |E|)$ finden:

Können wir die Laufzeit noch weiter verbessern?

Ja, falls der Graph **maximalen Knotengrad** 2 hat, dann lässt sich ein optimales Vertex-Cover in Zeit $O(|V| + |E|)$ finden:

- Ein solcher Graph besteht aus paarweise disjunkten Pfaden und Kreisen.
- Für jede Zusammenhangskomponente wähle einen beliebigen Knoten und dann jeden zweiten Knoten auf dem Kreis bzw. Pfad.

Können wir die Laufzeit noch weiter verbessern?

Ja, falls der Graph **maximalen Knotengrad** 2 hat, dann lässt sich ein optimales Vertex-Cover in Zeit $O(|V| + |E|)$ finden:

- Ein solcher Graph besteht aus paarweise disjunkten Pfaden und Kreisen.
- Für jede Zusammenhangskomponente wähle einen beliebigen Knoten und dann jeden zweiten Knoten auf dem Kreis bzw. Pfad.

Idee: Verändere **Struktur der Rekursion** um zu einem Basisfall zu kommen, in dem der Knotengrad maximal 2 ist.

Sei $\mathbf{x}$ ein Knoten in G und sei $\mathbf{N} = \{\mathbf{y} \in \mathbf{V} | (\mathbf{x}, \mathbf{y}) \in \mathbf{E}\}$ die Menge der Nachbarn von x.

Sei $\mathbf{x}$ ein Knoten in G und sei $\mathbf{N} = \{\mathbf{y} \in \mathbf{V} | (\mathbf{x}, \mathbf{y}) \in \mathbf{E}\}$ die Menge der Nachbarn von x. Ein Vertex-Cover V' für G enthält entweder x oder alle Knoten in N (oder beides gilt).

Sei $\mathbf{x}$ ein Knoten in G und sei $\mathbf{N} = \{\mathbf{y} \in \mathbf{V} | (\mathbf{x}, \mathbf{y}) \in \mathbf{E}\}$ die Menge der Nachbarn von x. Ein Vertex-Cover V' für G enthält entweder x oder alle Knoten in N (oder beides gilt).

Falls $\mathbf{x} \in \mathbf{V}'$, dann ist $V' \setminus \{x\}$ ein Vertex-Cover für G_x.

Sei $\mathbf{x}$ ein Knoten in G und sei $\mathbf{N} = \{\mathbf{y} \in \mathbf{V} | (\mathbf{x}, \mathbf{y}) \in \mathbf{E}\}$ die Menge der Nachbarn von x. Ein Vertex-Cover V' für G enthält entweder x oder alle Knoten in N (oder beides gilt).

Falls $\mathbf{x} \in \mathbf{V'}$, dann ist $V' \setminus \{x\}$ ein Vertex-Cover für G_x.

Falls $\mathbf{N} \subseteq \mathbf{V'}$, dann ist $V' \setminus N$ ein Vertex-Cover für G_N.

Sei $\mathbf{x}$ ein Knoten in G und sei $\mathbf{N} = \{\mathbf{y} \in \mathbf{V} | (\mathbf{x}, \mathbf{y}) \in \mathbf{E}\}$ die Menge der Nachbarn von x. Ein Vertex-Cover V' für G enthält entweder x oder alle Knoten in N (oder beides gilt).

Falls $\mathbf{x} \in \mathbf{V}'$, dann ist $V' \setminus \{x\}$ ein Vertex-Cover für G_x.

Falls $\mathbf{N} \subseteq \mathbf{V}'$, dann ist $V' \setminus N$ ein Vertex-Cover für G_N.

Der folgende Algorithmus VC-Backtracking-Knoten nutzt diese Fallunterscheidung um eine bessere Laufzeit zu erreichen.

VC-BACKTRACKING-KNOTEN$(G = (V, E), k)$

if $(k < 0)$ **return** false;
if $(|E| = 0)$ **return** true;
if (G besitzt nur Knoten mit Grad ≤ 2)
    Berechne die Größe opt eines minimalen Vertex Covers.
    if $(k \geq \mathrm{opt})$ **return** true; **else return** false;
Sei $x \in V$ ein beliebiger Knoten mit Grad ≥ 3.
Sei $N = \{y \in V \mid (x, y) \in E\}$ die Nachbarschaft von x.
Sei G_x der von $V \setminus \{x\}$ induzierte Teilgraph.
Sei G_N der von $V \setminus N$ induzierte Teilgraph.
if (VC-BACKTRACKING-KNOTEN$(G_x, k - 1)$) **return** true;
if (VC-BACKTRACKING-KNOTEN$(G_N, k - |N|)$) **return** true;
return false;

Theorem 7.5

Der Algorithmus VC-Backtracking-Knoten entscheidet in Zeit $O(1{,}5^k(n+m))$, ob der Graph G ein Vertex Cover der Größe k besitzt.

Beweis (Korrektheit):

Theorem 7.5

Der Algorithmus VC-Backtracking-Knoten entscheidet in Zeit $O(1{,}5^k(n+m))$, ob der Graph G ein Vertex Cover der Größe k besitzt.

Beweis (Korrektheit): Der Graph G besitzt genau dann ein Vertex Cover der Größe k, wenn entweder G_x ein Vertex Cover der Größe $k-1$ oder G_N ein Vertex Cover der Größe $k-|N|$ besitzt.

Theorem 7.5

Der Algorithmus VC-Backtracking-Knoten entscheidet in Zeit $O(1{,}5^k(n+m))$, ob der Graph G ein Vertex Cover der Größe k besitzt.

Beweis (Korrektheit): Der Graph G besitzt genau dann ein Vertex Cover der Größe k, wenn entweder G_x ein Vertex Cover der Größe $k-1$ oder G_N ein Vertex Cover der Größe $k-|N|$ besitzt.

Basisfälle der Rekursion:

Für $k<0$ existiert kein Vertex Cover (Zeile 1).

Ist $k \geq 0$ und E leer, so existiert ein Vertex Cover der Größe 0 (Zeile 2). $\square$

Beweis (Laufzeit):

Beweis (Laufzeit): Sei G ein Graph mit n Knoten und m Kanten.

Sei $N(k)$ die Anzahl an Knoten im Rekursionsbaum.

Beweis (Laufzeit): Sei G ein Graph mit n Knoten und m Kanten.

Sei $N(k)$ die Anzahl an Knoten im Rekursionsbaum.

Für $k \leq 2$ ist $N(k)$ durch eine geeignete Konstante c nach oben beschränkt.

Beweis (Laufzeit): Sei G ein Graph mit n Knoten und m Kanten.

Sei $N(k)$ die Anzahl an Knoten im Rekursionsbaum.

Für $k \leq 2$ ist $N(k)$ durch eine geeignete Konstante c nach oben beschränkt.

Für $k \geq 3$ gilt

$$N(k) \leq N(k-1) + N(k-3) + 1$$

denn ein Aufruf mit Parameter k generiert höchstens zwei rekursive Aufrufe mit den Parametern $k-1$ und $k - |N| \leq k-3$.

Beweis (Laufzeit): Sei G ein Graph mit n Knoten und m Kanten.

Sei $N(k)$ die Anzahl an Knoten im Rekursionsbaum.

Für $k \leq 2$ ist $N(k)$ durch eine geeignete Konstante c nach oben beschränkt.

Für $k \geq 3$ gilt

$$N(k) \leq N(k-1) + N(k-3) + 1$$

denn ein Aufruf mit Parameter k generiert höchstens zwei rekursive Aufrufe mit den Parametern $k - 1$ und $k - |N| \leq k - 3$.

Behauptung: Für eine geeignete Konstante a, die nur von c abhängt, gilt $N(k) \leq 1{,}5^{k+a} - 1$ für jedes $k \in \mathbb{N}_0$.

Behauptung: Für eine geeignete Konstante a, die nur von c abhängt, gilt $N(k) \leq 1{,}5^{k+a} - 1$ für jedes $k \in \mathbb{N}_0$.

Behauptung: Für eine geeignete Konstante a, die nur von c abhängt, gilt $N(k) \leq 1{,}5^{k+a} - 1$ für jedes $k \in \mathbb{N}_0$.

Dies können wir induktiv beweisen. Der Induktionsanfang für $k \leq 2$ folgt durch eine hinreichend große Wahl der Konstante a.

Behauptung: Für eine geeignete Konstante a, die nur von c abhängt, gilt $N(k) \leq 1{,}5^{k+a} - 1$ für jedes $k \in \mathbb{N}_0$.

Dies können wir induktiv beweisen. Der Induktionsanfang für $k \leq 2$ folgt durch eine hinreichend große Wahl der Konstante a.

Für $k \geq 3$ folgt mit der Induktionsannahme

$$\begin{aligned} N(k) &\leq N(k-1) + N(k-3) + 1 \\ &\leq (1{,}5^{k-1+a} - 1) + (1{,}5^{k-3+a} - 1) + 1 \\ &= 1{,}5^{k+a} \left(\frac{1}{1{,}5} + \frac{1}{1{,}5^3} \right) - 1 \\ &\leq 0{,}97 \cdot 1{,}5^{k+a} - 1 \\ &\leq 1{,}5^{k+a} - 1. \end{aligned}$$

Es gibt also insgesamt höchstens $O(1{,}5^k)$ rekursive Aufrufe von VC-Backtracking-Knoten.

Es gibt also insgesamt höchstens $O(1{,}5^k)$ rekursive Aufrufe von VC-Backtracking-Knoten.

Jeder dieser Aufrufe benötigt eine Laufzeit von $O(n + m)$:

Es gibt also insgesamt höchstens $O(1{,}5^k)$ rekursive Aufrufe von VC-Backtracking-Knoten.

Jeder dieser Aufrufe benötigt eine Laufzeit von $O(n + m)$:

Ist der Graph G als Adjazenzliste gespeichert, dann können G_x und G_N in $O(n + m)$ Zeit erstellt werden.

Es gibt also insgesamt höchstens $O(1{,}5^k)$ rekursive Aufrufe von VC-Backtracking-Knoten.

Jeder dieser Aufrufe benötigt eine Laufzeit von $O(n+m)$:

Ist der Graph G als Adjazenzliste gespeichert, dann können G_x und G_N in $O(n+m)$ Zeit erstellt werden.

Ist der Knotengrad maximal 2 (Basisfall) kann ein optimales Vertex Cover in $O(n+m)$ Zeit berechnet werden. $\square$

Korollar 7.6

Es gibt eine Algorithmus für die Optimierungsvariante des Vertex-Cover-Problems, dessen Worst-Case-Laufzeit für Graphen mit n Knoten und m Kanten $O(1{,}5^{k^*} \cdot p(n, m))$ für ein geeignetes Polynom p beträgt, wobei k^* die Größe eines optimalen Vertex Cover bezeichnet.

Korollar 7.6

Es gibt eine Algorithmus für die Optimierungsvariante des Vertex-Cover-Problems, dessen Worst-Case-Laufzeit für Graphen mit n Knoten und m Kanten $O(1{,}5^{k^*} \cdot p(n, m))$ für ein geeignetes Polynom p beträgt, wobei k^* die Größe eines optimalen Vertex Cover bezeichnet.

Algorithmus für die Wertvariante: Teste mit $k = 0$ beginnend ob ein Vertex-Cover der Größe k existiert. In jedem Schritt erhöhe k um eins, bis der Entscheidungsalgorithmus true zurückgibt. Anzahl Schritte ist k^*.

Korollar 7.6

Es gibt eine Algorithmus für die Optimierungsvariante des Vertex-Cover-Problems, dessen Worst-Case-Laufzeit für Graphen mit n Knoten und m Kanten $O(1{,}5^{k^*} \cdot p(n, m))$ für ein geeignetes Polynom p beträgt, wobei k^* die Größe eines optimalen Vertex Cover bezeichnet.

Algorithmus für die Wertvariante: Teste mit $k = 0$ beginnend ob ein Vertex-Cover der Größe k existiert. In jedem Schritt erhöhe k um eins, bis der Entscheidungsalgorithmus true zurückgibt. Anzahl Schritte ist k^*.

Algorithmus für die Optimierungsvariante: Ähnlich kann dann die Optimierungsvariante auf die Wertvariante reduziert werden, wodurch ein weiterer polynomieller Faktor zur Laufzeit hinzukommt.

Parametrisierte Algorithmen

Algorithmen für NP-schwere Probleme deren Laufzeit sich für eine Funktion f und ein Polynom p durch

$$O(f(k) \cdot p(n))$$

beschränken lässt, wobei der Parameter k mit der Eingabe definiert ist (z.B. bei Entscheidungsproblemen) und n die Eingabelänge bezeichnet.

Parametrisierte Algorithmen

Algorithmen für NP-schwere Probleme deren Laufzeit sich für eine Funktion f und ein Polynom p durch

$$O(f(k) \cdot p(n))$$

beschränken lässt, wobei der Parameter k mit der Eingabe definiert ist (z.B. bei Entscheidungsproblemen) und n die Eingabelänge bezeichnet.

Die Funktion f wächst in der Regel exponentiell in dem Parameter k. Worst-Case-Laufzeit noch immer exponentiell, da der Parameter k groß werden kann.

Parametrisierte Algorithmen

Algorithmen für NP-schwere Probleme deren Laufzeit sich für eine Funktion f und ein Polynom p durch

$$O(f(k) \cdot p(n))$$

beschränken lässt, wobei der Parameter k mit der Eingabe definiert ist (z.B. bei Entscheidungsproblemen) und n die Eingabelänge bezeichnet.

Die Funktion f wächst in der Regel exponentiell in dem Parameter k. Worst-Case-Laufzeit noch immer exponentiell, da der Parameter k groß werden kann.

Entkopplung des exponentiellen Wachstums von der Länge der Eingabe. Oft praktikabel solange der Parameter k nicht allzu groß wird.