

Konditionale untere Schranken basierend auf SAT

Seminar Satisfiability

Alexander Kulpe

Ruhr-Universität Bochum

2023-07-05

Inhaltsverzeichnis

Motivation

Konsequenzen aus ETH für harte Probleme

Konsequenzen aus SETH für einfache Probleme

Superlineare untere Schranken basierend auf SAT

Resümee

Inhaltsverzeichnis

Motivation

Konsequenzen aus ETH für harte Probleme

Konsequenzen aus SETH für einfache Probleme

Superlineare untere Schranken basierend auf SAT

Resümee

Problem: REPATTERNMATCHING (REPM)

- **Gegeben:** Regulärer Ausdruck R der Länge n und einen Textstring T der Länge m
- **Frage:** Matcht ein Teilstring von T mit R
- Der beste bekannte Algorithmus hat Laufzeit $\mathcal{O}(nm)$

Problem: REPATTERNMATCHING (REPM)

- **Gegeben:** Regulärer Ausdruck R der Länge n und einen Textstring T der Länge m
- **Frage:** Matcht ein Teilstring von T mit R

- Der beste bekannte Algorithmus hat Laufzeit $\mathcal{O}(nm)$
- **Offene Frage:** Kann das verbessert werden?
- **Genauer:** Kann REPM in Zeit $\tilde{\mathcal{O}}((nm)^{1-\varepsilon})$ für $\varepsilon > 0$ gelöst werden?
- Notation: $f(n) \in \tilde{\mathcal{O}}(g(n))$, wenn es eine Konstante c gibt mit $f(n) \in \mathcal{O}(g(n)(\log n)^c)$

Problem: REPATTERNMATCHING (REPM)

- **Gegeben:** Regulärer Ausdruck R der Länge n und einen Textstring T der Länge m
- **Frage:** Matcht ein Teilstring von T mit R

- Der beste bekannte Algorithmus hat Laufzeit $\mathcal{O}(nm)$
- **Offene Frage:** Kann das verbessert werden?
- **Genauer:** Kann REPM in Zeit $\tilde{\mathcal{O}}((nm)^{1-\varepsilon})$ für $\varepsilon > 0$ gelöst werden?
- Notation: $f(n) \in \tilde{\mathcal{O}}(g(n))$, wenn es eine Konstante c gibt mit $f(n) \in \mathcal{O}(g(n)(\log n)^c)$
- Wir werden sehen: Bessere Laufzeit ist unwahrscheinlich

- Wir haben viele NP-schwierige Probleme kennengelernt
- Untere Schranken werden hierdurch nicht impliziert
- Was können wir also über untere Schranken lernen?
 - $P \neq NP$ impliziert untere Schranke $n^{\omega(1)}$
 - **Frage:** Was können wir aus stärkeren Annahmen für NP-schwierige Probleme lernen?

- Wir haben viele NP-schwierige Probleme kennengelernt
- Untere Schranken werden hierdurch nicht impliziert
- Was können wir also über untere Schranken lernen?
 - $P \neq NP$ impliziert untere Schranke $n^{\omega(1)}$
 - **Frage:** Was können wir aus stärkeren Annahmen für NP-schwierige Probleme lernen?
Was können wir für leichtere Probleme lernen?

k -SAT

- geg: Aussagenlogische Formel φ in KNF mit n Variablen, m Klauseln, $\leq k$ Literalen pro Klauseln
- Frage: Gibt es eine erfüllende Belegung für φ ?
- Wie schnell kann k -SAT gelöst werden?

k -SAT

- geg: Aussagenlogische Formel φ in KNF mit n Variablen, m Klauseln, $\leq k$ Literalen pro Klauseln
- Frage: Gibt es eine erfüllende Belegung für φ ?
- Wie schnell kann k -SAT gelöst werden?
- **Notation:** $f(n) \in \hat{\mathcal{O}}(g(n)) \Leftrightarrow$ wenn es ein Polynom $p(n)$ gibt mit

$$f(n) \in \mathcal{O}(g(n)p(n))$$

- **Beste bekannte Algorithmen:**

- 3-SAT kann in Zeit $\hat{\mathcal{O}}(1.3^n)$ gelöst werden (Makino, Tamaki, Yamamoto 2013)
- k -SAT kann in Zeit $\hat{\mathcal{O}}(2^{(1-c_k)n})$ gelöst werden, mit $c_k = \Theta\left(\frac{1}{k}\right)$

Exponentialzeithypothesen

Exponentialzeithypothese (ETH)

Es gibt ein $\varepsilon > 0$, s.d. 3-SAT nicht in $\hat{O}(2^{\varepsilon n})$ gelöst werden kann

“Es gibt keine Algorithmen, die 3-SAT in subexponentieller Laufzeit lösen”

Exponentialzeithypothesen

Exponentialzeithypothese (ETH)

Es gibt ein $\varepsilon > 0$, s.d. 3-SAT nicht in $\hat{O}(2^{\varepsilon n})$ gelöst werden kann

“Es gibt keine Algorithmen, die 3-SAT in subexponentieller Laufzeit lösen”

Starke Exponentialzeithypothese (SETH)

Für jedes $\varepsilon > 0$, gibt es ein k , s.d. k -SAT nicht in Zeit $\hat{O}(2^{(1-\varepsilon)n})$ gelöst werden kann

“Es gibt keine effizienteren Algorithmen für k -SAT als Brute-Force”

Exponentialzeithypothesen

Exponentialzeithypothese (ETH)

Es gibt ein $\varepsilon > 0$, s.d. 3-SAT nicht in $\hat{O}(2^{\varepsilon n})$ gelöst werden kann

“Es gibt keine Algorithmen, die 3-SAT in subexponentieller Laufzeit lösen”

Starke Exponentialzeithypothese (SETH)

Für jedes $\varepsilon > 0$, gibt es ein k , s.d. k -SAT nicht in Zeit $\hat{O}(2^{(1-\varepsilon)n})$ gelöst werden kann

“Es gibt keine effizienteren Algorithmen für k -SAT als Brute-Force”

- ETH impliziert $P \neq NP$
- SETH impliziert ETH

Inhaltsverzeichnis

Motivation

Konsequenzen aus ETH für harte Probleme

Konsequenzen aus SETH für einfache Probleme

Superlineare untere Schranken basierend auf SAT

Resümee

ETH und Graphenprobleme

- 3-SAT ist NP-vollständiges Problem

ETH und Graphenprobleme

- 3-SAT ist NP-vollständiges Problem
- **Frage:** Hat ETH auch Auswirkungen auf andere NP-vollständige Probleme?

ETH und Graphenprobleme

- 3-SAT ist NP-vollständiges Problem
 - **Frage:** Hat ETH auch Auswirkungen auf andere NP-vollständige Probleme?
- ⇒ Wir schauen uns Reduktionen von 3-SAT auf andere NP-vollständige Probleme an. Hier: Zwei Graphenprobleme

ETH und Graphenprobleme

- 3-SAT ist NP-vollständiges Problem
 - **Frage:** Hat ETH auch Auswirkungen auf andere NP-vollständige Probleme?
- ⇒ Wir schauen uns Reduktionen von 3-SAT auf andere NP-vollständige Probleme an. Hier: Zwei Graphenprobleme

DOMINATINGSET

- **Geg:** Ungerichteter Graph $G = (V, E)$ und natürliche Zahl k
- **Frage:** Gibt es eine Knotenmenge $U \subseteq V, |U| = k$ s.d. für alle $v \in V \setminus U$ ein Knoten $u \in U$ existiert mit $(u, v) \in E$
- “Jeder Knoten des Graphen ist selbst in U oder durch eine Kante mit einem Knoten aus U verbunden”

ETH und Graphenprobleme

- 3-SAT ist NP-vollständiges Problem
 - **Frage:** Hat ETH auch Auswirkungen auf andere NP-vollständige Probleme?
- ⇒ Wir schauen uns Reduktionen von 3-SAT auf andere NP-vollständige Probleme an. Hier: Zwei Graphenprobleme

DOMINATINGSET

- **Geg:** Ungerichteter Graph $G = (V, E)$ und natürliche Zahl k
- **Frage:** Gibt es eine Knotenmenge $U \subseteq V, |U| = k$ s.d. für alle $v \in V \setminus U$ ein Knoten $u \in U$ existiert mit $(u, v) \in E$
- “Jeder Knoten des Graphen ist selbst in U oder durch eine Kante mit einem Knoten aus U verbunden”

INDEPENDENTSET

- **Geg:** Ungerichteter Graph $G = (V, E)$ und eine natürliche Zahl k
- **Frage:** Gibt es eine Knotenmenge $U \subseteq V, |U| = k$ s.d. für $v_1, v_2 \in U$ gilt, dass $(v_1, v_2) \notin E$.
- “Zwischen den Knoten aus U gibt es keine Kanten”

ETH und DOMINATINGSET I

Reduktion 3-SAT zu DOMINATINGSET

- Sei φ 3-KNF mit N Variablen und M Klauseln

Beispiel

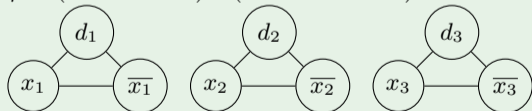
$$\varphi = (x_1 \vee x_2 \vee x_3) \wedge (\neg x_1 \vee \neg x_2 \vee x_3)$$

Reduktion 3-SAT zu DOMINATINGSET

- Sei φ 3-KNF mit N Variablen und M Klauseln
- Wir konstruieren G_φ wie folgt:
 - Für jede Variable x_i konstruiere Dreieck mit Literalknoten x_i, \bar{x}_i und Dummyknoten d_i als Knoten

Beispiel

$$\varphi = (x_1 \vee x_2 \vee x_3) \wedge (\neg x_1 \vee \neg x_2 \vee x_3)$$

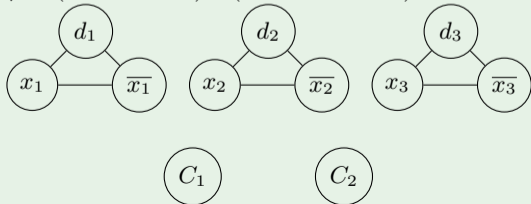


Reduktion 3-SAT zu DOMINATINGSET

- Sei φ 3-KNF mit N Variablen und M Klauseln
- Wir konstruieren G_φ wie folgt:
 - Für jede Variable x_i konstruiere Dreieck mit Literalknoten x_i, \bar{x}_i und Dummyknoten d_i als Knoten
 - Für jede Klausel C_j füge Knoten C_j hinzu

Beispiel

$$\varphi = (x_1 \vee x_2 \vee x_3) \wedge (\neg x_1 \vee \neg x_2 \vee x_3)$$

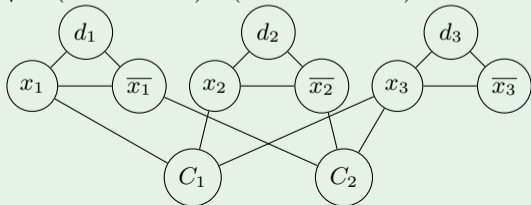


Reduktion 3-SAT zu DOMINATINGSET

- Sei φ 3-KNF mit N Variablen und M Klauseln
- Wir konstruieren G_φ wie folgt:
 - Für jede Variable x_i konstruiere Dreieck mit Literalknoten x_i, \bar{x}_i und Dummyknoten d_i als Knoten
 - Für jede Klausel C_j füge Knoten C_j hinzu
 - Verbinde Literalknoten ℓ mit Klauselknoten C_j genau dann wenn C_j das Literal ℓ enthält

Beispiel

$$\varphi = (x_1 \vee x_2 \vee x_3) \wedge (\neg x_1 \vee \neg x_2 \vee x_3)$$

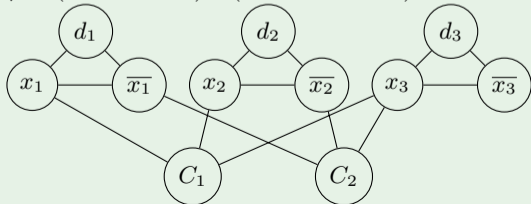


Reduktion 3-SAT zu DOMINATINGSET

- Sei φ 3-KNF mit N Variablen und M Klauseln
- Wir konstruieren G_φ wie folgt:
 - Für jede Variable x_i konstruiere Dreieck mit Literalknoten x_i, \bar{x}_i und Dummyknoten d_i als Knoten
 - Für jede Klausel C_j füge Knoten C_j hinzu
 - Verbinde Literalknoten ℓ mit Klauselknoten C_j genau dann wenn C_j das Literal ℓ enthält
 - Setze k auf N

Beispiel

$$\varphi = (x_1 \vee x_2 \vee x_3) \wedge (\neg x_1 \vee \neg x_2 \vee x_3)$$

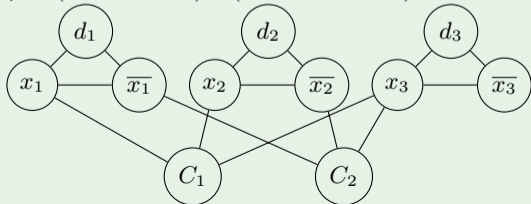


Reduktion 3-SAT zu DOMINATINGSET

- Sei φ 3-KNF mit N Variablen und M Klauseln
- Wir konstruieren G_φ wie folgt:
 - Für jede Variable x_i konstruiere Dreieck mit Literalknoten x_i, \bar{x}_i und Dummyknoten d_i als Knoten
 - Für jede Klausel C_j füge Knoten C_j hinzu
 - Verbinde Literalknoten ℓ mit Klauselknoten C_j genau dann wenn C_j das Literal ℓ enthält
 - Setze k auf N
- G_φ hat $3N + M$ Knoten

Beispiel

$$\varphi = (x_1 \vee x_2 \vee x_3) \wedge (\neg x_1 \vee \neg x_2 \vee x_3)$$



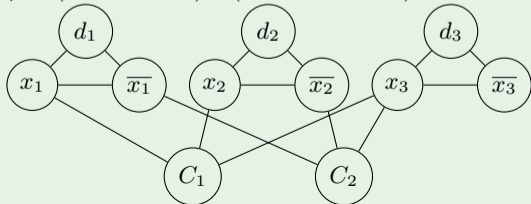
ETH und DOMINATINGSET I

Reduktion 3-SAT zu DOMINATINGSET

- Sei φ 3-KNF mit N Variablen und M Klauseln
- Wir konstruieren G_φ wie folgt:
 - Für jede Variable x_i konstruiere Dreieck mit Literalknoten x_i, \bar{x}_i und Dummyknoten d_i als Knoten
 - Für jede Klausel C_j füge Knoten C_j hinzu
 - Verbinde Literalknoten ℓ mit Klauselknoten C_j genau dann wenn C_j das Literal ℓ enthält
 - Setze k auf N
- G_φ hat $3N + M$ Knoten

Beispiel

$$\varphi = (x_1 \vee x_2 \vee x_3) \wedge (\neg x_1 \vee \neg x_2 \vee x_3)$$



Satz

G_φ besitzt ein DominatingSet der Größe N genau dann wenn φ ist erfüllbar

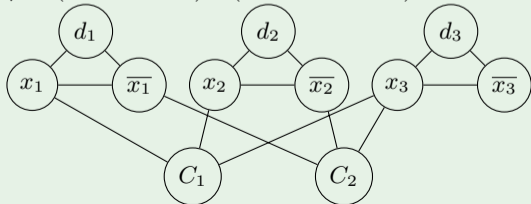
ETH und DOMINATINGSET I

Reduktion 3-SAT zu DOMINATINGSET

- Sei φ 3-KNF mit N Variablen und M Klauseln
- Wir konstruieren G_φ wie folgt:
 - Für jede Variable x_i konstruiere Dreieck mit Literalknoten x_i, \bar{x}_i und Dummyknoten d_i als Knoten
 - Für jede Klausel C_j füge Knoten C_j hinzu
 - Verbinde Literalknoten ℓ mit Klauselknoten C_j genau dann wenn C_j das Literal ℓ enthält
 - Setze k auf N
- G_φ hat $3N + M$ Knoten

Beispiel

$$\varphi = (x_1 \vee x_2 \vee x_3) \wedge (\neg x_1 \vee \neg x_2 \vee x_3)$$



Satz

G_φ besitzt ein DominatingSet der Größe N genau dann wenn φ ist erfüllbar

Beweis.

⇐

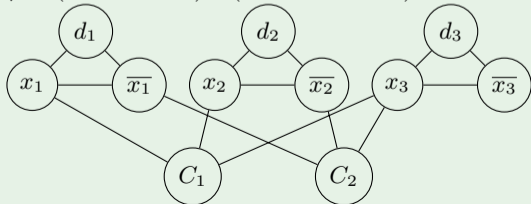
- Angenommen es existiert eine erfüllende Belegung für φ

Reduktion 3-SAT zu DOMINATINGSET

- Sei φ 3-KNF mit N Variablen und M Klauseln
- Wir konstruieren G_φ wie folgt:
 - Für jede Variable x_i konstruiere Dreieck mit Literalknoten x_i, \bar{x}_i und Dummyknoten d_i als Knoten
 - Für jede Klausel C_j füge Knoten C_j hinzu
 - Verbinde Literalknoten ℓ mit Klauselknoten C_j genau dann wenn C_j das Literal ℓ enthält
 - Setze k auf N
- G_φ hat $3N + M$ Knoten

Beispiel

$$\varphi = (x_1 \vee x_2 \vee x_3) \wedge (\neg x_1 \vee \neg x_2 \vee x_3)$$



Satz

G_φ besitzt ein DominatingSet der Größe N genau dann wenn φ ist erfüllbar

Beweis.

←

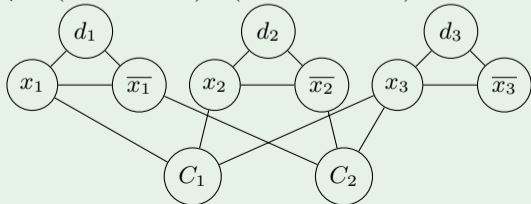
- Angenommen es existiert eine erfüllende Belegung für φ
- Sei S die Menge der Literalknoten ℓ , s.d. ℓ unter dieser Belegung wahr ist
- S enthält genau einen der Knoten x_i, \bar{x}_i . Es gilt $|S| = N$

Reduktion 3-SAT zu DOMINATINGSET

- Sei φ 3-KNF mit N Variablen und M Klauseln
- Wir konstruieren G_φ wie folgt:
 - Für jede Variable x_i konstruiere Dreieck mit Literalknoten x_i, \bar{x}_i und Dummyknoten d_i als Knoten
 - Für jede Klausel C_j füge Knoten C_j hinzu
 - Verbinde Literalknoten ℓ mit Klauselknoten C_j genau dann wenn C_j das Literal ℓ enthält
 - Setze k auf N
- G_φ hat $3N + M$ Knoten

Beispiel

$$\varphi = (x_1 \vee x_2 \vee x_3) \wedge (\neg x_1 \vee \neg x_2 \vee x_3)$$



Satz

G_φ besitzt ein DominatingSet der Größe N genau dann wenn φ ist erfüllbar

Beweis.

⇐

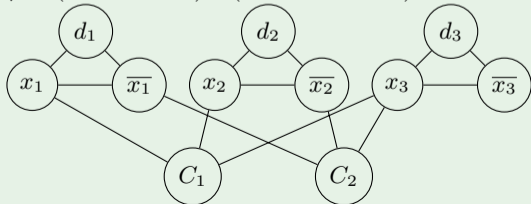
- Angenommen es existiert eine erfüllende Belegung für φ
- Sei S die Menge der Literalknoten ℓ , s.d. ℓ unter dieser Belegung wahr ist
- S enthält genau einen der Knoten x_i, \bar{x}_i . Es gilt $|S| = N$
- Für jede Variable x_i : Die Knoten x_i, \bar{x}_i, d_i sind dominiert

Reduktion 3-SAT zu DOMINATINGSET

- Sei φ 3-KNF mit N Variablen und M Klauseln
- Wir konstruieren G_φ wie folgt:
 - Für jede Variable x_i konstruiere Dreieck mit Literalknoten x_i, \bar{x}_i und Dummyknoten d_i als Knoten
 - Für jede Klausel C_j füge Knoten C_j hinzu
 - Verbinde Literalknoten ℓ mit Klauselknoten C_j genau dann wenn C_j das Literal ℓ enthält
 - Setze k auf N
- G_φ hat $3N + M$ Knoten

Beispiel

$$\varphi = (x_1 \vee x_2 \vee x_3) \wedge (\neg x_1 \vee \neg x_2 \vee x_3)$$



Satz

G_φ besitzt ein DominatingSet der Größe N genau dann wenn φ ist erfüllbar

Beweis.

⇐

- Angenommen es existiert eine erfüllende Belegung für φ
- Sei S die Menge der Literalknoten ℓ , s.d. ℓ unter dieser Belegung wahr ist
- S enthält genau einen der Knoten x_i, \bar{x}_i . Es gilt $|S| = N$
- Für jede Variable x_i : Die Knoten x_i, \bar{x}_i, d_i sind dominiert
- Jeder Klauselknoten C_j ist dominiert

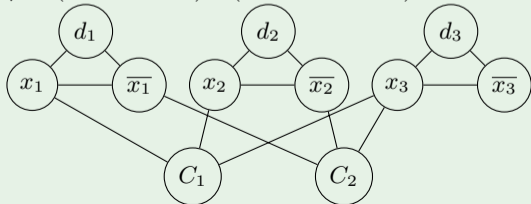
ETH und DOMINATINGSET II

Reduktion 3-SAT zu DOMINATINGSET

- Sei φ 3-KNF mit N Variablen und M Klauseln
- Wir konstruieren G_φ wie folgt:
 - Für jede Variable x_i konstruiere Dreieck mit Literalknoten x_i, \bar{x}_i und Dummyknoten d_i als Knoten
 - Für jede Klausel C_j füge Knoten C_j hinzu
 - Verbinde Literalknoten ℓ mit Klauselknoten C_j genau dann wenn C_j das Literal ℓ enthält
 - Setze k auf N
- G_φ hat $3N + M$ Knoten

Beispiel

$$\varphi = (x_1 \vee x_2 \vee x_3) \wedge (\neg x_1 \vee \neg x_2 \vee x_3)$$



Satz

G_φ besitzt ein DominatingSet der Größe N genau dann wenn φ ist erfüllbar

Beweis.

\Rightarrow

- Angenommen, S sei ein DominatingSet von G_φ der Größe N

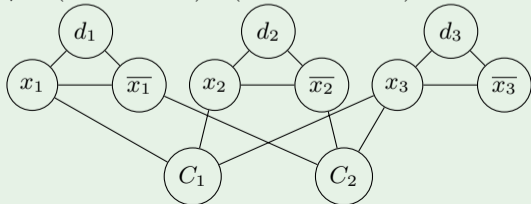
ETH und DOMINATINGSET II

Reduktion 3-SAT zu DOMINATINGSET

- Sei φ 3-KNF mit N Variablen und M Klauseln
- Wir konstruieren G_φ wie folgt:
 - Für jede Variable x_i konstruiere Dreieck mit Literalknoten x_i, \bar{x}_i und Dummyknoten d_i als Knoten
 - Für jede Klausel C_j füge Knoten C_j hinzu
 - Verbinde Literalknoten ℓ mit Klauselknoten C_j genau dann wenn C_j das Literal ℓ enthält
 - Setze k auf N
- G_φ hat $3N + M$ Knoten

Beispiel

$$\varphi = (x_1 \vee x_2 \vee x_3) \wedge (\neg x_1 \vee \neg x_2 \vee x_3)$$



Satz

G_φ besitzt ein DominatingSet der Größe N genau dann wenn φ ist erfüllbar

Beweis.

\Rightarrow

- Angenommen, S sei ein DominatingSet von G_φ der Größe N
- Um alle Literalknoten x, \bar{x}_i, d_i zu dominieren, muss S genau einen dieser Knoten enthalten

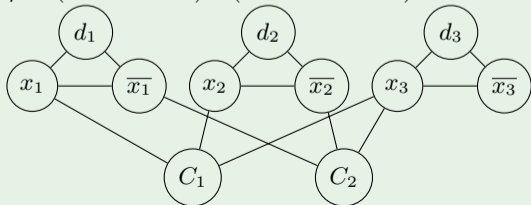
ETH und DOMINATINGSET II

Reduktion 3-SAT zu DOMINATINGSET

- Sei φ 3-KNF mit N Variablen und M Klauseln
- Wir konstruieren G_φ wie folgt:
 - Für jede Variable x_i konstruiere Dreieck mit Literalknoten x_i, \bar{x}_i und Dummyknoten d_i als Knoten
 - Für jede Klausel C_j füge Knoten C_j hinzu
 - Verbinde Literalknoten ℓ mit Klauselknoten C_j genau dann wenn C_j das Literal ℓ enthält
 - Setze k auf N
- G_φ hat $3N + M$ Knoten

Beispiel

$$\varphi = (x_1 \vee x_2 \vee x_3) \wedge (\neg x_1 \vee \neg x_2 \vee x_3)$$



Satz

G_φ besitzt ein DominatingSet der Größe N genau dann wenn φ ist erfüllbar

Beweis.

\Rightarrow

- Angenommen, S sei ein DominatingSet von G_φ der Größe N
- Um alle Literalknoten x, \bar{x}_i, d_i zu dominieren, muss S genau einen dieser Knoten enthalten
- Definiere $\varphi : x_i$ ist TRUE genau dann wenn $x_i \in S$ gilt.

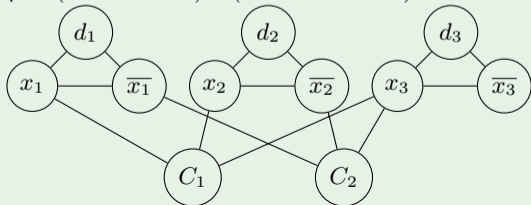
ETH und DOMINATINGSET II

Reduktion 3-SAT zu DOMINATINGSET

- Sei φ 3-KNF mit N Variablen und M Klauseln
- Wir konstruieren G_φ wie folgt:
 - Für jede Variable x_i konstruiere Dreieck mit Literalknoten x_i, \bar{x}_i und Dummyknoten d_i als Knoten
 - Für jede Klausel C_j füge Knoten C_j hinzu
 - Verbinde Literalknoten ℓ mit Klauselknoten C_j genau dann wenn C_j das Literal ℓ enthält
 - Setze k auf N
- G_φ hat $3N + M$ Knoten

Beispiel

$$\varphi = (x_1 \vee x_2 \vee x_3) \wedge (\neg x_1 \vee \neg x_2 \vee x_3)$$



Satz

G_φ besitzt ein DominatingSet der Größe N genau dann wenn φ ist erfüllbar

Beweis.

\Rightarrow

- Angenommen, S sei ein DominatingSet von G_φ der Größe N
- Um alle Literalknoten x, \bar{x}_i, d_i zu dominieren, muss S genau einen dieser Knoten enthalten
- Definiere $\varphi : x_i$ ist TRUE genau dann wenn $x_i \in S$ gilt.
- Da jeder Klauselknoten C_j durch einen Literalknoten $\ell \in S$ dominiert wird, muss C_j per Konstruktion erfüllt sein

□

Satz

Falls ETH gilt, dann lässt sich DOMINATINGSET nicht in Zeit $\hat{O}(2^{o(n)})$ lösen

ETH und DOMINATINGSET III

Satz

Falls ETH gilt, dann lässt sich DOMINATINGSET nicht in Zeit $\hat{O}(2^{o(n)})$ lösen

Beweisskizze

Beweis durch Widerspruch

Satz

Falls ETH gilt, dann lässt sich DOMINATINGSET nicht in Zeit $\hat{O}(2^{o(n)})$ lösen

Beweisskizze

Beweis durch Widerspruch

- **Annahme:** DOMINATINGSET ist in Zeit $\hat{O}(2^{o(n)})$ lösbar

Satz

Falls ETH gilt, dann lässt sich DOMINATINGSET nicht in Zeit $\hat{O}(2^{o(n)})$ lösen

Beweisskizze

Beweis durch Widerspruch

- **Annahme:** DOMINATINGSET ist in Zeit $\hat{O}(2^{o(n)})$ lösbar
- Die Reduktion von 3-SAT auf DOMINATINGSET bildet eine Formel φ mit N Variablen und M Klauseln auf einen Graphen mit $3N + M$ Knoten ab.

Satz

Falls ETH gilt, dann lässt sich DOMINATINGSET nicht in Zeit $\hat{O}(2^{o(n)})$ lösen

Beweisskizze

Beweis durch Widerspruch

- **Annahme:** DOMINATINGSET ist in Zeit $\hat{O}(2^{o(n)})$ lösbar
- Die Reduktion von 3-SAT auf DOMINATINGSET bildet eine Formel φ mit N Variablen und M Klauseln auf einen Graphen mit $3N + M$ Knoten ab.
- Sei nun φ eine 3-KNF-Formel mit N Variablen und M Klauseln ($N \leq 3M$)

Satz

Falls ETH gilt, dann lässt sich DOMINATINGSET nicht in Zeit $\hat{O}(2^{o(n)})$ lösen

Beweisskizze

Beweis durch Widerspruch

- **Annahme:** DOMINATINGSET ist in Zeit $\hat{O}(2^{o(n)})$ lösbar
- Die Reduktion von 3-SAT auf DOMINATINGSET bildet eine Formel φ mit N Variablen und M Klauseln auf einen Graphen mit $3N + M$ Knoten ab.
- Sei nun φ eine 3-KNF-Formel mit N Variablen und M Klauseln ($N \leq 3M$)
- Nach Annahme kann der aus φ entstehende Graph in Zeit $\hat{O}(2^{o(3N+M)})$ überprüft werden

Problem

- ETH macht nur eine Aussage über die Anzahl der Variablen N , nicht über die Anzahl der Klauseln M
 - Gilt ETH auch für M ?
- ⇒ Sparsification Lemma

Lemma (Sparsification Lemma)

- Seien $k, \varepsilon > 0$ und sei $C = C(k, \varepsilon)$ eine Konstante. Jede k -KNF-Formel φ kann in Zeit $\hat{O}(2^{\varepsilon n})$ zu einer Disjunktion $\varphi_1 \vee \dots \vee \varphi_t$ umgeformt werden, s.d.
 - (1) $t \leq 2^{\varepsilon n}$
 - (2) φ erfüllbar \Leftrightarrow eine der Formeln $\varphi_1, \dots, \varphi_t$ Erfüllbarkeit
 - (3) In $\varphi_1, \dots, \varphi_t$ kommt jede Variable höchstens in C Klauseln vor

Lemma (Sparsification Lemma)

- Seien $k, \varepsilon > 0$ und sei $C = C(k, \varepsilon)$ eine Konstante. Jede k -KNF-Formel φ kann in Zeit $\hat{O}(2^{\varepsilon n})$ zu einer Disjunktion $\varphi_1 \vee \dots \vee \varphi_t$ umgeformt werden, s.d.
 - (1) $t \leq 2^{\varepsilon n}$
 - (2) φ erfüllbar \Leftrightarrow eine der Formeln $\varphi_1, \dots, \varphi_t$ Erfüllbarkeit
 - (3) In $\varphi_1, \dots, \varphi_t$ kommt jede Variable höchstens in C Klauseln vor

Beispiel

- $\varphi = (x_1 \vee \neg x_2 \vee x_3) \wedge (\neg x_1 \vee x_3 \vee x_5) \wedge (x_1 \vee \neg x_2 \vee x_4) \wedge (x_1 \vee \neg x_2 \vee x_3 \vee \neg x_5) \wedge (x_3 \vee x_4)$

Lemma (Sparsification Lemma)

- Seien $k, \varepsilon > 0$ und sei $C = C(k, \varepsilon)$ eine Konstante. Jede k -KNF-Formel φ kann in Zeit $\hat{O}(2^{\varepsilon n})$ zu einer Disjunktion $\varphi_1 \vee \dots \vee \varphi_t$ umgeformt werden, s.d.
 - (1) $t \leq 2^{\varepsilon n}$
 - (2) φ erfüllbar \Leftrightarrow eine der Formeln $\varphi_1, \dots, \varphi_t$ Erfüllbarkeit
 - (3) In $\varphi_1, \dots, \varphi_t$ kommt jede Variable höchstens in C Klauseln vor

Beispiel

- $\varphi = (x_1 \vee \neg x_2 \vee x_3) \wedge (\neg x_1 \vee x_3 \vee x_5) \wedge (x_1 \vee \neg x_2 \vee x_4) \wedge (x_1 \vee \neg x_2 \vee x_3 \vee \neg x_5) \wedge (x_3 \vee x_4)$
- Die Literale x_1 und $\neg x_2$ treten zusammen in drei Klauseln auf

Lemma (Sparsification Lemma)

- Seien $k, \varepsilon > 0$ und sei $C = C(k, \varepsilon)$ eine Konstante. Jede k -KNF-Formel φ kann in Zeit $\hat{O}(2^{\varepsilon n})$ zu einer Disjunktion $\varphi_1 \vee \dots \vee \varphi_t$ umgeformt werden, s.d.
 - (1) $t \leq 2^{\varepsilon n}$
 - (2) φ erfüllbar \Leftrightarrow eine der Formeln $\varphi_1, \dots, \varphi_t$ Erfüllbarkeit
 - (3) In $\varphi_1, \dots, \varphi_t$ kommt jede Variable höchstens in C Klauseln vor

Beispiel

- $\varphi = (x_1 \vee \neg x_2 \vee x_3) \wedge (\neg x_1 \vee x_3 \vee x_5) \wedge (x_1 \vee \neg x_2 \vee x_4) \wedge (x_1 \vee \neg x_2 \vee x_3 \vee \neg x_5) \wedge (x_3 \vee x_4)$
- Die Literale x_1 und $\neg x_2$ treten zusammen in drei Klauseln auf
- **Idee:** Erstelle φ_1 und φ_2 nach Fallunterscheidung, ob
 - φ_1 : $x_1 \vee \neg x_2$ wahr ist,
 - φ_2 : $x_1 \vee \neg x_2$ falsch ist

Lemma (Sparsification Lemma)

- Seien $k, \varepsilon > 0$ und sei $C = C(k, \varepsilon)$ eine Konstante. Jede k -KNF-Formel φ kann in Zeit $\hat{O}(2^{\varepsilon n})$ zu einer Disjunktion $\varphi_1 \vee \dots \vee \varphi_t$ umgeformt werden, s.d.
 - (1) $t \leq 2^{\varepsilon n}$
 - (2) φ erfüllbar \Leftrightarrow eine der Formeln $\varphi_1, \dots, \varphi_t$ Erfüllbarkeit
 - (3) In $\varphi_1, \dots, \varphi_t$ kommt jede Variable höchstens in C Klauseln vor

Beispiel

- $\varphi = (x_1 \vee \neg x_2 \vee x_3) \wedge (\neg x_1 \vee x_3 \vee x_5) \wedge (x_1 \vee \neg x_2 \vee x_4) \wedge (x_1 \vee \neg x_2 \vee x_3 \vee \neg x_5) \wedge (x_3 \vee x_4)$
- Die Literale x_1 und $\neg x_2$ treten zusammen in drei Klauseln auf
- **Idee:** Erstelle φ_1 und φ_2 nach Fallunterscheidung, ob
 - φ_1 : $x_1 \vee \neg x_2$ wahr ist,
 - φ_2 : $x_1 \vee \neg x_2$ falsch ist
- $\varphi_1 = (x_1 \vee \neg x_2) \wedge (\neg x_1 \vee x_3 \vee x_5) \wedge (x_3 \vee x_4)$
- $\varphi_2 = x_3 \wedge (\neg x_1 \vee x_3 \vee x_5) \wedge x_4 \wedge (x_3 \vee \neg x_5) \wedge (x_3 \vee x_4)$

Lemma (Sparsification Lemma)

- Seien $k, \varepsilon > 0$ und sei $C = C(k, \varepsilon)$ eine Konstante. Jede k -KNF-Formel φ kann in Zeit $\hat{O}(2^{\varepsilon n})$ zu einer Disjunktion $\varphi_1 \vee \dots \vee \varphi_t$ umgeformt werden, s.d.
 - (1) $t \leq 2^{\varepsilon n}$
 - (2) φ erfüllbar \Leftrightarrow eine der Formeln $\varphi_1, \dots, \varphi_t$ Erfüllbarkeit
 - (3) In $\varphi_1, \dots, \varphi_t$ kommt jede Variable höchstens in C Klauseln vor

Beispiel

- $\varphi = (x_1 \vee \neg x_2 \vee x_3) \wedge (\neg x_1 \vee x_3 \vee x_5) \wedge (x_1 \vee \neg x_2 \vee x_4) \wedge (x_1 \vee \neg x_2 \vee x_3 \vee \neg x_5) \wedge (x_3 \vee x_4)$
- Die Literale x_1 und $\neg x_2$ treten zusammen in drei Klauseln auf
- **Idee:** Erstelle φ_1 und φ_2 nach Fallunterscheidung, ob
 - φ_1 : $x_1 \vee \neg x_2$ wahr ist,
 - φ_2 : $x_1 \vee \neg x_2$ falsch ist
- $\varphi_1 = (x_1 \vee \neg x_2) \wedge (\neg x_1 \vee x_3 \vee x_5) \wedge (x_3 \vee x_4)$
- $\varphi_2 = x_3 \wedge (\neg x_1 \vee x_3 \vee x_5) \wedge x_4 \wedge (x_3 \vee \neg x_5) \wedge (x_3 \vee x_4)$
- Die wiederholte Anwendungen "verdünnt" φ und wir erhalten $\varphi_1, \dots, \varphi_t$

Lemma (Sparsification Lemma)

- Seien $k, \varepsilon > 0$ und sei $C = C(k, \varepsilon)$ eine Konstante. Jede k -KNF-Formel φ kann in Zeit $\hat{O}(2^{\varepsilon n})$ zu einer Disjunktion $\varphi_1 \vee \dots \vee \varphi_t$ umgeformt werden, s.d.
 - (1) $t \leq 2^{\varepsilon n}$
 - (2) φ erfüllbar \Leftrightarrow eine der Formeln $\varphi_1, \dots, \varphi_t$ Erfüllbarkeit
 - (3) In $\varphi_1, \dots, \varphi_t$ kommt jede Variable höchstens in C Klauseln vor

Beispiel

- $\varphi = (x_1 \vee \neg x_2 \vee x_3) \wedge (\neg x_1 \vee x_3 \vee x_5) \wedge (x_1 \vee \neg x_2 \vee x_4) \wedge (x_1 \vee \neg x_2 \vee x_3 \vee \neg x_5) \wedge (x_3 \vee x_4)$
- Die Literale x_1 und $\neg x_2$ treten zusammen in drei Klauseln auf
- **Idee:** Erstelle φ_1 und φ_2 nach Fallunterscheidung, ob
 - φ_1 : $x_1 \vee \neg x_2$ wahr ist,
 - φ_2 : $x_1 \vee \neg x_2$ falsch ist
- $\varphi_1 = (x_1 \vee \neg x_2) \wedge (\neg x_1 \vee x_3 \vee x_5) \wedge (x_3 \vee x_4)$
- $\varphi_2 = x_3 \wedge (\neg x_1 \vee x_3 \vee x_5) \wedge x_4 \wedge (x_3 \vee \neg x_5) \wedge (x_3 \vee x_4)$
- Die wiederholte Anwendungen "verdünnt" φ und wir erhalten $\varphi_1, \dots, \varphi_t$

Folgerung

Falls ETH gilt, gibt es ein $s > 0$, s.d. 3-SAT nicht in Zeit $\hat{O}(2^{sm})$ entschieden werden kann

Satz

Falls ETH gilt, dann lässt sich DOMINATINGSET nicht in Zeit $\hat{O}(2^{o(n)})$ lösen

Beweisskizze

Beweis durch Widerspruch

- **Annahme:** DOMINATINGSET ist in Zeit $\hat{O}(2^{o(n)})$ lösbar
- Die Reduktion von 3-SAT auf DOMINATINGSET bildet eine Formel φ mit N Variablen und M Klauseln auf einen Graphen mit $3N + M$ Knoten ab.
- Sei nun φ eine 3-KNF-Formel mit N Variablen und M Klauseln ($N \leq 3M$)
- Nach Annahme kann der aus φ entstehende Graph in Zeit $\hat{O}(2^{o(3N+M)})$ überprüft werden

Problem

- ETH macht nur eine Aussage über die Anzahl der Variablen N , nicht über die Anzahl der Klauseln M
 - Gilt ETH auch für M ?
- ⇒ Sparsification Lemma

Satz

Falls ETH gilt, dann lässt sich DOMINATINGSET nicht in Zeit $\hat{O}(2^{o(n)})$ lösen

Beweisskizze

Beweis durch Widerspruch

- **Annahme:** DOMINATINGSET ist in Zeit $\hat{O}(2^{o(n)})$ lösbar
- Die Reduktion von 3-SAT auf DOMINATINGSET bildet eine Formel φ mit N Variablen und M Klauseln auf einen Graphen mit $3N + M$ Knoten ab.
- Sei nun φ eine 3-KNF-Formel mit N Variablen und M Klauseln ($N \leq 3M$)
- Nach Annahme kann der aus φ entstehende Graph in Zeit $\hat{O}(2^{o(3N+M)})$ überprüft werden

Kein Problem mehr

- ETH macht eine Aussage über die Anzahl der Variablen N und über die Anzahl der Klauseln M

Satz

Falls ETH gilt, dann lässt sich DOMINATINGSET nicht in Zeit $\hat{O}(2^{o(n)})$ lösen

Beweisskizze

Beweis durch Widerspruch

- **Annahme:** DOMINATINGSET ist in Zeit $\hat{O}(2^{o(n)})$ lösbar
 - Die Reduktion von 3-SAT auf DOMINATINGSET bildet eine Formel φ mit N Variablen und M Klauseln auf einen Graphen mit $3N + M$ Knoten ab.
 - Sei nun φ eine 3-KNF-Formel mit N Variablen und M Klauseln ($N \leq 3M$)
 - Nach Annahme kann der aus φ entstehende Graph in Zeit $\hat{O}(2^{o(3N+M)})$ überprüft werden
- $\Rightarrow \varphi$ kann in Zeit $\hat{O}(2^{o(M)})$ auf Erfüllbarkeit getestet werden \nexists

Kein Problem mehr

- ETH macht eine Aussage über die Anzahl der Variablen N und über die Anzahl der Klauseln M

Reduktion 3-SAT zu INDEPENDENTSET

- Sei φ 3-KNF mit N Variablen und M Knoten

Beispiel

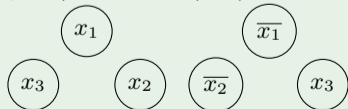
$$\varphi = (x_1 \vee x_2 \vee x_3) \wedge (\neg x_1 \vee \neg x_2 \vee x_3)$$

Reduktion 3-SAT zu INDEPENDENTSET

- Sei φ 3-KNF mit N Variablen und M Knoten
- Wir konstruieren G_φ wie folgt:
 - Für jedes Literal ℓ in einer Klausel füge einen Knoten ℓ hinzu

Beispiel

$$\varphi = (x_1 \vee x_2 \vee x_3) \wedge (\neg x_1 \vee \neg x_2 \vee x_3)$$

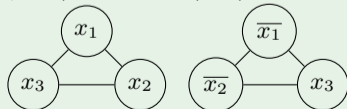


Reduktion 3-SAT zu INDEPENDENTSET

- Sei φ 3-KNF mit N Variablen und M Knoten
- Wir konstruieren G_φ wie folgt:
 - Für jedes Literal ℓ in einer Klausel füge einen Knoten ℓ hinzu
 - Verbinde die drei Literale in einer Klausel zu einem Dreieck

Beispiel

$$\varphi = (x_1 \vee x_2 \vee x_3) \wedge (\neg x_1 \vee \neg x_2 \vee x_3)$$

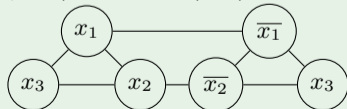


Reduktion 3-SAT zu INDEPENDENTSET

- Sei φ 3-KNF mit N Variablen und M Knoten
- Wir konstruieren G_φ wie folgt:
 - Für jedes Literal ℓ in einer Klausel füge einen Knoten ℓ hinzu
 - Verbinde die drei Literale in einer Klausel zu einem Dreieck
 - Verbinde jedes Literal x_i mit jedem Literal $\overline{x_i}$

Beispiel

$$\varphi = (x_1 \vee x_2 \vee x_3) \wedge (\neg x_1 \vee \neg x_2 \vee x_3)$$

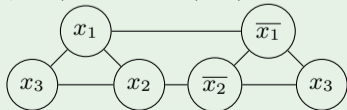


Reduktion 3-SAT zu INDEPENDENTSET

- Sei φ 3-KNF mit N Variablen und M Knoten
- Wir konstruieren G_φ wie folgt:
 - Für jedes Literal ℓ in einer Klausel füge einen Knoten ℓ hinzu
 - Verbinde die drei Literale in einer Klausel zu einem Dreieck
 - Verbinde jedes Literal x_i mit jedem Literal $\overline{x_i}$
 - Setze k auf M

Beispiel

$$\varphi = (x_1 \vee x_2 \vee x_3) \wedge (\neg x_1 \vee \neg x_2 \vee x_3)$$

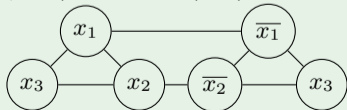


Reduktion 3-SAT zu INDEPENDENTSET

- Sei φ 3-KNF mit N Variablen und M Klauseln
- Wir konstruieren G_φ wie folgt:
 - Für jedes Literal ℓ in einer Klausel füge einen Knoten ℓ hinzu
 - Verbinde die drei Literale in einer Klausel zu einem Dreieck
 - Verbinde jedes Literal x_i mit jedem Literal $\overline{x_i}$
 - Setze k auf M
- G_φ hat $3M$ Knoten

Beispiel

$$\varphi = (x_1 \vee x_2 \vee x_3) \wedge (\neg x_1 \vee \neg x_2 \vee x_3)$$



Reduktion 3-SAT zu INDEPENDENTSET

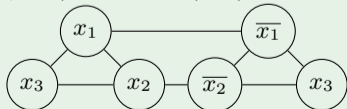
- Sei φ 3-KNF mit N Variablen und M Klauseln
- Wir konstruieren G_φ wie folgt:
 - Für jedes Literal ℓ in einer Klausel füge einen Knoten ℓ hinzu
 - Verbinde die drei Literale in einer Klausel zu einem Dreieck
 - Verbinde jedes Literal x_i mit jedem Literal $\overline{x_i}$
 - Setze k auf M
- G_φ hat $3M$ Knoten

Satz

G_φ besitzt ein IndependentSet der Größe M genau dann wenn φ erfüllbar ist

Beispiel

$$\varphi = (x_1 \vee x_2 \vee x_3) \wedge (\neg x_1 \vee \neg x_2 \vee x_3)$$

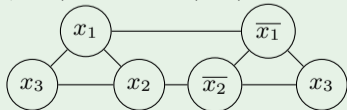


Reduktion 3-SAT zu INDEPENDENTSET

- Sei φ 3-KNF mit N Variablen und M Klauseln
- Wir konstruieren G_φ wie folgt:
 - Für jedes Literal ℓ in einer Klausel füge einen Knoten ℓ hinzu
 - Verbinde die drei Literale in einer Klausel zu einem Dreieck
 - Verbinde jedes Literal x_i mit jedem Literal $\overline{x_i}$
 - Setze k auf M
- G_φ hat $3M$ Knoten

Beispiel

$$\varphi = (x_1 \vee x_2 \vee x_3) \wedge (\neg x_1 \vee \neg x_2 \vee x_3)$$



Satz

G_φ besitzt ein IndependentSet der Größe M genau dann wenn φ erfüllbar ist

Beweis.

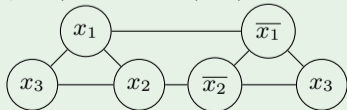
←

Reduktion 3-SAT zu INDEPENDENTSET

- Sei φ 3-KNF mit N Variablen und M Klauseln
- Wir konstruieren G_φ wie folgt:
 - Für jedes Literal ℓ in einer Klausel füge einen Knoten ℓ hinzu
 - Verbinde die drei Literale in einer Klausel zu einem Dreieck
 - Verbinde jedes Literal x_i mit jedem Literal $\overline{x_i}$
 - Setze k auf M
- G_φ hat $3M$ Knoten

Beispiel

$$\varphi = (x_1 \vee x_2 \vee x_3) \wedge (\neg x_1 \vee \neg x_2 \vee x_3)$$



Satz

G_φ besitzt ein IndependentSet der Größe M genau dann wenn φ erfüllbar ist

Beweis.

←

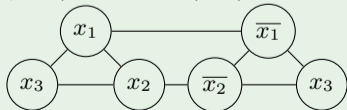
- Angenommen es existiert eine erfüllende Belegung für φ

Reduktion 3-SAT zu INDEPENDENTSET

- Sei φ 3-KNF mit N Variablen und M Klauseln
- Wir konstruieren G_φ wie folgt:
 - Für jedes Literal ℓ in einer Klausel füge einen Knoten ℓ hinzu
 - Verbinde die drei Literale in einer Klausel zu einem Dreieck
 - Verbinde jedes Literal x_i mit jedem Literal $\overline{x_i}$
 - Setze k auf M
- G_φ hat $3M$ Knoten

Beispiel

$$\varphi = (x_1 \vee x_2 \vee x_3) \wedge (\neg x_1 \vee \neg x_2 \vee x_3)$$



Satz

G_φ besitzt ein IndependentSet der Größe M genau dann wenn φ erfüllbar ist

Beweis.

⇐

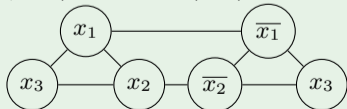
- Angenommen es existiert eine erfüllende Belegung für φ
- Wähle aus jedem Dreieck einen Knoten, dessen Literal unter dieser Belegung wahr ist
- Dies ist ein IndependentSet der Größe M

Reduktion 3-SAT zu INDEPENDENTSET

- Sei φ 3-KNF mit N Variablen und M Klauseln
- Wir konstruieren G_φ wie folgt:
 - Für jedes Literal ℓ in einer Klausel füge einen Knoten ℓ hinzu
 - Verbinde die drei Literale in einer Klausel zu einem Dreieck
 - Verbinde jedes Literal x_i mit jedem Literal $\overline{x_i}$
 - Setze k auf M
- G_φ hat $3M$ Knoten

Beispiel

$$\varphi = (x_1 \vee x_2 \vee x_3) \wedge (\neg x_1 \vee \neg x_2 \vee x_3)$$



Satz

G_φ besitzt ein IndependentSet der Größe M genau dann wenn φ erfüllbar ist

Beweis.

\Rightarrow

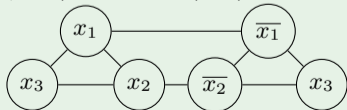
- Angenommen S sei ein IndependentSet von G_φ der Größe M .

Reduktion 3-SAT zu INDEPENDENTSET

- Sei φ 3-KNF mit N Variablen und M Klauseln
- Wir konstruieren G_φ wie folgt:
 - Für jedes Literal ℓ in einer Klausel füge einen Knoten ℓ hinzu
 - Verbinde die drei Literale in einer Klausel zu einem Dreieck
 - Verbinde jedes Literal x_i mit jedem Literal $\overline{x_i}$
 - Setze k auf M
- G_φ hat $3M$ Knoten

Beispiel

$$\varphi = (x_1 \vee x_2 \vee x_3) \wedge (\neg x_1 \vee \neg x_2 \vee x_3)$$



Satz

G_φ besitzt ein IndependentSet der Größe M genau dann wenn φ erfüllbar ist

Beweis.

\Rightarrow

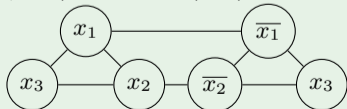
- Angenommen S sei ein IndependentSet von G_φ der Größe M .
- S muss aus jeder Klausel genau einen Knoten enthalten
- S kann keine konfligierende Knoten enthalten

Reduktion 3-SAT zu INDEPENDENTSET

- Sei φ 3-KNF mit N Variablen und M Klauseln
- Wir konstruieren G_φ wie folgt:
 - Für jedes Literal ℓ in einer Klausel füge einen Knoten ℓ hinzu
 - Verbinde die drei Literale in einer Klausel zu einem Dreieck
 - Verbinde jedes Literal x_i mit jedem Literal $\overline{x_i}$
 - Setze k auf M
- G_φ hat $3M$ Knoten

Beispiel

$$\varphi = (x_1 \vee x_2 \vee x_3) \wedge (\neg x_1 \vee \neg x_2 \vee x_3)$$



Satz

G_φ besitzt ein IndependentSet der Größe M genau dann wenn φ erfüllbar ist

Beweis.

\Rightarrow

- Angenommen S sei ein IndependentSet von G_φ der Größe M .
- S muss aus jeder Klausel genau einen Knoten enthalten
- S kann keine konfligierende Knoten enthalten
- Wähle Belegung, die Literale in S wahr macht



Satz

Falls ETH gilt, dann lässt sich INDEPENDENTSET nicht in Zeit $\hat{O}(2^{o(n)})$ lösen

Satz

Falls ETH gilt, dann lässt sich INDEPENDENTSET nicht in Zeit $\hat{O}(2^{o(n)})$ lösen

Beweisskizze

Beweis durch Widerspruch

- **Annahme:** INDEPENDENTSET ist in Zeit $\hat{O}(2^{o(n)})$ lösbar

Satz

Falls ETH gilt, dann lässt sich INDEPENDENTSET nicht in Zeit $\hat{O}(2^{o(n)})$ lösen

Beweisskizze

Beweis durch Widerspruch

- **Annahme:** INDEPENDENTSET ist in Zeit $\hat{O}(2^{o(n)})$ lösbar
- Die Reduktion von 3-SAT auf INDEPENDENTSET bildet eine Formel φ mit N Variablen und M Klauseln auf einen Graphen mit $3M$ Knoten ab
- Sei nun φ eine 3-KNF-Formel mit N Variablen und M Klauseln ($N \leq 3M$)

Satz

Falls ETH gilt, dann lässt sich INDEPENDENTSET nicht in Zeit $\hat{O}(2^{o(n)})$ lösen

Beweisskizze

Beweis durch Widerspruch

- **Annahme:** INDEPENDENTSET ist in Zeit $\hat{O}(2^{o(n)})$ lösbar
 - Die Reduktion von 3-SAT auf INDEPENDENTSET bildet eine Formel φ mit N Variablen und M Klauseln auf einen Graphen mit $3M$ Knoten ab
 - Sei nun φ eine 3-KNF-Formel mit N Variablen und M Klauseln ($N \leq 3M$)
 - Nach Annahme kann der aus φ entstehende Graph in Zeit $\hat{O}(2^{o(3M)})$ überprüft werden
- $\Rightarrow \varphi$ kann in Zeit $\hat{O}(2^{o(M)})$ gelöst werden \nmid

Satz

Falls ETH gilt, dann lässt sich INDEPENDENTSET nicht in Zeit $\hat{O}(2^{o(n)})$ lösen

Beweisskizze

Beweis durch Widerspruch

- **Annahme:** INDEPENDENTSET ist in Zeit $\hat{O}(2^{o(n)})$ lösbar
 - Die Reduktion von 3-SAT auf INDEPENDENTSET bildet eine Formel φ mit N Variablen und M Klauseln auf einen Graphen mit $3M$ Knoten ab
 - Sei nun φ eine 3-KNF-Formel mit N Variablen und M Klauseln ($N \leq 3M$)
 - Nach Annahme kann der aus φ entstehende Graph in Zeit $\hat{O}(2^{o(3M)})$ überprüft werden
- ⇒ φ kann in Zeit $\hat{O}(2^{o(M)})$ gelöst werden ↯

Übungsaufgabe

Finde und beweise ähnliche Aussagen für VERTEXCOVER, CLIQUE, HAMILTONIANCYCLE, TSP, 3-COLORABILITY, ...

Inhaltsverzeichnis

Motivation

Konsequenzen aus ETH für harte Probleme

Konsequenzen aus SETH für einfache Probleme

Superlineare untere Schranken basierend auf SAT

Resümee

Interessante Probleme und Vermutungen

3-SUM

- **Geg:** Menge $M \subset \{-n^3, \dots, n^3\}$ von ganzen Zahlen der Größe n
- **Frage:** Existieren drei paarweise verschiedene Elemente $a, b, c \in M$, s.d. $a + b = c$?

ALL-PAIRS-SHORTEST-PATH (APSP)

- **Geg:** Graph $G = (V, E)$ mit ganzzahligen Kantengewichten
- **Aufgabe:** Bestimme die Distanzen zwischen jedem Knotenpaar

ORTHOGONAL VECTORS (OV)

- **Geg:** Zwei Mengen A, B von n Vektoren aus $\{0, 1\}^d$
- **Frage:** Gibt es $a = (a_1, \dots, a_d) \in A$ und $b = (b_1, \dots, b_d) \in B$, s.d. $\sum_i a_i \cdot b_i = 0$?

Interessante Probleme und Vermutungen

3-SUM

- **Geg:** Menge $M \subset \{-n^3, \dots, n^3\}$ von ganzen Zahlen der Größe n
- **Frage:** Existieren drei paarweise verschiedene Elemente $a, b, c \in M$, s.d. $a + b = c$?
- Algorithmus mit Laufzeit $\tilde{O}(n^2)$:
 - Sortiere M
 - Prüfe für jedes Paar (a, b) , ob die Summe $a + b$ in der Liste vorkommt
- **Vermutung:** Es existiert kein Algorithmus, der 3-SUM in Zeit $\mathcal{O}(n^{2-\varepsilon})$ für ein $\varepsilon > 0$ löst

ALL-PAIRS-SHORTEST-PATH (APSP)

- **Geg:** Graph $G = (V, E)$ mit ganzzahligen Kantengewichten
- **Aufgabe:** Bestimme die Distanzen zwischen jedem Knotenpaar

ORTHOGONAL VECTORS (OV)

- **Geg:** Zwei Mengen A, B von n Vektoren aus $\{0, 1\}^d$
- **Frage:** Gibt es $a = (a_1, \dots, a_d) \in A$ und $b = (b_1, \dots, b_d) \in B$, s.d. $\sum_i a_i \cdot b_i = 0$?

Interessante Probleme und Vermutungen

3-SUM

- **Geg:** Menge $M \subset \{-n^3, \dots, n^3\}$ von ganzen Zahlen der Größe n
- **Frage:** Existieren drei paarweise verschiedene Elemente $a, b, c \in M$, s.d. $a + b = c$?
- Algorithmus mit Laufzeit $\tilde{O}(n^2)$:
 - Sortiere M
 - Prüfe für jedes Paar (a, b) , ob die Summe $a + b$ in der Liste vorkommt
- **Vermutung:** Es existiert kein Algorithmus, der 3-SUM in Zeit $O(n^{2-\varepsilon})$ für ein $\varepsilon > 0$ löst

ALL-PAIRS-SHORTEST-PATH (APSP)

- **Geg:** Graph $G = (V, E)$ mit ganzzahligen Kantengewichten
- **Aufgabe:** Bestimme die Distanzen zwischen jedem Knotenpaar
- Algorithmus mit Laufzeit $O(n^3)$:
 - Floyd-Warshall
- **Vermutung:** Es existiert kein Algorithmus, der APSP in Zeit $O(n^{3-\varepsilon})$ für ein $\varepsilon > 0$ löst

ORTHOGONAL VECTORS (OV)

- **Geg:** Zwei Mengen A, B von n Vektoren aus $\{0, 1\}^d$
- **Frage:** Gibt es $a = (a_1, \dots, a_d) \in A$ und $b = (b_1, \dots, b_d) \in B$, s.d. $\sum_i a_i \cdot b_i = 0$?

Interessante Probleme und Vermutungen

3-SUM

- **Geg:** Menge $M \subset \{-n^3, \dots, n^3\}$ von ganzen Zahlen der Größe n
- **Frage:** Existieren drei paarweise verschiedene Elemente $a, b, c \in M$, s.d. $a + b = c$?
- Algorithmus mit Laufzeit $\tilde{O}(n^2)$:
 - Sortiere M
 - Prüfe für jedes Paar (a, b) , ob die Summe $a + b$ in der Liste vorkommt
- **Vermutung:** Es existiert kein Algorithmus, der 3-SUM in Zeit $\mathcal{O}(n^{2-\varepsilon})$ für ein $\varepsilon > 0$ löst

ALL-PAIRS-SHORTEST-PATH (APSP)

- **Geg:** Graph $G = (V, E)$ mit ganzzahligen Kantengewichten
- **Aufgabe:** Bestimme die Distanzen zwischen jedem Knotenpaar
- Algorithmus mit Laufzeit $\mathcal{O}(n^3)$:
 - Floyd-Warshall
- **Vermutung:** Es existiert kein Algorithmus, der APSP in Zeit $\mathcal{O}(n^{3-\varepsilon})$ für ein $\varepsilon > 0$ löst

ORTHOGONAL VECTORS (OV)

- **Geg:** Zwei Mengen A, B von n Vektoren aus $\{0, 1\}^d$
- **Frage:** Gibt es $a = (a_1, \dots, a_d) \in A$ und $b = (b_1, \dots, b_d) \in B$, s.d. $\sum_i a_i \cdot b_i = 0$?
- Algorithmus mit Laufzeit $\tilde{O}(n^2 d)$:
 - Naives Testen aller Kombinationen
- **Vermutung:** Es existiert kein Algorithmus, der OV in Zeit $\mathcal{O}(n^{2-\varepsilon})$ löst

Feinkörnige Reduktionen

- **Frage:** Wie können wir untere Schranken für andere Probleme finden?

Feinkörnige Reduktionen

- **Frage:** Wie können wir untere Schranken für andere Probleme finden?
- ⇒ Betrachte eine Reduktion der Art: "Wenn Problem A nicht schneller gelöst werden kann als in Zeit T_A , dann kann Problem B nicht schneller gelöst werden als in Zeit T_B "

Feinkörnige Reduktionen

- **Frage:** Wie können wir untere Schranken für andere Probleme finden?
- ⇒ Betrachte eine Reduktion der Art: "Wenn Problem A nicht schneller gelöst werden kann als in Zeit T_A , dann kann Problem B nicht schneller gelöst werden als in Zeit T_B "

Definition

Für Probleme A, B mit Zeitschranken T_A, T_B nennen wir einen Algorithmus, der aus Instanz x von A eine Instanz y von B berechnet, eine Feinkörnige Reduktion, Falls

- x JA-Instanz $\Leftrightarrow y$ JA-Instanz
- für jedes $\varepsilon > 0$ gibt es ein $\delta > 0$, s.d. $T_B(|y|)^{1-\varepsilon} = \mathcal{O}(T_A(|x|)^{1-\delta})$
- Laufzeit der Reduktion beträgt $\mathcal{O}(T_A(|x|)^{1-\gamma})$ für ein $\gamma > 0$.

Feinkörnige Reduktionen

- **Frage:** Wie können wir untere Schranken für andere Probleme finden?
- ⇒ Betrachte eine Reduktion der Art: "Wenn Problem A nicht schneller gelöst werden kann als in Zeit T_A , dann kann Problem B nicht schneller gelöst werden als in Zeit T_B "

Definition

Für Probleme A, B mit Zeitschranken T_A, T_B nennen wir einen Algorithmus, der aus Instanz x von A eine Instanz y von B berechnet, eine Feinkörnige Reduktion, Falls

- x JA-Instanz $\Leftrightarrow y$ JA-Instanz
- für jedes $\varepsilon > 0$ gibt es ein $\delta > 0$, s.d. $T_B(|y|)^{1-\varepsilon} = \mathcal{O}(T_A(|x|)^{1-\delta})$
- Laufzeit der Reduktion beträgt $\mathcal{O}(T_A(|x|)^{1-\gamma})$ für ein $\gamma > 0$.

Satz

- Sei (A, T_A) *reduzierbar auf* (B, T_B) und es existiert ein Algorithmus für B mit Laufzeit $\mathcal{O}(T_B(n)^{1-\varepsilon})$ für ein $\varepsilon > 0$
- Dann existiert ein $\delta > 0$ und ein Algorithmus für A mit Laufzeit $\mathcal{O}(T_A(n)^{1-\delta})$

MATCHING TRIANGLES

- **Geg:** Graph $G = (V, E)$ mit Knotenfärbung $c : V \rightarrow \{1, \dots, n\}$ und natürliche Zahl k
- **Frage:** Gibt es ein Farbtupel $a, b, c \in \{1, \dots, n\}$, s.d. mindestens k Dreiecke in G vorkommen mit Knotenfarben a, b, c ?

MATCHING TRIANGLES

- **Geg:** Graph $G = (V, E)$ mit Knotenfärbung $c : V \rightarrow \{1, \dots, n\}$ und natürliche Zahl k
- **Frage:** Gibt es ein Farbtupel $a, b, c \in \{1, \dots, n\}$, s.d. mindestens k Dreiecke in G vorkommen mit Knotenfarben a, b, c ?

TRIANGLE COLLECTIONS

- **Geg:** Graph $G = (V, E)$ mit Knotenfärbung $c : V \rightarrow \{1, \dots, n\}$
- **Frage:** Gibt es ein Farbtupel $a, b, c \in \{1, \dots, n\}$ paarweise unterschiedlicher Farben, s.d. kein Dreieck in G mit Knotenfarben a, b, c existiert?

MATCHINGTRIANGLES

- **Geg:** Graph $G = (V, E)$ mit Knotenfärbung $c : V \rightarrow \{1, \dots, n\}$ und natürliche Zahl k
- **Frage:** Gibt es ein Farbtupel $a, b, c \in \{1, \dots, n\}$, s.d. mindestens k Dreiecke in G vorkommen mit Knotenfarben a, b, c ?

TRIANGLECOLLECTIONS

- **Geg:** Graph $G = (V, E)$ mit Knotenfärbung $c : V \rightarrow \{1, \dots, n\}$
- **Frage:** Gibt es ein Farbtupel $a, b, c \in \{1, \dots, n\}$ paarweise unterschiedlicher Farben, s.d. kein Dreieck in G mit Knotenfarben a, b, c existiert?

Beispiel

Farben: 

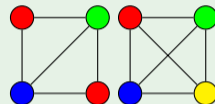
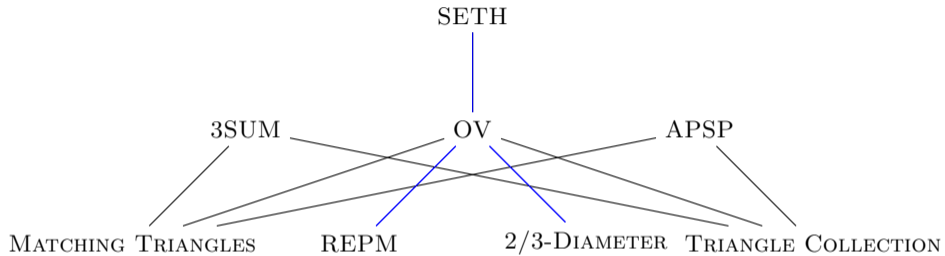


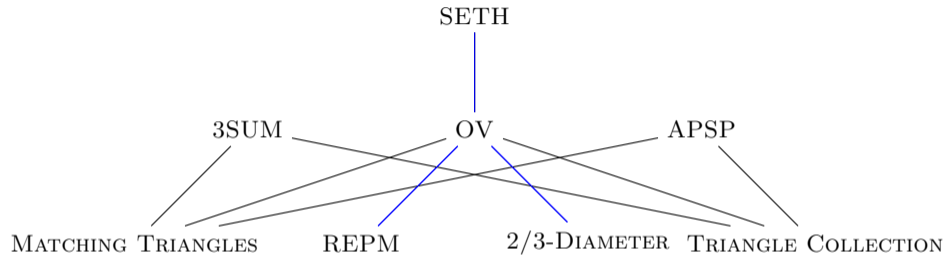
Abbildung: G_1 und G_2

- $(G_1, 1) \in \text{MATCHINGTRIANGLES}$,
 $(G_1, 2) \in \text{MATCHINGTRIANGLES}$
- $G_1 \in \text{TRIANGLECOLLECTIONS}$
- $(G_2, 1) \in \text{MATCHINGTRIANGLES}$,
 $(G_2, 2) \notin \text{MATCHINGTRIANGLES}$
- $G_2 \notin \text{TRIANGLECOLLECTIONS}$

Übersicht Probleme

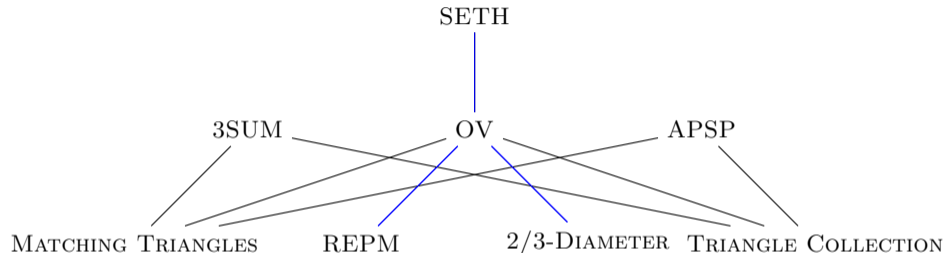


Übersicht Probleme



- Keine bekannten Reduktionen zwischen 3SUM, OV und APSP

Übersicht Probleme



- Keine bekannten Reduktionen zwischen 3SUM, OV und APSP
- Aber: Reduktionen von den drei Problemen zu MATCHING TRIANGLES und TRIANGLE COLLECTION sind bekannt
- MATCHING TRIANGLES und TRIANGLE COLLECTION sind in Zeit $\mathcal{O}(n^3)$ lösbar. Wenn ein Algorithmus mit Laufzeit $\mathcal{O}(n^{3-\epsilon})$ existiert, dann sind alle drei Vermutungen und die SETH falsch

Satz

Wenn SETH gilt, gibt es kein $\varepsilon > 0$ und $c > 0$, s.d. sich OV in Zeit $\mathcal{O}(n^{2-\varepsilon} d^c)$ lösen lässt

SETH zu OV

Satz

Wenn SETH gilt, gibt es kein $\varepsilon > 0$ und $c > 0$, s.d. sich OV in Zeit $\mathcal{O}(n^{2-\varepsilon}d^c)$ lösen lässt

Beweisidee

- Wir konstruieren eine Reduktion von (SAT, $2^n m^{c_1}$) zu (OV, $n^2 d^{c_2}$)

Satz

Wenn SETH gilt, gibt es kein $\varepsilon > 0$ und $c > 0$, s.d. sich OV in Zeit $\mathcal{O}(n^{2-\varepsilon} d^c)$ lösen lässt

Beweisidee

- Wir konstruieren eine Reduktion von (SAT, $2^n m^{c_1}$) zu (OV, $n^2 d^{c_2}$)
- Sei φ eine KNF-Formel mit n Variablen und m Klauseln (ObdA: n gerade)
- Partitioniere Variablen von φ in Mengen V_1, V_2 der Größe $n/2$

Satz

Wenn SETH gilt, gibt es kein $\varepsilon > 0$ und $c > 0$, s.d. sich OV in Zeit $\mathcal{O}(n^{2-\varepsilon}d^c)$ lösen lässt

Beweisidee

- Wir konstruieren eine Reduktion von (SAT, $2^n m^{c_1}$) zu (OV, $n^2 d^{c_2}$)
- Sei φ eine KNF-Formel mit n Variablen und m Klauseln (*ObdA: n gerade*)
- Partitioniere Variablen von φ in Mengen V_1, V_2 der Größe $n/2$
- Für jede partielle Wahrheitsbelegung $\alpha \in V_i$ erstelle einen Vektor $a_\alpha \in \{0, 1\}^m$. Setze i -te Komponente auf 1 genau dann wenn die i -te Klausel nicht durch diese Wahrheitsbelegung erfüllt wird.

SETH zu OV

Satz

Wenn SETH gilt, gibt es kein $\varepsilon > 0$ und $c > 0$, s.d. sich OV in Zeit $\mathcal{O}(n^{2-\varepsilon}d^c)$ lösen lässt

Beweisidee

- Wir konstruieren eine Reduktion von (SAT, $2^n m^{c_1}$) zu (OV, $n^2 d^{c_2}$)
- Sei φ eine KNF-Formel mit n Variablen und m Klauseln (*ObdA: n gerade*)
- Partitioniere Variablen von φ in Mengen V_1, V_2 der Größe $n/2$
- Für jede partielle Wahrheitsbelegung $\alpha \in V_i$ erstelle einen Vektor $a_\alpha \in \{0, 1\}^m$. Setze i -te Komponente auf 1 genau dann wenn die i -te Klausel nicht durch diese Wahrheitsbelegung erfüllt wird.

- Erstelle Menge W_i , die aus diesen Vektoren besteht. Es gilt $|W_i| = 2^{n/2} =: N$
- Laufzeit Reduktion: $\mathcal{O}(ND)$

SETH zu OV

Satz

Wenn SETH gilt, gibt es kein $\varepsilon > 0$ und $c > 0$, s.d. sich OV in Zeit $\mathcal{O}(n^{2-\varepsilon}d^c)$ lösen lässt

Beweisidee

- Wir konstruieren eine Reduktion von (SAT, $2^n m^{c_1}$) zu (OV, $n^2 d^{c_2}$)
- Sei φ eine KNF-Formel mit n Variablen und m Klauseln (*ObdA: n gerade*)
- Partitioniere Variablen von φ in Mengen V_1, V_2 der Größe $n/2$
- Für jede partielle Wahrheitsbelegung $\alpha \in V_i$ erstelle einen Vektor $a_\alpha \in \{0, 1\}^m$. Setze i -te Komponente auf 1 genau dann wenn die i -te Klausel nicht durch diese Wahrheitsbelegung erfüllt wird.

- Erstelle Menge W_i , die aus diesen Vektoren besteht. Es gilt $|W_i| = 2^{n/2} =: N$
- Laufzeit Reduktion: $\mathcal{O}(ND)$

Beispiel

- $\varphi = (x_1 \vee x_2) \wedge (x_1 \vee \neg x_2) \wedge (\neg x_1 \vee x_2) \wedge x_1$
- $V_1 = \{x_1\}, V_2 = \{x_2\}$
- $W_1 = \{(0, 0, 1, 0), (1, 1, 0, 1)\},$
 $W_2 = \{(0, 1, 0, 1), (1, 0, 1, 1)\}$
- $(0, 0, 1, 0) \in W_1, (0, 1, 0, 1) \in W_2$ orthogonal.
 $\{x_1 := \text{TRUE}, x_2 := \text{FALSE}\}$ erfüllende Belegung

SETH zu OV

Satz

Wenn SETH gilt, gibt es kein $\varepsilon > 0$ und $c > 0$, s.d. sich OV in Zeit $\mathcal{O}(n^{2-\varepsilon}d^c)$ lösen lässt

Beweisidee

- Wir konstruieren eine Reduktion von (SAT, $2^n m^{c_1}$) zu (OV, $n^2 d^{c_2}$)
- Sei φ eine KNF-Formel mit n Variablen und m Klauseln (*ObdA: n gerade*)
- Partitioniere Variablen von φ in Mengen V_1, V_2 der Größe $n/2$
- Für jede partielle Wahrheitsbelegung $\alpha \in V_i$ erstelle einen Vektor $a_\alpha \in \{0, 1\}^m$. Setze i -te Komponente auf 1 genau dann wenn die i -te Klausel nicht durch diese Wahrheitsbelegung erfüllt wird.

- Erstelle Menge W_i , die aus diesen Vektoren besteht. Es gilt $|W_i| = 2^{n/2} =: N$
- Laufzeit Reduktion: $\mathcal{O}(ND)$

Beispiel

- $\varphi = (x_1 \vee x_2) \wedge (x_1 \vee \neg x_2) \wedge (\neg x_1 \vee x_2) \wedge x_1$
- $V_1 = \{x_1\}, V_2 = \{x_2\}$
- $W_1 = \{(0, 0, 1, 0), (1, 1, 0, 1)\},$
 $W_2 = \{(0, 1, 0, 1), (1, 0, 1, 1)\}$
- $(0, 0, 1, 0) \in W_1, (0, 1, 0, 1) \in W_2$ orthogonal.
 $\{x_1 := \text{TRUE}, x_2 := \text{FALSE}\}$ erfüllende Belegung
- **Beobachtung:** φ erfüllbar \Leftrightarrow es gibt orthogonale $a \in W_1, b \in W_2$

SETH zu OV

Satz

Wenn SETH gilt, gibt es kein $\varepsilon > 0$ und $c > 0$, s.d. sich OV in Zeit $\mathcal{O}(n^{2-\varepsilon}d^c)$ lösen lässt

Beweisidee

- Wir konstruieren eine Reduktion von (SAT, $2^n m^{c_1}$) zu (OV, $n^2 d^{c_2}$)
- Sei φ eine KNF-Formel mit n Variablen und m Klauseln (*ObdA: n gerade*)
- Partitioniere Variablen von φ in Mengen V_1, V_2 der Größe $n/2$
- Für jede partielle Wahrheitsbelegung $\alpha \in V_i$ erstelle einen Vektor $a_\alpha \in \{0, 1\}^m$. Setze i -te Komponente auf 1 genau dann wenn die i -te Klausel nicht durch diese Wahrheitsbelegung erfüllt wird.

- Erstelle Menge W_i , die aus diesen Vektoren besteht. Es gilt $|W_i| = 2^{n/2} =: N$
- Laufzeit Reduktion: $\mathcal{O}(ND)$

Beispiel

- $\varphi = (x_1 \vee x_2) \wedge (x_1 \vee \neg x_2) \wedge (\neg x_1 \vee x_2) \wedge x_1$
- $V_1 = \{x_1\}, V_2 = \{x_2\}$
- $W_1 = \{(0, 0, 1, 0), (1, 1, 0, 1)\},$
 $W_2 = \{(0, 1, 0, 1), (1, 0, 1, 1)\}$
- $(0, 0, 1, 0) \in W_1, (0, 1, 0, 1) \in W_2$ orthogonal.
 $\{x_1 := \text{TRUE}, x_2 := \text{FALSE}\}$ erfüllende Belegung

- **Beobachtung:** φ erfüllbar \Leftrightarrow es gibt orthogonale $a \in W_1, b \in W_2$
- Angenommen OV in Zeit $\mathcal{O}(N^{2-\varepsilon}D^c)$ lösbar.
- Dann SAT in Zeit $\mathcal{O}(N^{2-\varepsilon}D^c) = \mathcal{O}(2^{n-\varepsilon'} m^c)$ lösbar



REPM (Wdh.)

- **Geg:** Regulärer Ausdruck R und einen Textstring T
- **Frage:** Matcht ein Teilstring von T mit R

Satz

Wenn OV gilt, gibt es kein $\varepsilon > 0$, s.d. sich REPM in Zeit $\mathcal{O}((nm)^{1-\varepsilon})$ lösbar ist

REPM (Wdh.)

- **Geg:** Regulärer Ausdruck R und einen Textstring T
- **Frage:** Matcht ein Teilstring von T mit R

Satz

Wenn OV gilt, gibt es kein $\varepsilon > 0$, s.d. sich REPM in Zeit $\mathcal{O}((nm)^{1-\varepsilon})$ lösbar ist

Beweisidee

- Seien $A, B \subseteq \{0, 1\}^d$ der Größe n

REPM (Wdh.)

- **Geg:** Regulärer Ausdruck R und einen Textstring T
- **Frage:** Matcht ein Teilstring von T mit R

Satz

Wenn OV gilt, gibt es kein $\varepsilon > 0$, s.d. sich REPM in Zeit $\mathcal{O}((nm)^{1-\varepsilon})$ lösbar ist

Beweisidee

- Seien $A, B \subseteq \{0, 1\}^d$ der Größe n
- Für $a \in A$ konstruiere RE r_a :
 - Ersetze Koordinate 1 durch RE 0
 - Ersetze Koordinate 0 durch RE $(0 + 1)$
- Für $b \in B$ konstruiere kanonischen Binärstring s_b

REPM (Wdh.)

- **Geg:** Regulärer Ausdruck R und einen Textstring T
- **Frage:** Matcht ein Teilstring von T mit R

Satz

Wenn OV gilt, gibt es kein $\varepsilon > 0$, s.d. sich REPM in Zeit $\mathcal{O}((nm)^{1-\varepsilon})$ lösbar ist

Beweisidee

- Seien $A, B \subseteq \{0, 1\}^d$ der Größe n
- Für $a \in A$ konstruiere RE r_a :
 - Ersetze Koordinate 1 durch RE 0
 - Ersetze Koordinate 0 durch RE $(0 + 1)$
- Für $b \in B$ konstruiere kanonischen Binärstring s_b

Beispiel

- $a = (0, 1, 0, 1), b = (1, 0, 0, 0)$
- $r_a = (0 + 1)0(0 + 1)0, s_b = 1000$

REPM (Wdh.)

- **Geg:** Regulärer Ausdruck R und einen Textstring T
- **Frage:** Matcht ein Teilstring von T mit R

Satz

Wenn OV gilt, gibt es kein $\varepsilon > 0$, s.d. sich REPM in Zeit $\mathcal{O}((nm)^{1-\varepsilon})$ lösbar ist

Beweisidee

- Seien $A, B \subseteq \{0, 1\}^d$ der Größe n
- Für $a \in A$ konstruiere RE r_a :
 - Ersetze Koordinate 1 durch RE 0
 - Ersetze Koordinate 0 durch RE $(0 + 1)$
- Für $b \in B$ konstruiere kanonischen Binärstring s_b

Beispiel

- $a = (0, 1, 0, 1), b = (1, 0, 0, 0)$
- $r_a = (0 + 1)0(0 + 1)0, s_b = 1000$
- Aus $A = \{a_1, \dots, a_n\}$ konstruiere $R = r_{a_1} + \dots r_{a_n}$
- Aus $B = \{b_1, \dots, b_n\}$ konstruiere $T = s_{b_1} \# s_{b_2} \# \dots \# s_{b_n}$

REPM (Wdh.)

- **Geg:** Regulärer Ausdruck R und einen Textstring T
- **Frage:** Matcht ein Teilstring von T mit R

Satz

Wenn OV gilt, gibt es kein $\varepsilon > 0$, s.d. sich REPM in Zeit $\mathcal{O}((nm)^{1-\varepsilon})$ lösbar ist

Beweisidee

- Seien $A, B \subseteq \{0, 1\}^d$ der Größe n
- Für $a \in A$ konstruiere RE r_a :
 - Ersetze Koordinate 1 durch RE 0
 - Ersetze Koordinate 0 durch RE $(0 + 1)$
- Für $b \in B$ konstruiere kanonischen Binärstring s_b

Beispiel

- $a = (0, 1, 0, 1), b = (1, 0, 0, 0)$
- $r_a = (0 + 1)0(0 + 1)0, s_b = 1000$
- Aus $A = \{a_1, \dots, a_n\}$ konstruiere $R = r_{a_1} + \dots + r_{a_n}$
- Aus $B = \{b_1, \dots, b_n\}$ konstruiere $T = s_{b_1} \# s_{b_2} \# \dots \# s_{b_n}$
- **Beobachtung:** Es gibt orthogonale Vektoren $a \in A, b \in B \Leftrightarrow$ Teilstring von T matcht mit R

REPM (Wdh.)

- **Geg:** Regulärer Ausdruck R und einen Textstring T
- **Frage:** Matcht ein Teilstring von T mit R

Satz

Wenn OV gilt, gibt es kein $\varepsilon > 0$, s.d. sich REPM in Zeit $\mathcal{O}((nm)^{1-\varepsilon})$ lösbar ist

Beweisidee

- Seien $A, B \subseteq \{0, 1\}^d$ der Größe n
- Für $a \in A$ konstruiere RE r_a :
 - Ersetze Koordinate 1 durch RE 0
 - Ersetze Koordinate 0 durch RE $(0 + 1)$
- Für $b \in B$ konstruiere kanonischen Binärstring s_b

Beispiel

- $a = (0, 1, 0, 1), b = (1, 0, 0, 0)$
- $r_a = (0 + 1)0(0 + 1)0, s_b = 1000$
- Aus $A = \{a_1, \dots, a_n\}$ konstruiere $R = r_{a_1} + \dots + r_{a_n}$
- Aus $B = \{b_1, \dots, b_n\}$ konstruiere $T = s_{b_1} \# s_{b_2} \# \dots \# s_{b_n}$
- **Beobachtung:** Es gibt orthogonale Vektoren $a \in A, b \in B \Leftrightarrow$ Teilstring von T matcht mit R
- Laufzeit Reduktion und Größe von R, T : $\mathcal{O}(nd)$
- Angenommen REPM in Zeit $\mathcal{O}((|R||T|)^{1-\varepsilon} d^c)$ lösbar
- Dann OV in Zeit $\mathcal{O}((|R||T|)^{1-\varepsilon}) = \mathcal{O}((n^2 d^2)^{1-\varepsilon}) = \mathcal{O}(n^{2-\varepsilon'} d^c)$ lösbar



Bonus: 2/3-DIAMETER

Satz

Wenn OV-Annahme gilt, dann ist das Unterscheiden von Durchmesser 2 zu Durchmesser 3 nicht in Zeit $\mathcal{O}(n^{2-\varepsilon})$ möglich, selbst wenn $m = \mathcal{O}(n)$

Bonus: 2/3-DIAMETER

Satz

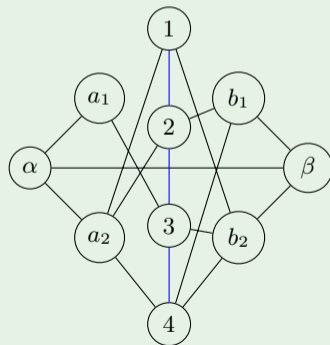
Wenn OV-Annahme gilt, dann ist das Unterscheiden von Durchmesser 2 zu Durchmesser 3 nicht in Zeit $\mathcal{O}(n^{2-\varepsilon})$ möglich, selbst wenn $m = \mathcal{O}(n)$

Beweisskizze

- Seien $A, B \subseteq \{0, 1\}^d$ der Größe n

Beispiel

$A = \{a_1 = (0, 0, 1, 0), a_2 = (1, 1, 0, 1)\}$,
 $B = \{b_1 = (0, 1, 0, 1), b_2 = (1, 0, 1, 1)\}$



Koordinatenknoten bilden Clique

Bonus: 2/3-DIAMETER

Satz

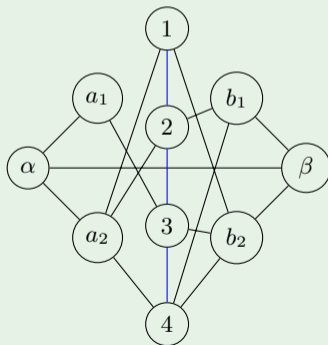
Wenn OV-Annahme gilt, dann ist das Unterscheiden von Durchmesser 2 zu Durchmesser 3 nicht in Zeit $\mathcal{O}(n^{2-\varepsilon})$ möglich, selbst wenn $m = \mathcal{O}(n)$

Beweisskizze

- Seien $A, B \subseteq \{0, 1\}^d$ der Größe n
- Für jeden Vektor u erstelle Vektorknoten v_u
- Für jede Koordinate c erstelle Koordinatenknoten v_c
- Füge Knoten α, β hinzu

Beispiel

$A = \{a_1 = (0, 0, 1, 0), a_2 = (1, 1, 0, 1)\}$,
 $B = \{b_1 = (0, 1, 0, 1), b_2 = (1, 0, 1, 1)\}$



Koordinatenknoten bilden Clique

Bonus: 2/3-DIAMETER

Satz

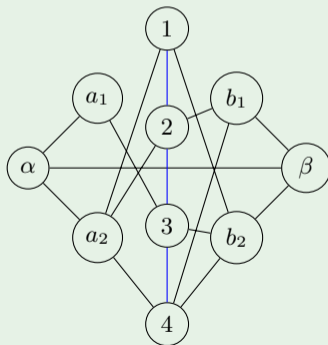
Wenn OV-Annahme gilt, dann ist das Unterscheiden von Durchmesser 2 zu Durchmesser 3 nicht in Zeit $\mathcal{O}(n^{2-\varepsilon})$ möglich, selbst wenn $m = \mathcal{O}(n)$

Beweisskizze

- Seien $A, B \subseteq \{0, 1\}^d$ der Größe n
- Für jeden Vektor u erstelle Vektorknoten v_u
- Für jede Koordinate c erstelle Koordinatenknoten v_c
- Füge Knoten α, β hinzu
- Füge Kanten hinzu zwischen

Beispiel

$A = \{a_1 = (0, 0, 1, 0), a_2 = (1, 1, 0, 1)\}$,
 $B = \{b_1 = (0, 1, 0, 1), b_2 = (1, 0, 1, 1)\}$



Koordinatenknoten bilden Clique

Bonus: 2/3-DIAMETER

Satz

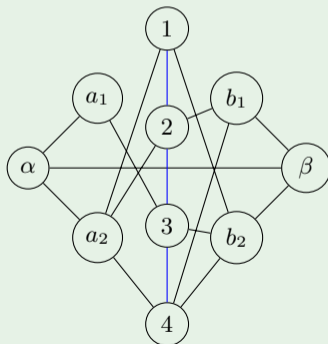
Wenn OV-Annahme gilt, dann ist das Unterscheiden von Durchmesser 2 zu Durchmesser 3 nicht in Zeit $\mathcal{O}(n^{2-\varepsilon})$ möglich, selbst wenn $m = \mathcal{O}(n)$

Beweisskizze

- Seien $A, B \subseteq \{0, 1\}^d$ der Größe n
- Für jeden Vektor u erstelle Vektorknoten v_u
- Für jede Koordinate c erstelle Koordinatenknoten v_c
- Füge Knoten α, β hinzu
- Füge Kanten hinzu zwischen
 - Vektorknoten v_u und Koordinatenknoten v_c falls $v_u[c] = 1$
 - α und β
 - α und jedem Vektorknoten $v_a, a \in A$
 - β und jedem Vektorknoten $v_b, b \in B$
 - jedem Koordinatenknoten

Beispiel

$$A = \{a_1 = (0, 0, 1, 0), a_2 = (1, 1, 0, 1)\},$$
$$B = \{b_1 = (0, 1, 0, 1), b_2 = (1, 0, 1, 1)\}$$



Koordinatenknoten bilden Clique

Bonus: 2/3-DIAMETER

Satz

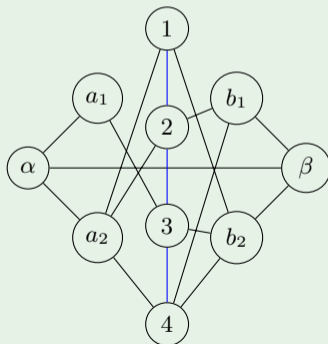
Wenn OV-Annahme gilt, dann ist das Unterscheiden von Durchmesser 2 zu Durchmesser 3 nicht in Zeit $\mathcal{O}(n^{2-\varepsilon})$ möglich, selbst wenn $m = \mathcal{O}(n)$

Beweisskizze

- Seien $A, B \subseteq \{0, 1\}^d$ der Größe n
- Für jeden Vektor u erstelle Vektorknoten v_u
- Für jede Koordinate c erstelle Koordinatenknoten v_c
- Füge Knoten α, β hinzu
- Füge Kanten hinzu zwischen
 - Vektorknoten v_u und Koordinatenknoten v_c falls $v_u[c] = 1$
 - α und β
 - α und jedem Vektorknoten $v_a, a \in A$
 - β und jedem Vektorknoten $v_b, b \in B$
 - jedem Koordinatenknoten
- **Beobachtung:** Existieren $a \in A, b \in B$ orthogonal, dann hat der Graph Durchmesser 3.

Beispiel

$$A = \{a_1 = (0, 0, 1, 0), a_2 = (1, 1, 0, 1)\},$$
$$B = \{b_1 = (0, 1, 0, 1), b_2 = (1, 0, 1, 1)\}$$



Koordinatenknoten bilden Clique

Inhaltsverzeichnis

Motivation

Konsequenzen aus ETH für harte Probleme

Konsequenzen aus SETH für einfache Probleme

Superlineare untere Schranken basierend auf SAT

Resümee

Von SAT zu LOGCIRCUITSAT

SAT

Erfüllbarkeit KNF-Formeln

Von SAT zu LOGCIRCUITSAT

SAT

Erfüllbarkeit KNF-Formeln

CIRCUITSAT

Erfüllbarkeit Boolescher Schaltkreise

Von SAT zu LOGCIRCUITSAT

SAT

Erfüllbarkeit KNF-Formeln

CIRCUITSAT

Erfüllbarkeit Boolescher Schaltkreise

- CIRCUITSAT ist eine Verallgemeinerung von SAT

Von SAT zu LOGCIRCUITSAT

SAT

Erfüllbarkeit KNF-Formeln

CIRCUITSAT

Erfüllbarkeit Boolescher Schaltkreise

- CIRCUITSAT ist eine Verallgemeinerung von SAT

LOGCIRCUITSAT

Erfüllbarkeit Boolescher Schaltkreise mit beschränktem Fan-in, m Gattern und $\log m$ Inputs

Von SAT zu LOGCIRCUITSAT

SAT

Erfüllbarkeit KNF-Formeln

CIRCUITSAT

Erfüllbarkeit Boolescher Schaltkreise

- CIRCUITSAT ist eine Verallgemeinerung von SAT

LOGCIRCUITSAT

Erfüllbarkeit Boolescher Schaltkreise mit beschränktem Fan-in, m Gattern und $\log m$ Inputs

- LOGCIRCUITSAT in polynomieller Laufzeit entscheidbar

Von SAT zu LOGCIRCUITSAT

SAT

Erfüllbarkeit KNF-Formeln

CIRCUITSAT

Erfüllbarkeit Boolescher Schaltkreise

- CIRCUITSAT ist eine Verallgemeinerung von SAT

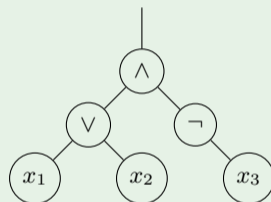
LOGCIRCUITSAT

Erfüllbarkeit Boolescher Schaltkreise mit beschränktem Fan-in, m Gattern und $\log m$ Inputs

- LOGCIRCUITSAT in polynomieller Laufzeit entscheidbar

Beispiel

$$\varphi = (x_1 \vee x_2) \wedge \neg x_3$$



Von SAT zu LOGCIRCUITSAT

SAT

Erfüllbarkeit KNF-Formeln

CIRCUITSAT

Erfüllbarkeit Boolescher Schaltkreise

- CIRCUITSAT ist eine Verallgemeinerung von SAT

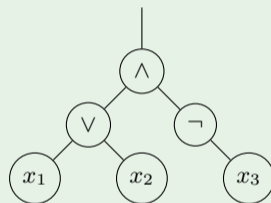
LOGCIRCUITSAT

Erfüllbarkeit Boolescher Schaltkreise mit beschränktem Fan-in, m Gattern und $\log m$ Inputs

- LOGCIRCUITSAT in polynomieller Laufzeit entscheidbar

Beispiel

$$\varphi = (x_1 \vee x_2) \wedge \neg x_3$$



- **Ziel:** Wenn ETH gilt, dann ist LOGCIRCUITSAT nicht in essentiell-linearer Zeit lösbar
- Notation: Ein Problem ist in essentiell-linearer Zeit lösbar, falls das Problem für alle $\varepsilon > 0$ in Zeit $\mathcal{O}(n^{1+\varepsilon})$ lösbar ist.

Beschränkter Nicht-Determinismus

- Gibt es bessere Ansätze als Brute-Force bei effizient lösbaren Problemen?

Beschränkter Nicht-Determinismus

- Gibt es bessere Ansätze als Brute-Force bei effizient lösbaren Problemen?
- **Idee:** Beschränkter Nicht-Determinismus: Beschränke den Grad von Nicht-Determinismus in einer Berechnung
- Halte im Input Platz für Bits vom Zeugen frei

Beschränkter Nicht-Determinismus

- Gibt es bessere Ansätze als Brute-Force bei effizient lösbaren Problemen?
- **Idee:** Beschränkter Nicht-Determinismus: Beschränke den Grad von Nicht-Determinismus in einer Berechnung
- Halte im Input Platz für Bits vom Zeugen frei

⇒ Zeuge beeinflusst Länge nicht!

Beschränkter Nicht-Determinismus

- Gibt es bessere Ansätze als Brute-Force bei effizient lösbaren Problemen?
 - **Idee:** Beschränkter Nicht-Determinismus: Beschränke den Grad von Nicht-Determinismus in einer Berechnung
 - Halte im Input Platz für Bits vom Zeugen frei
- ⇒ Zeuge beeinflusst Länge nicht!

Definition

Eine Sprache L ist in $\text{TIWI}(t(n), w(n))$, wenn eine Verifikationssprache V für L existiert, s.d.

- $V \in \text{TIME}(t(n))$
- Zeugen sind binär mit Länge $\leq w(n)$
- Kombination von x mit Zeugen von y verändert die Länge nicht!
- $x \in L$, genau dann wenn ein Zeuge z existiert, s.d. $x' \in V$, wobei x' das x aufgefüllt mit dem Zeugen z ist.

⇒ Um zu testen, ob x der Länge n in L ist, reicht es aus, Strings der Länge n in V zu testen

Simulation von Turingmaschinen mit Booleschen Schaltkreisen

Wir stellen eine Verbindung zwischen
 $TIWI(n, \log n)$ und $LOGCIRCUITSAT$ her

Simulation von Turingmaschinen mit Booleschen Schaltkreisen

Wir stellen eine Verbindung zwischen
 $\text{TIWI}(n, \log n)$ und LOGCIRCUITSAT her

Satz

*Jedes $L \in \text{TIWI}(n, \log n)$ ist reduzierbar auf
logarithmisch viele Instanzen von LOGCIRCUITSAT
in essentiell-linearer Zeit durch eine Turingmaschine*

Simulation von Turingmaschinen mit Booleschen Schaltkreisen

Wir stellen eine Verbindung zwischen $\text{TIWI}(n, \log n)$ und LOGCIRCUITSAT her

Satz

Jedes $L \in \text{TIWI}(n, \log n)$ ist reduzierbar auf logarithmisch viele Instanzen von LOGCIRCUITSAT in essentiell-linearer Zeit durch eine Turingmaschine

Lemma (Ohne Beweis)

Sei eine zeitkonstruierbare Funktion t gegeben. Wenn $L \in \text{TIME}(t(n))$, dann können wir Boolesche Schaltkreise für L der Größe $\mathcal{O}(t(n) \log t(n))$ in Zeit $\mathcal{O}(t(n) \cdot \text{poly}(\log t(n)))$ berechnen

Simulation von Turingmaschinen mit Booleschen Schaltkreisen

Wir stellen eine Verbindung zwischen $\text{TIWI}(n, \log n)$ und LOGCIRCUITSAT her

Satz

Jedes $L \in \text{TIWI}(n, \log n)$ ist reduzierbar auf logarithmisch viele Instanzen von LOGCIRCUITSAT in essentiell-linearer Zeit durch eine Turingmaschine

Lemma (Ohne Beweis)

Sei eine zeitkonstruierbare Funktion t gegeben. Wenn $L \in \text{TIME}(t(n))$, dann können wir Boolesche Schaltkreise für L der Größe $\mathcal{O}(t(n) \log t(n))$ in Zeit $\mathcal{O}(t(n) \cdot \text{poly}(\log t(n)))$ berechnen

Beweis

- Sei $L \in \text{TIWI}(n, \log n)$. Sei $V \in \text{TIME}(n)$ die Verifikationssprache für L .



Simulation von Turingmaschinen mit Booleschen Schaltkreisen

Wir stellen eine Verbindung zwischen $\text{TIWI}(n, \log n)$ und LOGCIRCUITSAT her

Satz

Jedes $L \in \text{TIWI}(n, \log n)$ ist reduzierbar auf logarithmisch viele Instanzen von LOGCIRCUITSAT in essentiell-linearer Zeit durch eine Turingmaschine

Lemma (Ohne Beweis)

Sei eine zeitkonstruierbare Funktion t gegeben. Wenn $L \in \text{TIME}(t(n))$, dann können wir Boolesche Schaltkreise für L der Größe $\mathcal{O}(t(n) \log t(n))$ in Zeit $\mathcal{O}(t(n) \cdot \text{poly}(\log t(n)))$ berechnen

Beweis

- Sei $L \in \text{TIWI}(n, \log n)$. Sei $V \in \text{TIME}(n)$ die Verifikationssprache für L .
- Sei x Input der Länge n

- Aus dem Lemma erhalten wir einen Schaltkreis C für V mit $\mathcal{O}(n \log n)$ Gattern in essentiell-linearer Zeit



Beispiel

$x = \perp \perp 1 0 1$, $n = 5$, $\lfloor \log n \rfloor = 2$

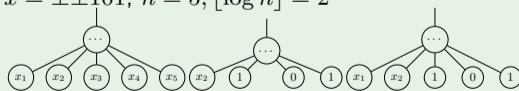


Abbildung: C, C_1, C_2

Simulation von Turingmaschinen mit Booleschen Schaltkreisen

Wir stellen eine Verbindung zwischen $\text{TIWI}(n, \log n)$ und LOGCIRCUITSAT her

Satz

Jedes $L \in \text{TIWI}(n, \log n)$ ist *reduzierbar auf logarithmisch viele Instanzen von LOGCIRCUITSAT in essentiell-linearer Zeit durch eine Turingmaschine*

Lemma (Ohne Beweis)

Sei eine *zeitkonstruierbare Funktion t* gegeben. Wenn $L \in \text{TIME}(t(n))$, dann können wir *Boolesche Schaltkreise für L der Größe $\mathcal{O}(t(n) \log t(n))$ in Zeit $\mathcal{O}(t(n) \cdot \text{poly}(\log t(n)))$ berechnen*

Beweis

- Sei $L \in \text{TIWI}(n, \log n)$. Sei $V \in \text{TIME}(n)$ die Verifikationssprache für L .
- Sei x Input der Länge n

- Aus dem Lemma erhalten wir einen Schaltkreis C für V mit $\mathcal{O}(n \log n)$ Gattern in essentiell-linearer Zeit
- Wir konstruieren Schaltkreisfamilie $\{C_i\}_{i=1}^{\lfloor \log n \rfloor}$ aus C , indem wir
 - x fixieren
 - i Bits für Zeugen freihalten

□

Beispiel

$x = \perp \perp 101$, $n = 5$, $\lfloor \log n \rfloor = 2$

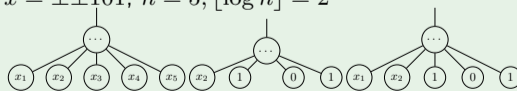


Abbildung: C, C_1, C_2

Simulation von Turingmaschinen mit Booleschen Schaltkreisen

Wir stellen eine Verbindung zwischen $\text{TIWI}(n, \log n)$ und LOGCIRCUITSAT her

Satz

Jedes $L \in \text{TIWI}(n, \log n)$ ist *reduzierbar auf logarithmisch viele Instanzen von LOGCIRCUITSAT in essentiell-linearer Zeit durch eine Turingmaschine*

Lemma (Ohne Beweis)

Sei eine *zeitkonstruierbare Funktion t* gegeben. Wenn $L \in \text{TIME}(t(n))$, dann können wir *Boolesche Schaltkreise für L der Größe $\mathcal{O}(t(n) \log t(n))$ in Zeit $\mathcal{O}(t(n) \cdot \text{poly}(\log t(n)))$ berechnen*

Beweis

- Sei $L \in \text{TIWI}(n, \log n)$. Sei $V \in \text{TIME}(n)$ die Verifikationssprache für L .
- Sei x Input der Länge n

- Aus dem Lemma erhalten wir einen Schaltkreis C für V mit $\mathcal{O}(n \log n)$ Gattern in essentiell-linearer Zeit
- Wir konstruieren Schaltkreisfamilie $\{C_i\}_{i=1}^{\lfloor \log n \rfloor}$ aus C , indem wir
 - x fixieren
 - i Bits für Zeugen freihalten
- C_i hat $\leq \log n$ Inputs und höchstens $\mathcal{O}(n \log n)$ Gatter

□

Beispiel

$x = \perp \perp 101$, $n = 5$, $\lfloor \log n \rfloor = 2$

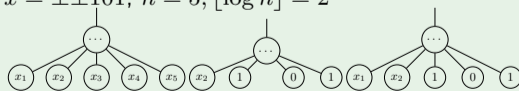


Abbildung: C, C_1, C_2

Simulation von Turingmaschinen mit Booleschen Schaltkreisen

Wir stellen eine Verbindung zwischen $\text{TIWI}(n, \log n)$ und LOGCIRCUITSAT her

Satz

Jedes $L \in \text{TIWI}(n, \log n)$ ist *reduzierbar auf logarithmisch viele Instanzen von LOGCIRCUITSAT in essentiell-linearer Zeit durch eine Turingmaschine*

Lemma (Ohne Beweis)

Sei eine *zeitkonstruierbare Funktion t* gegeben. Wenn $L \in \text{TIME}(t(n))$, dann können wir *Boolesche Schaltkreise für L der Größe $\mathcal{O}(t(n) \log t(n))$ in Zeit $\mathcal{O}(t(n) \cdot \text{poly}(\log t(n)))$ berechnen*

Beweis

- Sei $L \in \text{TIWI}(n, \log n)$. Sei $V \in \text{TIME}(n)$ die Verifikationssprache für L .
- Sei x Input der Länge n

- Aus dem Lemma erhalten wir einen Schaltkreis C für V mit $\mathcal{O}(n \log n)$ Gattern in essentiell-linearer Zeit
- Wir konstruieren Schaltkreisfamilie $\{C_i\}_{i=1}^{\lfloor \log n \rfloor}$ aus C , indem wir
 - x fixieren
 - i Bits für Zeugen freihalten
- C_i hat $\leq \log n$ Inputs und höchstens $\mathcal{O}(n \log n)$ Gatter
- Es gilt $x \in L \Leftrightarrow \exists i \in \{1, \dots, \lfloor \log n \rfloor\}$ s.d. C_i erfüllbar

□

Beispiel

$x = \perp \perp 101$, $n = 5$, $\lfloor \log n \rfloor = 2$

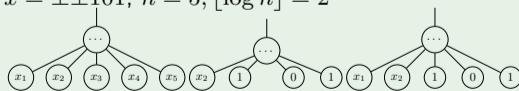


Abbildung: C, C_1, C_2

- **Ziel:** Zeige folgenden Satz

Satz

Wenn $\text{LOGCIRCUITSAT} \in \text{TIME}(n^\alpha)$ für alle $\alpha > 1$, dann ist ETH falsch.

- **Ziel:** Zeige folgenden Satz

Satz

Wenn $\text{LOGCIRCUITSAT} \in \text{TIME}(n^\alpha)$ für alle $\alpha > 1$, dann ist *ETH falsch*.

- Für den Beweis benutzen wir folgendes Theorem

Satz

Wenn für alle $\alpha > 1$ $\text{LOGCIRCUITSAT} \in \text{TIME}(n^\alpha)$, dann

$$(\forall \varepsilon > 0) \text{CIRCUITSAT} \in \text{TIME}(\text{poly}(n)2^{\varepsilon m}),$$

wobei m die Anzahl der Gatter ist.

- **Ziel:** Zeige folgenden Satz

Satz

Wenn $\text{LOGCIRCUITSAT} \in \text{TIME}(n^\alpha)$ für alle $\alpha > 1$, dann ist *ETH* falsch.

- Für den Beweis benutzen wir folgendes Theorem

Satz

Wenn für alle $\alpha > 1$ $\text{LOGCIRCUITSAT} \in \text{TIME}(n^\alpha)$, dann

$$(\forall \varepsilon > 0) \text{CIRCUITSAT} \in \text{TIME}(\text{poly}(n)2^{\varepsilon m}),$$

wobei m die Anzahl der Gatter ist.

Beweis.

- SAT ist Spezialfall von CIRCUITSAT
- Aus dem Satz folgt:

$$(\forall \varepsilon > 0) \text{SAT} \in \text{TIME}(\text{poly}(n)2^{\varepsilon m}),$$

wobei m die Anzahl der Gatter ist

ETH und LOGCIRCUITSAT I

- **Ziel:** Zeige folgenden Satz

Satz

Wenn $\text{LOGCIRCUITSAT} \in \text{TIME}(n^\alpha)$ für alle $\alpha > 1$, dann ist **ETH falsch**.

- Für den Beweis benutzen wir folgendes Theorem

Satz

Wenn für alle $\alpha > 1$ $\text{LOGCIRCUITSAT} \in \text{TIME}(n^\alpha)$, dann

$$(\forall \varepsilon > 0) \text{CIRCUITSAT} \in \text{TIME}(\text{poly}(n)2^{\varepsilon m}),$$

wobei m die Anzahl der Gatter ist.

Beweis.

- SAT ist Spezialfall von CIRCUITSAT
- Aus dem Satz folgt:

$$(\forall \varepsilon > 0) \text{SAT} \in \text{TIME}(\text{poly}(n)2^{\varepsilon m}),$$

wobei m die Anzahl der Gatter ist

- Anwendung des Sparsification Lemma liefert

$$(\forall \varepsilon > 0) \text{SAT} \in \text{TIME}(\text{poly}(n)2^{\varepsilon v}),$$

wobei v die Anzahl der Variablen ist.

⇒ **ETH falsch**



Satz (Wdh.)

Wenn für alle $\alpha > 1$ LOGCIRCUITSAT $\in \text{TIME}(n^\alpha)$, dann

$$(\forall \varepsilon > 0) \text{CIRCUITSAT} \in \text{TIME}(\text{poly}(n)2^{\varepsilon m}),$$

wobei m die Anzahl der Gatter ist.

Satz (Wdh.)

Wenn für alle $\alpha > 1$ LOGCIRCUITSAT \in TIME(n^α), dann

$$(\forall \varepsilon > 0) \text{CIRCUITSAT} \in \text{TIME}(\text{poly}(n)2^{\varepsilon m}),$$

wobei m die Anzahl der Gatter ist.

- Für den Beweis verwenden wir das folgende Lemma

Lemma

Sei g eine berechenbare Funktion. Wenn für alle $\alpha > 1$

$$\text{LOGCIRCUITSAT} \in \text{TIME}(n^\alpha),$$

dann gilt

$$(\forall \varepsilon > 0) \text{TIWI}(\text{poly}(n), g(n)) \subseteq \text{TIME}(2^{\varepsilon g(n)}).$$

Satz (Wdh.)

Wenn für alle $\alpha > 1$ LOGCIRCUITSAT \in TIME(n^α), dann

$$(\forall \varepsilon > 0) \text{CIRCUITSAT} \in \text{TIME}(\text{poly}(n)2^{\varepsilon m}),$$

wobei m die Anzahl der Gatter ist.

- Für den Beweis verwenden wir das folgende Lemma

Lemma

Sei g eine berechenbare Funktion. Wenn für alle $\alpha > 1$

$$\text{LOGCIRCUITSAT} \in \text{TIME}(n^\alpha),$$

dann gilt

$$(\forall \varepsilon > 0) \text{TIWI}(\text{poly}(n), g(n)) \subseteq \text{TIME}(2^{\varepsilon g(n)}).$$

Beweis.

- Angenommen LOGCIRCUITSAT \in TIME(n^α) für alle $\alpha > 1$.
- Sei $\lg(x) := \max\{1, \log x\}$. Wähle $g(n) = \frac{n}{\lg(n)}$.

Satz (Wdh.)

Wenn für alle $\alpha > 1$ LOGCIRCUITSAT \in TIME(n^α), dann

$$(\forall \varepsilon > 0) \text{CIRCUITSAT} \in \text{TIME}(\text{poly}(n)2^{\varepsilon m}),$$

wobei m die Anzahl der Gatter ist.

- Für den Beweis verwenden wir das folgende Lemma

Lemma

Sei g eine berechenbare Funktion. Wenn für alle $\alpha > 1$

$$\text{LOGCIRCUITSAT} \in \text{TIME}(n^\alpha),$$

dann gilt

$$(\forall \varepsilon > 0) \text{TIWI}(\text{poly}(n), g(n)) \subseteq \text{TIME}(2^{\varepsilon g(n)}).$$

Beweis.

- Angenommen LOGCIRCUITSAT \in TIME(n^α) für alle $\alpha > 1$.
- Sei $\lg(x) := \max\{1, \log x\}$. Wähle $g(n) = \frac{n}{\lg(n)}$.
- Anwendung des Lemmas liefert

$$(\forall \varepsilon > 0) \text{TIWI}(\text{poly}(n), \frac{n}{\log n}) \subseteq \text{TIME}(2^{\varepsilon \frac{n}{\log n}})$$

Satz (Wdh.)

Wenn für alle $\alpha > 1$ LOGCIRCUITSAT \in TIME(n^α), dann

$$(\forall \varepsilon > 0) \text{CIRCUITSAT} \in \text{TIME}(\text{poly}(n)2^{\varepsilon m}),$$

wobei m die Anzahl der Gatter ist.

- Für den Beweis verwenden wir das folgende Lemma

Lemma

Sei g eine berechenbare Funktion. Wenn für alle $\alpha > 1$

$$\text{LOGCIRCUITSAT} \in \text{TIME}(n^\alpha),$$

dann gilt

$$(\forall \varepsilon > 0) \text{TIWI}(\text{poly}(n), g(n)) \subseteq \text{TIME}(2^{\varepsilon g(n)}).$$

Beweis.

- Angenommen LOGCIRCUITSAT \in TIME(n^α) für alle $\alpha > 1$.
- Sei $\lg(x) := \max\{1, \log x\}$. Wähle $g(n) = \frac{n}{\lg(n)}$.
- Anwendung des Lemmas liefert

$$(\forall \varepsilon > 0) \text{TIWI}(\text{poly}(n), \frac{n}{\log n}) \subseteq \text{TIME}(2^{\varepsilon \frac{n}{\log n}})$$

- Da $\frac{n}{\log n} = \Theta(m)$ gilt, gilt

$$\text{CIRCUITSAT} \in \text{TIWI}(\text{poly}(n), \frac{n}{\log n})$$

$$\Rightarrow (\forall \varepsilon > 0) \text{CIRCUITSAT} \in \text{TIME}(2^{\varepsilon \frac{n}{\log n}})$$

$$\Rightarrow (\forall \varepsilon > 0) \text{CIRCUITSAT} \in \text{TIME}(\text{poly}(n)2^{\varepsilon m})$$

□

Lemma (Wdh.)

Sei g eine berechenbare Funktion. Wenn für alle $\alpha > 1$

$$\text{LOGCIRCUITSAT} \in \text{TIME}(n^\alpha),$$

dann gilt

$$(\forall \varepsilon > 0) \text{TIWI}(\text{poly}(n), g(n)) \subseteq \text{TIME}(2^{\varepsilon g(n)}).$$

Lemma (Wdh.)

Sei g eine berechenbare Funktion. Wenn für alle $\alpha > 1$

$$\text{LOGCIRCUITSAT} \in \text{TIME}(n^\alpha),$$

dann gilt

$$(\forall \varepsilon > 0) \text{TIWI}(\text{poly}(n), g(n)) \subseteq \text{TIME}(2^{\varepsilon g(n)}).$$

- Das Lemma folgt aus dem Satz über die Simulation von Turingmaschinen mit Booleschen Schaltkreisen und weiteren Resultaten aus dem Paper

Satz (Wdh.)

Jedes $L \in \text{TIWI}(n, \log n)$ ist reduzierbar auf logarithmisch viele Instanzen von LOGCIRCUITSAT in essentiell-linearer Zeit durch eine Turingmaschine

- Die Ergebnisse haben nicht nur Auswirkungen für LOGCIRCUITSAT

Satz

Wenn für alle $\alpha > 1$ gilt, dass $\text{LOGCIRCUITSAT} \in \text{TIME}(n^\alpha)$, dann gilt

$$k\text{-CLIQUE} \in \text{TIME}(n^\alpha)$$

für alle $k \in \mathbb{N}$ und jedes $\alpha > 1$.

Inhaltsverzeichnis

Motivation

Konsequenzen aus ETH für harte Probleme

Konsequenzen aus SETH für einfache Probleme

Superlineare untere Schranken basierend auf SAT




Resümee

- SETH hat Auswirkungen auf einfache Probleme wie REPM

- SETH hat Auswirkungen auf einfache Probleme wie REPM
- ETH hat Auswirkungen auf schwierige Probleme wie DOMINATINGSET oder INDEPENDENTSET

- SETH hat Auswirkungen auf einfache Probleme wie REPM
- ETH hat Auswirkungen auf schwierige Probleme wie DOMINATINGSET oder INDEPENDENTSET
- Überraschenderweise gibt ETH auch superlineare untere Schranken für LOGCIRCUITSAT, k -CLIQUE etc.

-  Abboud, Amir, Virginia Vassilevska Williams und Huacheng Yu (2018). „Matching Triangles and Basing Hardness on an Extremely Popular Conjecture“. In: *SIAM Journal on Computing* 47.3, S. 1098–1122.
-  Bringmann, Karl (2019). „Fine-Grained Complexity Theory (Tutorial)“. In: *36th International Symposium on Theoretical Aspects of Computer Science (STACS 2019)*. Hrsg. von Rolf Niedermeier und Christophe Paul. Bd. 126. Leibniz International Proceedings in Informatics (LIPIcs). Dagstuhl, Germany: Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 4:1–4:7. ISBN: 978-3-95977-100-9. DOI: 10.4230/LIPIcs.STACS.2019.4. URL: <http://drops.dagstuhl.de/opus/volltexte/2019/10243>.
-  Buhrman, Harry, Subhasree Patro und Florian Speelman (2019). *The Quantum Strong Exponential-Time Hypothesis*. arXiv: 1911.05686 [quant-ph].
-  Pilipczuk, Michał (2023). *Parameterized algorithms and Fine-grained complexity*. Presentation EPIT 2023, Île d’Oléron.
-  Salamon, András Z. und Michael Wehar (2022). „Superlinear Lower Bounds Based on ETH“. en. In: Schloss Dagstuhl - Leibniz-Zentrum für Informatik. DOI: 10.4230/LIPIcs.STACS.2022.55. URL: <https://drops.dagstuhl.de/opus/volltexte/2022/15865/>.

-  Williams, Virginia Vassilevska (2015). „Hardness of Easy Problems: Basing Hardness on Popular Conjectures such as the Strong Exponential Time Hypothesis (Invited Talk)“. In: *International Symposium on Parameterized and Exact Computation*.
-  Zeume, Thomas (2022). *Fine-grained complexity theory*. Lecture slides for Computational Complexity Theory, Version from 2022-06-15T11:48.
-  — (2023). *Algorithmen für SAT und die Exponentialzeithypothese*. Vorlesungsfolien für Theoretische Informatik, Version vom 2023-01-24T10:06.

Fragen?

Ausblick - Parametrisierte Algorithmen

- Probleme wie k -Clique sind abhängig von einem Parameter.

Ausblick - Parametrisierte Algorithmen

- Probleme wie k -Clique sind abhängig von einem Parameter.
- ⇒ In der Praxis können Parameter klein sein

Ausblick - Parametrisierte Algorithmen

- Probleme wie k -Clique sind abhängig von einem Parameter.

⇒ In der Praxis können Parameter klein sein

Definition

Ein parametrisiertes Problem ist eine Menge Q von Paaren (x, k) , wobei $x \in \Sigma^*$, $k \in \mathbb{N}$.

P-CLIQUE

- **Geg:** Ungerichteter Graph $G = (V, E)$ und natürliche Zahl $k \in \mathbb{N}$
- **Parameter:** k
- **Frage:** Gib es eine Knotenmenge $U \subseteq V$ so, dass
 - $(u, v) \in E$ oder $u \neq v$, für alle $u, v \in U$
 - $|U| = k$?
- **Frage:** Können wir die Laufzeit vom Parameter "entzerren"?

Ausblick - Parametrisierte Algorithmen

- Probleme wie k -Clique sind abhängig von einem Parameter.

⇒ In der Praxis können Parameter klein sein

Definition

Ein parametrisiertes Problem ist eine Menge Q von Paaren (x, k) , wobei $x \in \Sigma^*$, $k \in \mathbb{N}$.

P-CLIQUE

- **Geg:** Ungerichteter Graph $G = (V, E)$ und natürliche Zahl $k \in \mathbb{N}$
- **Parameter:** k
- **Frage:** Gib es eine Knotenmenge $U \subseteq V$ so, dass
 - $(u, v) \in E$ oder $u \neq v$, für alle $u, v \in U$
 - $|U| = k$?
- **Frage:** Können wir die Laufzeit vom Parameter "entzerren"?

Definition (FPT)

Ein parametrisiertes Problem Q wird fixed-parameter tractable (fpt) genannt, wenn es einen Algorithmus \mathcal{A} , eine Konstante c und eine berechenbare Funktion $f : \mathbb{N} \rightarrow \mathbb{N}$ gibt, so dass

- \mathcal{A} entscheidet Q
- \mathcal{A} läuft in Zeit $\leq f(k)|x|^c$ bei Eingabe (x, k)

Ausblick - Parametrisierte Algorithmen

- Probleme wie k -Clique sind abhängig von einem Parameter.

⇒ In der Praxis können Parameter klein sein

Definition

Ein parametrisiertes Problem ist eine Menge Q von Paaren (x, k) , wobei $x \in \Sigma^*$, $k \in \mathbb{N}$.

P-CLIQUE

- **Geg:** Ungerichteter Graph $G = (V, E)$ und natürliche Zahl $k \in \mathbb{N}$
- **Parameter:** k
- **Frage:** Gib es eine Knotenmenge $U \subseteq V$ so, dass
 - $(u, v) \in E$ oder $u \neq v$, für alle $u, v \in U$
 - $|U| = k$?
- **Frage:** Können wir die Laufzeit vom Parameter "entzerren"?

Definition (FPT)

Ein parametrisiertes Problem Q wird fixed-parameter tractable (fpt) genannt, wenn es einen Algorithmus \mathcal{A} , eine Konstante c und eine berechenbare Funktion $f : \mathbb{N} \rightarrow \mathbb{N}$ gibt, so dass

- \mathcal{A} entscheidet Q
- \mathcal{A} läuft in Zeit $\leq f(k)|x|^c$ bei Eingabe (x, k)
- Untere Schranken für $f(k)$ für FPT-Algorithmen können hergeleitet werden

Satz

Wenn ETH gilt, dann kann P-VERTEXCOVER nicht in Zeit $\hat{O}(2^{o(k)})$ gelöst werden

Ausblick - Parametrisierte Algorithmen

- Probleme wie k -Clique sind abhängig von einem Parameter.

⇒ In der Praxis können Parameter klein sein

Definition

Ein parametrisiertes Problem ist eine Menge Q von Paaren (x, k) , wobei $x \in \Sigma^*$, $k \in \mathbb{N}$.

P-CLIQUE

- **Geg:** Ungerichteter Graph $G = (V, E)$ und natürliche Zahl $k \in \mathbb{N}$
- **Parameter:** k
- **Frage:** Gib es eine Knotenmenge $U \subseteq V$ so, dass
 - $(u, v) \in E$ oder $u \neq v$, für alle $u, v \in U$
 - $|U| = k$?
- **Frage:** Können wir die Laufzeit vom Parameter "entzerren"?

Definition (FPT)

Ein parametrisiertes Problem Q wird fixed-parameter tractable (fpt) genannt, wenn es einen Algorithmus \mathcal{A} , eine Konstante c und eine berechenbare Funktion $f : \mathbb{N} \rightarrow \mathbb{N}$ gibt, so dass

- \mathcal{A} entscheidet Q
- \mathcal{A} läuft in Zeit $\leq f(k)|x|^c$ bei Eingabe (x, k)
- Untere Schranken für $f(k)$ für FPT-Algorithmen können hergeleitet werden

Satz

Wenn ETH gilt, dann kann P-VERTEXCOVER nicht in Zeit $\hat{O}(2^{o(k)})$ gelöst werden

Satz

Wenn ETH gilt, dann kann P-CLIQUE nicht in Zeit $\hat{O}(f(k)n^{o(k)})$ gelöst werden

Definition (Optimierungsprobleme)

- Ein Optimierungs-Minimierungs-Problem O ist ein Tupel (I, S, v) , wobei
 - $I \subseteq \Sigma^*$
 - $S : I \rightarrow \mathfrak{P}(\Sigma^*)$
 - $v : \{(x, y) \mid x \in I, y \in S(x)\} \rightarrow \mathbb{Q}^+$
- $\text{OPT-V}(x) = \min\{v(x, y) \mid y \in S(x)\}$
- $\text{OPT}(x) = \{y \in S(x) \mid v(x, y) = \text{OPT-V}(x)\}$

Definition (Optimierungsprobleme)

- Ein Optimierungs-Minimierungs-Problem O ist ein Tupel (I, S, v) , wobei
 - $I \subseteq \Sigma^*$
 - $S : I \rightarrow \mathfrak{P}(\Sigma^*)$
 - $v : \{(x, y) \mid x \in I, y \in S(x)\} \rightarrow \mathbb{Q}^+$
- $\text{OPT-V}(x) = \min\{v(x, y) \mid y \in S(x)\}$
- $\text{OPT}(x) = \{y \in S(x) \mid v(x, y) = \text{OPT-V}(x)\}$

Definition (Approximationsalgorithmus)

Ein Approximationsalgorithmus für O ist ein Algorithmus \mathcal{A} mit $\mathcal{A}(x) \in S(x)$ für alle $x \in I$

Definition (Optimierungsprobleme)

- Ein Optimierungs-Minimierungs-Problem O ist ein Tupel (I, S, v) , wobei
 - $I \subseteq \Sigma^*$
 - $S : I \rightarrow \mathfrak{P}(\Sigma^*)$
 - $v : \{(x, y) \mid x \in I, y \in S(x)\} \rightarrow \mathbb{Q}^+$
- $\text{OPT-V}(x) = \min\{v(x, y) \mid y \in S(x)\}$
- $\text{OPT}(x) = \{y \in S(x) \mid v(x, y) = \text{OPT-V}(x)\}$

Definition (Approximationsalgorithmus)

Ein Approximationsalgorithmus für O ist ein Algorithmus \mathcal{A} mit $\mathcal{A}(x) \in S(x)$ für alle $x \in I$

Definition (Approximationsgüte)

- Die Approximationsgüte einer Lösung y für die Eingabe x ist $Q(x, y) = \frac{v(x, y)}{\text{OPT-V}(x)}$
- \mathcal{A} ist ein δ -Approximationsalgorithmus, wenn $Q(x, \mathcal{A}(x)) \leq \delta(|x|)$ für alle $x \in I$

Definition (Optimierungsprobleme)

- Ein Optimierungs-Minimierungs-Problem O ist ein Tupel (I, S, v) , wobei
 - $I \subseteq \Sigma^*$
 - $S : I \rightarrow \mathfrak{P}(\Sigma^*)$
 - $v : \{(x, y) \mid x \in I, y \in S(x)\} \rightarrow \mathbb{Q}^+$
- $\text{OPT-V}(x) = \min\{v(x, y) \mid y \in S(x)\}$
- $\text{OPT}(x) = \{y \in S(x) \mid v(x, y) = \text{OPT-V}(x)\}$

Definition (Approximationsalgorithmus)

Ein Approximationsalgorithmus für O ist ein Algorithmus \mathcal{A} mit $\mathcal{A}(x) \in S(x)$ für alle $x \in I$

Definition (Approximationsgüte)

- Die Approximationsgüte einer Lösung y für die Eingabe x ist $Q(x, y) = \frac{v(x, y)}{\text{OPT-V}(x)}$
- \mathcal{A} ist ein δ -Approximationsalgorithmus, wenn $Q(x, \mathcal{A}(x)) \leq \delta(|x|)$ für alle $x \in I$

Definition (PTAS)

Ein Problem $O = (I, S, v)$ ist in PTAS, wenn es einen Algorithmus \mathcal{A} mit Eingaben x und ε gibt, s.d.

- $Q(x, \mathcal{A}(x, \varepsilon)) \leq 1 + \varepsilon$ für alle $x \in I$ und $\varepsilon > 0$
- \mathcal{A} hat polynomielle Laufzeit in $|x|$ für alle festen $\varepsilon > 0$

Definition (Optimierungsprobleme)

- Ein Optimierungs-Minimierungs-Problem O ist ein Tupel (I, S, v) , wobei
 - $I \subseteq \Sigma^*$
 - $S : I \rightarrow \mathfrak{P}(\Sigma^*)$
 - $v : \{(x, y) \mid x \in I, y \in S(x)\} \rightarrow \mathbb{Q}^+$
- $\text{OPT-V}(x) = \min\{v(x, y) \mid y \in S(x)\}$
- $\text{OPT}(x) = \{y \in S(x) \mid v(x, y) = \text{OPT-V}(x)\}$

Definition (Approximationsalgorithmus)

Ein Approximationsalgorithmus für O ist ein Algorithmus \mathcal{A} mit $\mathcal{A}(x) \in S(x)$ für alle $x \in I$

Definition (Approximationsgüte)

- Die Approximationsgüte einer Lösung y für die Eingabe x ist $Q(x, y) = \frac{v(x, y)}{\text{OPT-V}(x)}$
- \mathcal{A} ist ein δ -Approximationsalgorithmus, wenn $Q(x, \mathcal{A}(x)) \leq \delta(|x|)$ für alle $x \in I$

Definition (PTAS)

Ein Problem $O = (I, S, v)$ ist in PTAS, wenn es einen Algorithmus \mathcal{A} mit Eingaben x und ε gibt, s.d.

- $Q(x, \mathcal{A}(x, \varepsilon)) \leq 1 + \varepsilon$ für alle $x \in I$ und $\varepsilon > 0$
- \mathcal{A} hat polynomielle Laufzeit in $|x|$ für alle festen $\varepsilon > 0$

ClosestString

- **Geg:** Strings s_1, \dots, s_k der Länge ℓ , d
- **Frage:** Gibt es einen String s mit Hammingdistanz $\leq d$ für alle i ?

Definition (Optimierungsprobleme)

- Ein Optimierungs-Minimierungs-Problem O ist ein Tupel (I, S, v) , wobei
 - $I \subseteq \Sigma^*$
 - $S : I \rightarrow \mathfrak{P}(\Sigma^*)$
 - $v : \{(x, y) \mid x \in I, y \in S(x)\} \rightarrow \mathbb{Q}^+$
- $\text{OPT-V}(x) = \min\{v(x, y) \mid y \in S(x)\}$
- $\text{OPT}(x) = \{y \in S(x) \mid v(x, y) = \text{OPT-V}(x)\}$

Definition (Approximationsalgorithmus)

Ein Approximationsalgorithmus für O ist ein Algorithmus \mathcal{A} mit $\mathcal{A}(x) \in S(x)$ für alle $x \in I$

Definition (Approximationsgüte)

- Die Approximationsgüte einer Lösung y für die Eingabe x ist $Q(x, y) = \frac{v(x, y)}{\text{OPT-V}(x)}$
- \mathcal{A} ist ein δ -Approximationsalgorithmus, wenn $Q(x, \mathcal{A}(x)) \leq \delta(|x|)$ für alle $x \in I$

Definition (PTAS)

Ein Problem $\mathcal{O} = (I, S, v)$ ist in PTAS, wenn es einen Algorithmus \mathcal{A} mit Eingaben x und ε gibt, s.d.

- $Q(x, \mathcal{A}(x, \varepsilon)) \leq 1 + \varepsilon$ für alle $x \in I$ und $\varepsilon > 0$
- \mathcal{A} hat polynomielle Laufzeit in $|x|$ für alle festen $\varepsilon > 0$

ClosestString

- **Geg:** Strings s_1, \dots, s_k der Länge ℓ , d
- **Frage:** Gibt es einen String s mit Hammingdistanz $\leq d$ für alle i ?

Satz

- *Es gibt einen PTAS-Algorithmus für ClosestString mit Laufzeit $n^{\mathcal{O}(1/\varepsilon^2)}$*
- *Wenn ETH gilt, dann gibt es keinen PTAS-Algorithmus für ClosestString mit Laufzeit $\hat{O}(f(\frac{1}{\varepsilon})n^{o(\log(1/\varepsilon))})$*

Quantum SETH

Basic QSETH

Es gibt keinen fehlerbeschränkten Quantenalgorithmus, der SAT in Zeit $\hat{O}(2^{\frac{n}{2}(1-\varepsilon)} m^{O(1)})$ löst.