



THEORETISCHE INFORMATIK UND LOGIK

9. Vorlesung: NP und NP-Vollständigkeit

Markus Krötzsch

Lehrstuhl Wissensbasierte Systeme

TU Dresden, 10. Mai 2017

Rückblick

PTime und LogSpace als mathematische Modelle für Effizienz:

- PTime als robuste Verallgemeinerung der in linearer Zeit lösbaren Probleme
- LogSpace als typische (vermutlich) subpolynomielle Klasse

Wichtige Anwendung: Reduktionen, die ein Problem mit wenig Aufwand auf ein anderes zurückführen

- **Polynomielle (Many-One-)Reduktionen:** verbreitetste Form von effizienter Reduktion; Notation: \leq_p
- **LogSpace-Reduktionen:** häufig anzutreffen, aber selten im Detail definiert

Warum sollten Reduktionen effizient sein?

Intuition: Eine aufwändige Reduktion kann jedes Problem indirekt lösen, aber dadurch lernt man nichts interessantes über dessen Komplexität.

Warum sollten Reduktionen effizient sein?

Intuition: Eine aufwändige Reduktion kann jedes Problem indirekt lösen, aber dadurch lernt man nichts interessantes über dessen Komplexität.

Satz: Sei $L = \{a\}$ eine Sprache über dem Alphabet $\{a\}$. Falls P entscheidbar ist, dann gibt es eine Many-One-Reduktion $P \leq_m L$.

Warum sollten Reduktionen effizient sein?

Intuition: Eine aufwändige Reduktion kann jedes Problem indirekt lösen, aber dadurch lernt man nichts interessantes über dessen Komplexität.

Satz: Sei $L = \{a\}$ eine Sprache über dem Alphabet $\{a\}$. Falls P entscheidbar ist, dann gibt es eine Many-One-Reduktion $P \leq_m L$.

Beweis: Die Reduktion f funktioniert wie folgt:

- Wenn $w \in P$, dann sei $f(w) = a$.
- Andernfalls, wenn $w \notin P$, dann sei $f(w) = aa$. □

Effizient und doch nicht praktikabel

Es gibt Situationen, in denen ein Problem in PTime liegt und dennoch nicht praktisch algorithmisierbar ist.

Satz: Jede endliche Sprache kann in $D\text{TIME}(1)$ erkannt werden und liegt daher insbesondere in PTime und LogSpace.

Effizient und doch nicht praktikabel

Es gibt Situationen, in denen ein Problem in PTime liegt und dennoch nicht praktisch algorithmisierbar ist.

Satz: Jede endliche Sprache kann in $D\text{TIME}(1)$ erkannt werden und liegt daher insbesondere in PTime und LogSpace.

Beispiel: Sei L die Sprache, die alle wahren Aussagen aus der folgenden Menge enthält: $\{„P = NP“, „P \neq NP“\}$. Dann ist $L \in P\text{Time}$ und damit effizient berechenbar.

Effizient und doch nicht praktikabel

Es gibt Situationen, in denen ein Problem in PTime liegt und dennoch nicht praktisch algorithmisierbar ist.

Satz: Jede endliche Sprache kann in $D\text{TIME}(1)$ erkannt werden und liegt daher insbesondere in PTime und LogSpace.

Beispiel: Sei L die Sprache, die alle wahren Aussagen aus der folgenden Menge enthält: $\{„P = NP“, „P \neq NP“\}$. Dann ist $L \in P\text{Time}$ und damit effizient berechenbar.

Erkenntnis: Man kann manchmal die Existenz eines effizienten Algorithmus beweisen, ohne zu wissen, wie er aussehen müsste.

↪ nichtkonstruktiver Beweis

NP

Der Vortrag ohne Worte



Frank Nelson Cole

Rückblick: Polynomielle Verifikatoren

In der Vorlesung Formale Systeme haben wir die folgende Definition kennengelernt:

Ein **polynomieller Verifikator** für eine Sprache $\mathbf{L} \subseteq \Sigma^*$ ist eine polynomiell-zeitbeschränkte, deterministische TM \mathcal{M} , für die gilt:

- \mathcal{M} akzeptiert nur Wörter der Form $w\#z$ mit:
 - $w \in \mathbf{L}$
 - $z \in \Sigma^*$ ist ein **Zertifikat** polynomieller Länge (d.h. für \mathcal{M} gibt es ein Polynom p mit $|z| \leq p(|w|)$)
- Für jedes Wort $w \in \mathbf{L}$ gibt es ein solches Wort $w\#z \in \mathbf{L}(\mathcal{M})$.

Intuition:

- Das Zertifikat z kodiert die Lösung des Problems w , die der Verifikator lediglich nachprüft.
- Zertifikate sollten kurz sein, damit die Prüfung selbst nicht länger dauert als die Lösung des Problems.

Zertifikate werden auch **Nachweis**, **Beweis** oder **Zeuge** genannt

Rückblick: Nachweis-polynomielle Sprachen

Daraus ergibt sich die Definition einer Sprachklasse:

Eine Sprache L ist **nachweis-polynomiell** wenn es für sie einen polynomiellen Verifikator gibt.

Rückblick: Nachweis-polynomielle Sprachen

Daraus ergibt sich die Definition einer Sprachklasse:

Eine Sprache L ist **nachweis-polynomiell** wenn es für sie einen polynomiellen Verifikator gibt.

Beispiel: Die Entscheidung, ob ein gegebener Graph einen Hamilton-Pfad zulässt, ist nachweis-polynomiell. Als Zertifikat dient der entsprechende Pfad.

NP bedeutet „nachweis-polynomiell“

Wir hatten sodann gezeigt:

Satz: Eine Sprache L ist genau dann nachweis-polynomiell wenn $L \in NP$.

Beweisidee:

„ \Rightarrow “ Gibt es einen polynomiellen Verifikator, dann gibt es auch eine polynomiell zeitbeschränkte NTM, die das Zertifikat rät und anschließend verifiziert

„ \Leftarrow “ Gibt es eine polynomiell zeitbeschränkte NTM, so gibt es einen polynomiellen Verifikator, der diese NTM simuliert: das Zertifikat ist ein akzeptierender Lauf □

Weitere Beispiele für Probleme in NP

SAT (aussagenlogische Erfüllbarkeit)

Gegeben: Eine aussagenlogische Formel F

Frage: Gibt es für F eine erfüllende Belegung?

Weitere Beispiele für Probleme in NP

SAT (aussagenlogische Erfüllbarkeit)

Gegeben: Eine aussagenlogische Formel F

Frage: Gibt es für F eine erfüllende Belegung?

Teilmengen-Summe (subset sum)

Gegeben: Eine Menge von Gegenständen $S = \{a_1, \dots, a_n\}$, wobei jedem Gegenstand a_i ein Wert $v(a_i)$ zugeordnet ist; eine gewünschte Zahl z

Frage: Gibt es eine Teilmenge $T \subseteq S$ mit $\sum_{a \in T} v(a) = z$?

Weitere Beispiele für Probleme in NP

SAT (aussagenlogische Erfüllbarkeit)

Gegeben: Eine aussagenlogische Formel F

Frage: Gibt es für F eine erfüllende Belegung?

Teilmengen-Summe (subset sum)

Gegeben: Eine Menge von Gegenständen $S = \{a_1, \dots, a_n\}$, wobei jedem Gegenstand a_i ein Wert $v(a_i)$ zugeordnet ist; eine gewünschte Zahl z

Frage: Gibt es eine Teilmenge $T \subseteq S$ mit $\sum_{a \in T} v(a) = z$?

Zusammengesetzte Zahl (Nicht-Primzahl)

Gegeben: Eine natürliche Zahl $z > 1$

Frage: Gibt es eine natürliche Zahlen $p, q > 1$ mit $p \cdot q = z$?

NP ist nicht symmetrisch

Es ist leicht zu sehen:

Satz: Die Klasse P ist unter Komplement abgeschlossen.

Beweis: Wenn es für L eine polynomiell-zeitbeschränkte TM \mathcal{M} gibt, dann erhält man eine TM für \bar{L} indem man akzeptierende und nicht-akzeptierende Zustände von \mathcal{M} vertauscht. \square

Allgemein gilt: Jede deterministische Komplexitätsklasse ist unter Komplement abgeschlossen.

NP ist nicht symmetrisch

Es ist leicht zu sehen:

Satz: Die Klasse P ist unter Komplement abgeschlossen.

Beweis: Wenn es für L eine polynomiell-zeitbeschränkte TM \mathcal{M} gibt, dann erhält man eine TM für \bar{L} indem man akzeptierende und nicht-akzeptierende Zustände von \mathcal{M} vertauscht. \square

Allgemein gilt: Jede deterministische Komplexitätsklasse ist unter Komplement abgeschlossen.

Für nichtdeterministische Klassen wie NP ist das nicht so einfach:

Beispiel: Es scheint kein einfaches Zertifikat dafür zu geben, dass ein Graph **keinen** Hamiltonpfad hat.

NP ist nicht symmetrisch

Es ist leicht zu sehen:

Satz: Die Klasse P ist unter Komplement abgeschlossen.

Beweis: Wenn es für L eine polynomiell-zeitbeschränkte TM M gibt, dann erhält man eine TM für \bar{L} indem man akzeptierende und nicht-akzeptierende Zustände von M vertauscht. \square

Allgemein gilt: Jede deterministische Komplexitätsklasse ist unter Komplement abgeschlossen.

Für nichtdeterministische Klassen wie NP ist das nicht so einfach:

Beispiel: Es scheint kein einfaches Zertifikat dafür zu geben, dass ein Graph **keinen** Hamiltonpfad hat.

Die Klasse aller Sprachen L , für die $\bar{L} \in NP$ gilt, heißt coNP.

Jede NTM-Klasse kann komplementiert werden: coNL, coNExp, ...

In NP oder nicht?

Vermutung: $\text{coNP} \neq \text{NP}$, d.h. Komplemente von „typischen“ Problemen in NP sind nicht in NP.

Aber: Es gibt viele Probleme in $\text{coNP} \cap \text{NP}$. Zum Beispiel ist $P \subseteq \text{coNP} \cap \text{NP}$.

In NP oder nicht?

Vermutung: $\text{coNP} \neq \text{NP}$, d.h. Komplemente von „typischen“ Problemen in NP sind nicht in NP.

Aber: Es gibt viele Probleme in $\text{coNP} \cap \text{NP}$. Zum Beispiel ist $P \subseteq \text{coNP} \cap \text{NP}$.

Primzahl (= Zusammengesetzte Zahl)

Gegeben: Eine natürliche Zahl $z > 1$

Frage: Gibt es eine keine natürliche Zahlen $p, q > 1$ mit $p \cdot q = z$?

In NP oder nicht?

Vermutung: $\text{coNP} \neq \text{NP}$, d.h. Komplemente von „typischen“ Problemen in NP sind nicht in NP.

Aber: Es gibt viele Probleme in $\text{coNP} \cap \text{NP}$. Zum Beispiel ist $P \subseteq \text{coNP} \cap \text{NP}$.

Primzahl (= Zusammengesetzte Zahl)

Gegeben: Eine natürliche Zahl $z > 1$

Frage: Gibt es eine keine natürliche Zahlen $p, q > 1$ mit $p \cdot q = z$?

Seit 1975 ist bekannt: **Primzahl** $\in \text{NP}$, also **Primzahl** $\in \text{NP} \cap \text{coNP}$
(Zertifikat: „Primality certificate“)

In NP oder nicht?

Vermutung: $\text{coNP} \neq \text{NP}$, d.h. Komplemente von „typischen“ Problemen in NP sind nicht in NP.

Aber: Es gibt viele Probleme in $\text{coNP} \cap \text{NP}$. Zum Beispiel ist $P \subseteq \text{coNP} \cap \text{NP}$.

Primzahl (= Zusammengesetzte Zahl)

Gegeben: Eine natürliche Zahl $z > 1$

Frage: Gibt es eine keine natürliche Zahlen $p, q > 1$ mit $p \cdot q = z$?

Seit 1975 ist bekannt: **Primzahl** $\in \text{NP}$, also **Primzahl** $\in \text{NP} \cap \text{coNP}$
(Zertifikat: „Primality certificate“)

Seit 2002 ist bekannt: **Primzahl** $\in P$
(Primzahlentest nach Agrawal, Kayal und Saxena)

Randbemerkung: Ist Kryptografie sicher?

Das Wirkprinzip asymmetrischer Verschlüsselungsverfahren:

- Es ist **leicht**, zwei Zahlen zu multiplizieren
- Es ist **schwer**, eine Zahl in ihre Faktoren zu zerlegen

Aber seit 2002 wissen wir: Man kann in polynomieller Zeit entscheiden, ob es Faktoren mit $p \cdot q = z$ gibt.

Haben Agrawal, Kayal und Saxena die Kryptografie geknackt?

Randbemerkung: Ist Kryptografie sicher?

Das Wirkprinzip asymmetrischer Verschlüsselungsverfahren:

- Es ist **leicht**, zwei Zahlen zu multiplizieren
- Es ist **schwer**, eine Zahl in ihre Faktoren zu zerlegen

Aber seit 2002 wissen wir: Man kann in polynomieller Zeit entscheiden, ob es Faktoren mit $p \cdot q = z$ gibt.

Haben Agrawal, Kayal und Saxena die Kryptografie geknackt?

Nein:

- Es ist **leicht** zu entscheiden, ob eine Zahl echte Faktoren hat
- Aber es ist dennoch **schwer** die Faktoren zu bestimmen (glauben wir zumindest . . .)

Faktorisierung

(Nicht)Existenz von Faktoren (**Primzahl**) erfasst nicht wirklich, wie schwer Faktorisierung ist

~> Wie kann man das komplexitätstheoretisch ausdrücken?

Faktorisierung

(Nicht)Existenz von Faktoren (**Primzahl**) erfasst nicht wirklich, wie schwer Faktorisierung ist

↪ Wie kann man das komplexitätstheoretisch ausdrücken?

Faktor-7

Gegeben: Eine natürliche Zahl $z > 1$

Frage: Hat z einen Primfaktor, der mit der Ziffer 7 endet?

Faktorisierung

(Nicht)Existenz von Faktoren (**Primzahl**) erfasst nicht wirklich, wie schwer Faktorisierung ist

→ Wie kann man das komplexitätstheoretisch ausdrücken?

Faktor-7

Gegeben: Eine natürliche Zahl $z > 1$

Frage: Hat z einen Primfaktor, der mit der Ziffer 7 endet?

Ein interessantes Entscheidungsproblem:

- **Faktor-7** erfordert (vermutlich) die Kenntnis der Primfaktoren, nicht nur die Bestimmung deren Existenz

Faktorisierung

(Nicht)Existenz von Faktoren (**Primzahl**) erfasst nicht wirklich, wie schwer Faktorisierung ist

→ Wie kann man das komplexitätstheoretisch ausdrücken?

Faktor-7

Gegeben: Eine natürliche Zahl $z > 1$

Frage: Hat z einen Primfaktor, der mit der Ziffer 7 endet?

Ein interessantes Entscheidungsproblem:

- **Faktor-7** erfordert (vermutlich) die Kenntnis der Primfaktoren, nicht nur die Bestimmung deren Existenz
- **Faktor-7** \in NP

Faktorisierung

(Nicht)Existenz von Faktoren (**Primzahl**) erfasst nicht wirklich, wie schwer Faktorisierung ist

↪ Wie kann man das komplexitätstheoretisch ausdrücken?

Faktor-7

Gegeben: Eine natürliche Zahl $z > 1$

Frage: Hat z einen Primfaktor, der mit der Ziffer 7 endet?

Ein interessantes Entscheidungsproblem:

- **Faktor-7** erfordert (vermutlich) die Kenntnis der Primfaktoren, nicht nur die Bestimmung deren Existenz
- **Faktor-7** \in NP: Zertifikat ist die Liste aller Primfaktoren*

* Kontrollfrage: Wieso kann man polynomiell verifizieren, dass dies wirklich Primfaktoren sind?

Faktorisierung

(Nicht)Existenz von Faktoren (**Primzahl**) erfasst nicht wirklich, wie schwer Faktorisierung ist

→ Wie kann man das komplexitätstheoretisch ausdrücken?

Faktor-7

Gegeben: Eine natürliche Zahl $z > 1$

Frage: Hat z einen Primfaktor, der mit der Ziffer 7 endet?

Ein interessantes Entscheidungsproblem:

- **Faktor-7** erfordert (vermutlich) die Kenntnis der Primfaktoren, nicht nur die Bestimmung deren Existenz
- **Faktor-7** \in NP: Zertifikat ist die Liste aller Primfaktoren*
- **Faktor-7** \in coNP

* Kontrollfrage: Wieso kann man polynomiell verifizieren, dass dies wirklich Primfaktoren sind?

Faktorisierung

(Nicht)Existenz von Faktoren (**Primzahl**) erfasst nicht wirklich, wie schwer Faktorisierung ist

→ Wie kann man das komplexitätstheoretisch ausdrücken?

Faktor-7

Gegeben: Eine natürliche Zahl $z > 1$

Frage: Hat z einen Primfaktor, der mit der Ziffer 7 endet?

Ein interessantes Entscheidungsproblem:

- **Faktor-7** erfordert (vermutlich) die Kenntnis der Primfaktoren, nicht nur die Bestimmung deren Existenz
- **Faktor-7** \in NP: Zertifikat ist die Liste aller Primfaktoren*
- **Faktor-7** \in coNP: Zertifikat ist die Liste aller Primfaktoren*

* Kontrollfrage: Wieso kann man polynomiell verifizieren, dass dies wirklich Primfaktoren sind?

Faktorisierung

(Nicht)Existenz von Faktoren (**Primzahl**) erfasst nicht wirklich, wie schwer Faktorisierung ist

→ Wie kann man das komplexitätstheoretisch ausdrücken?

Faktor-7

Gegeben: Eine natürliche Zahl $z > 1$

Frage: Hat z einen Primfaktor, der mit der Ziffer 7 endet?

Ein interessantes Entscheidungsproblem:

- **Faktor-7** erfordert (vermutlich) die Kenntnis der Primfaktoren, nicht nur die Bestimmung deren Existenz
- **Faktor-7** \in NP: Zertifikat ist die Liste aller Primfaktoren*
- **Faktor-7** \in coNP: Zertifikat ist die Liste aller Primfaktoren*
- Aber niemand konnte bisher zeigen, dass **Faktor-7** \in P
vermutlich gilt **Faktor-7** \notin P

* Kontrollfrage: Wieso kann man polynomiell verifizieren, dass dies wirklich Primfaktoren sind?

NP-Vollständigkeit

Probleme vergleichen mit Reduktionen

Intuition:

$$P \leq_p Q$$

bedeutet

„**Q** ist mindestens genauso schwer wie **P**“

- Einordnung in Komplexitätsklassen: **obere Schranke**
(„**Q** ist mit höchstens polynomiellen Aufwand lösbar“)
- Reduktion auf andere Probleme: **untere Schranke**
(„**Q** benötigt mindestens so viel Aufwand wie **P**“)

NP-Härte und NP-Vollständigkeit

Eine Sprache ist

- **NP-hart**, wenn jede Sprache in NP polynomiell darauf reduzierbar ist
- **NP-vollständig**, wenn sie NP-hart ist und in NP liegt

NP-Härte und NP-Vollständigkeit

Eine Sprache ist

- **NP-hart**, wenn jede Sprache in NP polynomiell darauf reduzierbar ist
- **NP-vollständig**, wenn sie NP-hart ist und in NP liegt

Beispiel: **SAT** ist NP-vollständig (Cook & Levin).

Beispiel: **Primzahl** ist in NP, aber **vermutlich** nicht NP-hart. Gleiches gilt für viele Probleme in P (bei einigen ist dagegen sicher, dass sie nicht NP-hart sind).

Beispiel: Das Halteproblem ist NP-hart aber sicher nicht in NP. Gleiches gilt für jedes unentscheidbare Problem.

Das erste NP-vollständige Problem

Der Beweis der NP-Vollständigkeit von **SAT** war ein wichtiger Durchbruch (siehe auch Vorlesung Formale Systeme)

Beweisidee:

- Durch direkte Reduktion auf das Wortproblem von polynomiell zeitbeschränkten NTMS (dies ist eigentlich „das erste“ NP-vollständige Problem!)
- Wir verwenden so viele aussagenlogische Variablen, dass sie jeden polynomiellen Lauf kodieren könnten (d.h. alle Speicherstellen, Zustände und Positionen in jedem Schritt!)
- Wir verwenden Formeln, so dass jede erfüllende Belegung einen korrekten, akzeptierenden Lauf kodiert

Das erste NP-vollständige Problem

Der Beweis der NP-Vollständigkeit von **SAT** war ein wichtiger Durchbruch (siehe auch Vorlesung Formale Systeme)

Beweisidee:

- Durch direkte Reduktion auf das Wortproblem von polynomiell zeitbeschränkten NTMs (dies ist eigentlich „das erste“ NP-vollständige Problem!)
- Wir verwenden so viele aussagenlogische Variablen, dass sie jeden polynomiellen Lauf kodieren könnten (d.h. alle Speicherstellen, Zustände und Positionen in jedem Schritt!)
- Wir verwenden Formeln, so dass jede erfüllende Belegung einen korrekten, akzeptierenden Lauf kodiert

„a whole buttload of variables [and] a shitload of logical relations between them“

– Scott Aaronson, Quantum Computing since Democritus

Weitere NP-vollständige Probleme

Um zu zeigen, dass ein Problem **P** NP-vollständig ist, genügen die folgenden beiden Schritte:

- (1) Zeige, dass **P** \in NP
- (2) Finde ein bereits bekanntes NP-vollständiges Problem **Q** und zeige **Q** \leq_p **P**

Seit Cook und Levin (frühe 1970-er) wurden tausende von NP-vollständigen Problemen gefunden

Wir werden jetzt einige Beispiele sehen

Clique

Eine **Clique** ist ein Graph, bei dem jeder Knoten mit jedem anderen direkt durch eine Kante verbunden ist

Clique

Gegeben: Ein Graph G und eine Zahl k

Frage: Enthält G eine Clique mit k Knoten?

Clique

Eine **Clique** ist ein Graph, bei dem jeder Knoten mit jedem anderen direkt durch eine Kante verbunden ist

Clique

Gegeben: Ein Graph G und eine Zahl k

Frage: Enthält G eine Clique mit k Knoten?

Satz: **Clique** ist NP-vollständig.

Clique

Eine **Clique** ist ein Graph, bei dem jeder Knoten mit jedem anderen direkt durch eine Kante verbunden ist

Clique

Gegeben: Ein Graph G und eine Zahl k

Frage: Enthält G eine Clique mit k Knoten?

Satz: **Clique** ist NP-vollständig.

Beweis:

(1) **Clique** \in NP: Die Clique selbst ist ein geeignetes Zertifikat.

(Kontrollfrage: ist dieses Zertifikat wirklich polynomiell? Ist k in Binärkodierung gegeben, dann ist diese Eingabe schließlich nur $\log(k)$ Zeichen lang ...)

Clique

Eine **Clique** ist ein Graph, bei dem jeder Knoten mit jedem anderen direkt durch eine Kante verbunden ist

Clique

Gegeben: Ein Graph G und eine Zahl k

Frage: Enthält G eine Clique mit k Knoten?

Satz: **Clique** ist NP-vollständig.

Beweis:

(1) **Clique** \in NP: Die Clique selbst ist ein geeignetes Zertifikat.

(Kontrollfrage: ist dieses Zertifikat wirklich polynomiell? Ist k in Binärkodierung gegeben, dann ist diese Eingabe schließlich nur $\log(k)$ Zeichen lang ...)

(2) Clique ist NP-hart. Dazu konstruieren wir eine Reduktion von **SAT**.

Clique: Härte (1)

Beweis Teil (2): Clique ist NP-hart.

- Sei F eine aussagenlogische Formel in CNF:

$$F = \left((L_1^1 \vee \dots \vee L_{n_1}^1) \wedge \dots \wedge (L_1^\ell \vee \dots \vee L_{n_\ell}^\ell) \right)$$

Clique: Härte (1)

Beweis Teil (2): Clique ist NP-hart.

- Sei F eine aussagenlogische Formel in CNF:
$$F = ((L_1^1 \vee \dots \vee L_{n_1}^1) \wedge \dots \wedge (L_1^\ell \vee \dots \vee L_{n_\ell}^\ell))$$
- Wir definieren einen Graphen G_F , so dass gilt:

G_F hat eine Clique der Größe ℓ gdw. F ist erfüllbar

Clique: Härte (1)

Beweis Teil (2): Clique ist NP-hart.

- Sei F eine aussagenlogische Formel in CNF:

$$F = ((L_1^1 \vee \dots \vee L_{n_1}^1) \wedge \dots \wedge (L_1^\ell \vee \dots \vee L_{n_\ell}^\ell))$$

- Wir definieren einen Graphen G_F , so dass gilt:

G_F hat eine Clique der Größe ℓ gdw. F ist erfüllbar

- **Knoten von G_F :** die Paare $\langle L_j^i, i \rangle$, für alle $i \in \{1, \dots, \ell\}$ und $j \in \{1, \dots, n_i\}$

Clique: Härte (1)

Beweis Teil (2): Clique ist NP-hart.

- Sei F eine aussagenlogische Formel in CNF:

$$F = ((L_1^1 \vee \dots \vee L_{n_1}^1) \wedge \dots \wedge (L_1^\ell \vee \dots \vee L_{n_\ell}^\ell))$$

- Wir definieren einen Graphen G_F , so dass gilt:

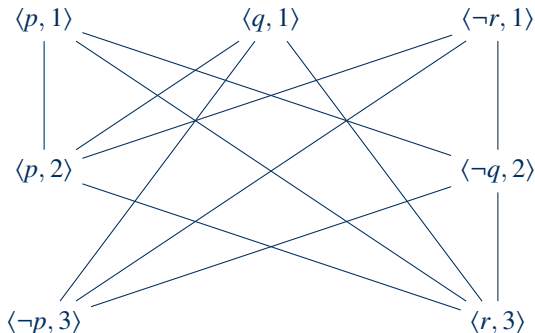
G_F hat eine Clique der Größe ℓ gdw. F ist erfüllbar

- **Knoten von G_F :** die Paare $\langle L_j^i, i \rangle$, für alle $i \in \{1, \dots, \ell\}$ und $j \in \{1, \dots, n_i\}$
- **Kanten von G_F :** alle Paare $\langle L, i \rangle - \langle L', j \rangle$, für die gilt:
 - (1) $i \neq j$ und
 - (2) $L \wedge L'$ ist erfüllbar, d.h. $L \neq \neg L'$ und $L' \neq \neg L$

Offensichtlich kann man G_F in polynomieller Zeit berechnen

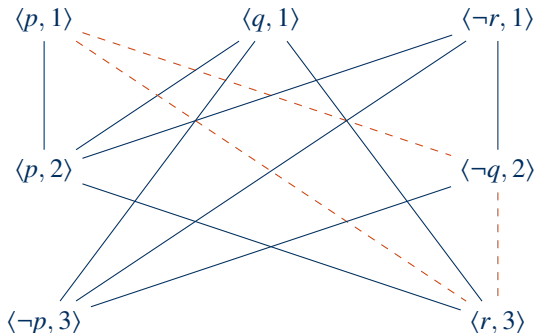
Clique: Härte (2)

Beispiel: $F = ((p \vee q \vee \neg r) \wedge (p \vee \neg q) \wedge (\neg p \vee r))$



Clique: Härte (2)

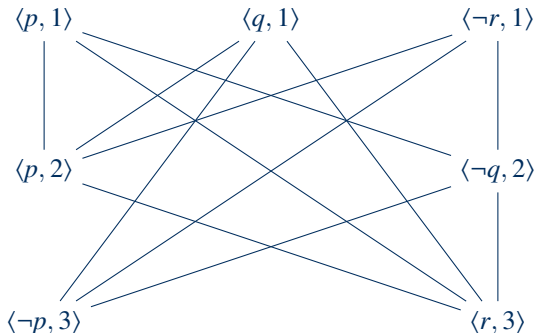
Beispiel: $F = ((p \vee q \vee \neg r) \wedge (p \vee \neg q) \wedge (\neg p \vee r))$



Erfüllende Belegung: $p \mapsto 1, q \mapsto 0, r \mapsto 1$

Clique: Härte (2)

Beispiel: $F = ((p \vee q \vee \neg r) \wedge (p \vee \neg q) \wedge (\neg p \vee r))$



Erfüllende Belegung: $p \mapsto 1, q \mapsto 0, r \mapsto 1$

Man sieht leicht, dass unsere Reduktion korrekt ist. □

Unabhängige Mengen

Eine **unabhängige Menge** ist eine Teilmenge von Knoten in einem Graph, bei der kein Knoten mit einem anderen direkt verbunden ist

Unabhängige Menge

Gegeben: Ein Graph G und eine Zahl k

Frage: Enthält G eine unabhängige Menge mit k Knoten?

Unabhängige Mengen

Eine **unabhängige Menge** ist eine Teilmenge von Knoten in einem Graph, bei der kein Knoten mit einem anderen direkt verbunden ist

Unabhängige Menge

Gegeben: Ein Graph G und eine Zahl k

Frage: Enthält G eine unabhängige Menge mit k Knoten?

Satz: Unabhängige Menge ist NP-vollständig.

Unabhängige Mengen

Eine **unabhängige Menge** ist eine Teilmenge von Knoten in einem Graph, bei der kein Knoten mit einem anderen direkt verbunden ist

Unabhängige Menge

Gegeben: Ein Graph G und eine Zahl k

Frage: Enthält G eine unabhängige Menge mit k Knoten?

Satz: **Unabhängige Menge** ist NP-vollständig.

Beweis: Die Reduktion auf **Clique** ist sehr einfach:

- Ein Graph hat eine unabhängige Menge der Größe k genau dann wenn
- sein Komplementgraph eine Clique der Größe k hat

↪ Komplementierung ist polynomiell

□

Zusammenfassung und Ausblick

NP entspricht der Klasse der nachweis-polynomiellen Probleme

Die Klasse der Komplemente von NP-Problemen ist coNP

Polynomielle Reduktionen erlauben uns, die Schwere von Problemen zu vergleichen

Es gibt sehr viele bekannte NP-vollständige Probleme

Was erwartet uns als nächstes?

- Mehr NP
- Pseudopolynomielle Probleme
- Komplexität jenseits von NP