Ladner's Theorem

Seminararbeit
am FG Theoretische Informatik / Formale Methoden
der Universität Kassel

Alexander Elia Rode
Philip Laskowicz

Sommersemester 2022

Zusammenfassung

Die Frage, ob P = NP oder P $\neq$ NP gilt, ist in der theoretischen Informatik ein bislang nicht gelöstes Problem. Unter der Annahme, dass P $\neq$ NP gilt, hat Ladner gezeigt, dass in der Komplexitätsklasse NP Probleme existieren, die weder in P liegen noch NP vollständig sind. Diese Probleme werden als NP-intermediate bezeichnet. Diese Ausarbeitung erläutert den Sachverhalt von Ladner's Theorem und beweist die Existenz von NP-intermediate Problemen. Der Beweis folgt dem Textbuch „Computational Complexity: A Modern Approach" von Sanjeev Arora und Boaz Barak. In dem Beweis wird ein Problem SAT_{H} konstruiert, indem das Problem SAT um ein bestimmtes Padding ergänzt wird, sodass anschließend gezeigt werden kann, dass das so konstruierte Problem weder in P liegt noch NP-vollständig ist.

1 Einführung

Die Komplexitätsklassen P und NP haben in der Komplexitätstheorie eine wichtige bedeutung. Die Komplexitätsklasse P beinhaltet dabei die effizient lösbaren Probleme und die Komplexitätsklasse NP die effizient verifizierbaren Probleme. In diesem Fall bedeutet effizient in polynomieller Zeit mittels einer deterministischen Turingmaschine. Zum Zeitpunkt der Ausarbeitung ist die Frage, ob P = NP oder P $\neq$ NP gilt, nicht gelöst. Die folgende Ausarbeitung beschäftigt sich mit Ladner's Theorem. Dieses Theorem untersucht den Zusammenhang zwischen P und NP und zeigt eine interessante Folgerung für den Fall, dass P $\neq$ NP gilt. Unter der Annahme, dass P $\neq$ NP gilt, zeigt Ladner's Theorem, dass in der Komplexitätsklasse NP nicht nur Probleme existieren, die in P liegen oder NP-vollständig sind. Ein Problem, dass die Eigenschaft besitzt in NP zu liegen und weder in P noch NP vollständig zu sein, wird in dieser Ausarbeitung konstruiert. Dieses Problem wird als SAT_H bezeichnet, welches durch gepaddete SAT Instanzen konstruiert wird. Durch eine Funktion H, welche die Länge eines Paddings einer SAT-Instanz bestimmt, kann die Komplexität von SAT_H beeinflusst werden. Das Ziel der Konstruktion ist die Länge des Paddings so zu wählen, dass SAT_H weder in polynomieller Zeit lösbar ist noch, dass SAT_H weiterhin genauso schwer zu lösen ist, wie das Erfüllbarkeitsproblem für aussagelogische Formeln SAT. In der folgenden Ausarbeitung wird im zweiten Abschnitt die für den Beweis notwendigen Definitionen und Konzepte erläutert. Im Abschnitt 3 wird aufgezeigt, was NP-intermediate Probleme sind. Zudem wird eine Leitfrage aufgestellt, welche in der Ausarbeitung beantwortet wird. Anschließend wird im Abschnitt 4 das als SAT_H bezeichnete Problem konstruiert, welches Ladners Theorem beweist und die Existenz von NP-intermediate Problemen bestätigt. Abschließend werden im letzten Abschnitt die Erkenntnisse der Ausarbeitung zusammengefasst und ein Ausblick auf sich ergebende Fragen gegeben.

2 Konzepte und Grundlagen

Die Komplexitätsklasse P beinhaltet die Probleme, die von einer deterministischen Turingmaschine in polynomieller Zeit entschieden werden können. Die Komplexitätsklasse NP beinhaltet die Probleme, die von einer nicht-deterministischen Turingmaschine in polynomieller Zeit entschieden werden können.

Durch eine Polynomialzeitreduktion lässt sich ein Probleme L auf L' polynomiell reduzieren, wenn es eine in polynomieller Zeit berechenbare Funktion f gibt, sodass für alle $w \in \Sigma^*$ gilt: $w \in L \leftrightarrow f(w) \in L'$. Ein Problem L wird genau dann als NP-schwer bezeichnet, wenn eine Polynomialzeitreduktion für alle Probleme aus NP auf L existiert. Wenn das Problem L zusätzlich zur NP-schwere in NP liegt, wird das Problem als NP-vollständig bezeichnet. Auffallend ist, dass eine große Anzahl an Problemen NP-vollständig ist. Die Frage, ob P = NP oder ob P $\neq$ NP gilt, ist in der theoretischen Informatik ein nicht gelöstes Problem. Jedoch gilt, falls ein Problem L NP-vollständig ist, dass L genau dann in P liegt, wenn P = NP gilt.

Ein für die Ausarbeitung relevantes NP-vollständiges Problem ist das Erfüllbarkeitsproblem für aussagelogische Formeln SAT. Das SAT Problem behandelt die Frage, ob für eine gegebene aussagelogische Formel ψ eine Belegung derart gefunden werden kann, sodass sich ψ zu wahr auswertet und damit erfüllbar ist. Nach dem Satz von Cook und Levin ist das SAT Problem NP-vollständig. Intuitiv betrachtet kann ein Algorithmus, der das SAT-Problem löst, in exponentieller Zeit konstruiert werden, indem für eine Formel ψ mit n Variablen alle 2^n Variablenbelegungen getestet werden. Ein polynomieller Algorithmus erweist sich als schwieriger zu konstruieren und ist nicht bekannt.

3 Was sind NP-intermediate Probleme?

In der Komplexitätstheorie sind verschiedene Probleme von Bedeutung. Durch die bekannten Probleme in der Komplexitätsklasse P und NP entsteht die Vermutung, dass jedes Problem aus NP in P liegt oder NP-vollständig ist. Mit anderen Worten ausgedrückt, bedeutet das, dass in NP keine Probleme existieren, die weder in P noch NP-vollständig sind. Unter der Annahme, dass P = NP gilt, ist diese Vermutung richtig, da dadurch alle NP-vollständigen Probleme in P liegen und die Komplexitätsklassen P und NP identisch sind, wie das Venn-Diagramm in Abbildung 1 verdeutlicht.

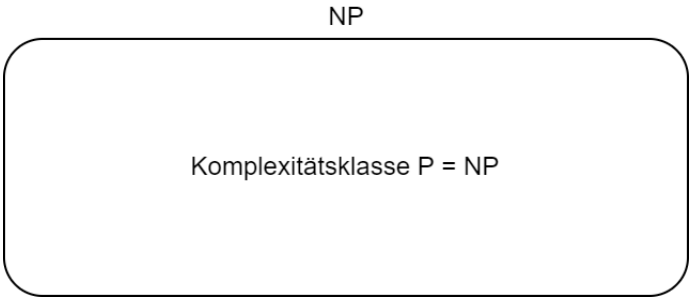

Abbildung 1: Komplexitätsklasse NP unter der Annahme P = NP

Unter der Annahme, dass P $\neq$ NP gilt, ergibt sich ein interessanter Sachverhalt. Die Entscheidung, ob die Vermutung zutrifft, dass keine Probleme in NP\P existieren, die nicht NP-vollständig sind, ist nicht mehr so offensichtlich. So lässt sich folgende Leitfrage für die Ausarbeitung formulieren, die die Ausarbeitung mit Hilfe eines Problems $\mathrm{SAT_H}$ beantwortet: Existieren Probleme in der Komplexitätsklasse NP, die weder in P liegen noch NP-vollständig sind?
Probleme mit diesen Eigenschaften werden als NP-intermediate bezeichnet. Die zugehörige Komplexitätsklasse wird mit NPI abgekürzt.

An dieser Stelle ist jedoch nicht nachgewiesen, dass Probleme mit den genannten Eigenschaften existieren. Das folgende Venn-Diagramm in Abbildung 2 verdeutlicht die oben definierte Fragestellung.

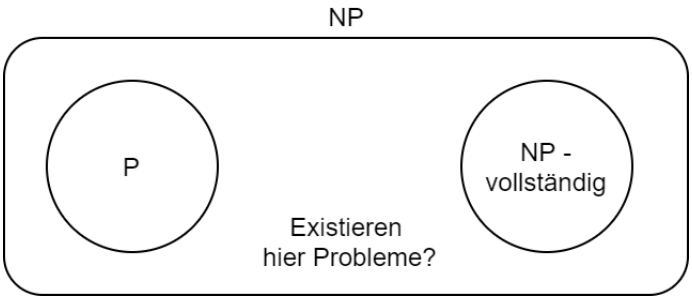

Abbildung 2: Komplexitätsklasse NP unter der Annahme P $\neq$ NP

4 Beweis der Existenz von NP-intermediate Problemen

Zur positiven Beantwortung der Frage, ob NP-intermediate Probleme unter der Annahme P $\neq$ NP existieren, muss ein Problem gefunden werden, das die im letzten Abschnitt genannten Eigenschaften erfüllt. Das bedeutet ein geeignetes Problem muss zum einen in NP\P liegen und zum anderen darf das Problem nicht NP-vollständig sein. Dieser Abschnitt beantwortet die Frage und beweist die Existenz von NP-intermediate Problemen. Der Beweis folgt dem Abschnitt Ladner's Theorem aus „Computational Complexity: A Modern Approach“ von Sanjeev Arora und Boaz Barak.[1]

Wie der vorherige Abschnitt gezeigt hat, ist die Voraussetzung für den Beweis, dass P $\neq$ NP gilt. Daher nehmen wir für diesen Abschnitt an, dass P $\neq$ NP gilt. Für den Beweis selbst wird im Folgenden ein künstliches Problem $\mathrm{SAT_H}$ unter Zuhilfenahme von gepaddeten Instanzen des SAT-Problems konstruiert.

4.1 Idee von Padding

Mit Padding kann die Komplexität von Problemen, die in unserem Fall von der Eingabelänge abhängt, beeinflusst werden. Aus Abschnitt 2 folgt, dass SAT in $O(2^n)$ entscheidbar ist, jedoch nicht in polynomieller Zeit unter der Annahme P $\neq$ NP, da das Erfüllbarkeitsproblem NP-vollständig ist. Durch Padding lassen sich nun die Probleme $\mathrm{SAT_1}$ und $\mathrm{SAT_2}$ erzeugen.

gegeben: $x \in SAT_1$ mit $SAT_1 = \{\psi 01^{2^n} \mid \psi \in SAT \text{ und } n = |\psi|\}$
Frage: Ist ψ erfüllbar?

gegeben: $x \in SAT_2$ mit $SAT_2 = \{\psi 01^{n^c} \mid \psi \in SAT \text{ und } n = |\psi|\}$ mit einem festen $c \in \mathbb{N}$
Frage: Ist ψ erfüllbar?

Das Problem SAT_1 liegt durch das so gewählte Padding in P. Ein Algorithmus zur Überprüfung der Formel ψ benötigt, wie bei SAT $O(2^n)$ Schritte. Jedoch ist die Eingabelänge von einer SAT_1 Instanz künstlich aufgepumpt, sodass für die Eingabelänge m einer SAT_1 Instanz gilt: $m > 2^n$. Damit kann ein Algorithmus für SAT_1 relativ zur Eingabelänge in $O(m^c)$ konstruiert werden, indem in linearer Zeit die Formel ψ ohne Padding bestimmt wird und anschließend alle 2^n Belegungen getestet werden. Also gilt $\mathrm{SAT}_1 \in$ P. Das Problem SAT_2 ist hingegen weiterhin NP-vollständig, da das Problem lediglich das Problem SAT mit polynomiell vielen Einsen ist, die Lösung jedoch mehr als polynomiell viele Schritte benötigen kann. SAT kann in polynomieller Zeit auf SAT_2 reduziert werden, indem in polynomieller Zeit für eine Formel ψ das entsprechende polynomielle Padding bestimmt wird.

4.2 Die Konstruktion von $\mathrm{SAT_H}$

Für den in diesem Abschnitt folgenden Beweis wird ein Kandidat für die Mitgliedschaft zu NPI benötigt. Dieser Kandidat ist das angepasste Problem $\mathrm{SAT_H}$, welches aus SAT mittels einer Funktion H hervorgeht. Eine $\mathrm{SAT_H}$ Instanz besteht aus einer Formel ψ, ergänzt um eine Zeichenkette, die aus einer Null, die das Ende der Formel ψ anzeigt, und einer gewissen Anzahl an Einsen besteht. ψ ist derart codiert, sodass die Zahl Null als eindeutiges Trennsymbol gewählt werden kann. Die Anzahl an Einsen wird durch die Funktion H bestimmt, die abhängig von der Eingabelänge von ψ ist.

Die Idee für die Funktion H ist, dass an ψ weniger als exponentiell viele Einsen angehangen werden, sodass gilt: $\mathrm{SAT_H} \notin$ P. Dennoch müssen genügend Einsen an $\mathrm{SAT_H}$ angehangen werden, damit $\mathrm{SAT_H}$ nicht NP-vollständig ist. An dieser Stelle wird $\mathrm{SAT_H}$ und die Funktion H(n) definiert. $\mathrm{SAT_H}$ ist gegeben durch die folgende Definition:

$$SAT_H = \{\psi 01^{n^{H(n)}} \mid \psi \in SAT \text{ und } n = |\psi|\}$$

Für die Funktion $H : \mathbb{N} \to \mathbb{N}$ gilt: H wird für $n \leq 4$ als 1 gesetzt und ansonsten wie folgt definiert:

$$H(n) = \begin{cases} \text{die kleinste Zahl } i < log_2(log_2(n)), \\ \text{sodass für jedes } x \in \{0,1\}^* \text{ mit } |x| \leq log_2(n) \text{ gilt:} \\ M_i \text{ hält innerhalb von } i|x|^i \text{ Schritten,} \\ \text{und gibt 1 aus, gdw. } x \in SAT_H, \text{ wobei } M_i \text{ die i-te DTM ist} \\ \\ \text{Wenn dieses i nicht existiert, dann ist } H(n) = log_2(log_2(n)) \end{cases}$$

Hierbei ist M_1, M_2, ... eine Aufzählung aller deterministischen Turingmaschinen und für die Turingmaschine M_i ist i die binäre Codierung von M. Zu jeder deterministischen Turingmaschine M_i, welche die Sprache $L(M_i)$ erkennt, existieren unendlich weitere deterministische Turingmaschinen M_j mit $j > i$ für die gilt: $L(M_i) = L(M_j)$. Diese Turingmaschinen M_j können konstruiert werden, indem zum Beispiel nicht erreichbare

Zustände zu M_j hinzugefügt werden und damit die erkannte Sprache nicht geändert wird.

Natürlich muss die Funktion H wohldefiniert sein. Allerdings hängt die Funktion H rekursiv von $\mathrm{SAT_H}$ ab, da $\mathrm{SAT_H}$ selbst in der Definition von H benutzt wird. Laut Definition überprüft die Funktion H für ein gegebenes n, ob alle $x \in \{0,1\}^{\leq log_2(n)}$ in $\mathrm{SAT_H}$ liegen. Diese Überprüfung kann berechnet werden, denn es werden nur Zeichenketten der Länge $x \leq log_2(n)$ überprüft. Für diese Werte ist H bereits induktiv definiert und damit wohldefiniert. Des Weiteren kann ein Algorithmus H(n) in $O(n^3)$ berechnen. Der folgende Algorithmus überprüft für maximal $log_2(log_2(n)) < n$ viele i, alle Wörter der Länge maximal $log_2(n)$, d.h. maximal $\sum_{i=0}^{log_2(n)}(2^i) = 2^{log_2(n)+1} - 1 \leq 2n$ Wörter, ob $x \in \mathrm{SAT_H}$ gilt. Für die innerste Berechnung werden höchstens $log_2(log_2(n)) \cdot log_2(n)^{log_2(log_2(n))} < cn$ Schritte für ein genügend großes $c \in \mathbb{N}$ benötigt. Damit ergibt sich ein Algorithmus bestehend aus zwei in jeweils O(n) laufenden For-Schleifen und einer in $O(n)$ laufenden Berechnung. Das Produkt aus $n, 2n$ und cn für ein genügend großes $c \in \mathbb{N}$ ergibt die Gesamtkomplexität $O(n^3)$ für den folgenden Algorithmus.

function `H(`n`):`
  for $i \leftarrow 0$ **to** $log_2(log_2(n))$ **do**
    for ***all*** $x \in \{0,1\}^{\leq log(n)}$ **do**
      Berechne auf M_i *maximal* $i|x|^i$ *Schritte*
      if M_i *akzeptiert und* $x \in SAT_H$ **then return** i
    end
  end
  return $log_2(log_2(n))$
end function

4.3 Schlussfolgerungen für $\mathrm{SAT_H}$

Für das Problem $\mathrm{SAT_H}$ lassen sich zwei Lemmas feststellen, abhängig von der Annahme, ob $\mathrm{SAT_H} \in \mathrm{P}$ oder $\mathrm{SAT_H} \notin \mathrm{P}$.

Lemma 4.1. *$SAT_H \in P$, gdw. für alle $n : H(n) \leq i$ für ein $i \in \mathbb{N}$*

Beweis. Für den Beweis wird zunächst die Hinrichtung gezeigt. $\mathrm{SAT_H} \in \mathrm{P}$ impliziert, dass für alle n gilt: $H(n) \leq i$ für ein $i \in \mathbb{N}$.
Angenommen es gibt eine DTM M, die $\mathrm{SAT_H}$ in cn^c Schritten löst. Dann gibt es auch eine DTM M_i mit $i > c$, sodass M und M_i $\mathrm{SAT_H}$ entscheiden.
Durch die Definition von H(n) gilt nun für $n > 2^{2^i}$: $log_2(log_2(n)) > i \geq H(n)$ und M_i hält nach Annahme nach $c|x|^c \leq i|x|^i$ Schritten und gibt 1 aus, gdw. $x \in \mathrm{SAT_H}$. Damit gilt $H(n) \leq i$.

An dieser Stelle wird die Rückrichtung der Äquivalenz bewiesen. Dafür wird gezeigt, dass wenn für alle n gilt: $H(n) \leq i$, dann impliziert das, dass $\mathrm{SAT_H} \in \mathrm{P}$ liegt.
Wenn für alle n gilt, dass $H(n) \leq i$, dann folgt daraus, dass ein j existiert, sodass $H(n) = j$ für unendlich viele n. Das bedeutet allerdings, dass eine DTM M_j existiert,

die $\mathrm{SAT_H}$ in jn^j Schritten löst. Angenommen, es gibt eine Eingabe x, für die die DTM M_j die falsche Antwort gibt, dann gilt für jedes $n > 2^{|x|}$: $H(n) \neq j$, da für eine Eingabe $n > 2^{|x|}$ alle Wörter der Länge $log(2^{|x|}) = |x|$ überprüft werden und damit auch das Wort x selbst. Das heißt aber, das nicht unendliche viele j existieren, sodass H(n) = j, wodurch ein Widerspruch zur Annahme entsteht, dass $H(n) = j$ für unendlich viele n. Damit gilt: Für alle $n : H(n) \leq i \Rightarrow \mathrm{SAT_H} \in \mathrm{P}$ □

Lemma 4.2. *$SAT_H \notin P$, gdw. $lim_{n\to\infty} H(n) = \infty$*

Beweis. Durch die im Lemma 4.1 geltende Äquivalenz, ist bereits ein Beweis durch Negation für die zweite Behauptung gegeben.
$\mathrm{SAT_H} \notin P \Leftrightarrow$ es gibt ein n für jede Konstante i, sodass $H(n) > i$
$\Leftrightarrow lim_{n\to\infty} H(n) = \infty$ □

Zusammenfassend gilt für $\mathrm{SAT_H}$, dass $\mathrm{SAT_H}$ genau dann in P liegt, wenn $H(n) \leq i$ für alle n und die äquivalente Aussage, dass $\mathrm{SAT_H}$ genau dann nicht in P liegt, wenn H(n) mit wachsenden n gegen unendlich strebt.

4.4 Beweis $\mathbf{SAT_H \in NPI}$

Das Problem $\mathrm{SAT_H}$ liegt in NP, da die Funktion H in polynomieller Zeit berechnet werden kann. Intuitiv kann für eine gegebene Formel mit Padding und eine Belegung in polynomieller Zeit überprüft werden, ob die Eingabe eine Instanz von $\mathrm{SAT_H}$ ist, indem in polynomieller Zeit geprüft wird, ob die Belegung erfüllend ist und in polynomieller Zeit geprüft wird, ob die Formel das richtige Padding besitzt. Mit den im vorherigen Abschnitt gezeigten Lemmas für $\mathrm{SAT_H}$ kann nun gezeigt werden, dass $\mathrm{SAT_H}$ weder in P liegt noch NP-vollständig ist und damit in NPI liegt. Für den Beweis wird vorher noch das folgende Lemma benötigt.

Lemma 4.3. *Eine Zahl n lässt sich nur polynomiell oft um einen polynomiellen Faktor, z.B. $\sqrt[k]{n}$, mit $k \in \mathbb{N}$ reduzieren, bis ein gewählter Grenzwert $c \geq \sqrt{2}$ unterschritten wird.*

Beweis. Sei dieser Faktor $\sqrt[k]{n}$ mit Grenzwert $\sqrt[k]{2}$, m die Anzahl der Reduktionen von n und $k \in \mathbb{N}$. Dann folgt $n^{(1/k)^m} \leq 2^{1/k} \Rightarrow n^{(1/k)^m \cdot k^m} \leq 2^{(1/k)\cdot k^m} \Rightarrow n^{1^m} \leq 2^{k^{-1}\cdot k^m}$
$\Rightarrow n \leq 2^{k^{m-1}} \Rightarrow m \geq log_k(log_2(n)) + 1 < n$
Als Ergebnis lässt sich feststellen, dass die Zahl n nachdem sie $log_k(log_2(n)) < n$ Mal reduziert wurde, den Grenzwert $\sqrt[k]{2}$ unterschreitet und damit weniger als n Reduktionen notwendig sind. □

Theorem 4.4. *Aus der Annahme, dass $P \neq NP$ gilt, folgt, dass in $NP \backslash P$ mindestens ein Problem existiert, dass nicht NP-vollständig ist. Damit gilt:*

$$P \neq NP \Rightarrow NPI \neq \emptyset$$

Beweis. $\mathrm{SAT_H}$ ist nicht in P.
Zum Widerspruch wird angenommen, dass $\mathrm{SAT_H}$ in P liegt. Aus Lemma 4.1 folgt, dass

für alle n gilt: $H(n) \leq i$ für eine Konstante i.
Das bedeutet, dass eine $\mathrm{SAT_H}$ Instanz, vereinfacht ausgedrückt, eine SAT Instanz ist, ergänzt um eine polynomiell lange Zeichenkette, wie beim Problem SAT_2.[4.1]
Nun gilt, dass wenn $\mathrm{SAT_H} \in$ P ist, dann ist auch SAT $\in$ P. Wenn $\mathrm{SAT_H}$ in polynomieller Zeit durch eine deterministische Turingmaschine gelöst werden kann, dann auch SAT, indem SAT in polynomieller Zeit auf $\mathrm{SAT_H}$ reduziert wird. Dafür wird an eine gegebene Formel ψ in polynomieller Zeit das Padding $01^{n^{H(n)}}$ gehangen unter der Beachtung, dass H in polynomieller Zeit berechnet werden kann. An dieser Stelle entsteht ein Widerspruch zur Annahme, dass P $\neq$ NP gilt, da SAT NP-vollständig ist und damit nicht in P liegt.
Somit gilt: $\mathrm{SAT_H} \notin$ P. □

Beweis. $\mathrm{SAT_H}$ ist nicht NP-vollständig:
Zum Widerspruch wird angenommen, dass $\mathrm{SAT_H}$ NP-vollständig ist.
Nach der Definition der NP-Vollständigkeit existiert eine polynomielle Reduktion, die das Problem SAT auf das Problem $\mathrm{SAT_H}$ reduziert. Das bedeutet, dass SAT Instanzen der Länge n auf $\mathrm{SAT_H}$ Instanzen der Länge $n + 1 + n^{H(n)}$ abgebildet werden.

Damit die Größe eines Paddings $n^{H(n)}$ eine polynomielle Länge besitzt, müsste für alle n und ein $i \in \mathbb{N}$ gelten, dass H(n) = i. Dies ist jedoch nicht möglich, da bereits bewiesen wurde, dass $\mathrm{SAT_H}$ nicht in P liegt. Daher ist die Funktion durch kein i beschränkt, sodass die Funktion H mit wachsendem n gegen unendlich strebt.[4.2] Somit sind $\mathrm{SAT_H}$ Instanzen für genügend große n mehr als polynomiell größer als SAT Instanzen, da das Padding aus $n^{H(n)}$ Einsen besteht. Die polynomielle Reduktion reduziert damit SAT-Instanzen ψ der Länge n auf $\mathrm{SAT_H}$ Instanzen der Form $\psi' 01^{m^{H(m)}}$, wobei ψ' mit der Länge m um einen polynomiellen Faktor kürzer ist als ψ, beispielsweise $m = \sqrt[3]{n}$, und $\psi' 01^{m^{H(m)}}$ eine polynomielle Länge besitzt, da nur polynomiell viele Schritte zur Verfügung stehen. Damit gilt $|\psi' 01^{m^{H(m)}}| \leq n^i$ für ein festes $i \in \mathbb{N}$.

Nun lässt sich ein Algorithmus erstellen, der SAT in polynomiell vielen Schritten löst, indem die Reduktion rekursiv angewandt wird, sodass das reduzierte ψ' ohne Padding auf die zugehörige $\mathrm{SAT_H}$ Instanz $\psi'' 01^{n^{H(n)}}$ mit $n = |\psi'|$ rekursiv reduziert wird bis SAT in O(1) lösbar ist.
Dieser Algorithmus besitzt eine polynomielle Laufzeit, da eine Eingabe der Länge n nur polynomiell oft um einen polynomiellen Faktor, wie $\sqrt[3]{n}$, reduziert werden kann bis die neue Länge einen gewählten Grenzwert, beispielsweise $i = \sqrt{2}$, unterschreitet.[4.3]
Damit existiert ein Algorithmus, der SAT in $O(n^c)$ löst, sodass SAT in P liegt. Durch die Existenz dieses Algorithmus entsteht ein Widerspruch zur Annahme P $\neq$ NP.
Somit gilt $\mathrm{SAT_H}$ ist nicht NP-vollständig. □

5 Zusammenfassung und Ausblick

Zum Ende der Ausarbeitung wird die Leitfrage abschließend in einer Zusammenfassung betrachtet und ein Ausblick auf natürliche Probleme gegeben, die im Zusammenhang zu Ladner's Theorem stehen.

5.1 Zusammenfassung

Der vorherige Abschnitt der Ausarbeitung hat gezeigt, dass unter der Annahme, dass P $\neq$ NP gilt, mindestens ein NP-intermediate Problem existiert. Dieses Problem wurde künstlich mit Hilfe des Erfüllbarkeitproblems SAT konstruiert. SAT_H ist definiert als $\text{SAT}_\text{H} = \{\psi 01^{n^{H(n)}} | \ \psi \in \text{SAT}, \ n = |\psi|\}$. Durch Padding kann die Komplexität von SAT_H beeinflusst werden, sodass die in polynomieller Zeit berechenbare Funktion H, welche die Größe des Paddings bestimmt, so gewählt wird, dass SAT_H weder in P liegt noch NP-vollständig ist. Für den endgültigen Beweis werden zwei Lemmas benötigt. Das erste Lemma zeigt, dass die Funktion H genau dann beschränkt ist, wenn SAT_H in P liegt. Mit dem ersten Lemma kann gezeigt werden, dass SAT_H in diesem Fall SAT mit polynomiell großen Padding ist. Durch diese Tatsache ist SAT_H nicht in P. Durch Hinzunahme des zweiten Lemmas, das aus dem ersten Lemma folgt, kann gezeigt werden, dass SAT_H nicht NP-vollständig ist. Wenn die Funktion H nicht mehr beschränkt ist, ist das Padding von SAT_H mehr als polynomiell größer als die Länge der Formel ψ. Unter der Annahme, dass SAT_H NP-vollständig ist, kann damit ein Algorithmus für das SAT Problem, der in polynomieller Zeit läuft, konstruiert werden, wie im Abschnitt 4.4 erläutert. Dieser Algorithmus reduziert dabei SAT_H Instanzen rekursiv auf SAT Instanzen. An dieser Stelle wird der zweite Widerspruch erzeugt unter der Annahme, dass P $\neq$ NP gilt. Das Ergebnis ist Ladner's Theorem. Wenn P $\neq$ NP gilt, dann ist NPI $\neq \emptyset$.

Hiermit kann nun die Leitfrage, ob Probleme in NP existieren, die weder in P liegen noch NP vollständig sind, beantwortet werden. Unter Annahme, dass P $\neq$ NP gilt, kann die Leitfrage mit Ja beantwortet werden. Dann existiert eine weitere nicht-triviale Komplexitätsklasse, die Klasse der NP-intermediate Probleme (NPI). In dieser Klasse liegt mindestens ein Problem, das Problem SAT_H. Jedoch muss beachtet werden, dass das in diesem Beweis gezeigte Problem SAT_H künstlich mittels einer Funktion H erzeugt worden ist und keine Relevanz in der realen Welt besitzt. Das Venn-Diagramm in Abbildung 3 zeigt den abschließenden Sachverhalt zur Fragestellung.

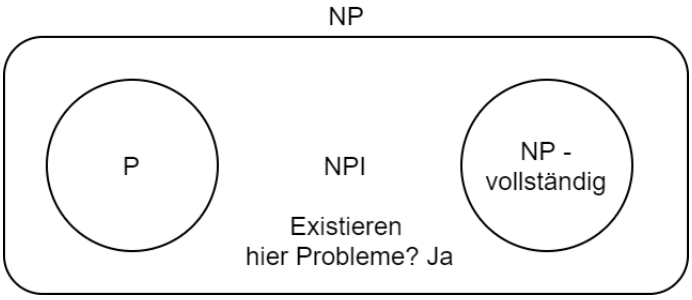

Abbildung 3: Komplexitätsklasse NP unter der Annahme P $\neq$ NP

5.2 Ausblick

Durch Ladner's Theorem kann die Beziehung zwischen P und NP besser verstanden werden. Mit der Erkenntnis von Ladner ergibt sich die Frage, ob weitere Probleme in der Komplexitätsklasse NPI liegen, insbesondere, ob auch natürliche Probleme in NPI liegen. In diesem Ausblick wird betrachtet, ob natürliche Probleme in der Komplexitätsklasse NPI liegen. Zum Zeitpunkt der Ausarbeitung ist kein natürliches Problem bekannt, welches NP-intermediate ist. Jedoch gibt es geeignete Kandidaten, von denen ausgegangen wird, dass sie in NPI liegen, da für diese Probleme weder ein Polynomialzeitalgorithmus zur Lösung des Problems bekannt ist noch nachgewiesen werden konnte, dass diese Probleme NP-vollständig sind. Ein eindeutiger Beweis zur Zugehörigkeit zu NPI konnte bislang nicht erstellt werden. Zwei dieser Kandidaten sind das Faktorisierungsproblem und das Graph Isomorphismus Problem, welche in diesem Ausblick kurz erläutert werden.

Faktorisierungsproblem: Das Faktorisierungsproblem ist relevant für die Kryptographie, da viele Verschlüsselungsmethoden Primzahlen und die damit verbundene Schwierigkeit der Faktorisierung ausnutzen. Das Problem fragt für drei gegebene natürliche Zahlen n, l und u, ob im Intervall [l,u] ein Primfaktor von n liegt.

Graph Isomorphismus Problem: Ein weiteres Problem, welches in NPI liegen könnte, ist das Graph Isomorphismus Problem. Bei der Betrachtung von Graphen ist in den meisten Fällen die Struktur eines Graphen relevanter als die Bezeichnung der Knoten. Das Graph Isomorphismus Problem kann, wie folgt, beschrieben werden:

Gegeben: Zwei endliche Graphen
Frage: Sind die beiden Graphen strukturell gleich, das heißt isomorph?

Ein Beispiel des Problems kann in der folgenden Darstellung betrachtet werden. Die beiden Graphen haben unterschiedliche Knoten, sind jedoch strukturell gleich, da die Graphen mit der bijektiven Abbildung f ineinander überführt werden können.

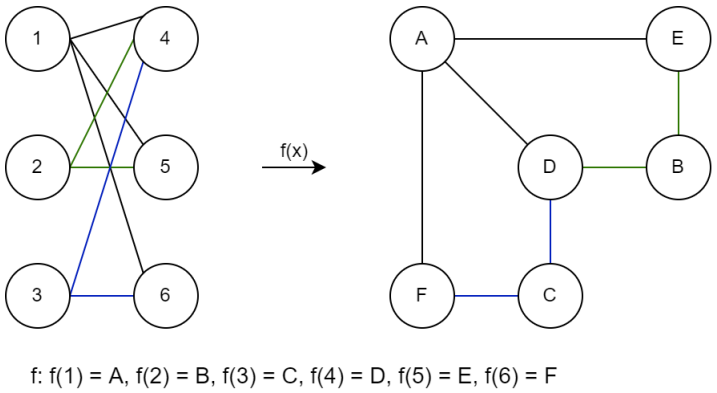

Abbildung 4: Zwei zueinander isomorphe Graphen

Literatur

[1] Arora, Sanjeev and Barak, Boaz. *Computational Complexity: A Modern Approach*. 2007.

[2] Blömer, Johannes. *Skript zur Vorlesung Komplexitätstheorie*. 2010. `https://www.hni.uni-paderborn.de/fileadmin/Fachgruppen/Algorithmen/Lehre/Vorlesungsarchiv/WS_2011_12/Komplexitaetstheorie/KT.SS2010.Skript_beta_v4.pdf`. Letzter Zugriff 13.07.2022

[3] *Ladner's theorem in TOC*. zuletzt geändert 2022. `https://www.geeksforgeeks.org/ladners-theorem-in-toc/`. Letzter Zugriff 13.07.2022

[4] Das, Sanjoy. *Understanding Ladner's Theorem*. 2020. `www.playingwithpointers.com/blog/ladners-theorem.html`. Letzter Zugriff 13.07.2022