LMU LUDWIG-MAXIMILIANS-UNIVERSITÄT MÜNCHEN
INSTITUT FÜR INFORMATIK
LEHRSTUHL FÜR MOBILE UND VERTEILTE SYSTEME

BACHELORARBEIT

Erstellung und empirische Auswertung synthetischer Daten für NP-schwere Optimierungsprobleme

Gordian Hunecke

LMU LUDWIG-MAXIMILIANS-UNIVERSITÄT MÜNCHEN
INSTITUT FÜR INFORMATIK
LEHRSTUHL FÜR MOBILE UND VERTEILTE SYSTEME

BACHELORARBEIT

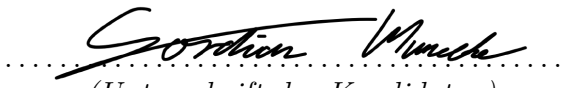
Erstellung und empirische Auswertung synthetischer Daten für NP-schwere Optimierungsprobleme

Gordian Hunecke

Aufgabensteller:	Prof. Dr. Claudia Linnhoff-Popien
Betreuer:	Jonas Nüßlein Julian Schönberger
Abgabetermin:	31. Juli 2025

Hiermit versichere ich, dass ich die vorliegende Bachelorarbeit selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet habe.

München, den 31. Juli 2025



(Unterschrift des Kandidaten)

Abstract

Jüngste Entwicklungen und Trends im Bereich der Künstlichen Intelligenz und des maschinellen Lernens - etwa das Aufkommen großer Sprachmodelle - haben einer breiteren Öffentlichkeit Konzepte wie trainierbare Algorithmen, neuronale Netze oder Deep Learning nähergebracht. Diese Fortschritte lassen sich jedoch nicht allein auf neue Erkenntnisse aus jahrzehntelanger Forschung oder den kontinuierlichen Anstieg der Rechenleistung zurückführen, sondern auch auf die Verfügbarkeit großer, annotierter Datenmengen. Insbesondere Letztere ermöglichen häufig erst das Training leistungsfähiger Modelle in verschiedensten Anwendungsdomänen. Die Auswertung von Rohdaten kann jedoch sehr zeitaufwendig sein, wenn sie nicht sogar gänzlich unpraktikabel ist.
Entscheidend ist dabei die Frage, inwieweit sich die entsprechenden Probleme effizient lösen lassen. Während für viele Problemklassen Algorithmen mit polynomialer Laufzeit existieren, wächst die Komplexität anderer exponentiell mit dem Umfang der Eingabe.
Eine mögliche Herangehensweise an diese Herausforderung besteht im inversen Generieren von Problemen. Dabei wird nicht versucht, ein gegebenes Problem einer Domäne zu lösen, sondern zu einer vorgegebenen Lösung ein Problem zu konstruieren, für das die gegebene Lösung weiterhin eine optimale Lösung bleibt. Auf diese Weise entsteht eine Problemstellung, deren Lösung bereits bekannt ist, ohne den Rechenaufwand für den eigentlichen Lösungsprozess aufbringen zu müssen.
In dieser Arbeit wird untersucht, inwiefern ein inverses Generieren von Problemen für das Maximum-Clique- und das Hamiltonian-Cycle-Problem möglich ist. Dabei werden neben historischen auch intuitive und eigene Generatoren betrachtet und die erzeugten Probleme hinsichtlich Korrektheit, Komplexität und spezifischer Eigenschaften analysiert und verglichen. Insbesondere wird der Frage nachgegangen, ob sich Probleminstanzen mit bekannter Lösung erzeugen lassen, deren Komplexität gleichwertig oder höher ist als die vergleichbarer, relevanter Instanzen innerhalb der jeweiligen Domäne.
Hierzu wurden verschiedene Verfahren implementiert und hinsichtlich Struktur, Lösungslaufzeit sowie Korrektheit empirisch untersucht.
Die Ergebnisse zeigen, dass es möglich ist, Probleme mit bekannter Lösung in einem breiten Spektrum von Struktur, Komplexität und Größe zu generieren, oftmals jedoch Vorteile in einer Dimension mit Einschränkungen in einer anderen einhergehen.
Darüber hinaus zeigt sich, dass identische Ansätze in unterschiedlichen Problemdomänen unterschiedlich effektiv sind. So lassen sich beispielsweise bestimmte, schwer lösbare Instanzen des Maximum-Clique-Problems vergleichsweise effizient im Hamiltonian-Cycle-Problem lösen.

Abstract

Recent developments and trends in the field of artificial intelligence and machine learning - such as the emergence of large language models - have brought concepts such as trainable algorithms, neural networks and deep learning to a wider audience. However, this progress cannot be attributed solely to new findings from decades of research or the continuous increase in computing power, but also to the availability of large, annotated amounts of data. The latter in particular often make it possible to train powerful models in a wide range of application domains. However, the evaluation of raw data can be very time-consuming, if not completely impractical.

The decisive factor here is the extent to which the corresponding problems can be solved efficiently. While algorithms with polynomial runtimes exist for many problem classes, the complexity of others grows exponentially with the size of the input.

One possible approach to this challenge is the inverse generation of problems. This does not attempt to solve a given problem in a domain, but to construct a problem for a given solution for which the given solution remains an optimal solution. In this way, a problem is created whose solution is already known, without having to expend the computational effort for the actual solution process.

This thesis examines the extent to which inverse problem generation is possible for the Maximum-Clique- and Hamiltonian-Cycle-Problem. In addition to historical generators, intuitive and custom generators are also considered, and the generated problems are analyzed and compared with respect to correctness, complexity, and specific properties. In particular, the question of whether problem instances with a known solution can be generated whose complexity is equivalent to or higher than that of comparable, relevant instances within the respective domain is investigated.

For this purpose, various methods were implemented and empirically examined with regard to structure, solution time, and correctness.

The results show that it is possible to generate problems with a known solution in a wide range of structure, complexity, and size, but often advantages in one dimension are accompanied by limitations in another.

Furthermore, it turns out that identical approaches are differently effective in different problem domains. For example, certain hard-to-solve instances of the Maximum-Clique-Problem can be solved comparatively efficiently in the Hamiltonian-Cycle-Problem.

Inhaltsverzeichnis

1 Einleitung

Jüngste Entwicklungen und Trends im Bereich der Künstlichen Intelligenz und des maschinellen Lernens - etwa das Aufkommen großer Sprachmodelle - haben einer breiteren Öffentlichkeit Konzepte wie trainierbare Algorithmen, neuronale Netze oder Deep Learning nähergebracht.
Doch seien es historisch wegweisende Modelle im Bereich der Bilderkennung wie das AlexNet-Modell [KSH12], weiterführende Architekturen wie die VGG- oder ResNet-Modelle [SSSG17], oder jüngste Untersuchungen von transformerbasierten Modellen [YZY+23] oder Pointer-Netzen [VFJ15] zur Lösung von Traveling-Salesman-Problemen. Sie alle sind nicht allein auf neue Erkenntnisse aus jahrzehntelanger Forschung oder den kontinuierlichen Anstieg der Rechenleistung zurückzuführen, sondern auch auf die Verfügbarkeit großer, annotierter Datenmengen. Ein bekanntes Beispiel für eine solche Datensammlung ist der sogenannte ImageNet-Datensatz von Deng et al. [DDS+09], mit über 3 Millionen aufbereiteten Bildern aus verschiedenen Themenbereichen inklusive entsprechender Annotation.
Doch eine solche Auswertung von Rohdaten kann sehr zeitaufwendig sein, wenn sie nicht sogar gänzlich unpraktikabel ist. Entscheidend ist dabei die Frage, inwieweit sich die entsprechenden Probleme effizient lösen bzw. annotieren lassen. Während für viele Problemklassen Algorithmen mit polynomialer Laufzeit existieren, wächst die Komplexität anderer exponentiell mit dem Umfang der Eingabe.
Eine mögliche Herangehensweise an diese Herausforderung besteht im inversen Generieren von Problemen. Dabei wird nicht versucht, ein gegebenes Problem einer Domäne zu lösen, sondern zu einer vorgegebenen Lösung ein Problem zu konstruieren, für das die gegebene Lösung weiterhin eine optimale Lösung bleibt. Auf diese Weise entsteht eine Problemstellung, deren Lösung bereits bekannt ist, ohne den Rechenaufwand für den eigentlichen Lösungsprozess aufbringen zu müssen.
Doch neben der Erstellung von Trainingsdaten für lernende Modelle ist auch die Anwendung zur Effektivitätsmessung von approximativen Lösungsalgorithmen für schwere Optimierungsprobleme von Interesse, wie es etwa Sanchis in ihrer Dissertation [San89] beschreibt. Solche Lösungsverfahren finden oftmals nur Annäherungen an die optimale Lösung eines Problems, nicht die exakte Lösung. Man kann die Effektivität eines Verfahrens jedoch nur messen, wenn Kenntnis über die Differenz der gefundenen zur optimalen Lösung besteht. Es wäre daher von Vorteil, ein Verfahren an Probleminstanzen mit bekannter optimaler Lösung testen zu können.
Aus diesem Anlass beschäftigt sich die Forschung wiederkehrend mit dem Erzeugen synthetischer Probleminstanzen, bei denen die optimale Lösung des Problems im Vornherein bekannt ist. Brockington und Cuberson [BC93] untersuchten etwa 1993 einen Ansatz, wie man eine optimale Lösung im Maximum-Clique-Problem möglichst gut in einem zufälligen Graphen verstecken kann. Nach ähnlichem Schema untersuchte Sanchis im selben Zeitraum ein Verfahren für das Minimal-Vertex-Cover-Problem [San94]. Zuletzt stellten Xu et al. [XBHL07] ein populäres Verfahren zur Erzeugung schwerer Probleminstanzen

mit bekannter Lösbarkeit für diverse Problemklassen mithilfe einer Konstruktion über den sogenannten Phasenübergang beim Erfüllbarkeitsproblem (SAT) [MZK+99] vor. Aus diesem Verfahren ist der bekannte BHOSLIB [Xu07] Datensatz hervorgegangen, welcher bis heute weitreichende Bedeutung als Vergleichsdatensatz zur Messung der Leistung von Lösungsalgorithmen hat.
Das Erzeugen von Problemen mit bekannter Lösung ist daher nach wie vor von Aktualität und Interesse. Diese Arbeit soll diesbezüglich einen Beitrag leisten, indem verschiedene mögliche Verfahren zum Erzeugen von Probleminstanzen mit bekannter Lösung für das Maximum-Clique- und Hamiltonian-Cycle-Problem analysiert, empirisch verglichen und hinsichtlich einer Eignung als möglicher Generator evaluiert werden.

Sämtliche in dieser Arbeit vorgestellten Generatoren wurden in Python implementiert und stehen unter `https://github.com/Jumagoro/nphard-generators` zur Verfügung.

1.1 Notation

Im Folgenden sind die in dieser Arbeit verwendeten Notationen, wenn lokal nicht anders bezeichnet, zusammengefasst:

Symbol	**Bedeutung**
$G = (V, E)$	Ein Graph mit Knotenmenge V und Kantenmenge E
$\overline{G} = (V, \overline{E})$	Kompl. Graph von G mit $\overline{E} = \{(u, v) \in V^2 : u \neq v \wedge (u, v) \notin E\}$
V	Menge der Knoten (engl. *vertices*)
E	Menge der Kanten (engl. *edges*)
$n = \|V\|$	Anzahl der Knoten
d	Dichte des Graphen
$G[u, v]$	Kante zwischen Knoten u und v in Graph G
a	Anzahl hinzugefügter Kanten
a_{zus}	Kanten, die zusätzlich hinzugefügt werden sollen
s	Größe der maximalen Clique
s_b	Größe der Hintergrund-Clique (siehe Kapitel 2.7)
C	Die betrachtete (meist maximale) Clique bzw. der betrachtete Pfad
u, v	Knoten des Graphen G (außer in Kapitel (4.1.5)
(u, v)	Eine Kante von u nach v
grad(v)	Grad eines Knotens v
n_*	Anzahl von *
verbindeKnoten(C)	Verbindet alle Knoten $c \in C$ miteinander
maxKanten(C)	Maximale Anzahl an Kanten im Teilgraph $G(C)$
kanten(C)	Tatsächliche Anzahl an Kanten im Teilgraph $G(C)$
zufallsKnoten(C)	Wählt zufällig einen Knoten $c \in C$ aus
zufall()	Erzeugt eine Zufallszahl $z \in [0; 1[$

Tabelle 1.1: Verwendete Notation

2 Hintergrund

In den nachfolgenden Abschnitten werden zunächst wichtige Grundkonzepte, welche in dieser Arbeit behandelt werden, kurz definiert und erläutert. Das Verständnis von Entscheidungs- / Optimierungsproblemen (2.1) und NP-schweren / -vollständigen Problemen (2.2) ist dabei fundamental für beide betrachteten Problemklassen, dem Maximum-Clique-Problem (2.3) und dem Hamiltonian-Cycle-Problem (2.4). Die Beschreibungen des Minimal-Vertex-Cover- (2.5), des Maximum-Independent-Set-Problems (2.6) und der sogenannten Hintergrund-Clique (2.7) sind jedoch hauptsächlich für die Konstruktion von Probleminstanzen des Maximum-Clique-Problems relevant.

2.1 Entscheidungs- vs. Optimierungsprobleme

Grundsätzlich lassen sich Probleme in zwei Arten unterteilen. Zum einen gibt es Entscheidungsprobleme, welche lediglich ein "ja" oder "nein" zur Antwort haben. Beispielsweise ist die Frage, ob es in einem Graphen einen vollständigen Zyklus gibt, der alle Knoten beinhaltet, mit einem einfachen "ja" oder "nein" zu beantworten.
Zum anderen gibt es Optimierungsprobleme, bei denen nach einer konkreteren Lösung als nur "ja" oder "nein" gefragt wird. Beispielsweise, welche Knoten einen vollständigen Zyklus in einem Graphen bilden.
Oftmals befinden sich dabei die zwei Arten desselben Problems in unterschiedlichen Komplexitätsklassen (siehe Abschnitt 2.2).
In dieser Arbeit erzeugen die meisten Generatoren Probleminstanzen für das Optimierungsproblem (da dadurch beide Arten abgedeckt werden), manche jedoch auch nur Instanzen für das entsprechende Entscheidungsproblem.

2.2 NP-schwere und NP-vollständige Probleme

Verschiedene Problemklassen sind oftmals unterschiedlich schwer zu lösen oder zu verifizieren. Das Generieren von Probleminstanzen mit bekannter Lösung ist dabei vor allem für Probleme interessant, welche nicht in polynomialer Laufzeit lösbar sind (also nicht in der sogenannten P-Klasse liegen). Man spricht bei solchen Problemen von Problemen, die mindestens so schwer sind wie Probleme in NP. Probleme in NP definiert, dass diese nicht polynomial lösbar sind, eine korrekte Lösung jedoch zumindest in polynomialer Laufzeit verifiziert werden kann. Lässt sich zudem beweisen (z.B. mithilfe einer Polynomialzeitreduktion), dass ein Problem genauso schwer ist wie die schwersten Probleme in NP, so bezeichnet man es als NP-vollständig.
Beispielhaft dafür ist das in dieser Arbeit betrachtete Hamiltonian-Cycle-Problem (2.4). Das Finden eines hamiltonschen Zyklus ist im Allgemeinen nicht polynomial lösbar, eine gegebene Lösung ist jedoch in polynomialer Laufzeit ($O(n)$) verifizierbar und es existiert eine Polynomialzeitreduktion auf bekannte NP-vollständige Probleme. [GJ79a]

Lässt sich die Lösung eines Problems nicht in polynomialer Laufzeit verifizieren, so bezeichnet man das Problem als NP-schwer. In diese Klasse fällt das ebenfalls in dieser Arbeit betrachtete Maximum-Clique-Problem. Die Gültigkeit einer gefundenen Clique (Clique-Entscheidungsproblem) lässt sich zwar in polynomialer Laufzeit überprüfen ($O(n^2)$), die Maximalität dieser Clique jedoch nicht. [GJ79a]
Neben der exakten Bestimmung einer Problemlösung gibt es jedoch verschiedenste Heuristiken, mit denen Annäherungen an die Lösungen mit wesentlich geringerer Zeitkomplexität erfolgen können. Eine solche Heuristik für das betrachtete Maximum-Clique-Problem könnte beispielsweise eine gierige Suche sein. Man prüft dabei zuerst die Knoten mit dem höchsten Knotengrad, da diese in der Theorie wahrscheinlicher Teil der maximalen Clique sind als andere Knoten. Mit dem inversen Generieren von Probleminstanzen mit bekannter Lösung können Datensätze erstellt werden, um die Effektivität solcher Heuristiken zu messen.

2.3 Maximum-Clique-Problem

Viele Probleme, insbesondere aus der Kodierungs- und Informationstheorie, lassen sich auf einen Graphen und das Finden einer möglichst großen Gruppe (z.B. entsprechender Codewörter) in diesem Graphen reduzieren [BSSS06]. Dies entspricht dem sogenannten "Maximum-Clique-Problem". Eine Clique in einem Graphen $G = (V, E)$ bezeichnet dabei eine Knotenmenge $C \subseteq V$, für die der resultierende Teilgraph $G(C)$ vollständig verbunden ist [HPV93]. In dieser Arbeit wird das Maximum-Clique-Problem für ungerichtete, ungewichtete Graphen untersucht.

2.4 Hamiltonian-Cycle-Problem

Das Hamiltonian-Cycle-Problem beinhaltet die Frage, ob ein Graph einen Pfad (Knotenfolge) enthält, sodass jeder Knoten genau einmal enthalten ist und man wieder zurück zum Startknoten kommt, um den Zyklus zu schließen. Es also eine Permutation $C = (v_1, v_2, \ldots, v_n) \in V^n$ gibt, für die gilt $\forall u, v \in C : u \neq v$ und $\forall v_i \in C \setminus \{v_n\} : (v_i, v_{i+1}) \in E$ und $(v_n, v_1) \in E$. In dieser Arbeit wird das Hamiltonian-Cycle-Problem für ungerichtete, ungewichtete Graphen untersucht.

2.5 Minimal-Vertex-Cover-Problem

Die kleinste, alle Kanten überdeckende Knotenmenge (im Nachfolgenden auch als "Minimal-Vertex-Cover" bezeichnet) eines Graphen $G = (V, E)$ bezeichnet die kleinste Teilmenge $V' \subseteq V$, für die alle Kanten $(u, v) \in E$ mit mindestens einem Knoten aus V' verbunden sind, also $\forall (u, v) \in E : u \in V' \vee v \in V'$.
Das Minimal-Vertex-Cover-Problem beschreibt daher das Problem, in einem Graphen G die kleinste kantenüberdeckende Knotenmenge V' zu finden und entspricht dabei mit $\overline{G} = (V, \overline{E})$ dem Maximum-Clique-Problem mit maximaler Clique $\overline{V}$. [GJ79b]
Eines der in dieser Arbeit vorgestellten Verfahren nutzt das Minimal-Vertex-Cover-Problem zur Konstruktion des Maximum-Clique-Problems.

2.6 Maximum-Independent-Set-Problem

Die größte unabhängige Knotenmenge (im Nachfolgenden auch als "Maximum-Independent-Set" bezeichnet) eines Graphen $G = (V, E)$ bezeichnet die größte Teilmenge $V' \subseteq V$, für die kein Knoten $u \in V'$ mit einem anderen Knoten $v \in V'$ verbunden ist, also $\forall u, v \in V' : (u, v) \notin E$.
Das Maximum-Independent-Set-Problem beschreibt das Problem, in einem Graphen G die größte unabhängige Knotenmenge V' zu finden und entspricht im komplementären Graphen $\overline{G} = (V, \overline{E})$ dem Maximum-Clique-Problem mit maximaler Clique V'. [GJ79b]
Eines der in dieser Arbeit vorgestellten Verfahren nutzt das Maximum-Independent-Set-Problem zur Konstruktion des Maximum-Clique-Problems.

2.7 Hintergrund-Clique

Als erstes stellte Matula um 1970 [Mat70] fest, dass für eine feste Dichte die Verteilung der maximalen Clique eines Zufallsgraphen stark konzentriert ist. Fast jeder Zufallsgraph einer gewissen Dichte d und Größe n hat also in etwa die gleiche Größe der maximalen Clique. Dieser Befund wurde später durch Grimett und McDiarmid [GM75] sowie Bollobás und Erdős [BE76] bestätigt. Die Größe s_b dieser Hintergrund-Clique kann als Basis für Problemgeneratoren für das Maximum-Clique-Problem genutzt werden.
Als Ansatz dient die Anzahl Y_r an Cliquen einer bestimmten Größe r. Die Wahrscheinlichkeit, dass r beliebige Knoten in einem Graphen der Dichte d vollständig miteinander verbunden sind, ist $d^{\binom{r}{2}}$. Die erwartete Anzahl an Cliquen der Größe r lässt sich damit aus der Anzahl aller möglichen Knotenmengen der Größe r und der Wahrscheinlichkeit, dass diese je vollständig verbunden sind, mit $E(Y_r) = \binom{n}{r} \cdot d^{\binom{r}{2}}$ bestimmen.
Die Größe s_b der maximalen Clique ist somit das $r \in \mathbb{N}$, für das $E(Y_r) \geq 1 \wedge E(Y_{r+1}) < 1$. Der Wert für r lässt sich dann entweder numerisch oder mithilfe der Stirling-Annäherung $n! \sim \sqrt{2\pi n}(\frac{n}{e})^n$ berechnen. [Bol01a]
Die obengenannte Bedingung lässt sich wie in (2.1) gezeigt umformen:

$$\begin{aligned} E(Y_r) = 1 &= \binom{n}{r} \cdot d^{\binom{r}{2}} \\ &= (2\pi)^{-\frac{1}{2}} \cdot n^{n+\frac{1}{2}} \cdot (n-r)^{r-n-\frac{1}{2}} \cdot r^{-r-\frac{1}{2}} \cdot d^{r(r-1)\frac{1}{2}} \end{aligned} \tag{2.1}$$

Ein r, welches diese Gleichung erfüllt, kann dann mit (2.2) angenähert werden:

$$r = 2 \cdot \log_b(n) - 2 \cdot \log_b(\log_b(n)) + 2 \cdot \log_b\left(\frac{e}{2}\right) + 1 \quad \text{mit } b = \frac{1}{d} \tag{2.2}$$

Eine detaillierte Ausführung dazu findet sich in [Bol01a].
Zwei der in dieser Arbeit vorgestellten Verfahren greifen auf die Größe dieser Hintergrund-Clique zurück.

3 Verwandte Arbeiten

Die Idee des inversen Generierens von Probleminstanzen mit bekannter optimaler Lösung ist nicht neu.
So untersuchte etwa Sanchis in ihrer Dissertation 1989 [San89] ein allgemeines Verfahren zur Generierung von Probleminstanzen für NP-schwere Probleme bzw. stellte 1994 [San94] explizit einen Ablauf zur Erzeugung von Problemen mit bekannter optimaler Lösung für das Minimal-Vertex-Cover bzw. das Maximum-Clique-Problem vor. In ihren Arbeiten werden jedoch nur einzelne Generatoren betrachtet und nur bedingt mit anderen Graphenklassen und Ansätzen bezüglich Komplexität, Struktur und Eigenschaften verglichen.
Ein weiterer historisch bekannter Generator mit dem Ziel der Erzeugung künstlicher Probleme mit einer eingebauten Lösung wurde 1994 von Brockington und Culberson [BC93] untersucht. Dabei wurden die erzeugten Graphen auch mit zufälligen Graphen verglichen. Der Fokus der Arbeit liegt jedoch auf dem Verfahren selbst, dem Verstecken großer Cliquen in Graphen für das Maximum-Clique-Problem. Es erfolgt kaum eine Analyse hinsichtlich anderer Verfahren.
Neben Verfahren für das Maximum-Clique-Problem gibt es eine Vielzahl von Generatoren für das bekanntere Traveling-Salesman-Problem (TSP). So stellten etwa Arthur und Frendewey 1988 einen Generator für das TSP vor [AF88], welcher Probleme mit bekannter optimaler Tour erzeugt. Das Verfahren ist jedoch auf das TSP beschränkt, und es erfolgt kaum ein Vergleich in Bezug auf andere Verfahren.
Ein weiterer Generator für das TSP wurde etwa von Hougardy und Zhong [HZ21] vorgestellt. Bei diesem Verfahren liegt der Fokus jedoch auf dem Erzeugen besonders schwerer Instanzen. Das Sicherstellen einer bekannten, optimalen Lösung wird nicht betrachtet.
Eine größere Sammlung verschiedener Verfahren für das Maximum-Clique-Problem veröffentlichten 1992 Hasselberg et al. [HPV93]. In ihrer Arbeit gehen sie insbesondere auf die Herkunft und den Ablauf verschiedener Verfahren ein. Es handelt sich jedoch um eine allgemeine Sammlung von Generatoren, das Ableiten bekannter Lösungen wird nicht betrachtet. Zudem gibt es keinen Vergleich bezüglich der Komplexität oder Struktur der erzeugten Graphen unter den Verfahren.
Neben den genannten Arbeiten gibt es drei weitere, weit verbreitete Problemsammlungen, welche insbesondere als Messverfahren für Lösungsalgorithmen für das Maximum-Clique-, Hamiltonian-Cycle- und Traveling-Salesman-Problem verwendet werden. Der erste der drei ist der sogenannte DIMACS-Datensatz, welcher aus der zweiten DIMACS Challenge im Jahre 1993 [JT96] hervorgegangen ist und unter anderem Graphen für das Maximum-Clique-Problem enthält. Dieser ist jedoch zunächst eine reine Sammlung von existierenden Graphen, nicht von Generatoren.
Eine zweite bekannte Sammlung, vor allem für das TSP, aber auch für das Hamiltonian-Cycle-Problem, ist die sogenannte TSPLIB-Sammlung [Rei91]. Diese enthält verschiedene Probleminstanzen, jedoch nur mit Ober- und Untergrenzen für die Lösungen der Probleme, keine exakten Lösungen. Zudem enthält auch diese Sammlung keine konkreten

Generatoren, sondern nur Probleminstanzen.
Ein dritter bekannter Datensatz ist die von Xu vorgestellte BHOSLIB-Sammlung [Xu07]. Diese enthält schwer zu lösende Graphen für etwa das Maximum-Clique-Problem, in die mithilfe einer Konstruktion über das Erfüllbarkeitsproblem (SAT) bekannte Lösungen eingesetzt wurden [XBHL07]. Xu et al. betrachten dabei einen anderen Ansatz als die in dieser Arbeit vorgestellten Generatoren, nämlich den sogenannten Phasenübergang bei Erfüllbarkeitsproblemen [MZK+99]. Dieser scheint in Bezug auf das Ziel, die Erzeugung (schwerer) Probleminstanzen mit bekannter Lösung, durchaus vielversprechend und hat sich bereits empirisch mehrfach bestätigen können.
Einen ähnlichen Ansatz nutzten Sleegers und van den Berg [SvdB22] in Kombination mit evolutionären Algorithmen, um besonders schwere Probleminstanzen für das Hamiltonian-Cycle-Problem zu erzeugen. Dabei stellte sich eine Gruppe von Graphen als besonders schwer heraus, deren Mitglieder alle eine verwandte Struktur teilen. Dieses Verfahren basiert jedoch nicht auf einer zuvor bekannten Lösung, und die Erkenntnisse konnten nicht genutzt werden, um einen allgemeinen Problemgenerator für Probleme konstant hoher Komplexität abzuleiten.
Ein eher unbekannter Datensatz für das Traveling-Salesman-Problem und Hamiltonian-Cycle-Problem wurde von Baniasadi et al. in [BEHR18] vorgestellt. Bei dieser Sammlung wurden verschiedenste schwere Graphen aus der Literatur oder abgeänderten Zufallsgeneratoren untersucht und zusammengestellt. Das Erzeugen von Problemen mit bekannter optimaler Lösung fand jedoch keine Betrachtung. Außerdem enthält die Sammlung Probleminstanzen speziellerer Strukturen, deren Erkenntnisse nicht ohne Weiteres auf eine Vielzahl unterschiedlicher Graphen übertragbar sind.
Abschließend sei noch auf eine konzeptionelle Perspektive des Erzeugens von Problemen mit bekannter Lösung verwiesen. Eine solche stellen beispielsweise Pilcher und Rardin [PR92] in ihrer Arbeit vor. Dabei handelt es sich um einen allgemeineren Ansatz, der theoretisch auf verschiedene Problemklassen übertragen werden könnte, um Probleme mit bekannter Lösung zu erzeugen. Es wird jedoch als konkrete Problemklasse nur das Traveling-Salesman-Problem betrachtet und dabei hauptsächlich die Theorie des Verfahrens, kein konkreter Generator.

4 Methodik

Zunächst soll dargestellt werden, welche Ansätze für die Untersuchung ausgewählt wurden, inwiefern diese sich für das Erstellen synthetischer Probleme mit bekannter Lösung eignen könnten, sowie deren Prozedur beschrieben werden.

4.1 Maximum-Clique-Problem

4.1.1 Intuitiver Ansatz

Zunächst soll ein Ansatz gezeigt werden, der auf einer intuitiven Herangehensweise an das Erzeugen von Graphen mit einer bekannten Maximum-Clique basiert. Dieser soll insbesondere als ein erster Test dienen, ob das synthetische Erzeugen von Instanzen für das Maximum-Clique-Problem mit bekannter Lösung grundsätzlich möglich ist.

4.1.1.1 Idee

Die Grundidee hinter vielen Ansätzen beim Erzeugen synthetischer Probleme ist das Einbetten einer beliebigen Lösung in das Problemszenario, hier den Graphen, und anschließendes Verstecken der Lösung durch Generieren einer vermeintlich zufälligen Umgebung, in diesem Fall weiterer Kanten.
Der folgende intuitive Ansatz soll den Erhalt der ursprünglichen Lösung sicherstellen, indem Knoten außerhalb der eingebetteten Clique nie mehr Kanten erhalten können, als eine Clique, die größer als die ursprüngliche ist, benötigen würde.

4.1.1.2 Problemgenerator

Um nach oben genannter Idee eine Probleminstanz zu erzeugen, wird zunächst, wie in Algorithmus 1 dargestellt, eine beliebige Clique C der Größe $s \in \mathbb{N}$ gebildet und anschließend in den Graphen gesetzt. Für jeden Knoten $c \in C$ gilt nun $grad(c) = s - 1$.
Die Kenntnis über den Grad der Knoten in C ermöglicht nun im zweiten Schritt das Verbinden der verbliebenen Knoten $v \notin C$ zu je maximal $s-1$ anderen Knoten. Dadurch wird sichergestellt, dass keine Cliquen der Größe $s_i > s$ entstehen können.
In Bezug auf die Dichte des Graphen gibt es allerdings eine Obergrenze, welche sich aus der Anzahl der Kanten der Clique C und den möglichen Kanten der verbleibenden Knoten $v \notin C$ ergibt. Die Dichte ist maximal, wenn letztere stets nur zu Knoten der Clique C verbunden werden und lässt sich nach (4.1) bestimmen.

$$d_{\max}(s,n) = \frac{0.5s(s-1) + (n-s)(s-1)}{0.5n(n-1)} = \frac{(s-1)(n-0.5s)}{0.5n(n-1)} \tag{4.1}$$

Algorithm 1 Intuitiver Generator für das Maximum-Clique-Problem

Require: $n \geq s \geq 0, \quad 0 \leq d \leq 1$

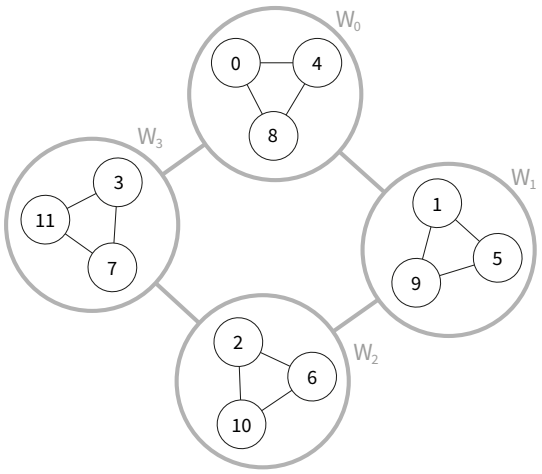
$C \leftarrow$ Wähle s zufällige Knoten aus V
verbindeKnoten(C)
$a_{zus} \leftarrow d \cdot$maxKanten(G) - kanten($C$) ▷ Anzahl zusätzlicher Kanten
for all $u \in \overline{C}$ **do**
    $andere \leftarrow V \setminus \{u\}$
    $b \leftarrow 0$ ▷ Grad von u
    for all $v \in andere$ **do**
        if $G[u, v]$ **then**
            $b \leftarrow b + 1$
            $andere \leftarrow andere \setminus \{v\}$
        end if
    end for
    while $b < s - 1 \wedge |andere| > 0$ **do**
        if $a_{zus} \leq 0$ **then**
            return
        end if
        $v \leftarrow$ zufallsKnoten($andere$)
        $G[u, v] \leftarrow$ true
        $andere \leftarrow andere \setminus \{v\}$
        $a_{zus} \leftarrow a_{zus} - 1$
    end while
end for

4.1.2 C-Fat Graphen

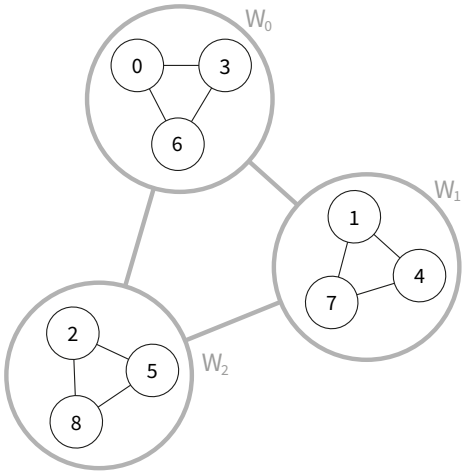
Neben der Entwicklung eigener Verfahren besteht eine weitere Möglichkeit für das Erzeugen geeigneter Probleminstanzen in der Analyse und Verwendung bereits existierender Verfahren. Eine solche, historisch aus dem DIMACS-Datensatz [JT96] bekannte Klasse von Graphen sind die sogenannten C-Fat-Ringe. Im Folgenden soll die Struktur dieser Graphenklasse untersucht und daraus Möglichkeiten zum Ableiten einer enthaltenen maximalen Clique aufgezeigt werden.

4.1.2.1 Hintergrund

Graphen der C-Fat-Klasse entstammen ursprünglich einem Verfahren zur Fehlerdiagnose von großen Multiprozessor-Systemen nach Berman und Pelc [BP90], Teil dessen unter anderem das Finden einer maximalen Clique in den C-Fat-Graphen ist.
Der Graph besteht dabei aus $n \in \mathbb{N}$ Knoten, welche in $k = \lfloor \frac{n}{c \cdot ln(n)} \rfloor$ möglichst gleich große Partitionen $W_0, \ldots, W_{k-1}$ mit $W_i = \{u \in \{0, \ldots, n-1\} \mid u \bmod k = i\}$ unterteilt werden. Dabei gilt für je zwei Knoten $u \in W_i$ und $v \in W_j$: $(u, v) \in E \iff u \neq v \wedge |i - j| \in \{0, 1, k-1\}$. [HPV93]
Zwei unterschiedliche Knoten sind also genau dann verbunden, wenn sie in derselben oder in zwei aneinandergrenzenden Partitionen sind. Dadurch entsteht ein ringartiger Aufbau, wie in Abbildung 4.1 zu sehen.



(a) Struktur eines C-Fat-Graphen für $k = 4$



(b) Struktur eines C-Fat-Graphen für $k = 3$

Abbildung 4.1: Ringartige Struktur der C-Fat-Graphen für $k = 4$ Partitionen (4.1a) und $k = 3$ Partitionen (4.1b).

Da benachbarte Partitionen vollständig miteinander verbunden sind, lässt sich die größte Clique C wie folgt ableiten:

$$C = \begin{cases} W_0 \cup W_1 \cup W_2 & \text{für } k = 3 \\ W_0 \cup W_1 & \text{sonst} \end{cases} \tag{4.2}$$

beziehungsweise gilt für die Anzahl der Knoten in C:

$$|C| = \begin{cases} 2 \cdot \lfloor \frac{n}{k} \rfloor + \min(n \bmod k, 2) & \text{für } k \geq 4 \\ n & \text{sonst} \end{cases} \tag{4.3}$$

4.1.2.2 Problemgenerator

Um aus diesen Erkenntnissen einen Problemgenerator mit Lösungsbereitstellung zu gewinnen, wird die ursprüngliche C-Fat-Prozedur (vgl. [HPV93]) um eine entsprechende Logik zur Bestimmung der maximalen Clique erweitert (siehe Algorithmus 2). Für die trivialen Fälle $k \leq 3$ besteht die maximale Clique aus allen Knoten. Für die nicht trivialen Fälle $k \geq 4$ wird für jeden Knoten $u \in \{0, \ldots, n-1\}$ geprüft, ob $u \in W_0 \cup W_1$ und falls ja, der maximalen Clique C hinzugefügt:

4.1.3 Sanchis Graphen

Ein weiteres bekanntes Verfahren ist der sogenannte Sanchis-Generator, nach Sanchis [San94], bei dem explizit eine definierte Lösung in einem Graphen versteckt wird. Dieser wurde ursprünglich für das Vertex-Cover-Problem entwickelt, kann jedoch auf das Maximum-Clique-Problem übertragen werden [HPV93].

Algorithm 2 C-Fat Generator für das Maximum-Clique-Problem

Require: $c > 0, \quad n > 0$
1: $k \leftarrow \lfloor \frac{n}{c \cdot ln(n)} \rfloor$
2: **for** $u = 0$ **to** $n - 2$ **do** ▷ Partitionen erstellen
3: $part_u \leftarrow u \bmod k$
4: **for** $v = u + 1$ **to** $n - 1$ **do**
5: $part_v \leftarrow v \bmod k$
6: $diff \leftarrow |part_v - part_u|$
7: **if** $diff \leq 1 \vee diff = k - 1$ **then**
8: $G[u, v] \leftarrow$ true
9: **end if**
10: **end for**
11: **end for**
12:
13: $C \leftarrow \{\}$ ▷ Bestimmen der maximalen Clique
14: **if** $k \leq 3$ **then**
15: $C \leftarrow \{0, \ldots, n - 1\}$
16: **else**
17: **for** $u = 0$ **to** $n - 1$ **do**
18: $part_u \leftarrow u \bmod k$
19: **if** $part_u = 0 \vee part_u = 1$ **then**
20: $C \leftarrow C \cup \{u\}$
21: **end if**
22: **end for**
23: **end if**

4.1.3.1 Hintergrund

Bei diesem Ansatz wird zuerst im komplementären Graphen $\overline{G}(V, \overline{E})$ durch das Bilden von möglichst gleich großen, ineinander vollständig verbundenen Partitionen eine minimale Knotenüberdeckungsmenge erzeugt. Diese besteht aus allen bis auf einen Knoten jeder Partition. Die einzelnen, übrigen Knoten jeder Partition bilden zusammen im Graphen $G(V, E)$ eine maximale Clique. [PX94]

4.1.3.2 Problemgenerator

Zum Erzeugen einer maximalen Clique der Größe $s \in \mathbb{N}$ wird der komplementäre Graph in s möglichst gleich große Partitionen $W_0, \ldots, W_{s-1}$ mit $W_i = \{u \in \{0, \ldots, n-1\} \mid u \bmod s = i\}$ aufgeteilt und diese in sich vollständig verbunden (siehe Algorithmus 3).
Anschließend werden die ersten s Knoten $u \in \{0, \ldots, s-1\}$ als maximale Clique C definiert (aus jeder Partition der erste Knoten). Die verbleibenden Knoten sind Teil der minimalen Knotenüberdeckungsmenge.
In einem zweiten Schritt können weitere Kanten im komplementären Graphen $\overline{G}$ bis zu einer Maximalzahl von $a_{max} = \binom{n}{2} - \binom{s}{2}$ hinzugefügt werden, um die Dichte des eigentlichen Graphen G (des Maximum-Clique-Problems) zu verringern. Dabei muss je mindestens ein Kantenende u oder v ein Knoten außerhalb der maximalen Clique C sein, da sonst in G die maximale Clique nicht mehr vollständig verbunden wäre.

Algorithm 3 Sanchis Generator für das Maximum-Clique-Problem

Require: $n \geq s \geq 0, \quad 0 \leq d \leq 1$

$C \leftarrow \{0, \ldots, s-1\}$
$a \leftarrow 0$ ▷ Anzahl hinzugefügter Kanten
for $u = 0$ **to** $n-2$ **do** ▷ Partitionen erstellen
    $part_u \leftarrow u \bmod s$
    for $v = u+1$ **to** $n-1$ **do**
        $part_v \leftarrow v \bmod s$
        if $part_u = part_v$ **then**
            $\overline{G}[u, v] \leftarrow$ true
            $a \leftarrow a + 1$
        end if
    end for
end for
$a_{max} \leftarrow \binom{n}{2} - \binom{s}{2}$
$a_{soll} \leftarrow \binom{n}{2} \cdot (1 - d)$
while $a < \min(a_{max}, a_{soll})$ **do** ▷ Zufällige Kanten einfügen
    $u \leftarrow v$
    while $u = v \vee \overline{G}[u, v]$ **do**
        $u \leftarrow$ zufallsKnoten$(V \setminus C)$
        $v \leftarrow$ zufallsKnoten(V)
    end while
    $\overline{G}[u, v] \leftarrow$ true
    $a \leftarrow a + 1$
end while

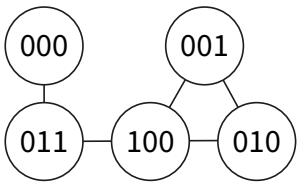
4.1.4 Hamming Graphen

Aus der Kodierungstheorie ist der sogenannte Hamming-Abstand ein bekanntes Mittel zur Fehlererkennung und -behebung. Dieser Abstand beschreibt die Anzahl an Bits, in denen sich zwei binäre Worte voneinander unterscheiden. Dabei können beispielsweise mit einem Abstand von d bis zu $d-1$ Fehler erkannt und $\lfloor \frac{d-1}{2} \rfloor$ Fehler korrigiert werden [MS77].
Möchte man eine Menge binärer Wörter finden, die zueinander stets einen Hammingabstand von d haben, so kann dies mithilfe des Findens einer maximalen Clique in einem entsprechenden Graphen modelliert werden [HPV93].

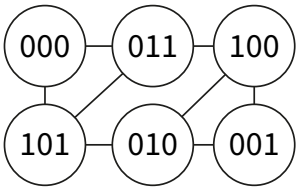
4.1.4.1 Hintergrund

Zum Finden einer solchen maximalen Clique für binäre Wörter der Länge $l \in \mathbb{N}$ wird ein Graph derart konstruiert, dass jeder Knoten eines der möglichen binären Worte in Form dessen Dezimaldarstellung repräsentiert. Dabei werden zwei Knoten $u, v \in V$ genau dann verbunden, wenn die binäre Darstellung ihres Dezimalwertes einen Hammingabstand $d_{u,v} \geq d$ aufweist (siehe Abbildung 4.2). [PX94]
Im Kontext des Konstruierens mit bekannter Lösung wurden insbesondere Graphen mit einer Hammingdistanz von $d = 2$ betrachtet, da dabei der Graph in genau zwei Gruppen unterteilt wird, eine mit allen binären Worten mit gerader Anzahl gesetzter Bits,

die andere mit ungerader Anzahl. Die maximale Clique ist dann diejenige Gruppe mit den meisten Knoten. Für Abstände $d > 2$ konnten in dieser Arbeit keine Regeln aus den Strukturen für die Zusammensetzung der Cliquen abgeleitet werden. In Kapitel 5.1.4 wird jedoch für $d = 3$ eine empirisch hergeleitete Formel für die Größe der maximalen Clique vorgestellt.
Das in [HPV93] vorgestellte Verfahren erzeugt nur Graphen mit einer Knotenanzahl von $n = 2^l$ für eine gegebene Wortlänge l. Im Rahmen dieser Arbeit wurde das Verfahren so angepasst, dass das Erzeugen für ein beliebiges $n \in \mathbb{N}$ möglich ist, durch Weglassen der letzten $2^{\lceil \log_2(n) \rceil} - n$ Knoten.



(a) Struktur eines Hamming-Graph mit $n = 5$ und $d = 2$



(b) Struktur eines Hamming-Graph mit $n = 6$ und $d = 2$

Abbildung 4.2: Struktur von Hamming-Graphen für $n = 5$ (4.2a) und $n = 6$ (4.2b) bei einer Hammingdistanz von $d = 2$.

4.1.4.2 Problemgenerator

Aufgrund der Modifizierung zu beliebigen Knotenzahlen wird als erstes aus der Knotenzahl n die Wortlänge $l = \lceil \log_2(n) \rceil$ berechnet. Anschließend wird für jedes Knotenpaar $(u, v) \in V^2$ der Hammingabstand $d_{u,v}$ berechnet und das Paar verbunden, wenn $d_{u,v} \geq 2$. Anschließend kann die maximale Clique durch Aufzählen der beiden Gruppen bestimmt werden (siehe Algorithmus 4).

4.1.5 Brock Graphen

In den vorangehenden Kapiteln wurden vor allem strukturierte Ansätze betrachtet. Die nachfolgend vorgestellten Verfahren nutzen dagegen verstärkt Zufallskomponenten, um den Graphen zu erstellen. Bei solchen Verfahren wird häufig auf die sogenannte Hintergrund-Clique zurückgegriffen, welche in Kapitel 2.7 erläutert wurde.
Ein historisch bekannter Ansatz ist der sogenannte DELTA Graph Generator, im Nachfolgenden auch als “Brock” bezeichnet, nach Brockington und Culberson [BC93].
Die Idee ist dabei ähnlich wie beim Sanchis-Ansatz (vergleiche Kapitel 4.1.3): Es wird versucht, eine vorher definierte Clique in einem Graphen mit zufälligen weiteren Kanten zu verstecken. Das Verfahren legt den Fokus jedoch auf das Verstecken von Cliquen (wesentlich) größer als die Größe der Hintergrund-Clique.

4.1.5.1 Hintergrund

Das Verfahren nach Brockington und Culberson nutzt eine Konstruktion über das Maximum-Independent-Set-Problem (siehe Kapitel 2.6), welches im komplementären Graphen dem Maximum-Clique-Problem entspricht. Dabei wird versucht, den Knotengrad von

Algorithm 4 Hamming Generator ($d = 2$) für das Maximum-Clique-Problem

Require: $n > 0$
1: $l \leftarrow \lceil \log_2(n) \rceil$
2: **for** $u = 0$ **to** $n - 2$ **do** ▷ Knoten verbinden
3: **for** $v = u + 1$ **to** $n - 1$ **do**
4: $dist \leftarrow 0$
5: **for** $pos = 0$ **to** $l - 1$ **do** ▷ Hammingdistanz berechnen
6: **if** $\lfloor \frac{u}{2^{pos}} \rfloor \bmod 2 \neq \lfloor \frac{v}{2^{pos}} \rfloor \bmod 2$ **then**
7: $dist \leftarrow dist + 1$
8: **end if**
9: **end for**
10: **if** $dist \geq 2$ **then**
11: $G[u, v] \leftarrow$ true
12: **end if**
13: **end for**
14: **end for**
15: $V_{\text{gerade}} \leftarrow \{\}$
16: $V_{\text{ungerade}} \leftarrow \{\}$
17: **for** $u = 0$ **to** $n - 1$ **do** ▷ Maximale Clique ermitteln
18: **if** anzahlEinsen(u) $\bmod 2 = 0$ **then**
19: $V_{\text{gerade}} \leftarrow V_{\text{gerade}} \cup \{u\}$
20: **else**
21: $V_{\text{ungerade}} \leftarrow V_{\text{ungerade}} \cup \{u\}$
22: **end if**
23: **if** $|V_{\text{gerade}}| \geq |V_{\text{ungerade}}|$ **then**
24: $C \leftarrow V_{\text{gerade}}$
25: **else**
26: $C \leftarrow V_{\text{ungerade}}$
27: **end if**
28: **end for**

Knoten außerhalb der versteckten Clique basierend auf einer sogenannten Versteckungstiefe h so anzugleichen, dass die Knoten der Clique schwerer anhand ihres Knotengrads als potenzielle Mitglieder zu identifizieren sind. [BC93]

4.1.5.2 Problemgenerator

Der Generator erzeugt als erstes eine beliebige Knotenmenge C mit $|C| = s$, welche die größte unabhängige Knotenmenge (beziehungsweise die größte Clique) repräsentiert.
Anschließend werden basierend auf der Knotenzahl n, der Größe der versteckten Clique s, der Zieldichte d und einem Versteckungsparameter $h \in [0; 4]$ zwei Dichten p_0 und p_1 für den Graphen $\overline{G}$ berechnet. p_0 definiert die Dichte der Verbindungen von je zwei Knoten $v_1, v_2 \in V \setminus C$, also außerhalb der unabhängigen Menge. p_1 die Dichte an Verbindungen von Knoten $v_1 \in V \setminus C$ zu Knoten $v_2 \in C$. Für eine Versteckungstiefe von $h = u + 1$ beschreibt I_u die Menge von noch nicht gefundenen Knoten aus C, wenn bereits u Knoten korrekt aus C identifiziert wurden. $\overline{I}_u$ die Menge der Knoten, die noch zu durchsuchen

sind, also keine Verbindung zu bereits gefundenen Knoten haben und nicht in I_u sind. Die Wahrscheinlichkeiten p_0 und p_1 sollen so berechnet werden, dass nach u korrekt gefundenen Knoten ein Lösungsalgorithmus keine Informationen aus den zu erwartenden Knotengraden μ beider Gruppen ziehen kann, also $\mu(I_u) = \mu(\overline{I}_u)$. [BC93]
Der erwartete Anfangsknotengrad für Knoten in $C = I_0$ setzt sich aus der Anzahl statistischer Verbindungen zu Knoten in $\overline{I}_0 = V \setminus I_0$ zusammen:

$$\mu(I_0) = p_1 \cdot (n - s) \tag{4.4}$$

Der für Knoten in $\overline{I}_0$ aus der Anzahl statistischer Verbindungen zu anderen Knoten in $\overline{I}_0$ und den Verbindungen zu Knoten in I_0:

$$\mu(\overline{I}_0) = p_0 \cdot (n - s - 1) + p_1 \cdot s \tag{4.5}$$

Da man jedoch den Erwartungswert nach u korrekt gefundenen Knoten aus I_0 betrachtet und durch jeden korrekt gefundenen Knoten die Anzahl der Knoten in $\overline{I}_0$ um den Faktor $(1 - p_1)$ verringert wird, muss der erwartete Knotengrad beider Gruppen somit noch um $(1 - p_1)^u$ korrigiert werden. Außerdem entfallen die u gefundenen Knoten aus I_0 als mögliche Verbindungspunkte für Verbindungen von Knoten aus $\overline{I}_0$.

$$\mu(I_u) = p_1 \cdot (n - s) \cdot (1 - p_1)^u \tag{4.6}$$

$$\mu(\overline{I}_u) = p_0 \cdot ((n - s) \cdot (1 - p_1)^u - 1) + p_1 \cdot (s - u) \tag{4.7}$$

Da über $\overline{G}$ insgesamt eine Dichte von $p = 1 - d$ gelten soll, lässt sich der durchschnittliche Knotengrad t der verbliebenen Knoten nach u korrekt gefundenen Knoten wie folgt definieren:

$$t = p \cdot ((s - u) + (n - s)(1 - p_1)^u - 1) \tag{4.8}$$

Aus $\mu(I_u) = \mu(\overline{I}_u) = t$ folgt dann

$$\begin{aligned} t &= \mu(I_u) \\ 0 &= \mu(I_u) - t \\ &= p_1(n - s)(1 - p_1)^u - p((s - u) + (n - s)(1 - p_1)^u - 1) \\ &= (p_1 - p)(n - s)(1 - p_1)^u - p(s - u - 1) \end{aligned} \tag{4.9}$$

Hieraus lässt sich p_1 bei festem u, p, n und s numerisch bestimmen. p_0 ergibt sich anschließend aus $\mu(\overline{I}_u) = \mu(I_u)$:

$$p_0 = p_1 \frac{(n - s)(1 - p_1)^u - (s - u)}{(n - s)(1 - p_1)^u - 1} \tag{4.10}$$

Basierend auf diesen Dichten werden dann alle Knoten aus $V \setminus C$ untereinander bzw. zu Knoten in C verbunden (siehe Algorithmus 5). Eine ausführlichere Herleitung findet sich in der entsprechenden Arbeit von Brockington und Culberson [BC93].

4.1.6 Zufallsgraphen und Hintergrund-Clique

Bei der Untersuchung des Brock-Verfahrens zeigte sich, dass auch für eine Versteckungstiefe von $h = 0$, also ohne Verwendung einer Versteckungslogik, die Graphen nicht leichter

Algorithm 5 Brock Generator für das Maximum-Clique-Problem

Require: $n \geq s \geq 0, \quad 0 \leq d \leq 1, \quad 0 \leq h \leq 4$

$C \leftarrow$ Wähle s zufällige Knoten aus V
$u \leftarrow h - 1$
$p_1 \leftarrow \text{berechneP1}(n, 1 - d, s, u)$
$p_0 \leftarrow \text{berechneP0}(n, s, u, p1)$
for all $v_1 \in V \setminus C$ **do**
    for all $v_2 \in V \setminus C$ **do** ▷ Verbinde zu Knoten außerhalb C
        if $v_1 < v_2 \wedge \text{zufall}() < p_0$ **then**
            $\overline{G}[u, v] \leftarrow \text{true}$
        end if
    end for
    $a_{soll} \leftarrow s \cdot \lfloor p_1 \rceil$
    while $a_{soll} > 0$ **do** ▷ Verbinde zu Knoten in C
        while *true* **do**
            $v_2 \leftarrow \text{zufallsKnoten}(C)$
            if $\overline{G}[u, v] = \text{false}$ **then**
                $\overline{G}[u, v] \leftarrow \text{true}$
                $a_{soll} \leftarrow a_{soll} - 1$
                break
            end if
        end while
    end while
end for

zu lösen waren als mit Verstecken. Es stellt sich daher die Frage, ob schlicht das Einsetzen einer größeren Clique als die Hintergrund-Clique zu vergleichbaren Ergebnissen führen kann.

4.1.6.1 Hintergrund

Die Idee ist, dass allein aufgrund der Zufallskomponente in zufälligen Graphen vereinzelt Knoten mit demselben oder einem höheren Grad als der von Knoten der eingesetzten Clique C auftreten und daher automatisch zu einem gewissen "Verstecken" führen.
Setzt man nun eine Clique C um s_o größer als die errechnete Größe der Hintergrund-Clique ein, könnte diese unter Umständen als optimale Lösung erhalten bleiben.

4.1.6.2 Problemgenerator

Der Generator (im Nachfolgenden auch als "SynA3" bezeichnet) erzeugt als Erstes, wie in Algorithmus 6 dargestellt, einen beliebigen Zufallsgraphen mit n Knoten und einer Dichte d. Anschließend wird die Größe s_b der theoretischen Hintergrund-Clique errechnet und darauf basierend eine zufällige Knotenmenge der Größe $s_b + s_o$ als maximale Clique C in den Graphen eingesetzt. Dabei muss lediglich ein Teil der benötigten Kanten eingefügt werden, da die Knoten der Clique bereits Teil des Zufallsgraphen sind.

Algorithm 6 SynA3 Generator für das Maximum-Clique-Problem

Require: $n \geq s_o \geq 0, \quad 0 \leq d \leq 1$

$a_{zus} \leftarrow d \cdot \text{maxKanten}(G)$ ▷ Erstelle Zufallsgraphen
while $a_{zus} > 0$ **do**
    $u \leftarrow \text{zufallsKnoten}(V)$
    $v \leftarrow \text{zufallsKnoten}(V)$
    if $u \neq v \wedge G[u, v] = false$ **then**
        $G[u, v] \leftarrow \text{true}$
        $a_{zus} \leftarrow a_{zus} - 1$
    end if
end while
$s_b \leftarrow \text{berechneBackgroundClique}(n, d)$
$C \leftarrow$ Wähle $s_b + s_o$ zufällige Knoten aus V
$\text{verbindeKnoten}(C)$

4.1.7 Johnson- und Keller-Graphen

Der Vollständigkeit wegen seien hier noch die bekannten Johnson- und Keller-Graphen erwähnt.
Die Johnson-Graphen stammen, ähnlich wie die Hamming-Graphen (siehe Kapitel 4.1.4), aus dem Bereich der Kodierungstheorie. Bei diesen werden gewichtete binäre Wörter betrachtet, also nur Worte, die stets w gesetzte Bits beinhalten. Zwei Knoten werden ab einem gegebenen Hamming-Abstand d miteinander verbunden [HPV93].
Mit diesem Verfahren konnten jedoch nur Probleme generiert werden, deren maximale Clique trivial oder, im Rahmen dieser Arbeit, nach keinem Muster bestimmbar war. Zudem lassen die Konstruktionsparameter, ähnlich wie beim Hamming-Ansatz, nur eine geringe Variation der Struktur zu.
Die Keller-Graphen entstanden aus Überlegungen bezüglich Keller's Vermutung [Kel30], dass sich bei der Aufteilung jedes n-dimensionalen Raumes in uniforme "Würfel" mindestens zwei "Würfel" an einer $(n-1)$-dimensionalen "Fläche" vollständig berühren. Die Gültigkeit der Vermutung konnte für viele Dimensionen über ein entsprechendes Clique-Problem bewiesen oder widerlegt werden, zuletzt auch für die verbliebene 7. Dimension [BHMN20].
Die Konstruktion der Kellergraphen erfolgt über eine gegebene Wortgröße v, aus der ein Graph mit $n = 4^v$ entsteht [HPV93]. Bereits daraus lässt sich ableiten, dass ein Generator nach diesem Verfahren ungeeignet ist, da die Knotenzahl sehr schnell wächst und zudem bisher keine Systematik zur Ableitung einer optimalen Lösung bekannt ist.

4.2 Hamiltonian-Cycle-Problem

4.2.1 Intuitiver Ansatz - Hamiltonsch

Wie beim vorangehend untersuchten Maximum-Clique-Problem 4.1.1 soll auch für das Hamiltonian-Cycle-Problem zunächst ein intuitiver Ansatz als Probe dienen, ob das Erzeugen von Probleminstanzen mit bekannter Lösung grundsätzlich möglich ist.

4.2.1.1 Idee

Eine triviale Möglichkeit für das Erzeugen eines garantiert hamiltonschen Graphen ist das Verbinden aller aufeinanderfolgender Knoten inklusive einer Verbindung wieder zum Ursprungsknoten. Die Gültigkeit dieser Lösung kann durch Hinzufügen weiterer zufälliger Kanten nicht eingeschränkt werden. Dies soll daher als ein erster Einstiegspunkt dienen.

4.2.1.2 Problemgenerator

Wie in Algorithmus 7 dargestellt, wird zunächst eine beliebige Knotenfolge C bestimmt, welche alle Knoten einmal beinhaltet, bis auf den ersten. Dieser wird zusätzlich ans Ende gestellt, um einen geschlossenen Zyklus zu erzeugen. Anschließend werden je zwei aufeinanderfolgende Knoten im Graphen verbunden.
Abschließend werden zufällige Kanten entsprechend der vorgegebenen Dichte gesetzt.

Algorithm 7 Intuitiver Generator (hamilt.) für das Hamiltonian-Cycle-Problem

Require: $n \geq 0, \quad 0 \leq d \leq 1$
1: $C \leftarrow$ Alle Knoten in zufälliger Reihenfolge aus V
2: $C \leftarrow C + C[0]$ ▷ Erster Knoten schließt am Ende
3: **for** $i = 0$ **to** $n - 1$ **do**
4: $u \leftarrow C[i]$
5: $v \leftarrow C[i + 1]$
6: $G[u, v] \leftarrow$ true
7: **end for**
8: $a_{zus} \leftarrow d \cdot \text{maxKanten}(G) - n$ ▷ Anzahl zusätzlicher Kanten
9: **while** $a_{zus} > 0$ **do**
10: $u \leftarrow \text{zufallsKnoten}(C)$
11: $v \leftarrow \text{zufallsKnoten}(C)$
12: **if** $u \neq v \wedge G[u, v] = \text{false}$ **then**
13: $G[u, v] \leftarrow$ true
14: $a_{zus} \leftarrow a_{zus} - 1$
15: **end if**
16: **end while**

4.2.2 Intuitiver Ansatz - Nicht hamiltonsch

Im Kapitel 4.2.1 wurde ein Ansatz vorgestellt, welcher stets hamiltonsche Probleminstanzen für das Hamiltonian-Cycle-Problem erzeugt. Im Folgenden soll nun ein Ansatz betrachtet werden, welcher das Erzeugen nicht-hamiltonscher Probleminstanzen ermöglicht.

4.2.2.1 Idee

Das Erzeugen nicht-hamiltonscher Graphen ist nicht ganz so einfach wie das Erzeugen hamiltonscher, da sichergestellt werden muss, dass durch das Hinzufügen weiterer (zufälliger) Kanten kein Zyklus entsteht.
Es muss also eine Struktur in den Graphen implementiert werden, welche den Graph

grundsätzlich nicht-hamiltonsch macht, unabhängig beliebig hinzugefügter Kanten.
Eine solche Struktur zeigt Abbildung 4.3, welche aus praktischen Überlegungen hervorgegangen ist.

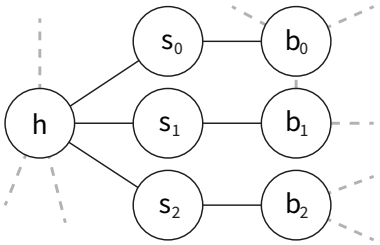


Abbildung 4.3: Flaschenhals-Struktur in SynH2-Graphen. Knoten $h \in V$ stellt sicher, dass maximal 2 der 3 Schulterknoten $s_i \in V$, $i \in \{0, 1, 2\}$ durchlaufen werden können.

Da jeder Schulterknoten s_i nur über h erreicht beziehungsweise verlassen werden kann, ist es nie möglich, alle drei Schulterknoten zu durchlaufen, da h nicht mehrfach durchlaufen werden darf. Ein Graph, der diese Struktur beinhaltet, kann also nie hamiltonsch sein.
Anschließend können beliebige Kanten dem Graphen hinzugefügt werden, insofern sichergestellt ist, dass keine außer den dargestellten Kanten zu den Knoten s_i führen.
Für die Knoten h und b_i bestehen keine Einschränkungen; die Bodenknoten $b_i \in V$, $i \in \{0, 1, 2\}$ können sogar auf einen einzigen Bodenknoten reduziert werden.

4.2.2.2 Problemgenerator

Wie Algorithmus 8 zeigt, werden zunächst 7 unterschiedliche Knoten für die Flaschenhalsstruktur ausgewählt. Der erste Knoten stellt den eigentlichen Flaschenhals dar, die verbleibenden 6 Knoten werden der Reihe nach in je einen Schulter- und einen Bodenknoten unterteilt. Anschließend wird der Flaschenhals-Knoten mit allen Schultern und die Schultern mit ihren entsprechenden Böden verbunden. Als letztes werden dann basierend auf der vorgegebenen Dichte weitere Kanten dem Graphen hinzugefügt. Dabei werden die Schulterknoten als mögliche Endknoten ausgeschlossen.

4.2.3 Petersen Graphen

Nach den zuvor gezeigten Intuitiven Ansätzen soll abschließend das Generieren von Probleminstanzen für bekannte Lösungen anhand der historisch bekannten Graphenfamilie der Allgemeinen Petersen-Graphen nach Watkins [Wat69] erfolgen. Für diese konnte Alspach [Als83] vervollständigend zeigen, dass diese für bestimmte Konstruktionsparameter immer hamiltonsch bzw. nicht hamiltonsch sind.

4.2.3.1 Hintergrund

Allgemeine Petersen-Graphen sind von ihrer Struktur immer ähnlich aufgebaut. Während die eine Hälfte der Knoten der Reihe nach zyklisch verbunden wird (regelmäßiges Polygon), wird die andere Hälfte zu einer Art Sternpolygon verbunden. Anschließend werden die zueinander gehörigen Eckpunkte beider Polygone miteinander verbunden.

Algorithm 8 Intuitiver Generator (nicht hamilt.) für das Hamiltonian-Cycle-Problem

Require: $n \geq 7, \quad 0 \leq d \leq 1$
1: $F \leftarrow$ Wähle 7 zufällige Knoten aus V
2: $h \leftarrow F[0]$
3: $X \leftarrow V$ ▷ Alle Knoten außer Schulterknoten
4: **for all** $i \in \{1, 3, 5\}$ **do**
5: $\quad s \leftarrow F[i]$
6: $\quad b \leftarrow F[i+1]$
7: $\quad G[h, s] \leftarrow$ true
8: $\quad G[s, b] \leftarrow$ true
9: $\quad X \leftarrow X \setminus \{s\}$ ▷ Keine weiteren Verbindungen zu Schulterknoten
10: **end for**
11: $a_{zus} \leftarrow d \cdot \text{maxKanten}(G) - 6$ ▷ Anzahl zusätzlicher Kanten
12: **while** $a_{zus} > 0$ **do**
13: $\quad u \leftarrow \text{zufallsKnoten}(X)$
14: $\quad v \leftarrow \text{zufallsKnoten}(X)$
15: $\quad$ **if** $u \neq v \wedge G[u, v] =$ false **then**
16: $\quad\quad G[u, v] \leftarrow$ true
17: $\quad\quad a_{zus} \leftarrow a_{zus} - 1$
18: $\quad$ **end if**
19: **end while**

Die exakte Formation des Sternpolygons wird durch einen Parameter $k \in \mathbb{N}$ bestimmt. Dieser gibt an, mit dem wie vielten nächsten Knoten $v_{i+k} \in V_{Stern}$ ein Knoten $v_i \in V_{Stern}$ verbunden wird. Die Knotennummerierung innerhalb einer Gruppe wird dabei immer modulo n betrachtet.

4.2.3.2 Problemgenerator

Zum Erzeugen eines allgemeinen Petersen-Graphen $\text{GP}(n, k)$ werden zunächst zwei gleich große Knotengruppen U und V mit $|U| = |V| = n$ gebildet. Anschließend werden aus U je die Knoten $u_i, u_{i+1} \in U$ für $i \in \{0, \ldots, n-1\}$ mit Indizes modulo n verbunden. Aus V werden nach ähnlichem Schema je die Knoten $v_i, v_{i+k} \in V,\ i \in \{0, \ldots, n-1\}$ mit Indizes modulo n verbunden. Zuletzt erfolgt das Verbinden der Eckpunkte beider Polygone mit einer Verbindung zwischen je $u_i, v_i \in U \cup V,\ i \in \{0, \ldots, n-1\}$.

Ein allgemeiner Petersen-Graph ist dann immer hamiltonsch, außer für:

- $n \equiv 5 \mod 6 \quad$ für $\quad k \in \{2, n-2, \frac{n-1}{2}, \frac{n+1}{2}\}$
- $n \equiv 0 \mod 4 \quad$ für $\quad n \geq 8 \wedge k = \frac{n}{2}$

[Als83]
Als konkrete Eingabe für den Problemgenerator (siehe Algorithmus 9) wurde die Gesamtknotenzahl $n_{ges} = 2 \cdot n$ gewählt, um dem Nutzer ein Umrechnen der eigentlichen Knotenzahl in den Parameter n zu ersparen.

Algorithm 9 Petersen Generator für das Hamiltonian-Cycle-Problem

Require: $n_{ges} \geq 4, n_{ges} \equiv 0 \mod 2, \quad 1 \leq k \leq \frac{n_{ges}}{2} - 1$

1: $n \leftarrow \frac{n_{ges}}{2}$
2: **for** $i = 0$ **to** $n - 1$ **do**
3: $u_1 \leftarrow i$
4: $u_2 \leftarrow (i + 1) \mod n$
5: $G[u_1, u_2] \leftarrow \text{true}$
6:
7: $v_1 \leftarrow i + n$
8: $v_2 \leftarrow (i + k \mod n) + n$
9: $G[v_1, v_2] \leftarrow \text{true}$
10:
11: $G[u_1, v_1] \leftarrow \text{true}$
12: **end for**

5 Evaluation

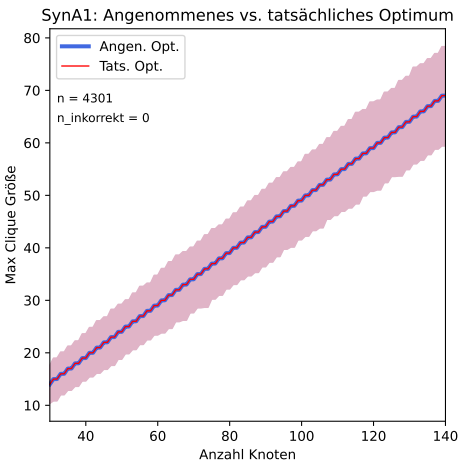
Im Folgenden sollen die im vorangegangenen Kapitel vorgestellten Ansätze und Ideen bezüglich ihrer Korrektheit, Komplexität und Struktureigenschaften ausgewertet werden. Die Korrektheit wird anhand des Anteils korrekt erzeugter Probleme basierend auf der angenommenen optimalen Lösung an der Gesamtzahl der Probleme bestimmt. Ein Wert von 0.78 bedeutet also, dass in 78% der Fälle der Generator ein Problem vorgeschlagen hat, dessen optimale Lösung der gewünschten Lösung entspricht. Je höher dieser Wert, desto verlässlicher und besser der Generator.
Die Komplexität eines Problems wird anhand der benötigten Rechenzeit (in Sekunden) gemessen, die ein Lösungsprogramm benötigt, um die Lösung des Problems (exakt) zu bestimmen. Je höher die benötigte Laufzeit, desto komplexer das Problem und dementsprechend besser der Generator. Jede Instanz wird dabei mit einer zufällig generierten Instanz derselben Dichte und Knotenzahl verglichen, da diese aufgrund ihrer erhöhten Schwierigkeit als gute Vergleichsnorm dienen. Nach Möglichkeit wurden in der Regel je zwei Lösungsalgorithmen verwendet, CliSAT [SSFÁP23] und Fast Max-Cliquer [PPG$^+$15] für das Maximum-Clique-Problem, Concorde [ABCC11] und LKH [Hel00] für das Hamiltonian-Cycle-Problem. Da die Knotenzahlunabhängigen Verläufe jeweils nicht sonderlich voneinander abwichen, wurden die nachfolgenden Darstellungen auf je einen Verlauf reduziert.
Bei den Struktureigenschaften soll insbesondere auf die Freiheit bezüglich der Wahl von Dichte, Knotenzahl und Größe der maximalen Clique eingegangen, aber auch problemspezifische Eigenschaften aufgezeigt werden. Grundsätzlich gilt hier: je mehr Freiheitsgrade, desto besser.

5.1 Maximum-Clique-Problem

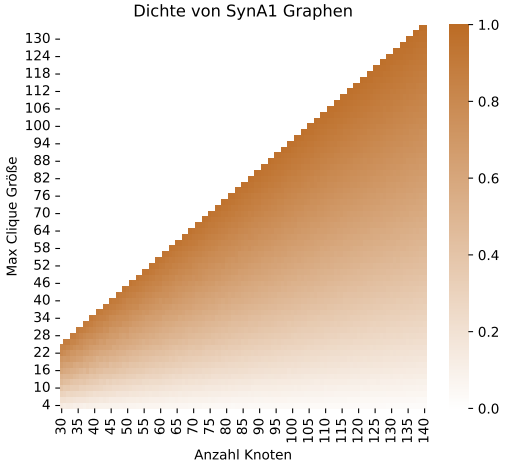
5.1.1 Intuitiver Ansatz

Zur Untersuchung des Intuitiven Ansatzes (im Nachfolgenden auch als “SynA1” bezeichnet) wurden insgesamt 4301 Probleminstanzen mit einer Knotenzahl im Bereich $n \in \{30, \ldots, 140\}$ generiert, mit je eingesetzten Cliquen der Größe $s \in \{4, 6, 8, \ldots, n-5\}$. Ziel war dabei, Instanzen mit maximaler Dichte zu erzeugen.
Bezüglich der Korrektheit bestätigt die empirische Auswertung die Theorie aus 4.1.1, denn wie aus Abbildung 5.1a zu entnehmen ist, bleibt nach dem Generieren zusätzlicher Kanten die Optimalität der ursprünglich eingesetzten Lösung für alle Instanzen erhalten. Wie aus der Formel für die Obergrenze der Dichte einer Instanz (4.1) zu entnehmen ist, zeigt Abbildung 5.1b jedoch, dass die maximale Dichte insbesondere für Instanzen mit kleiner eingesetzter Clique im Verhältnis zur Gesamtknotenzahl gering ist und somit in dieser Eigenschaft eine gewisse Einschränkung in der Wahlfreiheit herrscht.

Zudem zeigt sich bei der Auswertung der benötigten Zeit zum Lösen solcher Instanzen eine verhältnismäßig geringe Komplexität, wie aus Abbildung 5.2a zu entnehmen. Im Vergleich zu zufällig generierten Instanzen benötigen Instanzen des Intuitiven Ansatzes



(a) Angenommene und tatsächliche Lösungen für Probleme je Knotenanzahl



(b) Maximale Dichte nach Knotenanzahl und eingebetteter maximaler Clique

Abbildung 5.1: Korrektheit der Instanzen des Intuitiven Ansatzes (5.1a) und deren Dichte in Abhängigkeit von der Knotenzahl und eingesetzten Clique (5.1b).

nur in etwa 0.01% der Zeit zum Lösen, bei fast identischer Dichte pro Knotenzahl (siehe Abbildung 5.2b).

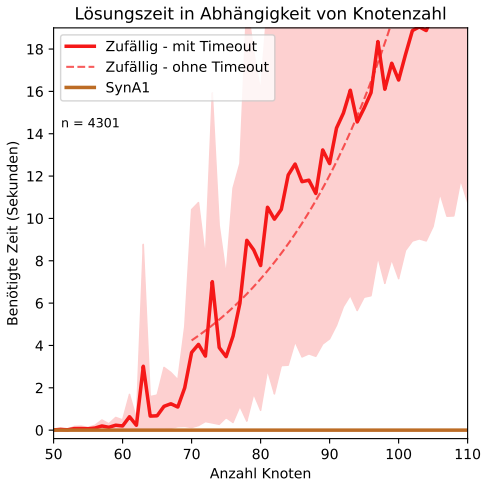
Eine Begründung dafür könnte darin liegen, dass aufgrund der Konstruktion der Graphen alle Knoten in drei Hauptgruppen unterteilt sind. Die erste Gruppe A, bestehend aus Knoten $a \in A$ mit $\text{grad}(a) > s-1$, welche sicher Teil der maximalen Clique sind, einer Gruppe B mit Knoten $b \in B$ und $\text{grad}(b) = s-1$, welche potenziell Teil einer maximalen Clique sind, und einer Gruppe C mit Knoten $c \in C$ und $\text{grad}(c) < s-1$, welche kein Teil einer maximalen Clique sind. Daraus lässt sich schnell ein Teil des Graphen ausschließen und für eine maximale Clique relevante Knoten identifizieren.

Eine zweite, damit einhergehende Begründung ist, dass es gierige Algorithmen besonders einfach bei Instanzen dieser Problemklasse haben, da diese sich oftmals am Grad der Knoten orientieren und daher sehr schnell zu einer optimalen Lösung finden.

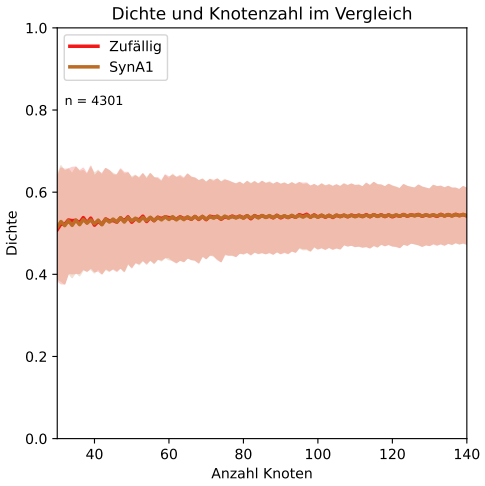
Der Intuitive Ansatz zeigt also, dass das Generieren von künstlichen Probleminstanzen mit bekannter Lösung für das Maximum-Clique-Problem durchaus möglich ist. Der Ansatz selbst erzeugt jedoch nicht sonderlich schwer zu lösende Probleme und hat Einschränkungen in der Wahlfreiheit der Dichte.

5.1.2 C-Fat Graphen

Zur Untersuchung der C-Fat-Graphen wurden insgesamt 9694 Probleminstanzen mit einer Knotenzahl im Bereich $n \in \{30, \ldots, 160\}$ generiert, mit je einem $c \in \{0.1, 0.2, 0.3, \ldots, 7.4\}$. Wie aus Abbildung 5.3a zu entnehmen, wurde die maximale Clique in allen Fällen korrekt abgeleitet. Die Dichte ist jedoch an die Größe der maximalen Clique gekoppelt und, wie in Abbildung 5.3b zu sehen, abhängig von dem Parameter c. Bei konstanter Knotenzahl führt ein niedrigeres c zu mehr Partitionen und dementsprechend zu einer kleineren maximalen Clique und einer geringeren Dichte.

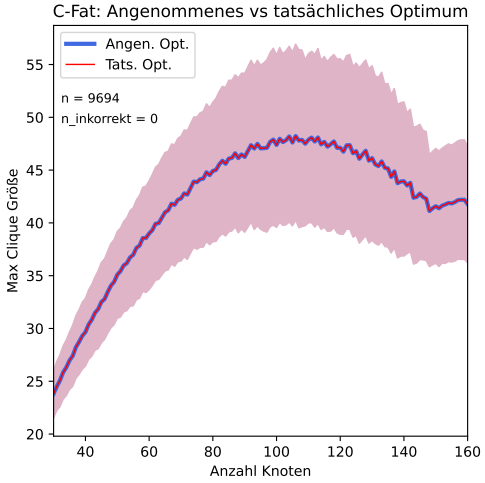


(a) Benötigte Lösungszeit zufälliger Probleme und Instanzen des Intuitiven Ansatzes

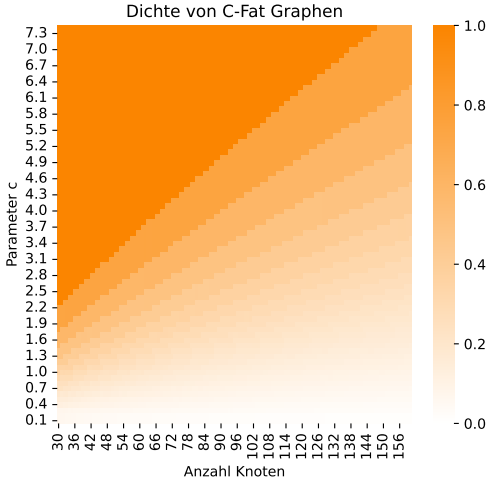


(b) Vergleich Dichte nach Knotenanzahl des Intuitiven Ansatzes und zufälligen Instanzen

Abbildung 5.2: Benötigte Zeit zum Lösen des Intuitiven Ansatzes und zufälligen Instanzen (5.2a), bei vergleichbarer Dichte je Anzahl Knoten (5.2b).



(a) Angenommene und tatsächliche Lösungen für Probleme je Knotenanzahl

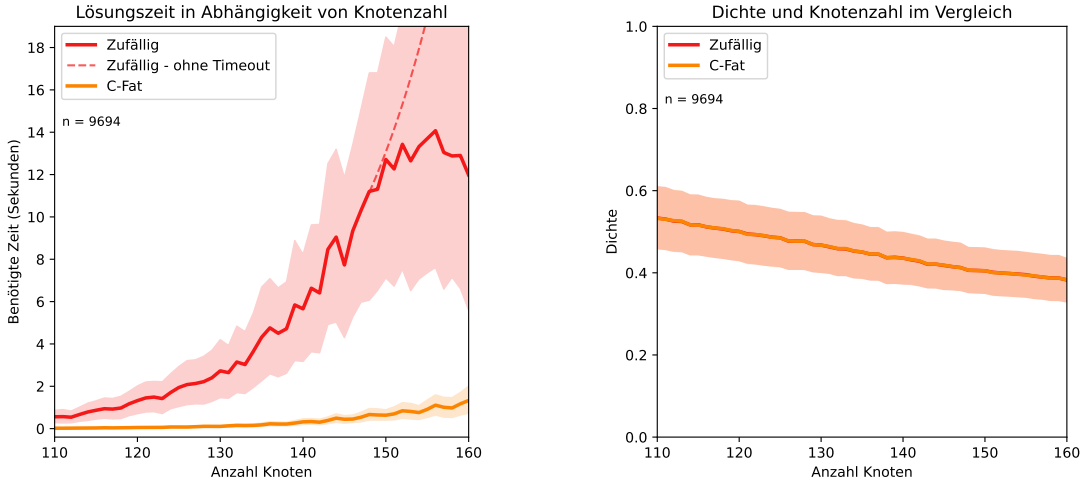


(b) Maximale Dichte nach Knotenanzahl und Parameter c

Abbildung 5.3: Korrektheit der C-Fat Instanzen (5.3a) und deren Dichte in Abhängigkeit von der Knotenzahl und dem Parameter c (5.3b).

Zur Messung der Komplexität wurden die generierten Probleme mit zufälligen Graphen derselben Dichte und Knotenanzahl verglichen (siehe Abbildung 5.4b). Wie Abbildung 5.4a zu entnehmen ist, kommen dabei die C-Fat-Graphen den zufälligen Vergleichsgraphen etwas näher als beispielsweise die Probleme des Intuitiven Ansatzes (C-Fat in etwa 5% der Schwere von Zufallsgraphen im Vergleich zu etwa 0.01% beim Intuitiven Ansatz).

Dennoch sind sie merkbar leichter zu lösen als die zufälligen Instanzen. Eine Begründung dafür könnte darin liegen, dass auch bei dieser Problemklasse oftmals aus dem Knotengrad eine Zugehörigkeit zur maximalen Clique ableitbar ist und Algorithmen dies nutzen können, um wichtige Knoten zu identifizieren.
Zudem gibt es insbesondere für ein n mit $n \bmod k \gg 0$ bzw. $n \bmod k = 0$ viele gleich große maximale Cliquen, was es einer Lösungsprozedur zudem erleichtern kann, schnell eine maximale Clique zu finden.



(a) Benötigte Lösungszeit zufälliger Probleme und Instanzen der C-Fat-Klasse

(b) Vergleich Dichte nach Knotenanzahl der C-Fat-Graphen und zufälligen Instanzen

Abbildung 5.4: Benötigte Zeit zum Lösen von C-Fat-Graphen und zufälligen Instanzen (5.4a), bei vergleichbarer Dichte je Anzahl Knoten (5.4b).

Die Verwendung der C-Fat-Problemklasse als Problemgenerator mit bekannter, optimaler Lösung ist also durchaus möglich. Man ist jedoch in der Wahl der Struktur und somit auch Dichte und Größe der maximalen Clique auf den Parameter c beschränkt. Außerdem sind Instanzen dieses Ansatzes im Vergleich zu zufälligen Graphen merkbar leichter zu lösen.

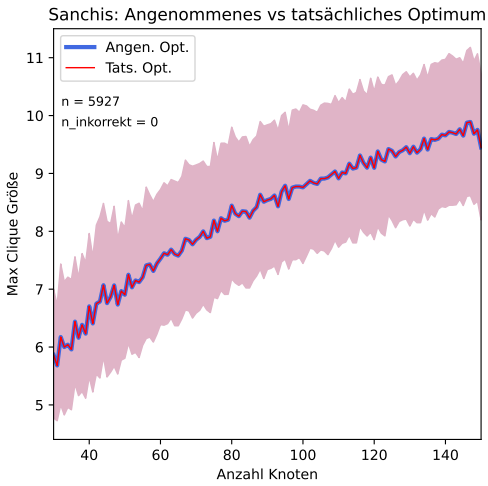
5.1.3 Sanchis Graphen

Zur Messung der Sanchis-Graphen wurden insgesamt 5927 Probleminstanzen mit einer Knotenzahl im Bereich $n \in \{30, \ldots, 150\}$ generiert, mit Dichten und maximalen Clique-Größen entsprechend dem Zufallsgraphen-Datensatz des C-Fat-Experiments (um eine möglichst einheitliche Vergleichsbasis der beiden Verfahren zu schaffen). Ungelöste Zufallsgraphen wurden dabei ausgeschlossen, da bei diesen keine maximale Clique entnommen werden kann.

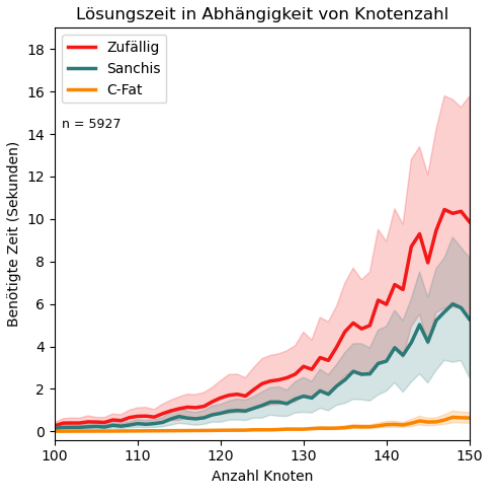
Wie aus Abbildung 5.5a zu entnehmen, bleibt die implementierte maximale Clique in allen Fällen auch nach dem Hinzufügen weiterer Kanten eine optimale Lösung.
Die Dichte ist grundsätzlich frei wählbar und nur in Abhängigkeit von der Knotenanzahl n und der maximalen Cliquegröße s durch die untere Schranke (5.1) und obere Schranke (5.2) begrenzt [San89].

$$d_{\min}(s,n) = 1 - \frac{\binom{n-s}{2} + s(n-s)}{\binom{n}{2}} \tag{5.1}$$

$$d_{\max}(s,n) = 1 - \frac{r\binom{b+1}{2} + (s-r)\binom{b}{2}}{\binom{n}{2}}, \quad \text{wobei } r = n \bmod s \text{ und } b = \left\lfloor \frac{n}{s} \right\rfloor \tag{5.2}$$



(a) Angenommene und tatsächliche Lösungen für Probleme je Knotenanzahl



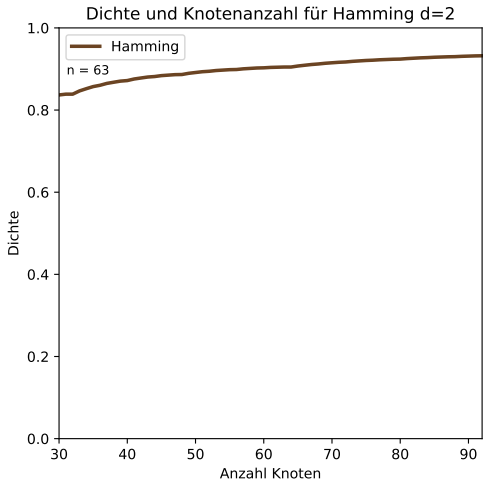
(b) Benötigte Lösungszeit zufälliger Probleme und Instanzen der Sanchis- und C-Fat-Klasse

Abbildung 5.5: Korrektheit der Sanchis-Instanzen (5.5a) und benötigte Zeit zum Lösen im Vergleich zu zufälligen und C-Fat-Graphen (5.5b).

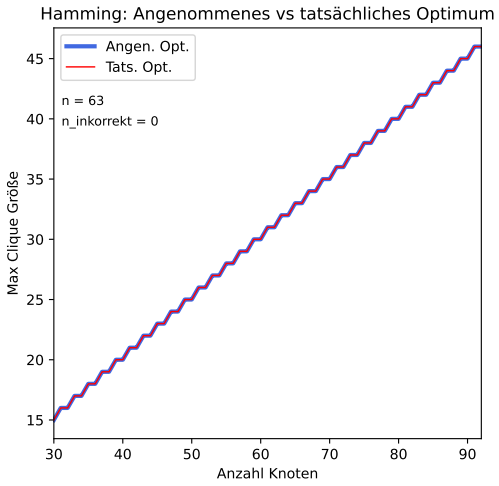
Zur Messung der Komplexität wurden die generierten Probleme mit zufälligen Graphen und C-Fat-Instanzen derselben Dichte und Knotenanzahl verglichen. Wie Abbildung 5.5b zu entnehmen ist, sind die Sanchis-Graphen schwieriger zu lösen als die C-Fat-Graphen, aber leichter als die zufälligen Vergleichsgraphen (Sanchis in etwa 55% der Schwere von Zufallsgraphen im Vergleich zu etwa 5% beim C-Fat-Ansatz).
Die erhöhte Schwierigkeit könnte damit erklärt werden, dass bei diesem Ansatz die Verteilung der Knotengrade breiter gestreut ist und auch über dem Knotengrad der maximalen Clique liegen kann. Eine offensichtliche Lösung ist also nicht mehr garantiert. Außerdem kann die Anzahl an optimalen Lösungen durchaus geringer ausfallen, als beispielsweise beim C-Fat-Verfahren.
Das Verfahren nach Sanchis eignet sich also durchaus, um Probleminstanzen mit bekannter Lösung und moderater Schwierigkeit für das Maximum-Clique-Problem zu generieren, da insbesondere für die Knotenanzahl, Dichte und eingesetzte maximale Clique je ein Parameter zur Verfügung steht. Außerdem können erzeugte Graphen innerhalb derselben Konfiguration aufgrund der Zufallskomponente variieren, was oftmals erwünscht ist.

5.1.4 Hamming Graphen

Zur Messung der Hamming-Graphen wurden insgesamt 63 Probleminstanzen mit einer Knotenzahl im Bereich $n \in \{30, \ldots, 93\}$ generiert. Im Verhältnis zu anderen verglichenen Problemen ist diese Anzahl gering, da ein Hamming-Graph für eine gegebene Knotenzahl stets dieselbe Struktur aufweist und daher keine Variation möglich ist.
Außerdem steigt die Dichte schnell stark an und verweilt auf hohem Niveau (siehe Abbildung 5.6a), da der Graph stets aus zwei vollständig verbundenen Cliquen besteht, welche in sich eine Dichte von 100% haben.
Bezüglich der Korrektheit lassen sich jedoch die Überlegungen aus 4.1.4 bestätigen, für alle generierten Instanzen war die berechnete Lösung korrekt (siehe Abbildung 5.6b).



(a) Dichte von Hamming-Graphen mit $d = 2$



(b) Angenommene und tatsächliche Lösungen für Probleme je Knotenanzahl

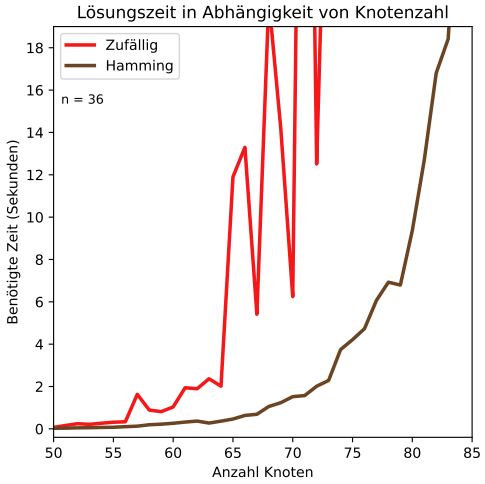
Abbildung 5.6: Dichte je Anzahl Knoten (5.6a) und Korrektheit der Hamming-Instanzen für $d = 2$ (5.6b).

Zur Messung der Komplexität wurden die generierten Probleme mit zufälligen Graphen derselben Dichte und Knotenanzahl verglichen. Wie Abbildung 5.7a zu entnehmen ist, sind die Hamming-Graphen nicht trivial zu lösen, im Vergleich zu zufälligen Graphen jedoch leicht (in etwa 8% der Schwere von Zufallsgraphen).
Für Hammingabstände von $d \geq 3$ konnte aus den Strukturen keine Regel für die Zusammensetzung der Cliquen abgeleitet werden. Experimente für $d = 3$ ergaben jedoch ein Muster für die Größe der maximalen Clique (siehe Abbildung 5.7b), woraus sich die Formel (5.3) empirisch herleiten ließ. Der erste Summand beschreibt dabei die längere Hauptstufe, welche ab 5 Knoten alle 16 Knoten um 2 erhöht wird. Der zweite Summand die zusätzliche kleine Stufe nach 12 Knoten innerhalb jeder Hauptstufe.

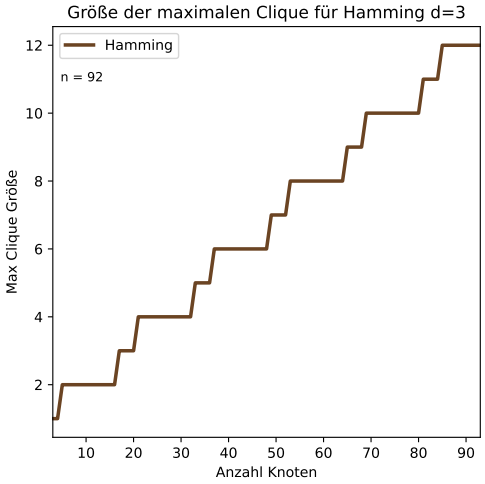
$$|C| = \begin{cases} 2 \cdot (\lfloor \frac{n-5}{16} \rfloor + 1) + \lfloor \frac{(n-5) \mod 16}{12} \rfloor & \text{für } n \geq 5 \\ 1 & \text{sonst} \end{cases} \tag{5.3}$$

Das Generieren von Probleminstanzen mit bekannter Lösung nach dem Hamming-Ansatz eignet sich also nur bedingt, da insbesondere bei der Dichte und Variation der Struktur

keine Freiheit besteht. Des Weiteren sind die erzeugten Graphen nicht außergewöhnlich komplex im Verhältnis zu zufälligen Graphen.



(a) Benötigte Lösungszeit zufälliger Probleme und Hamming-Graphen für $d = 2$

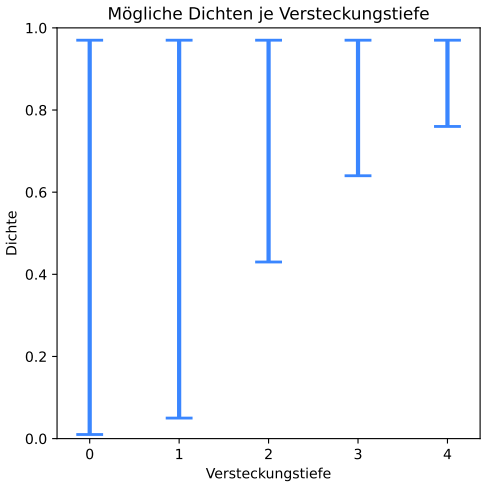


(b) Größe der maximalen Clique bei Hamming-Graphen für $d = 3$

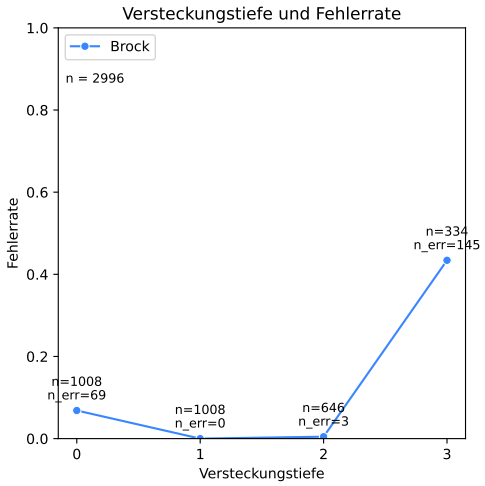
Abbildung 5.7: Benötigte Lösungszeit von Hamming-Graphen für $d = 2$ im Vergleich zu zufälligen Instanzen (5.7a) und empirische Analyse der Größe der maximalen Clique für $d = 3$ (5.7b).

5.1.5 Brock Graphen

Zur Messung der Brock-Graphen wurden 2996 Probleminstanzen mit einer Knotenzahl im Bereich $n \in \{80, \ldots, 135\}$ generiert, mit je 6 Graphen pro Dichte $d \in \{0.3, 0.5, 0.75\}$ und einer versteckten Clique, die 2 Knoten größer ist als die Hintergrund-Clique. Das Einsetzen kleinerer Cliquen als die Hintergrund-Clique führt bei dem Verfahren zu einer hohen Fehlerrate, da durch die Konstruktion über zufällige Zusatzkanten meist automatisch eine Clique dieser Größe entsteht.
Der tatsächliche Stichprobenumfang ist kleiner als die Anzahl der Konfigurationsmöglichkeiten, da das Brock-Verfahren insbesondere bei höheren Versteckungstiefen nur für bestimmte Konfigurationen möglich ist. Abbildung 5.8a zeigt beispielsweise, dass die mögliche Dichte mit steigender Versteckungstiefe immer weiter eingeschränkt wird.
Wie aus Abbildung 5.8b zu entnehmen, hängt der Erhalt der implementierten Clique C als optimale Lösung neben der Größe von C auch von der Versteckungstiefe ab. Während eine Versteckungstiefe von $h = 1$ oder $h = 2$ die Fehlerrate bei erzeugten Instanzen verringert, steigt diese für höhere Versteckungstiefen stark an. Dies kann sich damit erklären lassen, dass durch das Angleichen der Knotengrade manche Knoten außerhalb von C ebenfalls auf einen hohen Knoten gehoben werden und daraus potenziell größere Cliquen hervorgehen können.

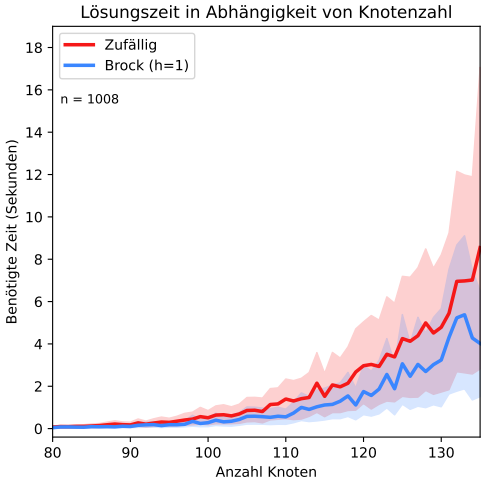


(a) Mögliche Dichten der Brock-Graphen nach Versteckungstiefe

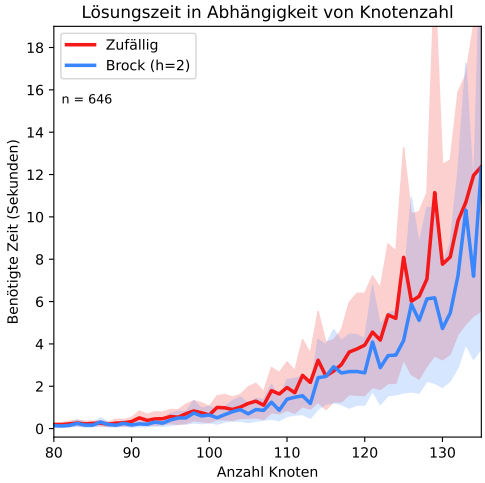


(b) Fehlerrate der Brock-Graphen nach Versteckungstiefe

Abbildung 5.8: Mögliche Dichten nach Versteckungstiefe (5.8a) und Fehlerraten nach Versteckungstiefe (5.8b) beim Brock-Verfahren.



(a) Benötigte Lösungszeit für eine Versteckungstiefe von $h = 1$



(b) Benötigte Lösungszeit für eine Versteckungstiefe von $h = 2$

Abbildung 5.9: Vergleich der benötigten Lösungszeit von Graphen des Brockverfahren für $h = 1$ (5.9a) und $h = 2$ (5.9b) mit zufälligen Graphen.

Zur Messung der Komplexität wurden die generierten Probleme mit zufälligen Graphen derselben Dichte und Knotenanzahl verglichen. Im Folgenden werden jedoch nur die Versteckungstiefen $h = 1$ und $h = 2$ betrachtet, da höhere Versteckungstiefen aufgrund höherer Fehlerraten und eingeschränkterer Konfigurationsmöglichkeiten von geringerer Bedeutung sind. Abbildung 5.9a zeigt dabei, dass die Schwierigkeit von Brock-Graphen mit einer Versteckungstiefe von $h = 1$ etwas unterhalb der Schwierigkeit von zufälligen

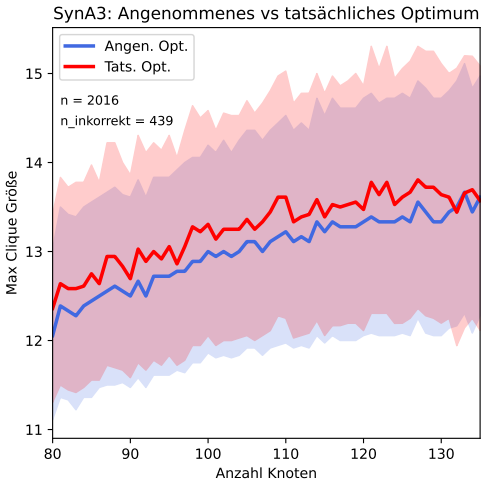
Graphen liegt (in etwa 65% der Schwere). Die Komplexität von Probleminstanzen der Versteckungstiefe $h = 2$ (siehe Abbildung 5.9b) verläuft ähnlich, jedoch sind diese mit rund 74% der Schwere von Zufallsgraphen etwas schwerer zu lösen. Im Vergleich zum Sanchis-Verfahren (circa 55% der Schwierigkeit von Zufallsgraphen) erreichen die vom Brock-Verfahren erzeugten Probleminstanzen bei geringer Fehlerrate also eine höhere Komplexität.

Das Verfahren nach Brockington und Culberson eignet sich also durchaus für das Generieren von Probleminstanzen für das Maximum-Clique-Problem mit bekannter optimaler Lösung. Es ist jedoch darauf zu achten, dass das Verfahren nur für eingesetzte Cliquen größer als die Hintergrund-Clique sinnhaft ist. Außerdem sollten beim Erstellen die unterschiedlichen Vor- und Nachteile der Konfigurationsmöglichkeiten bedacht werden.

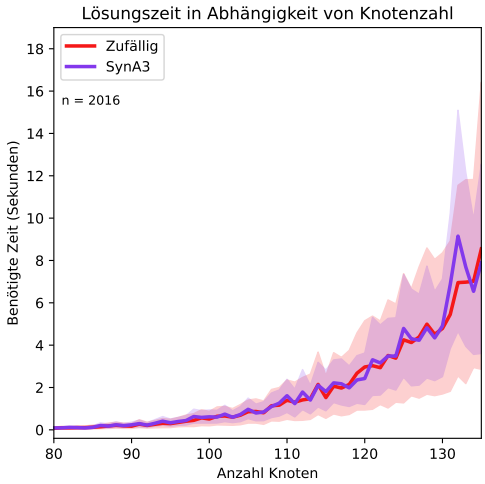
5.1.6 Zufallsgraphen und Hintergrund-Clique

Zur Messung der "SynA3"-Graphen wurden 2016 Probleminstanzen mit einer Knotenzahl im Bereich $n \in \{80, \ldots, 135\}$ generiert, mit je 6 Graphen pro Dichte $d \in \{0.3, 0.5, 0.75\}$ und einer versteckten Clique, die 2 Knoten größer ist als die Hintergrund-Clique.
Wie aus Abbildung 5.10a zu entnehmen, bleibt die eingesetzte Clique in rund 78% der Fälle als optimale Lösung der Probleminstanz erhalten. Die Differenz zum zuvor untersuchten Brock-Verfahren (rund 99%) kann damit erklärt werden, dass beim SynA3-Verfahren jede mögliche Verbindung dieselbe Wahrscheinlichkeit d hat. Beim Brock-Verfahren dagegen sind Verbindungen zwischen Knoten außerhalb der eingesetzten Clique wahrscheinlicher als Verbindungen zur Clique (durch ein erhöhtes p_0). Das Brock-Verfahren hat dadurch eine geringere Wahrscheinlichkeit, die eingesetzte Clique zufällig zu vergrößern.
Bezüglich der Komplexität zeigt Abbildung 5.10b, dass die generierten Probleminstanzen in etwa gleich schwer wie zufällige Instanzen sind, und damit etwas schwerer als die vom Brock-Verfahren generierten Probleme. Dies könnte damit erklärt werden, dass die SynA3-Instanzen näher an einer zufälligen Struktur orientiert sind, welche sich bisher in allen Experimenten als am schwersten zu lösen herausgestellt hat.
Die Komplexität und Korrektheit dieses Ansatzes hängen jedoch stark von dem gewählten Überhang s_o der eingesetzten Clique ab. Je größer dieser Überhang, desto sicherer bleibt die eingesetzte Lösung auch die optimale Lösung des Problems (siehe Abbildung 5.11a). Ab einem zu großen Überhang fällt die Komplexität allerdings ab, da die eingesetzte Clique für Lösungsalgorithmen immer offensichtlicher wird, wie Abbildung 5.11b zeigt. Ein Überhang von um die 2 hat sich in den Experimenten als am ausgeglichensten zwischen Korrektheit und Komplexität herausgestellt.

Dieses Experiment zeigt, dass es durchaus möglich ist, invers generierte Probleminstanzen mit bekannter Lösung derselben Komplexität wie zufällige Probleminstanzen zu erzeugen. Die Korrektheit der vorgeschlagenen Lösung kann jedoch nicht für alle generierten Instanzen garantiert werden.
Abschließend sei noch erwähnt, dass es sich bei diesem, wie auch dem Brock-Verfahren, um statistische Verfahren handelt. Wie Abbildung 5.11a zeigt, fällt etwa die Fehlerrate beim Erhalt der Lösung mit steigender Graphengröße, da die Unsicherheit über bestimmte Annahmen wie die Verteilung der Knotengrade oder die Größe der Hintergrundclique abnimmt.

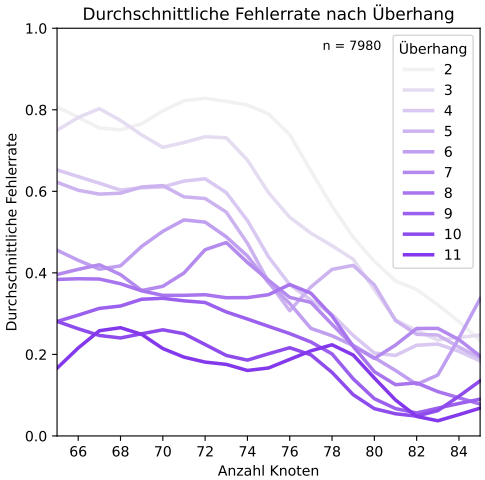


(a) Angenommene und tatsächliche Lösungen für Probleme je Knotenanzahl

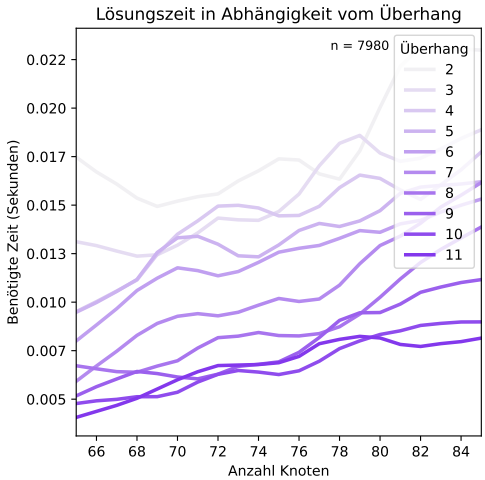


(b) Benötigte Lösungszeit zufälliger Probleme und Instanzen des SynA3-Generator

Abbildung 5.10: Korrektheit der SynA3-Instanzen (5.10a) und benötigte Zeit zum Lösen im Vergleich zu zufälligen Graphen (5.10b).



(a) Geringere Fehlerrate bei größerem Überhang und höherer Knotenzahl



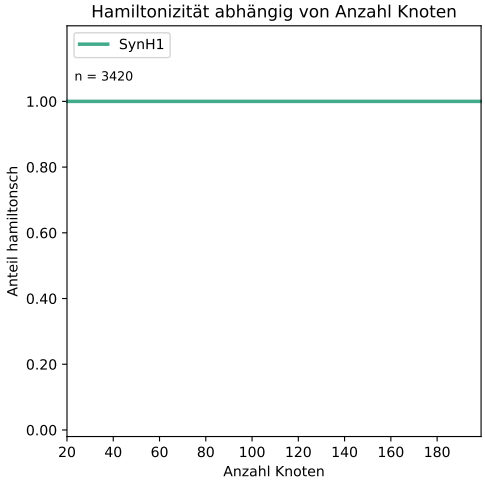
(b) Geringere Komplexität von SynA3-Instanzen bei größerem Überhang

Abbildung 5.11: Korrektheit (5.11a) und Komplexität (5.11b) von SynA3-Instanzen in Abhängigkeit vom Überhang s_o und Knotenanzahl.

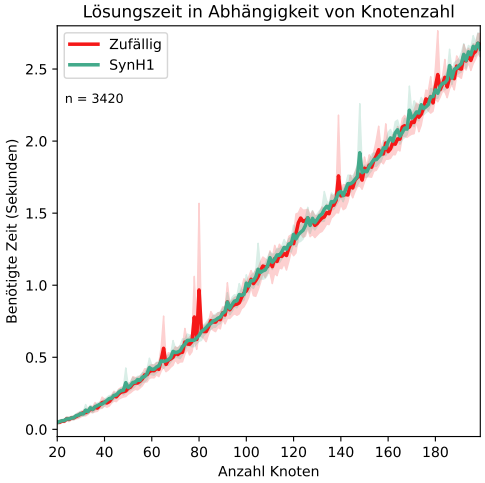
5.2 Hamiltonian-Cycle-Problem

5.2.1 Intuitiver Ansatz - Hamiltonsch

Zur Untersuchung des Hamiltonschen Intuitiven Ansatzes (im Nachfolgenden auch als "SynH1" bezeichnet) wurden insgesamt 3420 Probleminstanzen mit einer Knotenzahl im Bereich $n \in \{20, \ldots, 199\}$ generiert, mit je Dichten von $d \in \{0.05, 0.1, 0.15, \ldots, 0.95\}$.
Bezüglich der Korrektheit bestätigt die empirische Auswertung die Theorie aus 4.2.1, denn wie aus Abbildung 5.12a zu entnehmen ist, bleibt nach dem Generieren zusätzlicher Kanten die Hamiltonizität aller Probleminstanz erhalten.
Im Gegensatz zum Intuitiven Ansatz beim Maximum-Clique-Problem ist jedoch dieser Ansatz bereits mit der Schwierigkeit zufälliger Graphen vergleichbar, wie Abbildung 5.12b zeigt.



(a) Anteil hamiltonscher Instanzen je Knotenanzahl



(b) Benötigte Lösungszeit zufälliger Probleme und Instanzen des hamiltonschen Intuitiven Ansatzes

Abbildung 5.12: Korrektheit der Instanzen des hamiltonschen Intuitiven Ansatzes (5.12a) und benötigte Lösungszeit im Vergleich zu zufälligen Graphen (5.12b).

Dies ist darauf zurückzuführen, dass die Struktur der generierten Probleme der von zufälligen Graphen ähnlich ist und diese im Allgemeinen verhältnismäßig einfach zu lösen sind, da diese ab einer gewissen Dichte p (siehe 5.4) fast immer hamiltonsch sind [Bol01b]. Unterhalb dieser Schwelle gibt es in der Regel nur wenige Möglichkeiten für hamiltonsche Zyklen und diese sind dementsprechend schnell getestet. Hat beispielsweise nur ein Knoten $u \in V$ einen Knotengrad von $\text{grad}(u) < 2$ ist die ganze Probleminstanz nicht hamiltonsch. Umgekehrt lässt sich oberhalb dieser Schwelle schnell ein vollständiger Zyklus finden, da die Zahl der Möglichkeiten schnell ansteigt.

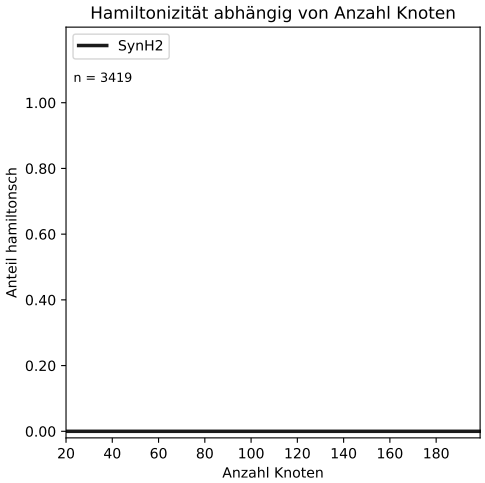
$$p > \frac{\ln(n) + \ln(\ln(n))}{n} \tag{5.4}$$

Grundsätzlich zeigt dieses Experiment, dass das Erzeugen von künstlichen Probleminstanzen für beliebige Dichten und Knotenzahlen, die immer hamiltonsch sind, möglich

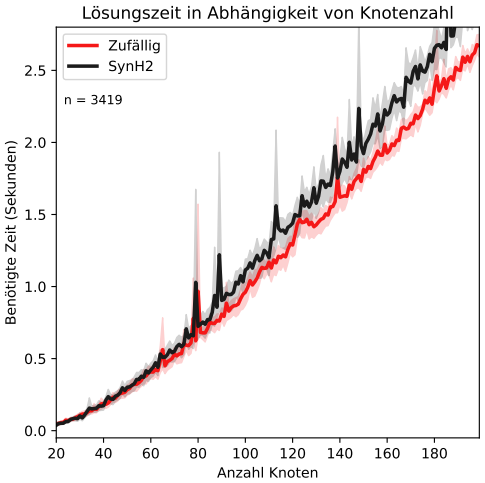
ist. Jedoch sind die erzeugten Graphen dieses Ansatzes nicht besonders schwer zu lösen und zufällige Graphen eignen sich nur bedingt als Maßstab für die Komplexität erzeugter Probleminstanzen.

5.2.2 Intuitiver Ansatz - Nicht hamiltonsch

Zur Untersuchung des nicht-hamiltonschen Intuitiven Ansatzes (im Nachfolgenden auch als "SynH2" bezeichnet) wurden insgesamt 3420[1] Probleminstanzen mit einer Knotenzahl im Bereich $n \in \{20, \ldots, 199\}$ generiert, mit je Dichten von $d \in \{0.05, 0.1, 0.15, \ldots, 0.95\}$. Bezüglich der Korrektheit bestätigt die empirische Auswertung die Theorie aus 4.2.2, denn wie aus Abbildung 5.13a zu entnehmen ist, wird nach dem Generieren zusätzlicher Kanten keine Probleminstanz hamiltonsch.



(a) Anteil hamiltonscher Instanzen je Knotenanzahl



(b) Benötigte Lösungszeit zufälliger Probleme und Instanzen des nicht-hamiltonschen Intuitiven Ansatzes

Abbildung 5.13: Korrektheit der Instanzen des SynH2-Ansatzes (5.13a) und benötigte Zeit zum Lösen im Vergleich zu zufälligen Graphen (5.13b).

Bezüglich der Schwere der erzeugten Graphen verhält sich die eingesetzte Struktur jedoch ähnlich wie die des hamiltonschen Intuitiven Ansatzes aus Kapitel 5.2.1. Wie Abbildung 5.13b zeigt, sind die nicht-hamiltonschen Probleminstanzen nur geringfügig komplexer als vergleichbare zufällige Graphen. Dies wird jedoch hauptsächlich darauf zurückzuführen sein, dass es bei den meisten zufälligen Graphen schwerer sein sollte, Nicht-Hamiltonizität festzustellen, als umgekehrt, da für den hamiltonschen Fall nur eine gefundene Lösung Hamiltonizität beweist, für Nicht-Hamiltonizität jedoch alle möglichen Lösungen ausgeschlossen werden müssen.

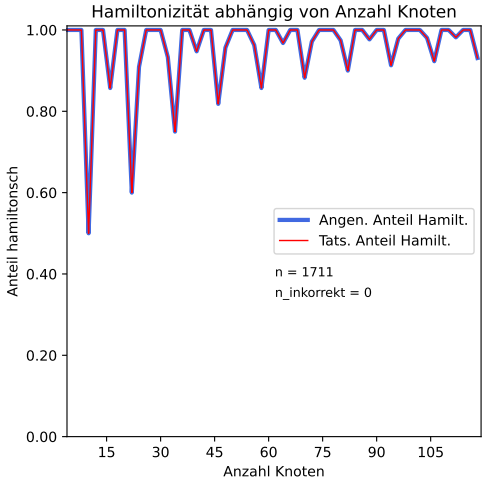
Im Allgemeinen ist es also möglich, künstlich hamiltonsche und nicht-hamiltonsche Probleminstanzen für verschiedenste Knotenzahlen und Dichten zu generieren. Die beiden untersuchten Verfahren führen jedoch nicht automatisch zu wesentlich schwereren Instanzen als zufällige Graphen. Tiefergehende Analysen bezüglich der Eigenschaften von

[1] Eine Instanz führte beim Concorde-Löser zu unerwartet starken Schwankungen.

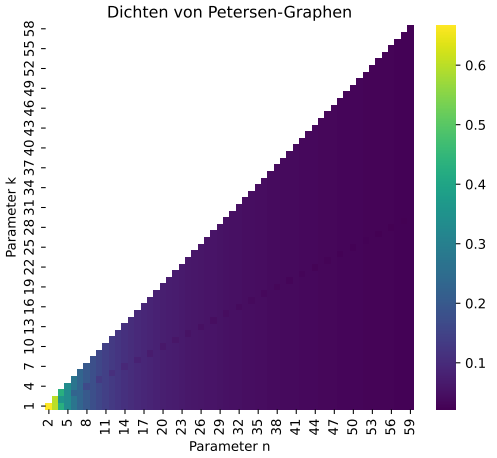
schweren HCP-Instanzen finden sich etwa in Arbeiten von Baniasada et al. [BEHR18] sowie Sleegers und van den Berg [SvdB22].

5.2.3 Petersen Graphen

Zur Untersuchung der Petersen-Graphen wurden insgesamt 1711 Probleminstanzen mit einer Knotenzahl $n_{ges} = 2 \cdot n$ mit $n \in \{2, \ldots, 60\}$ generiert, mit je einem $k \in \{1, \ldots, n-1\}$. Bezüglich der Korrektheit lassen sich die Regeln für Hamiltonizität von Alspach [Als83] bestätigen (siehe Abbildung 5.14a). Wie auch aus den genannten Regeln ableitbar, sind die meisten Konfigurationen hamiltonsch. Abbildung 5.14b zeigt jedoch, dass die möglichen Dichten dieses Ansatzes auf einen schmalen, niedrigen, mit der Anzahl der Knoten fallenden Bereich beschränkt sind.



(a) Anteil hamiltonscher Instanzen je Knotenanzahl



(b) Dichte der Petersen-Graphen nach Konfiguration

Abbildung 5.14: Korrektheit der Petersen-Instanzen (5.14a) und erzeugte Dichte je Konfiguration (5.14b).

Bezüglich der Komplexität zeigt die Auswertung in Abbildung 5.15, dass die meisten Probleminstanzen im Verhältnis zu zufälligen Instanzen einfacher zu lösen sind. Die einzigen schweren Instanzen (rund 3% aller Instanzen) wurden von Kombinationen $n \equiv 1 \mod 2$ mit einem $k \in \{2, n-2, \frac{n-1}{2}, \frac{n+1}{2}\}$ erzeugt. Dieses Muster ist ähnlich der Bedingung für Nicht-Hamiltonizität (siehe Kapitel 4.2.3), jedoch war Nicht-Hamiltonizität keine Bedingung für die schweren Instanzen. Es wäre daher unter Umständen für zukünftige Arbeiten interessant, diese Konfigurationen genauer zu analysieren.
Grundsätzlich zeigt dieses Experiment, dass das Erzeugen von künstlichen Probleminstanzen mit bekannter Lösung mithilfe der Konstruktion von allgemeinen Petersen-Graphen möglich ist.
Es gibt jedoch kaum Variation in den Dichten, und die meisten Instanzen sind leichter zu lösen als vergleichbare zufällige Graphen.

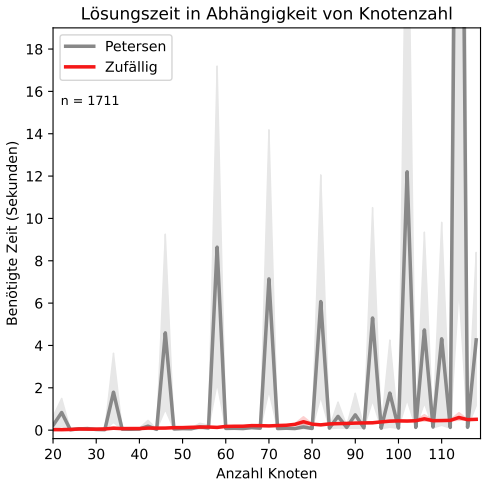


Abbildung 5.15: Benötigte Zeit zum Lösen der Petersen-Graphen im Vergleich zu zufälligen Graphen.

6 Diskussion

In dieser Arbeit wurden insgesamt 8 Verfahren für das Generieren von Probleminstanzen für das Maximum-Clique-Problem untersucht. Dabei war es für zwei Graphenklassen (Johnson- und Keller-Graphen) nicht möglich, Probleme mit einer bekannten optimalen Lösung zu generieren. Bei drei Ansätzen (Intuitiver Ansatz, C-Fat und Hamming) stellte sich nur eine bedingte Eignung heraus, da bei diesen starke Einschränkungen bezüglich der Komplexität oder der Wahlfreiheit der Struktur bestehen. Die drei verbleibenden Verfahren (Sanchis, Brock und SynA3) waren jedoch in der Lage, Probleme etwas unterhalb oder um die Schwierigkeit von zufälligen Instanzen mit bekannter Lösung zu erzeugen.
Alle drei Verfahren unterscheiden sich dennoch in der Sicherheit beim Erhalt der eingesetzten Lösung, der erzielten Schwere zum Lösen und der Komplexität des Verfahrens selbst. Während das Sanchis-Verfahren in allen Fällen eine korrekte Lösung garantieren kann, erzeugen der Brock- und SynA3-Ansatz schwerere Probleme, bei geringerer Sicherheit, da diese noch näher an der Struktur echter Zufallsgraphen orientiert sind.
Alle drei Verfahren zeigen jedoch, dass das Erreichen von Schwierigkeiten in der Nähe von Zufallsgraphen meist nur durch Nachahmen der zufälligen Struktur selbst möglich ist und insbesondere das Verstecken bzw. Angleichen der Knotengrade eine Rolle spielt. Um dabei eine eingesetzte Lösung als optimal zu erhalten, ist oftmals auf statistische Überlegungen zurückzugreifen, und diese können je nach Größenordnung der betrachteten Probleme stark variieren oder weitere Einschränkungen - wie eine Mindestgröße der eingesetzten Clique - mit sich bringen.
Abschließend lässt sich über das Maximum-Clique-Problem mit den hier gewonnenen Erkenntnissen sagen, dass für das Erzeugen von Problemen mit bekannter optimaler Lösung kein perfektes Verfahren existiert. Jedoch erscheinen, je nach Einsatzrahmen, manche Einschränkungen als hinnehmbar, da etwa Zufallsgraphen selbst in fast allen Fällen nur Cliquen weniger bestimmter Größen aufweisen.

Für das Hamiltonian-Cycle-Problem wurden 3 Verfahren für das Erzeugen von Problemen unter bekannter Lösung untersucht. Die beiden Intuitiven Ansätze (SynH1 und SynH2) erzeugen dabei auf trivialer Basis Probleminstanzen, welche die angedachte Lösung garantieren können und sich in Sachen Komplexität auf demselben Niveau wie Zufallsgraphen derselben Struktur befinden. Das betrachtete Verfahren nach Petersen erzeugt Probleme, die in der Regel leichter zu lösen sind als zufällige Probleme, vermutlich da diese stärker strukturiert sind.
Hier scheinen jedoch zufällige Probleminstanzen im Allgemeinen nicht repräsentativ als Maßstab für schwere Instanzen zu sein. Im entsprechenden Kapitel wurde daher auf weiterführende Literatur verwiesen. Es ist jedoch denkbar, dass auch hier mit dem richtigen Ansatz Probleminstanzen von höherer Komplexität erzeugbar sind.

Die Untersuchungen können an diesem Punkt noch weiter fortgeführt werden. Für beide aufgegriffenen Verfahren existiert noch eine Vielzahl nicht betrachteter Verfahren bzw. für

manche Konfigurationen wie die Hamming-Graphen mit einer Distanz von $d \geq 3$ konnte nur die Größe der Lösung beziffert werden, nicht die genaue Knotenmenge. Außerdem wurde nur eine Variante der jeweiligen Problemstellungen betrachtet. Ob und inwiefern hier betrachtete Ansätze auf beispielsweise gewichtete oder gerichtete Graphen übertragbar sind, bleibt offen. Außerdem wurden nur Probleme in einer für den Autor sinnvollen berechenbaren Größenordnung getestet. Jedoch kann sich diese Größenordnung je nach Einsatzgebiet deutlich unterscheiden und die vorgestellten Ergebnisse insbesondere bezüglich Korrektheit oder Komplexität dahingehend variieren.

Zum Schluss sei hier noch ein weiterer möglicherweise vielversprechender Ansatz zur Erzeugung schwerer Probleminstanzen mit bekannter Lösung erwähnt: Wie Monasson et al. [MZK$^+$99] in ihrer Arbeit beschreiben, gibt es für das sogenannte Erfüllbarkeitsproblem (kurz “SAT”; die Frage, ob, bzw. für welche Variablenbelegung, eine logische Aussage in konjunktiver Normalform, z.B. $(x \vee y) \wedge (\text{not } x)$, erfüllbar ist) einen sogenannten Phasenübergang. Das bedeutet, dass in Abhängigkeit von dem Verhältnis der Anzahl Klauseln zur Anzahl der Variablen ab einem gewissen Punkt die meisten Probleme von meistens erfüllbar zu meistens nicht erfüllbar kippen. Gleichzeitig scheinen Probleme an diesem Phasenübergang für gängige Lösungsalgorithmen schwerer zu sein, als außerhalb des Übergangs [MZK$^+$99]. Basierend auf dieser Erkenntnis stellten Xu et al. [XBHL07] einen Generator vor, der schwere, erfüllbare Probleminstanzen für das SAT-Problem entlang dieses Phasenübergangs erzeugt.
Es wäre daher noch interessant zu untersuchen, ob ein Übersetzen dieser Probleme, in beispielsweise das Maximum-Clique-Problem, zu schwereren Probleminstanzen als den hier vorgestellten führt, unter Erhalt einer bekannten Lösung. Der beschriebene Phasenübergang konnte in dieser Arbeit leider nicht mehr ausreichend untersucht werden.

7 Fazit

Zusammenfassend lässt sich sagen, dass sich diese Arbeit mit der Frage beschäftigt hat, ob und inwiefern das Erzeugen synthetischer Probleminstanzen mit bekannter Lösung für NP-schwere Optimierungsprobleme wie dem Maximum-Clique-Problem oder dem Hamiltonian-Cycle-Problem möglich ist.
Dazu wurden verschiedenste Verfahren analysiert und empirisch verglichen, von denen sich manche als geeignet und manche als ungeeignet herausstellten. Die zentrale Forschungsfrage kann daher insofern beantwortet werden, dass es für beide Problemfelder möglich war, Probleminstanzen verschiedener Komplexitäts- und Korrektheitsstufen zu erzeugen.
Je nach hinnehmbarer Einschränkung können die vorgestellten Ansätze etwa verwendet werden, um Trainingsdaten für lernende Modelle zu erzeugen oder als Maßstab für die Effektivität von (approximativen) Algorithmen zu dienen.
Für zukünftige Arbeiten wäre es jedoch noch interessant, inwiefern diese Ansätze auf Variationen der betrachteten Problemstellungen oder andere Problemstellungen übertragbar sind, oder welche anderen Verfahren entsprechend geeignet wären.

Abbildungsverzeichnis

Literaturverzeichnis

[ABCC11] Applegate, David L, Robert E Bixby, Vašek Chvátal und William J Cook: *The traveling salesman problem: a computational study*. In: *The Traveling Salesman Problem*. Princeton university press, 2011.

[AF88] Arthur, Jeffrey L und James O Frendewey: *Generating travelling-salesman problems with known optimal tours*. Journal of the Operational Research Society, 39(2):153–159, 1988.

[Als83] Alspach, Brian: *The classification of hamiltonian generalized Petersen graphs*. Journal of Combinatorial Theory, Series B, 34(3):293–312, 1983.

[BC93] Brockington, Mark und Joseph C Culberson: *Camouflaging independent sets in quasi-random graphs.* Cliques, coloring, and satisfiability, 26:75–88, 1993.

[BE76] Bollobás, Béla und Paul Erdös: *Cliques in random graphs*. In: *Mathematical Proceedings of the Cambridge Philosophical Society*, Band 80, Seiten 419–427. Cambridge University Press, 1976.

[BEHR18] Baniasadi, Pouya, Vladimir Ejov, Michael Haythorpe und Serguei Rossomakhine: *A new benchmark set for Traveling salesman problem and Hamiltonian cycle problem*. arXiv preprint arXiv:1806.09285, 2018.

[BHMN20] Brakensiek, Joshua, Marijn Heule, John Mackey und David Narváez: *The resolution of Keller's conjecture*. In: *International Joint Conference on Automated Reasoning*, Seiten 48–65. Springer, 2020.

[Bol01a] Bollobás, Béla: *Random Graphs*, Seiten 282–290. Cambridge University Press, Cambridge, UK, 2 Auflage, 2001.

[Bol01b] Bollobás, Béla: *Random Graphs*, Seiten 206–212. Cambridge University Press, Cambridge, UK, 2 Auflage, 2001.

[BP90] Berman, P und A Pelc: *Distributed Fault Diagnosis for Multiprocessor Systems Proc. of the 20th Annual Intern*. In: *Symp. on Fault-Tolerant Computing (Newcastle, UK)*, Seiten 340–346, 1990.

[BSSS06] Brouwer, Andries E, James B Shearer, Neil JA Sloane und Warren D Smith: *A new table of constant weight codes*. IEEE Transactions on Information Theory, 36(6):1334–1380, 2006.

[DDS+09] Deng, Jia, Wei Dong, Richard Socher, Li-Jia Li, Kai Li und Li Fei-Fei: *Imagenet: A large-scale hierarchical image database*. In: *2009 IEEE conference on computer vision and pattern recognition*, Seiten 248–255. Ieee, 2009.

[GJ79a] Garey, Michael R. und David S. Johnson: *Computers and Intractability: A Guide to the Theory of NP-Completeness*. In: *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman & Company, San Francisco, CA, First Auflage, 1979.

[GJ79b] Garey, Michael R. und David S. Johnson: *Computers and Intractability: A Guide to the Theory of NP-Completeness*. In: *Computers and Intractability: A Guide to the Theory of NP-Completeness*, Seiten 53–65. W. H. Freeman & Company, San Francisco, CA, First Auflage, 1979.

[GM75] Grimmett, Geoffrey R und Colin JH McDiarmid: *On colouring random graphs*. In: *Mathematical Proceedings of the Cambridge Philosophical Society*, Band 77, Seiten 313–324. Cambridge University Press, 1975.

[Hel00] Helsgaun, Keld: *An effective implementation of the Lin–Kernighan traveling salesman heuristic*. European journal of operational research, 126(1):106–130, 2000.

[HPV93] Hasselberg, Jonas, Panos M Pardalos und George Vairaktarakis: *Test case generators and computational results for the maximum clique problem*. Journal of Global Optimization, 3:463–482, 1993.

[HZ21] Hougardy, Stefan und Xianghui Zhong: *Hard to solve instances of the euclidean traveling salesman problem*. Mathematical Programming Computation, 13(1):51–74, 2021.

[JT96] Johnson, David S und Michael A Trick: *Cliques, coloring, and satisfiability: second DIMACS implementation challenge, October 11-13, 1993*, Band 26. American Mathematical Soc., 1996.

[Kel30] Keller, Ott-Heinrich: *Über die lückenlose Erfüllung des Raumes mit Würfeln*. 1930.

[KSH12] Krizhevsky, Alex, Ilya Sutskever und Geoffrey E Hinton: *Imagenet classification with deep convolutional neural networks*. Advances in neural information processing systems, 25, 2012.

[Mat70] Matula, DW: *On the complete subgraphs of a random graph, Combinatory Math, and its Applications, Chapel Hill, N*, 1970.

[MS77] MacWilliams, Florence Jessie und Neil James Alexander Sloane: *The theory of error-correcting codes*, Band 16. Elsevier, 1977.

[MZK+99] Monasson, Rémi, Riccardo Zecchina, Scott Kirkpatrick, Bart Selman und Lidror Troyansky: *Determining computational complexity from characteristic 'phase transitions'*. Nature, 400(6740):133–137, 1999.

[PPG+15] Pattabiraman, Bharath, Md Mostofa Ali Patwary, Assefaw H Gebremedhin, Wei-keng Liao und Alok Choudhary: *Fast algorithms for the maximum clique problem on massive graphs with applications to overlapping community detection*. Internet Mathematics, 11(4-5):421–448, 2015.

[PR92] Pilcher, Martha G und Ronald L Rardin: *Partial polyhedral description and generation of discrete optimization problems with known optima*. Naval Research Logistics (NRL), 39(6):839–858, 1992.

[PX94] Pardalos, Panos M und Jue Xue: *The maximum clique problem*. Journal of global Optimization, 4:301–328, 1994.

[Rei91] Reinelt, Gerhard: *TSPLIB—A traveling salesman problem library*. ORSA journal on computing, 3(4):376–384, 1991.

[San89] Sanchis, Laura: *Language instance generation and test case construction for NP-hard problems.* 1989.

[San94] Sanchis, Laura: *Test case construction for the vertex cover problem.* DIMACS Series in Discrete Mathematics and Theoretical Computer Science, 15:315–326, 1994.

[SSFÁP23] San Segundo, Pablo, Fabio Furini, David Álvarez und Panos M Pardalos: *CliSAT: A new exact algorithm for hard maximum clique problems.* European Journal of Operational Research, 307(3):1008–1025, 2023.

[SSSG17] Sun, Chen, Abhinav Shrivastava, Saurabh Singh und Abhinav Gupta: *Revisiting unreasonable effectiveness of data in deep learning era.* In: *Proceedings of the IEEE international conference on computer vision*, Seiten 843–852, 2017.

[SvdB22] Sleegers, Joeri und Daan van den Berg: *The hardest hamiltonian cycle problem instances: the plateau of yes and the cliff of no.* SN Computer Science, 3(5):372, 2022.

[VFJ15] Vinyals, Oriol, Meire Fortunato und Navdeep Jaitly: *Pointer networks.* Advances in neural information processing systems, 28, 2015.

[Wat69] Watkins, Mark E: *A theorem on Tait colorings with an application to the generalized Petersen graphs.* Journal of Combinatorial Theory, 6(2):152–164, 1969.

[XBHL07] Xu, Ke, Frédéric Boussemart, Fred Hemery und Christophe Lecoutre: *Random constraint satisfaction: Easy generation of hard (satisfiable) instances.* Artificial intelligence, 171(8-9):514–534, 2007.

[Xu07] Xu, K BHOSLIB: *Benchmarks with hidden optimum solutions for graph problems.* URL http://www.nlsde.buaa.edu.cn/kexu/benchmarks/graph-benchmarks.htm, 2007.

[YZY+23] Yang, Hua, Minghao Zhao, Lei Yuan, Yang Yu, Zhenhua Li und Ming Gu: *Memory-efficient transformer-based network model for traveling salesman problem.* Neural Networks, 161:589–597, 2023.