SoSe 25

Betreuer: Prof. Dr. Georg Loho

Zweitkorrektur: Dr. Olaf Parczyk

Maschineller mathematischer Erkenntnisgewinn

Grenzen und Möglichkeiten ausgewählter KI-Werkzeuge

Conrad Schweiker

Matr.nr: 5542095

Submission date: 10.09.25

Inhalt

Abkürzungen

Es seien folgende Schreibweisen für diese Arbeit vereinbart:

- $\mathbb{N} \coloneqq \mathbb{Z}_+ = \{1, 2, 3, ...\}$
- $\mathbb{N}_0 \coloneqq \mathbb{N} \cup \{0\} = \{0, 1, 2, 3, ...\}$
- Für ein beliebiges $n \in \mathbb{N}_0$ sei $[n] \coloneqq \{1, 2, 3, ..., n-1, n\} = \mathbb{N}_{\leq n}$
- Für eine Reihe $(x_i)_{i \in [n]} = (x_1, x_2, ..., x_n)$ sei die Abkürzung (x_i) für die Reihe und x_i für das i-te Element dieser Reihe verwendet.
- Entsprechend ist $\sum(x_i) \coloneqq \sum_{i=1}^{n} x_i$ und $\prod(x_i) \coloneqq \prod_{i=1}^{n} x_i$

Zur einfacheren Kompatibilität sei für mathematische Ausdrücke stets die englische Notation verwendet, d.h. Dezimalzahlen werden mit Punkt geschrieben und Koordinaten eines Vektors durch ein einfaches Komma getrennt.

Es gibt keinen Algebraiker oder Mathematiker, der in seiner Wissenschaft so bewandert ist, dass er sofort volles Vertrauen in eine von ihm entdeckte Gültigkeit setzt oder sie als etwas anderes als eine bloße Wahrscheinlichkeit betrachtet. Jedes Mal, wenn er seine Beweise durchgeht, wächst sein Vertrauen; aber noch mehr durch die Zustimmung seiner Freunde; und wird durch die allgemeine Zustimmung und den Beifall der gelehrten Welt zu seiner höchsten Vollkommenheit erhoben.

— David Hume

Einleitung

Seit der Veröffentlichung von ChatGPT 3.5 im November 2022 ist das Thema KI, trotz seiner Existenz seit fast 70 Jahren, plötzlich in die Aufmerksamkeit der breiten Gesellschaft gerückt. Medien, Bildungseinrichtungen, Unternehmen, Privatpersonen und natürlich auch Forschende begannen, sich intensiv damit auseinanderzusetzen.

Seitdem ist die Zeit nicht stehen geblieben. Ein Mitarbeiter von OpenAI sprach im Dezember 2024 davon, dass fünf Jahre alte KI-Forschung sich wie die Steinzeit der KI-Entwicklung anfühle. [12 25] Mittlerweile könnte ein LLM (Large Language Model) mit den entsprechenden Prompts eine beeindruckende Einleitung schreiben, die sich exakt an den eigenen Schreibstil hält. Das motiviert, einige originelle Worte zu finden, die sich von den Outputs bisheriger Sprachmodelle unterscheiden.

Stellen wir uns vor, es gäbe ein Buch, in dem alles Wissen des Universums steht, von jedem Baustein, jedem Zusammenhang, von jeder Zahl, alle Eigenschaften. Das Buch hätte unendlich viele Sprachen und – allein weil es unendlich viele Zahlen gibt – wäre zudem unendlich dick. Jeder Versuch, dieses Buch mit einer endlichen Sprache zu beschreiben, wäre letztlich nur eine Approximation, eine Modellierung, die in jedem Falle unvollständig wäre.

Menschen haben Sprache entwickelt, die mit nur wenigen Worten Bilder beschreibt, die andere Menschen nachvollziehen können. Dabei ist diese nicht eindeutig. So entsteht z.B. bei der Verwendung des Wortes „Haus“ in jeder Person eine andere Vorstellung. Die natürliche Sprache trägt so viele dieser Ungenauigkeiten in sich, dass wir uns fragen müssen, ob sie tatsächlich eine gute Approximation für das Buch allen Wissens ist.

Eine neue, alternative Sprache ist die der formalen Beweissysteme in der Mathematik. Sie darf nur nach festen Regeln auf- und umgestellt werden. Eine Relevanz erhält sie allerdings auch erst, wenn eine Bedeutungskorrelation der Symbole der Sprache in die natürliche Sprache vorhanden ist. Außerdem müssen einige Axiome und Umstelllogiken angenommen werden, ohne dass diese begründet werden können. Eine Entscheidung, ob diese Sprache also besser ist als die natürliche, lässt sich daher nicht treffen. Doch bieten formale Systeme einen entscheidenden Vorteil: Alle Aussagen darin sind entweder wahr oder falsch; alle korrekten Umformungen einer wahren Aussage bleiben wahr.

Die Sätze innerhalb des Systems lassen sich durch Anwendung der endlichen logischen Regeln auf endlich viele Axiome zurückführen. Das gesamte System bekommt dadurch die Struktur eines gerichteten Graphen, dessen Knoten die Sätze sind; die Kanten (A, B) bedeuten, dass es eine logische Regel φ gibt, mit der wir aus $A \overset{\varphi}{\rightsquigarrow} B$ folgern können.

Wir können auf diesem Graphen $\mathcal{G}$ für Sätze verschiedene Schwierigkeiten definieren, z.B. über die Größe des Teilgraphen, der nötig ist, um den Satz zu beweisen, oder über die Anzahl verschiedener Teilgraphen, die den Satz beweisen. Das Bild der Schwierigkeitsfunktion soll auf der reellen Zahlengerade liegen und so entsteht eine visuelle Vorstellung, wie kompliziert es ist, ein Problem zu beweisen.

Wenn es eine Korrelation gibt zwischen wachsender Schwierigkeit eines Problems und der Zeit, die benötigt wird, um das Problem zu lösen, so gibt es womöglich Probleme, deren Komplexität so groß ist, dass ein Menschenleben nicht ausreicht, um bis zur Lösung vorzudringen. Es ist daher nicht abwegig, die Lösungssuche auf eine digitale Intelligenz übertragen zu wollen, die beliebig skaliert werden kann – vorausgesetzt, wir können uns zu 100% auf diese verlassen.

Häufig aber arbeiten Mathematiker:innen nicht innerhalb der formalen Beweissysteme, da diese ihnen nicht die nötige Intuition liefern, um sich ihren Problemen anzunähern. Stattdessen verwenden sie die natürliche Sprache, ihr inhärentes dreidimensionales Vorstellungsvermögen und

nutzen die Fähigkeit aus, ihre Umwelt manipulieren zu können. Letztlich gibt das formale System auch nicht die Bedeutung eines Satzes an. Was wichtig bzw. relevant für ein beliebiges mathematisches Anwendungsgebiet ist, bleibt ein Thema der natürlichen Sprache.

Die mathematische Arbeit mit LLMs wird in verschiedenen wissenschaftlichen Artikeln untersucht [Lig+23, Liu+25], schließlich sind diese nicht 100% zuverlässig. Es erinnert daran, dass Menschen selbst von Zeit zu Zeit Fehler verschulden und diese teils sehr spät bemerken. Im Mittelpunkt dieser Arbeit stehen allerdings drei Anwendungsfälle, bei denen die unzuverlässigen Ausgaben von Sprachmodellen per formalem System gefiltert werden. Die resultierenden Erkenntnisse sind dann mathematisch korrekt.

Dieser Ansatz ist an der Art inspiriert, wie Menschen beim Problemlösen ihre Zwischenergebnisse verifizieren. Es hilft daher, sich anzusehen, wie Menschen mathematisches Wissen erzeugen und erfolgreich Probleme lösen. Das erste Kapitel widmet sich daher zunächst diesem eher didaktischen Bereich überblicksartig. Daraus können Schlüsse gezogen werden, wie Menschen effektiv mathematisch denken lernen können, Erkenntnisse, die sicherlich von Relevanz für die Umgestaltung von Lehrsystemen sind.

Darauf folgend sind die verschiedenen Fälle vorgestellt, bei denen neuronale Netze erfolgreich in der mathematischen Forschung genutzt worden sind. Im Anschluss zeige ich eine konkrete Anwendung an einem eigenen Beispiel und gebe abschließend ein paar Gedankenimpulse über die Limitationen formaler Systeme an.

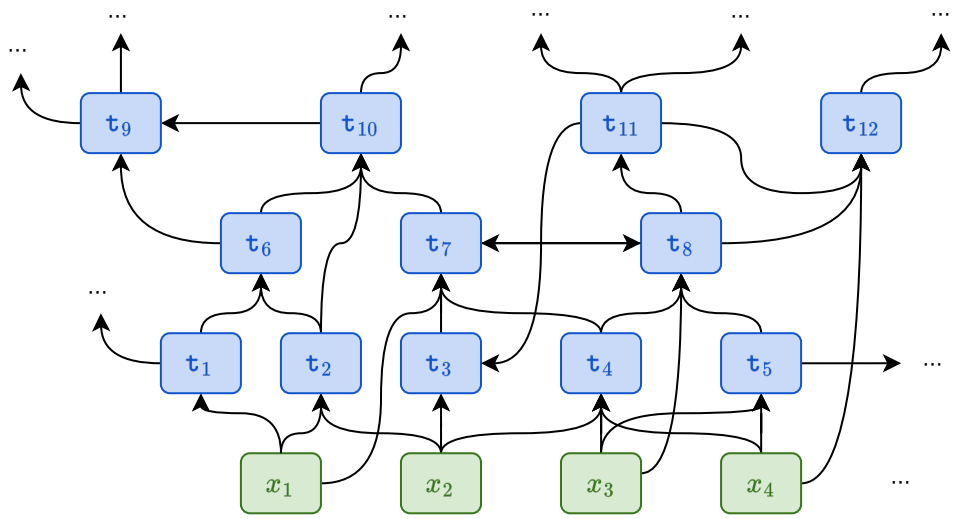
1. Überblick: Wie mathematisches Wissen geschaffen wird

In erster Linie sollten wir das Ziel klären, das wir mit der Wissensgenerierung verfolgen. Sicherlich stößt die Vereinbarung „Wir wollen relevante mathematische Zusammenhänge entdecken und lernen“ auf Zustimmung und Widerstand zugleich. Zustimmung, denn sicherlich ist nicht jede willkürliche Umstellung mathematischer Formeln interessant, auch wenn sie logisch korrekt ist. So ist beispielsweise $-2x(2-x)-x^2+4(2x-5)-1$ eine beliebige Umstellung eines quadratischen Funktionsterms. Die allgemeine Darstellung $x^2+4x-21$, die Scheitelpunktform $(x+2)^2-25$ oder die Nullstellenform $(x+7)(x-3)$ wären aber in den meisten Kontexten deutlich interessantere Darstellungen.

Auf der anderen Seite weiß eine forschende Person nicht, worauf sie stoßen wird. Was zunächst unwichtig wirkt, kann auf einmal relevant werden; die Arbeit an einem unwichtigen Thema kann zu neuen Lösungstechniken, Tricks und Strategien führen; das Entdecken des Wissens kann dem reinen Vergnügen dienen; oder aber das reine Umstellen dient dem Training einer expliziten mathematischen Fähigkeit [Ben05, Zei16]. Es ist daher schwer „Relevanz“ zu definieren, allerdings notwendig, wenn ein automatisiertes System nicht einfach beliebig umgestellte Formeln ausgeben soll.

Es hilft an dieser Stelle, den Grundgedanken von formalen Beweissystemen vorzuziehen (siehe Kapitel 2.3.1). Die mathematische Gesellschaft versucht seit jeher, logische Argumente so einfach wie möglich zu designen und diese auf nur wenigen Grundannahmen $\{x_1, x_2, ..., x_n\}$ zu fußen, die angenommen werden müssen [Hof17]. Diese sind nicht eindeutig. Es ist möglich, eine andere Menge an Grundwahrheiten zu wählen.

Der Aufwand wird betrieben, um sicherzugehen, dass die Ergebnisse tatsächlich wahr sind. Mit den logischen Umstellregeln können die Grundannahmen in neue Aussagen umgestellt werden, welche dann mit den selben Umstellregeln weiter verändert werden können. Stellt man sich die Grundannahmen und Aussagen als Knoten eines Graphen vor und valide Umstellregeln als gerichtete Kanten, so entsteht ein Bild, in dem alle Aussagen der Mathematik vorkommen:



Durch dieses Bild wird auch klar, dass es sehr viele willkürliche Sätze gibt, die sich mit logischen Regeln bilden lassen, aber nur wenige von genereller Bedeutung sind. Z.B. ist $\frac{3}{3}(x^2 + 4x - 21)$ eine beliebige Umstellung des zuvor genannten Funktionsterms und vermutlich weniger interessant als $(x^2 + 4x + 4) - 25$.

Mit diesem Bild haben wir gleichzeitig eine Intuition geschaffen für mathematisches Wissen selbst. Eine allgemein anerkannte Definition für „mathematisches Wissen“ mag es nicht geben, es macht jedoch Sinn, alle Knoten im Graphen als Teil dieses Wissens anzunehmen. Das *relevante* mathematische Wissen ist eine Teilmenge des gesamten Wissens; dementsprechend ist auch eine Teilmenge all der Knoten im Graphen *relevant*.

Ob bewusst oder zufällig, Menschen, die sich mit mathematischen Problemen jeder Art auseinandersetzen, durchsuchen diesen Graphen. Durch sie selbst, ihr persönliches Leben und das ihrer Umwelt entsteht die Relevanz. Eine willkürliche Formel kann durch das konkrete Problem im Leben eines spezifischen Menschen Relevanz erlangen. Will ein Mensch beispielsweise ein künstlerisches Kleidungsstück nähen und fragt sich, wie er dafür möglichst wenig Stoff verwenden kann, findet er womöglich eine Formel, die aussagt, wie viele x-förmige Schablonen er pro Quadratmeter maximal unterbringen kann.

Daraus ergeben sich zwei wichtige Aspekte:

1. **Mathematisches Wissen wird durch Personen aller Art geschaffen.** D.h. nicht nur durch solche, die in der mathematischen Forschung arbeiten, sondern auch Ingenieure, Forscher und Interessierte aller Art: Künstler, Logistiker, Spieleentwickler etc. Die heute weit verbreiteten Bézierkurven wurden von Mitarbeitenden im Automobilbau entwickelt.
2. **Mathematisches Wissen ist relevant, wenn es hilft, Probleme von Menschen zu lösen.** Je mehr Menschen davon profitieren, desto wichtiger. Außerdem ist die Relevanz rekursiv, d.h. Wissen ist dann relevant, wenn es hilft, anderes relevantes Wissen herzuleiten. So gewinnt beispielsweise der gesamte Zweig der Mathematik an Bedeutung, der notwendig ist, um die Informatik und Physik von Computern zu erklären, da Computer viele Probleme vieler Menschen lösen.

Anmerkung: Da eine KI keine „eigenen“ Probleme hat, muss sie so designt werden, dass sie die Probleme von Menschen bearbeitet und solche mathematischen Themen, die dafür lösungsbringend sind.

Im Mittelpunkt steht also stets ein konkretes Problem. Konkret heißt nicht, dass der Lösungsalgorithmus klar ist, lediglich, dass immer ein festes Problem aus der Menge aller Probleme $\mathcal{P}$ fixiert werden kann, das den Ausgangspunkt markiert, an dem der Wissensschaffungsprozess beginnt. Diese Formulierung erlaubt es, dass das Problem am Ende des Prozesses nicht gelöst ist, aber dennoch Wissen geschaffen worden ist.

Im Wesentlichen ist Wissenschaffen also ein Nebenprodukt des Problemlösens. Es ist gut, dass sich die Mathedidaktik bereits sehr intensiv mit der Kunst mathematische Probleme zu lösen auseinandergesetzt hat. Im Folgenden werden wir uns die Techniken bzw. die Verhaltensweisen erfolgreicher Problemlöser ansehen.

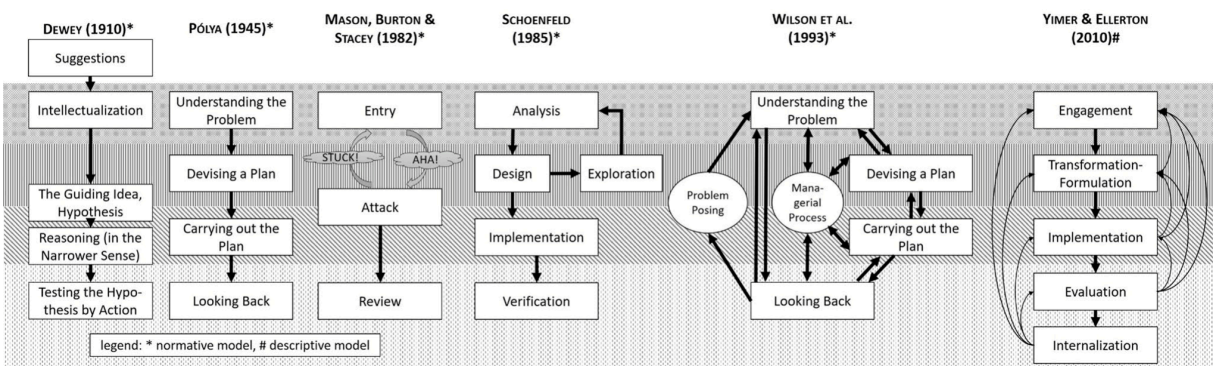
Es sei angemerkt, dass wir an dieser Stelle die vielfältigen Lösungsstrategien der realweltlichen Probleme auf rein mathematische Problemsituationen reduzieren. Wir verlassen also die Metaebene und verlieren damit die Möglichkeit, auf dieser den Problemlöseerfolg zu analysieren, wie z.B. den Einfluss von äußeren Faktoren, die Arbeitsweise in Teams oder Arten der menschlichen Interaktion.

1.1. Strategien mathematische Probleme zu lösen

Einer der prägendsten Mathedidaktiker des 20. Jahrhunderts war sicherlich der ungarische Mathematiker George Pólya. In seinem Buch *How to solve it!* [Pol14] stellt er die verschiedensten Techniken, Tricks und mentalen Einstellungen vor, die man im Allgemeinen für das Lösen mathematischer Probleme braucht. Nachfolgend sei eine beispielhafte, unvollständige Liste solcher groben Strategien nach Pólya für einen ersten Eindruck:

- Gib nicht zu schnell auf.
- Betrachte Beispiele.
- Visualisiere das Problem.
- Erkläre das Problem und deine Gedanken anderen.
- Lerne Tricks zum Thema.
- Wende verschiedene Standardtechniken an.
- Zerlege das Ziel in Teilschritte.
- Benutz eine andere Methode.
- Formuliere das Ziel um.
- Beschäftige dich mit etwas Ähnlichem.
- Gib auf.

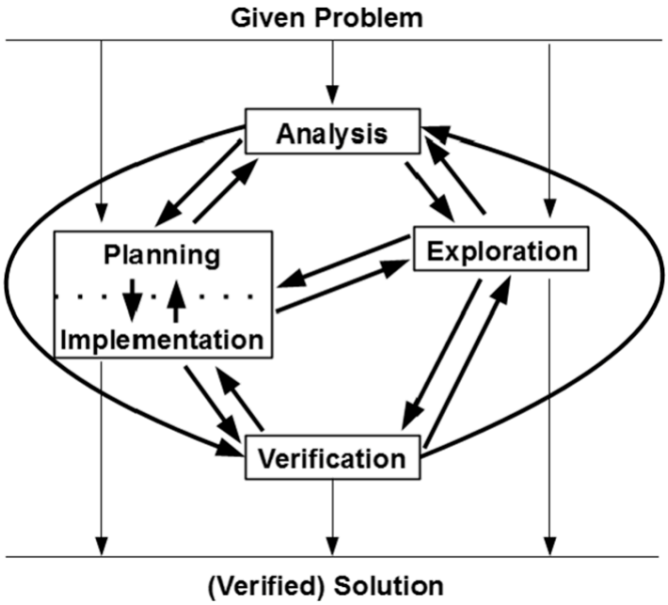
Neben diesen Strategien stellte Pólya auch ein konkretes Modell vor, das den gesamten Ablauf des Problemlösens graphisch in Teilschritte unterteilt. Seine Arbeit war so einflussreich, dass es Rott, Specht und Knipping gelang, alle später folgenden Problemlösemodelle daran auszurichten [RSK21]. Hier sind einige solcher Modelle im Überblick:



In der Grafik werden mehrere deskriptive und normative Modelle miteinander verglichen. Für die Entwicklung eines automatisierten Problemlösetools ist ein deskriptives Modell wegen seiner klaren

Struktur vermutlich einfacher zu implementieren, allerdings würde dies die Flexibilität des Tools hemmen. Einige der Modelle sind auch offensichtlich kritisch. Pólyas lineares Lösungskonzept sieht z.B. eine einzige Planungsphase vor und erkennt nicht das Vor- und Zurücktasten beim Problemlösen an. Insbesondere wenn wir auch freiere Problemthemen einbeziehen wollen und den Lösungsprozess ergebnisoffen gestalten wollen, lohnt es sich, deskriptive Modelle als Vorlage zu wählen.

[RSK21] schlägt ein neues deskriptives Modell vor:



Die Stärke dieses Modells liegt darin, dass eine vermeintliche Erkenntnis als Lösung markiert werden kann, ohne dass diese notwendigerweise richtig sein muss. Der Output kann nach fast jeder Phase ausgegeben werden und die lösende Person kann von jeder Phase in jede andere wechseln.

Beobachten wir Menschen dabei, wie sie mathematische Probleme angehen, so können wir eine konkrete Reihenfolge $(P_i)_{i\in[N]\subset\mathbb{N}}$ der Phasen beobachten. Diese, gepaart mit dem Output ω des Systems, der während des Prozesses ausgegeben wurde, könnte naiv als Inputpaar für ein neuronales Netz genutzt werden, welches so trainiert wird, selbst zu entscheiden, wann in welche Phase gewechselt wird. Die Bewertungsfunktion wäre die Richtigkeit der Lösung. Das System hätte also eine hohe Flexibilität, würde die Heuristik des mathematischen Problemlösens lernen, würde aber gleichzeitig keine Garantie darauf abgeben, dass die Ergebnisse korrekt sind.

Der Prozess ist dem Menschen nachempfunden, der genauso Fehler machen kann. So blieben beispielsweise die fälschlicherweise als korrekt geglaubten Kempe-Ketten im Beweisversuch des Vierfarbensatzes durch Alfred Kempe 1879 für 11 Jahre unentdeckt. Auch Andrew Wiles musste einige Monate nachdem er 1993 vermeintlich für den Beweis von Fermats letztem Satz gefeiert wurde, diesen aufgrund von unentdeckten Fehlern zunächst wieder zurückziehen.

Das Ziel soll aber nicht sein, ein System zu kreieren, das mathematisch unsicheres Halbwissen erzeugt. Solange wir der Korrektheit des Outputs nicht vertrauen können, stellt sich die Frage, wie wir die bisherigen Systeme trotzdem sinnvoll im alltäglichen Problemlöseprozess von Forschenden nutzen können.

Eine Antwort ist, eine KI so zu konstruieren, dass sie in einem formalen Verifikationssystem Aussagen generieren kann, mit den Regeln des formalen Systems selbst. Alle Aussagen sind dann korrekt, allerdings ist die KI auf das System limitiert. Die Implementierung und Analyse der

Ergebnisse wird von Menschen durchgeführt. Wir werden uns ein solches Beispiel im Kapitel 2.1 näher ansehen.

Oder aber man erlaubt der KI freiere Konstruktionen zu generieren, auch solche, die in der Untersuchung als „falsch“ gelten. Dadurch kann sie einfacher in einem großen und ggf. komplizierten Suchraum lernen. Die Ergebnisse können dann von Menschen analysiert werden; die KI unterstützt also bei der Exploration unübersichtlicher Strukturen und Beispielgenerierung. Implementierung und Verifikation obliegen ebenfalls dem Menschen. Im Kapitel 2.2 sei daher eine solche, flexiblere Methode vorgestellt.

Im letzten Beispiel, Kapitel 2.3, geht es um eine Implementierung, die dem oben vorgestellten deskriptiven Modell ähnelt. Das Modell insgesamt kann Fehler machen, nämlich genau dann, wenn die Übersetzung der Problemstellung in ein formales Verifikationssystem schiefgeht. Da das Modell ein sehr generelles formales System, ein formales Beweissystem, verwendet, kann es allerdings einfacher für einen breiteren Bereich an Problemen genutzt werden.

Bevor wir uns ein konkretes Fallbeispiel näher ansehen, an dem diverse Aspekte des mathematischen Problemlösens veranschaulicht werden sollen, eine Anmerkung zum Schluss: Keines der Modelle in [RSK21] ist vollständig. Alle Modelle, inklusive der deskriptiven, unterliegen dem Bias-Problem, dass die Forschenden konkrete Kategorien festgelegt haben, die nach festen Strukturen ineinander übergehen. Die Verhaltensweisen von Probanden werden dann auf das Modell projiziert.

Dem aktuellsten Modell von Rott, Specht und Knipping selbst fehlt beispielsweise ein interessanter Aspekt, der sicherlich für die mathematische Forschungsarbeit relevant ist. Eine Phase, die im Modell von Wilson u. a. aufgezeigt wird, ist die des Problem Posing, also nicht nur das Stellen von Unterproblemen, sondern auch das Formulieren neuer Probleme [RSK21].

1.2. Fallbeispiel: Tadaos Eiskunstlaufproblem

Das folgende, nach einem Rätsel von Tadao Kitazawa adaptierte Problem [Kit21] soll als Beispiel für mehrere Aspekte dieser Arbeit dienen.

Die erste Besonderheit ist, dass sich die Regeln recht einfach durch Abbildungen kommunizieren lassen. Die physische Situation inferiert zudem Regeln, die nicht explizit kommuniziert werden müssen. Die lesende Person sei eingeladen, zunächst selbst eine Lösung zu finden und sich beim Problemlöseprozess zu beobachten. Hier ist eine Niederschrift des Rätsels aus dem *Mathe im Adventskalender* des FU-Mentoring-Programms 2024:

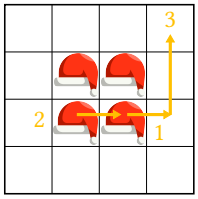
Dies ist zunächst nur ein Rätsel, d.h. es gibt ein konkretes Ziel und bekannte Regeln. Als Rätsel ist es vermutlich sogar so designt, dass die Suche nach einer Lösung Spaß macht oder es einen verblüffenden Twist gibt. Das Rätsel zu lösen, ist also im Vergleich zu aktueller mathematischer Forschungsarbeit ein deutlich einfacherer Fall des Problemlösens.

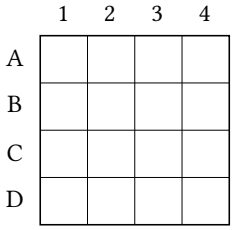
Das Rätsel ist von vielen nicht-repräsentativen Menschen aus meinem Umfeld betrachtet worden. Die Lösungsdauer hat sich zwischen den Bachelormathestudierenden mit 10min bis hin zu 5h bei einer Architekturstudierenden unterschieden. Einige Schüler:innen, die es für 2h versucht haben, sind zu keiner Lösung gekommen.

In der Problemanalyse stellten fast alle Probanden fest, dass der erste und letzte Zug vollständig symmetrisch und daher immer gleich ist. Die Planungsphase hielt sich bei allen in Grenzen, vermutlich da die offensichtlichste Lösungsstrategie das Ausprobieren war. In der Explorationsphase wurde entweder auf Papier skizzenhaft probiert, oder, eine deutlich effektivere Art, mit Münzen geschoben. Auf dem Papier kam es bei den Lösenden häufiger zu Regelverletzungen, was möglicherweise daran liegt, dass oft aus Unlust, das Spielfeld häufig aufzuzeichnen, mehrere Schritte hintereinander im Kopf durchgeführt wurden.

Nach Finden einer vermeintlichen Lösung gab es häufig eine kurze Planungsphase, nämlich die, wie die Lösung effektiv aufgeschrieben werden könnte. Neben der visuellen Darstellung mit Pfeilen wie in den Erklärbildern wurde die Grid häufig beschriftet, sodass die Bewegung eines Wichtels als Koordinatenpaar oder als zweistellige Hexadezimalzahl dargestellt werden konnte. Auch die

Zuordnung der Namen der Wichtlis in Kombination mit der Beschreibung, in welche Richtung und um wie viele Felder diese geschubst wurden, war eine gewählte Darstellung.

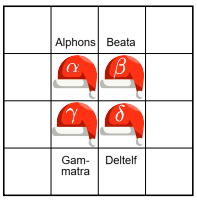




(3C,4C), (2C,3C), (4C,4A)

1	2	3	4
5	6	7	8
9	A	B	C
D	E	F	G

BC, AB, C4



$\delta \rightarrow 1, \gamma \rightarrow 1, \delta \uparrow 2$

Tatsächlich ist Darstellungswechsel in diesem Problem – eine von Pólyas Generalstrategien – äußerst sinnvoll. Die erste ist sicherlich die für Menschen am intuitivsten nachvollziehbare.

Die zweite erlaubt es, eine beliebige Stellung auf der Eisfläche durch vier Positionsvektoren in der x- und y-Achse darzustellen z.B. $\{(2,2),(3,2),(4,1),(3,3)\}$ (A,B,C,D werden in der y-Achse auch mit $1,2,3,4$ repräsentiert). So gelingt es einfach alle möglichen Züge zu bestimmen, indem über alle Kombinationen hinweg genau eine Koordinate in einem Koordinatenpaar auf eine der möglichen anderen Ziffern gebracht wird, z.B. $\{(2,2),(4,2),(4,1),(3,3)\}$. Dann müssen nur noch die ungültigen Fälle aussortiert werden.

Außerdem eignet sich die Darstellung gut, um die einzelnen Vektoren mit Symmetriematrizen zu multiplizieren, um die symmetrisch identischen Spielfeldpositionen zu berechnen.

Auf der anderen Seite ist diese Darstellung ungeeignet, um Zustände eindeutig zu beschreiben, denn wir können die Vektoren unter der Voraussetzung, dass die Namen der Wichtlis keine Bedeutung tragen, beliebig permutieren. $\{(2,2),(3,2),(4,1),(3,3)\}$ ist dann identisch mit $\{(4,1),(3,2),(2,2),(3,3)\}$.

Eine sortierte Darstellung der Eisfläche lässt sich besser mit der dritten Darstellung herleiten. Die binäre Matrix

$$\begin{pmatrix} 0 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}$$

ist ein direktes Abbild der Eisfläche nach den drei Zügen und lässt sich in zwei Bytes umwandeln: `0001 0110 0010 0000`, wobei 1 an der i. Stelle von links aufwärts gezählt dafür steht, dass sich auf dem i_{16}. Feld in Hexadezimaldarstellung ein Wichtli befindet. Es wird sofort ersichtlich, dass es nur $\binom{16}{4} = 1820$ verschiedene Stellungen geben kann.

Die letzte Darstellung mag unübersichtlich wirken, nutzt aber aus, dass die Wichtlis nur in einer Achse verschoben werden können. Während ein Zug in Hexadezimaldarstellung ein Byte groß ist $(2^4 \cdot 2^4)$, lässt sich mit der letzten Darstellung mit (Wichtli $\in [4]$, Richtung $\in [4]$, Distanz $\in [3]$) in $2^2 \cdot 2^2 \cdot 2^2$ auf 6 Bit komprimieren. Das Eiskunstlaufproblem soll später in das N-dimensionale erweitert werden. Ab Dimension 6 nimmt diese Schreibweise den halben Speicherplatz ein: 3 Byte lassen sich auf 12 Bit kürzen.

Eine Strategie, die in diesem Problem enorm hilfreich ist, ist das Rückwärtsarbeiten, d.h. die Regeln andersherum zu definieren. Wir werden das später noch verwenden.

In den meisten Fällen hat die Wahl der Darstellung dafür gesorgt, dass die Probanden Zug für Zug die Spielregeln kontrollierten. Der Akt der ausführlichen Niederschrift sorgte also für einen sauberen Verifikationsprozess. Trotzdem kam es auch zu falschen Abgaben, vor allem dann, wenn mehrere Züge hintereinander gleichzeitig visualisiert worden sind.

Die Problem-Posing-Phase wurde im Wesentlichen nur von den Bachelormathestudierenden erreicht, die im Rätsel bereits neue Forschungsprobleme sahen. Sie fragten sich, was die kürzeste Lösung sei und wie viele davon existieren; ob es Stellungen gibt, die nie erreicht werden können; wie das Rätsel mit hexagonalen Waben oder Dreiecken als Grundstruktur aussieht oder wie die Wichtlis sich in einem N-dimensionalen Gitter verhalten würden?

Schon wird das Rätsel zum Forschungsproblem. Verschiedenste Hypothesen können aufgestellt werden und mit allen Mitteln der Mathematik wird das Problem angegangen. Der Mehrwert für andere Disziplinen mag nicht auf Anhieb klar werden, Bendegem argumentiert jedoch, dass die Arbeit an „unterhaltsamen“ Problemen unerwartet zu Erkenntnissen oder Tools führen kann, die in anderen Bereichen von Wert werden können [Ben05].

Für eine Verallgemeinerung des Problems lohnt es sich, die Regeln zunächst noch einmal sauber aufzuschreiben. Wir sehen, dass Regel 4. in der visuellen Erklärung nicht genannt worden ist, sich aber aus der Situation ableiten lässt.

Sei ein 4x4 Spielfeld gegeben. Auf jedem der 4 Felder, die nicht am Rand liegen, liegt ein Stein. Jetzt darf, Zug um Zug, jeweils ein Stein bewegt werden. Das Ziel ist, dass die Steine in den vier Eckfeldern des Spielfelds liegen. Das sind die Regeln:

1. Pro Zug wird nur ein Stein bewegt.
2. Der Stein, der bewegt wird, muss zu Beginn des Zuges mindestens einen Stein auf einem horizontal oder vertikal benachbarten Feld liegen haben.
3. Gilt 2. für einen Stein, so darf dieser horizontal oder vertikal im Spielfeld um eine beliebige Anzahl an Spielfeldern verschoben werden.
4. Bei 3. darf ein Stein **nicht über** einen anderen Stein oder über das Spielfeldende hinaus verschoben werden.

1.2.1. Tadaos Eiskunstlauf im N-dimensionalen

Wir wollen das Eiskunstlaufproblem in der Dimension $N \in \mathbb{N}$ betrachten.

Das Spielfeld besteht dann aus 2^N Steinen, die als N-dimensionale Koordinatenvektoren $(x_i)_{i\in[N]} \in [4]^N$ dargestellt werden können.

Anmerkung: Wir schreiben auch (x_i) statt $(x_i)_{i\in[N]}$, wenn die Dimension des Vektors klar ist.

Die Menge aller möglichen Spielfelder ist $T_N := \{T \subset [4]^N \mid |T| = 2^N\}$ mit dem Start- und Zielspielfeld

$$T^{(N)}_{\text{Start}} := \{(x_i) \in [4]^N \mid x_i \in \{2,3\}\} \in T_N$$

$$T^{(N)}_{\text{Ziel}} := \{(x_i) \in [4]^N \mid x_i \in \{1,4\}\} \in T_N$$

Ein Spielzug ist ein Vektorpaar $(x, y) \in [4]^N \times [4]^N =: Z_N$

Alle möglichen Zug-Spielfeldkombinationen fassen wir in der folgenden Menge zusammen:

$$
\begin{aligned}
(T_N \times Z_N)^\star := \Big\{ & (T, x, y) \in T_N \times [4]^N \times [4]^N \mid \qquad x =: (x_i), y =: (y_i), \\
& 1)\ \ x \in T, y \notin T \qquad \text{(es wird genau ein Stein bewegt)} \\
& 2)\ \ \exists \tilde{e} \in \{\pm e_i\}_{i \in [N]} : x + \tilde{e} \in T \Big) \qquad \text{(der Stein ist zu Beginn benachbart)} \\
& \quad e_i \text{ sind die Einheitsvektoren, } + \text{ eine partielle Vektoraddition} \\
& 3)\ \ \exists! \hat{\imath} \in [N] : x_{\hat{\imath}} \neq y_{\hat{\imath}} \qquad \text{(es wird ausschließlich orthogonal bewegt)} \\
& 4)\ \ \forall z \in [4], \min\{x_{\hat{\imath}}, y_{\hat{\imath}}\} < z < \max\{x_{\hat{\imath}}, y_{\hat{\imath}}\} : \nexists (t_i) \in T : t_i = \begin{cases} z & , i = \hat{\imath} \\ x_i & , \text{sonst} \end{cases} \\
& \quad \text{(es gibt keinen Stein, der dazwischen liegt)}\}
\end{aligned}
$$

Einen Zug durchzuführen können wir als partielle, binäre Multiplikation darstellen:

$$
\begin{aligned}
\cdot : T_N \times Z_N &\ \rightharpoonup T_N \\
(T, (x, y)) &\mapsto (T \setminus \{x\}) \cup \{y\}
\end{aligned}
$$

Der Definitionsbereich $\mathrm{Def}(\cdot)$ soll isomorph zu $(T_N \times Z_N)^\star$ sein, um nur erlaubte Züge zuzulassen.

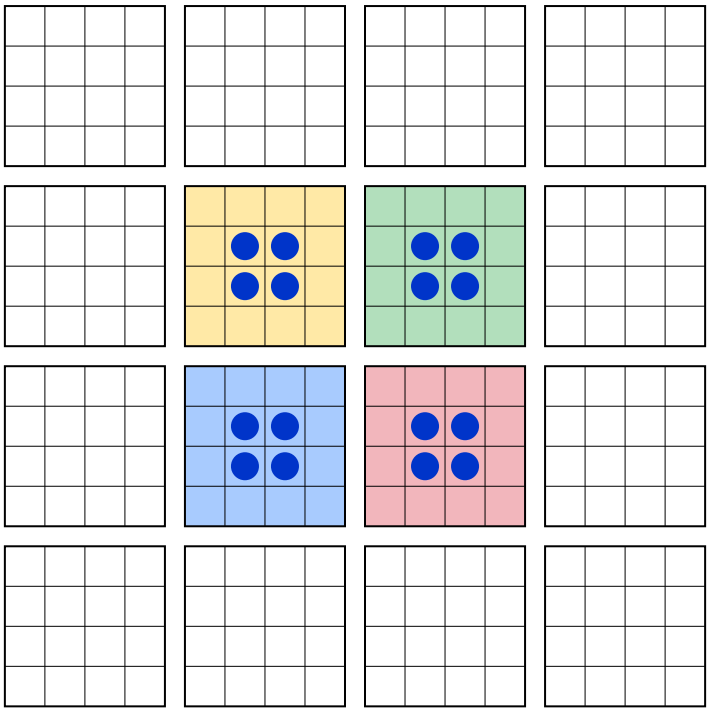
Das Ziel des Rätsels ist es, eine Reihe $(z_i)_{i \in [n]}$ zu finden für ein $n \in \mathbb{N}$, für die der folgende Ausdruck wohldefiniert und korrekt ist:

$$
\begin{aligned}
T^{(N)}_{\text{Start}} \cdot z_1 \cdot z_2 \cdot \ldots \cdot z_n = T^{(N)}_{\text{Ziel}} \qquad & \text{bzw.} \\
T^{(N)}_{\text{Start}} \cdot \prod_{i=1}^{n} z_i = T^{(N)}_{\text{Ziel}} \qquad & \text{oder kurz} \\
T^{(N)}_{\text{Start}} \cdot \prod (z_i) = T^{(N)}_{\text{Ziel}} &
\end{aligned}
$$

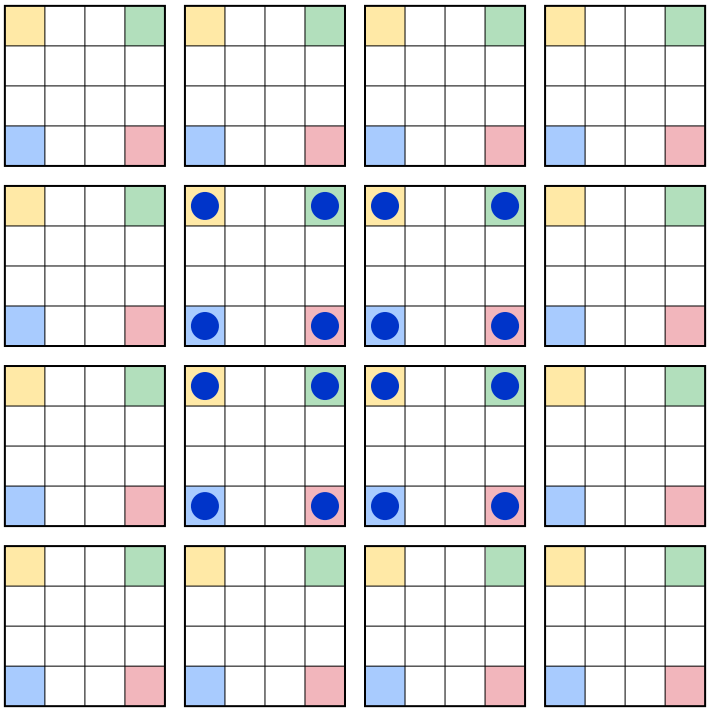
1.2.2. Intuition der Lösbarkeit im N-dimensionalen

Im Folgenden sei ein Algorithmus vorgestellt, der für jede beliebige Dimension eine, nicht notwendigerweise die kürzeste, Lösung angibt. Grundlage dafür ist die Vorstellung, dass sich der N-dimensionale Raum in 2-dimensionale Scheiben schneiden lässt. Wenn wir eine Lösung im 2-dimensionalen haben, können wir diese in jeder Scheibe anwenden und lösen so insgesamt das Problem in N.

Als Beispiel haben wir im 4-dimensionalen die „offensichtlich sichtbaren“ 2D-Scheiben und können die vier mittigen zuerst lösen:



Danach müssen wir auf den verbleibenden Koordinatenachsen die Steine an die richtige Position bringen. Wir können dort auch wieder innerhalb eines 2D-Problems lösen. Die zu lösenden Scheiben sind farblich markiert:

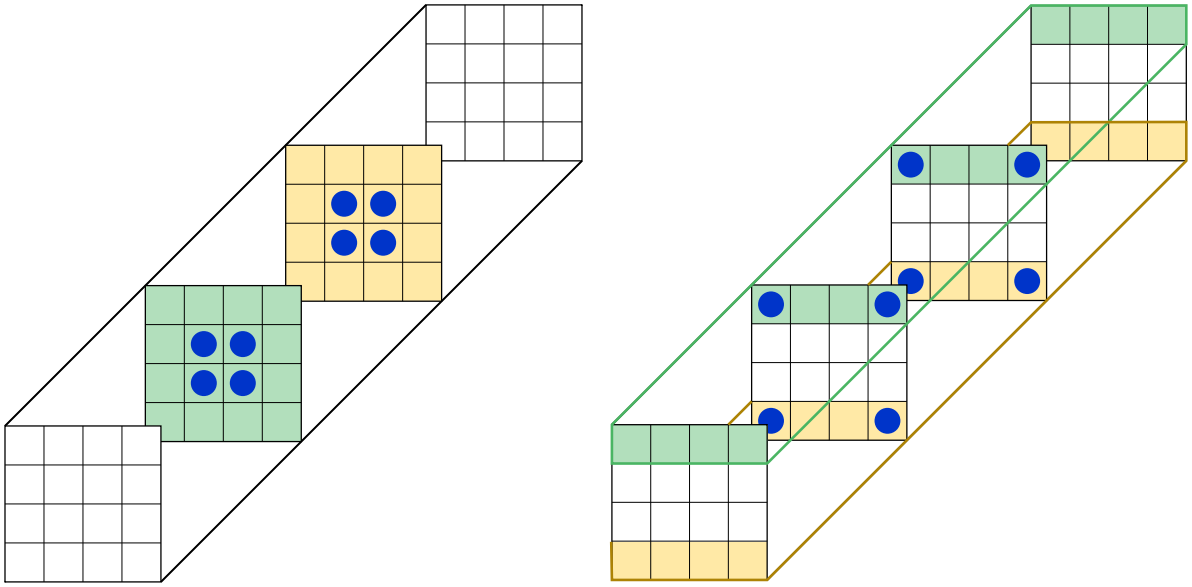


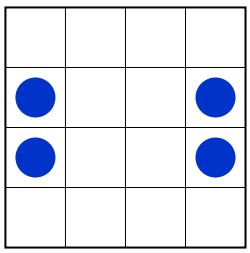
Bei dieser expliziten Lösung kann sehr einfach die Anzahl der Schritte bis zur Lösung gezählt werden. Dass diese Anzahl optimal ist, also eine kürzeste Lösung im N-dimensionalen konstituiert, bedarf eines seperaten Beweises, ist allerdings nicht zu vermuten (siehe Kapitel 2.2.3).

Allgemein können wir für eine Dimension N und eine beliebige Partitionierung $N = \sum(p_i)$, wobei $(p_i) \in \mathbb{N}^t, t \in \mathbb{N}$ auch den N-dimensionalen Raum in p_i-dimensionale „Scheiben“ zerschneiden. Haben wir dann eine Lösung für alle Dimensionen p_i (für alle $i \in [t]$), so können wir mit dem

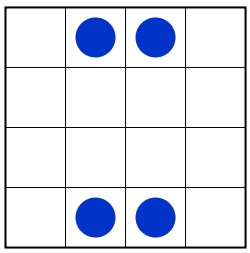
Algorithmus das Problem in N lösen. Ganz einfach geht das für alle geraden N mit einer Partitionierung nur aus 2en, denn dort haben wir bereits eine Lösung gefunden.

Solange wir noch keine Lösung im 3-dimensionalen haben, müssen wir uns aber Gedanken machen, wie wir das Problem auch für ungerade N lösen können. Dabei entsteht in jedem Fall eine Überlappung der Scheiben, wodurch wir Steine, die bereits auf einer Achse optimiert sind, noch einmal bewegen. Wir erkennen am Beispiel im 3-dimensionalen, dass dadurch eine andere Position entsteht, die es zu lösen gilt. Nachdem wir die „offensichtlich sichtbaren“ Scheiben gelöst haben (links), bleibt bei der anderen Zerschneidung (rechts), in jeder Scheibe die Position unten übrig:

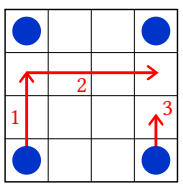




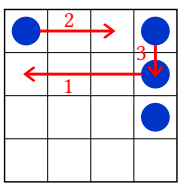
bzw.



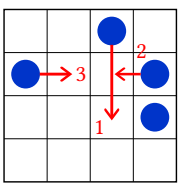
Und tatsächlich können wir für diese Position eine kürzeste Lösung in 8 Zügen finden. Die Lösung dieses Spielfelds und der des normalen Startspielfelds sind hier graphisch dargestellt, allerdings rückwärts vom Zielspielfeld ausgehend:



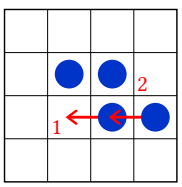
1 – 3



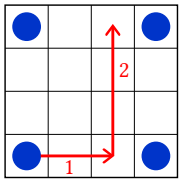
4 – 6



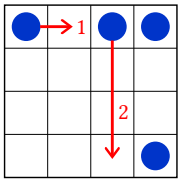
7 – 9



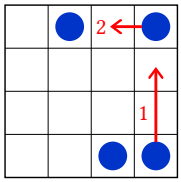
10, 11



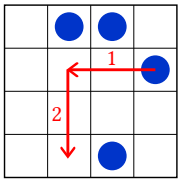
1, 2



3, 4



5, 6



7, 8

1.2.3. Beweis der Lösbarkeit im N-dimensionalen

Sei $(p_i) \in \mathbb{N}^t$ mit $\sum(p_i) = N$ eine Partitionierung von N mit der Eigenschaft, dass für alle $i \in [t]$ eine Lösung $\left({}_{p_i}z_j\right) \in Z_{p_i}^{\chi(p_i)}$ existiert, d.h. $T_{\text{Start}}^{(p_i)} \cdot \prod\left({}_{p_i}z_j\right) = T_{\text{Ziel}}^{(p_i)}$, wobei $\chi(p_i)$ die Anzahl an Zügen einer Lösung im p_i-dimensionalen angibt. Es gelte ${}_{p_i}z_j := \left(\left({}_{p_i,j}x_l\right), \left({}_{p_i,j}y_l\right)\right)$.

Außerdem sei das Achsenintervall definiert als

$$(a,b]_\Sigma := \left[\sum_{i=1}^{b} p_i\right] \setminus \left[\sum_{i=1}^{a} p_i\right]$$

Satz über die Lösbarkeit von Tadaos Eiskunstlaufproblem in $N \in 2\mathbb{N}$. Dann ist

$$T_{\text{Start}}^{(N)} \cdot \prod_{s=1}^{t} \prod_{\substack{(d_i)\in[4]^N \\ d_i\in\{1,4\} \text{ für } i\in(0,s-1]_\Sigma \\ d_i=1 \text{ für } i\in(s-1,s]_\Sigma \\ d_i\in\{2;3\} \text{ für } i\in(s,t]_\Sigma}} \prod_{k=1}^{\chi(p_s)} \mathfrak{z}(s,(d_i),k) = T_{\text{Ziel}}^{(N)}$$

wobei $\mathfrak{z}(s,(d_i),k) := ((x_i),(y_i))$ für

$$x_i := \begin{cases} {}_{p_s,k}x_r & \text{für } \; i = r + \sum_{j=1}^{s-1} p_j \;,\; r \in [p_s] \\ d_i & \text{sonst} \end{cases}$$

$$y_i := \begin{cases} {}_{p_s,k}y_r & \text{für } \; i = r + \sum_{j=1}^{s-1} p_j \;,\; r \in [p_s] \\ d_i & \text{sonst} \end{cases}$$

Für den Beweis zeigen wir per Induktion zunächst folgendes Lemma:

Lemma. Für alle $\tau \in [t] \cup \{0\}$ gilt:

1. Das Spielfeld

$$\mathcal{T}_\tau := T_{\text{Start}}^{(N)} \cdot \prod_{s=1}^{\tau} \prod_{\substack{(d_i)\in[4]^N \\ d_i\in\{1,4\} \text{ für } i\in(0,s-1]_\Sigma \\ d_i=1 \text{ für } i\in(s-1,s]_\Sigma \\ d_i\in\{2;3\} \text{ für } i\in(s,t]_\Sigma}} \prod_{k=1}^{\chi(p_s)} \mathfrak{z}(s,(d_i),k)$$

   ist wohldefiniert, d.h. für jeden Zug z der auf ein Spielfeld T in $\mathcal{T}_\tau$ wirkt, gilt $(T,z) \in (T_N \times Z_N)^\star$

2. Für alle Steine $(x_i) \in \mathcal{T}_\tau$ gilt, dass deren Einträge an den Koordinaten 1 bis $\sum_{i=1}^{\tau} p_i$ 1 oder 4 sind, sonst 2 oder 3.

Induktionsanfang: $\tau = 0$

Offensichtlich gilt: T^N_{Start} ist 1. wohldefiniert und 2. alle dessen Steine haben an allen Koordinaten eine 2 oder 3, genau das ist schließlich die Definition des Startspielfelds.

Induktionsschritt: $\tau - 1 \to \tau$

Wir wollen zeigen, dass

$$\mathcal{T}_\tau = \mathcal{T}_{\tau-1} \cdot \prod_{\substack{(d_i)\in[4]^N \\ d_i\in\{1,4\} \text{ für } i\in(0,\tau-1]_\Sigma \\ d_i=1 \text{ für } i\in(\tau-1,\tau]_\Sigma \\ d_i\in\{2;3\} \text{ für } i\in(\tau,t]_\Sigma}} \prod_{k=1}^{\chi(p_\tau)} \mathfrak{z}(\tau,(d_i),k)$$

1. wohldefiniert ist und 2. die Steine korrekt verändert werden.

Betrachte die Definition von $\mathfrak{z}$, insbesondere die der Steine, die in $\mathfrak{z}$ verändert werden. Hier ist offensichtlich, dass nur die Einträge an den Koordinaten in $(\tau-1,\tau]_\Sigma$ verändert werden. Laut Induktionsvoraussetzung müssten diese Koordinateneinträge $\in \{2,3\}$ liegen. Außerdem gilt für alle Züge des rechten Produktzeichens, dass ein (d_i) fixiert ist, d.h. es werden genau die Koordinaten so verändert wie in der p_τ-dimensionalen Lösung, nur jetzt an den Koordinaten $(\tau-1,\tau]_\Sigma$.

Daher gelten in jedem der Züge unter einer Fixierung (d_i) die Aspekte 1)–4) aus $\left(T_{p_\tau} \times Z_{p_\tau}\right)^\star$ auch für $(T_N \times Z_N)^\star$. Nach Abschluss des rechten Produkts erhalten wir ein neues Spielfeld $T'_{\tau-1}$, in dem für alle Steine $(x_i) \in \mathcal{T}'_{\tau-1}$ mit $x_i = d_i$ für $i \in [N]\setminus(\tau-1,\tau]_\Sigma$ gilt und $x_i \in \{1,4\}$ sonst, dank der zuletzt durchgeführten Züge.

Für alle anderen Steine $(x_i) \in \mathcal{T}'_{\tau-1}$ gilt immer noch $x_i \in \{1,4\}$ für $i \in (0,\tau-1]_\Sigma$ und $x_i \in \{2,3\}$ aufgrund der Induktionsvoraussetzung. Wir müssen also alle Kombinationen (d_i) durchgehen, um alle Steine zu verändern.

Die Anzahl solcher Kombinationen können wir von den Konditionen des mittleren Produkts ablesen: $2^{N-p_\tau} = \frac{2^N}{2^{p_\tau}}$ – diese Formel erinnert an die Intuition, dass wir den N-dimensionalen Raum in p_τ-dimensionale „Scheiben“ zerschneiden. In jeder p_τ-dimensionalen Lösung werden 2^{p_τ} viele Steine verändert, d.h. wir haben am Ende $2^{N-p_\tau} \cdot 2^{p_\tau} = 2^N$ viele veränderte Steine, die alle an $x_i \in \{1,4\}$ geworden sind für $i \in (\tau-1,\tau]_\Sigma$. Da sich die veränderten Steine nicht überlappen, wegen der einzigartigen Koordinatenachsenkombinationen (d_i), wurden tatsächlich alle Steine bewegt und tragen diese Eigenschaft. Somit gilt der 2. Aspekt des Lemmas. ∎

Korrular. Es gilt

$$\mathcal{T}_t = T^{(N)}_{\text{Ziel}}$$

Durch das Lemma wissen wir, dass in $\mathcal{T}_t$ gilt, dass für alle Steine stimmt, dass deren Einträge in den Koordinaten 1 bis $\sum_{i=1}^t p_i = N$ 1 oder 4 sind. Das ist aber genau die Definition des Zielspielfelds. Der Satz über die Lösbarkeit von Tadaos Eiskunstlaufproblem in N ist somit entschieden. ∎

Satz über die Lösbarkeit von Tadaos Eiskunstlaufproblem in $N \in 2\mathbb{N}+1$. Sei $(p_i) \in \mathbb{N}^{t-1}$ mit $\sum(p_i) = N-1$ eine Partitionierung und $p_t = 2$ eine Überlappung, sodass $\sum_{i=1}^t p_i = N+1$.

Dann ist

$$\underbrace{\left(T^{(N)}_{\text{Start}} \cdot \prod_{s=1}^{t-1} \prod_{\substack{(d_i)\in[4]^N \\ d_i\in\{1,4\} \text{ für } i\in(0,s-1]_\Sigma \\ d_i=1 \text{ für } i\in(s-1,s]_\Sigma \\ d_i\in\{2;3\} \text{ für } i\in(s,t]_\Sigma}} \prod_{k=1}^{\chi(p_s)} \mathfrak{z}(s,(d_i),k) \right)}_{\mathtt{T}} \cdot \prod_{\substack{(d_i)\in[4]^N \\ d_{N-1}=1=d_N \\ d_i\in\{1,4\}}} \prod_{k=1}^{8} \mathtt{z}(s,(d_i),k) = T^{(N)}_{\text{Ziel}}$$

wobei $\mathfrak{z}$ definiert wie zuvor und $\mathtt{z}(s,(d_i),k) := ((x_i),(y_i))$ für

$$x_i := \begin{cases} {}_k x'_1 & \text{für } \ i = N-1 \\ {}_k x'_2 & \text{für } \ i = N \\ d_i & \text{sonst} \end{cases} \qquad y_i := \begin{cases} {}_k y'_1 & \text{für } \ i = N-1 \\ {}_k y'_2 & \text{für } \ i = N \\ d_i & \text{sonst} \end{cases}$$

wenn $(({}_k x'_i),({}_k y'_i))$ der k-te Lösungszug des 2-dimensionalen Sonderfalls ist, bei dem alle Steine auf einer Achse bereits $\in \{1,4\}$ sind, in der anderen Achse aber noch $\in \{2,3\}$.

Der Beweis wird analog argumentiert wie der für N gerade. Das Spielfeld $\mathtt{T}$ hat die Eigenschaft, dass alle Vektoren in den Koordinatenachsen 1 bis $N-1$ bereits Einträge $\in \{1,4\}$ haben. Das rechte Doppelprodukt iteriert dann über alle möglichen Achsenkombinationen und löst 2-dimensional den Sonderfall auf. ■

1.2.4. Anzahl der Züge der allgemeinen Lösung in N

Die Produktformeln erlauben, dass man für ein beliebiges N sehr schnell eine obere Schranke für die Anzahl an benötigten Zügen zur Lösung ablesen kann. Partitionieren wir beispielsweise in 2er-Paaren, d.h. $p_i = 2 \ \forall i \in [t]$, so können wir die folgenden Formeln direkt von den Produkten ablesen, da wir eine Lösung im 2-dimensionalen mit $\chi(2) = 11$ kennen. Anzahl Züge ist:

$$\begin{aligned} N \text{ gerade:} \quad & \frac{N}{2} \cdot 2^{N-2} \cdot 11 && = 11N \cdot 2^{N-3} \\ N \text{ ungerade:} \quad & \frac{N-1}{2} \cdot 2^{N-2} \cdot 11 + 2^{N-2} \cdot 8 = (11N+5) \cdot 2^{N-3} \end{aligned}$$

Vergleichen wir das mit der unteren Schranke, also mit der minimalen Anzahl an Zügen, die ein Rätsel in jedem Fall dauert, nämlich $N \cdot 2^N$, da jeder der 2^N Steine in einem Zug immer nur in einer seiner N Achsen verändert werden kann:

N	$N \cdot 2^N$	Zuganzahl $\chi(N)$
2	8	11
3	24	38
4	64	88
5	160	240
6	384	528
...	...	...

Anmerkung. Bemerke, dass wir die obere Schranke 38 für $N = 3$ haben. Wir werden später eine KI-Lösung mit nur 28 Zügen finden. Damit lassen sich dann wieder neue, kleinere obere Schranken konstruieren, z.B. für $N = 5 = 3 + 2$ können wir mit $2^{N-2} \cdot 11 + 2^{N-3} \cdot 28 = 200 < 240$ eine deutlich kleinere Schranke konstruieren. Langfristig besteht die Hoffnung, dass man einen allgemeinen Zusammenhang finden kann, sodass man immer die kürzeste Lösung $\chi_{\min}(N)$ für alle Dimensionen N bestimmen kann.

2. Konkrete KI-Anwendungsfälle in der mathematischen Forschung

2.1. Falsifikation von Vermutungen

Einer der „einfachsten“ Anwendungsfälle von KI in der mathematischen Forschung ist das Generieren von Beispielkonstruktionen, die bestimmte Eigenschaften erfüllen. Der Algorithmus wird so gebaut, dass er nur korrekte Konstruktionen generieren kann, die dann hinsichtlich einer Eigenschaft bewertet werden können. Mit jeder Iteration werden die generierten Beispiele „besser“ hinsichtlich der Bewertungsfunktion. Im Idealfall entstehen so Konstruktionen, die dann mathematische Vermutungen widerlegen können.

1. Betrachte eine beliebige Struktur (z.B. Graphen, Matrizen, Permutationen), die sich einfach konstruieren lässt. Bezeichne die Menge aller Elemente dieser Struktur als $\mathcal{A}$.
2. Für eine konkrete Eigenschaft bauen wir eine Bewertungsfunktion $f : \mathcal{A} \to \mathbb{R}$. Zu widerlegen gilt, dass es keine Konstruktionen $k \in \mathcal{A}$ gibt, für die gilt $f(k) \sim t$ für einen Schwellwert $t \in \mathbb{R}$ und Vergleichsoperator $\sim \in \{<, >, \leq, \geq, =\}$.
3. Das neuronale Netz generiert Konstruktionen $\mathcal{C} \subset \mathcal{A}$, bewertet diese mit f und trainiert das Netz auf Paaren $(c, f(c))$ für $c \in \mathcal{C}$.
4. Wiederhole Schritt 3. bis eine Konstruktion $k \in \mathcal{C}$ entsteht, für die gilt $f(k) \nsim t$.

Der Artikel von Wagner zeigt auf, dass man auch andersherum arbeiten kann, d.h. man erzeugt „gute“ Konstruktionen und schafft so neue Upper- bzw. Lowerbounds hinsichtlich einer reellen Eigenschaft [Wag21]. Die generierten Beispiele können dann helfen, neue allgemeine Formeln zu finden.

Die erzeugten Beispiele in $\mathcal{C}$ können zwar reichen, eine allgemeine Vermutung zu widerlegen, genügen aber nicht, um die Vermutung allgemein abzulehnen: Die Vermutung über $\mathcal{A}$ mag durch die Konstruktion $k \in \mathcal{C} \subset \mathcal{A}$ widerlegt werden, könnte aber über dem fast identischen Fall $\mathcal{A} \setminus \{k\}$ wahr sein.

Um die Anwendung zu veranschaulichen, betrachten wir ein Beispiel von Wagner selbst [Wag21].

1. Als Struktur betrachten wir ungerichtete, zusammenhängende, schleifenfreie Graphen $G(V, E)$ für Knoten V und Kanten $E \subseteq V^2$, d.h.

   $\mathcal{A} = \{G(V, E) \mid E \subseteq V^2$ für eine endliche Menge V und
   1) $\forall (x, y) \in E : (y, x) \in E$ (ungerichtet)
   2) $\forall x, y \in V, x \neq y, (x, y) \notin E : \exists a_1, ..., a_m \in V, m \in \mathbb{N} : (x, a_1), (a_1, a_2), ..., (a_m, y) \in E$ (zusammenhängend)
   3) $\forall x \in V : (x, x) \notin E$ (schleifenfrei)$\}$

   Für eine einfache Konstruktion dieser Graphen wird eine feste Anzahl $n \in \mathbb{N}$ für die Knoten gewählt, sodass wir

   $$\mathcal{A}_n := \{G(V, E) \in \mathcal{A} \mid |V| = n\}$$

   betrachten. Eine geeignete Repräsentation eines konkreten Graphen G ist die dazugehörige binäre Matrix $M(G) \in \{0, 1\}^{n \times n}$, definiert durch

$$M(G)_{i,j} := \begin{cases} 1 & \text{, falls } (v_i, v_j) \in E \\ 0 & \text{, sonst} \end{cases}$$

für durchnummerierte Knoten $V = \{v_1, ..., v_n\}$.

Offenbar gilt,
- $M(G)$ ist symmetrisch, wegen 1)
- $M(G)$ hat in jeder Zeile/Spalte mindestens einen Eintrag mit 1, wegen 2)
- $M(G)$ hat auf der Hauptdiagonalen ausschließlich 0-Einträge, wegen 3)

2. Um eine passende Bewertungsfunktion f zu finden, müssen wir zunächst die Vermutung formulieren, die es zu widerlegen gilt.

   Definiere dazu den höchsten Eigenwert von $M(G)$ als $\lambda_{\max}$. Außerdem sei die Matchingzahl des Graphen G gegeben als

   $$\mu := \max\{|\mathcal{M}| : \mathcal{M} \subseteq E, \forall (x, y), (x', y') \in \mathcal{M} : |\{x, y, x', y'\}| = 4\},$$

   also die maximale Anzahl an Kanten im Graphen G, die sich nicht in einem gemeinsamen Knoten berühren.

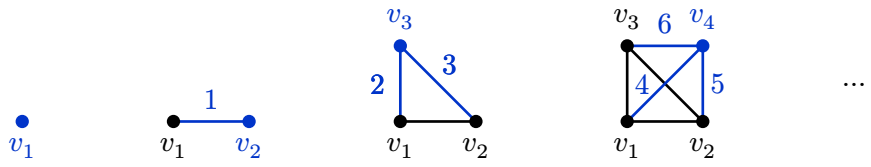
   Die Vermutung lautet:

   $$\lambda_{\max} + \mu \geq \sqrt{n-1} + 1$$

   Diese Werte lassen sich schnell per Computeralgorithmus approximieren. Da die Matrizen symmetrisch sind, ist $\lambda_{\max}$ in jedem Fall reell, μ lässt sich durch den Blossom-Algorithmus ermitteln [Edm65].

   Wir wählen daher $f : \mathcal{A}_n \to \mathbb{R}$ mit $G \mapsto \lambda_{\max} + \mu$ als Bewertungsfunktion und trainieren das neuronale Netz, diese zu minimieren.

3. Für das Generieren der günstigen Konstruktionen $\mathcal{C}_n \subset \mathcal{A}_n$ wird der Graph über seine Kanten codiert. Dabei werden die Kanten mit einer *sinnvollen* Ordnung nummeriert:



   Definiere dazu die Kantenkette $K(G)$ des Graphen G als

   $$K(G) \in \{0, 1\}^k \text{ für maximale Kantenanzahl } k = \frac{n(n-1)}{2}$$

   mit

   $$K(G)_i := \begin{cases} 1 & \text{, falls } \left(v_{p(i)+1}, v_{i-\Sigma(i)}\right) \in E \\ 0 & \text{, sonst} \end{cases}$$

   für die Kantenindex-zu-Knotenindex-Funktion

   $$p : \mathbb{N} \to \mathbb{N} \quad \text{mit} \qquad p(x) := \left\lceil \sqrt{2x + \frac{1}{4}} - \frac{1}{2} \right\rceil$$

   die sich aus den Grenzen

$$\underbrace{\frac{\tilde{p}(\tilde{p}-1)}{2}}_{=:\,\Sigma(x)} < x \leq \frac{\tilde{p}(\tilde{p}+1)}{2} \quad \text{für} \quad \tilde{p} := p(x)$$

herleiten lässt.

So muss ein Transformer-ähnliches Modell (Details werden in [Wag21] nicht genannt) in jedem Schritt aus zwei Tokens, 1 oder 0, bzw. Kombinationen dieser wählen.

4. Tatsächlich gelang es Wagner, eine Konstruktion $k \in \mathcal{C}_{19}$ zu finden, für die gilt $f(k) < \sqrt{n-1} + 1$, was ein Widerspruch zur allgemeinen Vermutung ist. Das ist eine enorme Verbesserung gegenüber Stevanovićs bisherigem Gegenbeispiel $k' \in \mathcal{C}_{600}$ [Ste10].

Erneut sei angemerkt, dass diese Gegenbeispiele nicht allgemein zeigen, dass die Ungleichung immer falsch ist. Die Ungleichung könnte für $n \notin \{19, 600\}$ trotzdem wahr sein oder das Hinzufügen einer simplen Bedingung würde widersprüchliche Graphen wie k und k' vermeiden.

Die Graphen selbst haben also ohne weiteres eine geringe Aussagekraft, liefern aber interessante Beispiele, die von Mathematiker:innen analysiert werden können, um generellere Erkenntnisse zu gewinnen.

Aufgrund der Kettendarstellung erzeugt das neuronale Netz fast ausschließlich Beispiele, die alle Bedingungen einhalten. Allerdings ist die Generierung eines nicht zusammenhängenden Graphens möglich. Solche werden mit der Bewertungsfunktion entsprechend bestraft und herausgefiltert. Im nächsten Kapitel werden wir uns ein Konzept ansehen, das insbesondere in Fällen eingesetzt werden kann, bei denen sehr viele Generierungen die notwendigen Bedingungen nicht erfüllen. Mit einem Algorithmus werden kaputte Konstruktionen „repariert“. So können auch bei komplexeren Bedingungen „gute“ Beispiele generiert werden.

2.2. Musterverstärkung zur Analyse spezieller Strukturen mit PatternBoost

Vier Mathematiker stellten im vergangenen Jahr ein Framework vor, mit dem mathematische Konstruktionen schnell und allgemein untersucht werden können: PatternBoost [Cha+24]. Im Folgenden soll erklärt werden, in welchen Kontexten dieses Framework angewandt werden kann – unter anderem verwende ich es im Kapitel 2.2.3, um eine neue untere Schranke in Tadaos Eiskunstlaufproblem zu ermitteln.

2.2.1. Anwendungsmöglichkeiten PatternBoost

PatternBoost kann in folgenden Szenarien angewandt werden:

1. Wir betrachten eine beliebige Struktur (z.B. Graphen, Matrizen, Permutationen), die sich einfach konstruieren lässt. Bezeichne die Menge aller Elemente dieser Struktur als $\mathcal{A}$.
2. Uns interessieren nur bestimmte Elemente aus $\mathcal{A}$, nämlich solche, die eine Reihe von Eigenschaften haben, die für unsere Untersuchung interessant sind. (z.B. nur Graphen mit 4- und 5-Zykeln, nur Matrizen M mit $\det(M) = \operatorname{trace}(M)$, etc.)

   Die Menge aller Elemente, die diese Eigenschaft haben, bezeichnen wir als $\mathcal{B} \subset \mathcal{A}$. Sie lassen sich dank der zusätzlichen Eigenschaften nicht so einfach konstruieren wie $\mathcal{A}$.
3. Die Konstruktionen in $\mathcal{B}$ lassen sich hinsichtlich verschiedener Eigenschaften bewerten. Wir wollen eine davon konkret untersuchen und erschaffen eine Bewertung $f : \mathcal{B} \to \mathbb{R}$.

   Das Forschungsziel ist $\max f(\mathcal{B})$ zu finden, bzw. zu untersuchen, welche Bedingungen für ein $b \in \mathcal{B}$ notwendig sind, damit $f(b) = \max f(\mathcal{B})$ gilt. PatternBoost findet nicht notwendigerweise $\max f(\mathcal{B})$, kann aber dabei unterstützen, solche $b_i \in \mathcal{B}$ zu finden, für die mit jeder Iteration $i \to i+1$ gilt: $f(b_{i+1}) \geq f(b_i)$.

   Die Hoffnung ist, dass man anhand der resultierenden Konstruktionen b_i ein Muster erkennt und eine mathematische Erkenntnis schlussfolgern kann.
4. Der erste Schritt besteht darin, per Algorithmus eine Menge $\mathcal{C} \subset \mathcal{B}$ zu generieren, also eine zufällige Ansammlung solcher Konstruktionen, die die notwendigen Eigenschaften einhalten.
5. Mit der Bewertung f können wir die resultierenden Konstruktionen bewerten. Wir wählen die besten Konstruktionen $\mathcal{C}_{\text{best}} \subset \mathcal{C}$ aus, z.B. mit $|\mathcal{C}_{\text{best}}| = p \cdot |\mathcal{C}|$ und $\forall c \in \mathcal{C}_{\text{best}}, c' \in \mathcal{C} \setminus \mathcal{C}_{\text{best}} : f(c) \geq f(c')$ für ein kleines $p \in [0,1]$ z.B. $p = 0.2$.

   Das neuronale Netz wird jetzt trainiert, den Fehler für alle Paare $(c, f(c))$ für alle $c \in \mathcal{C}_{\text{best}}$ so klein wie möglich zu halten.
6. Jetzt generiert der Transformer eine neue Menge an Konstruktionen $\mathcal{D} \in \mathcal{A}$.
7. Ein Reparaturalgorithmus $R : \mathcal{A} \longrightarrow \mathcal{B}$ korrigiert falsche Konstruktionen möglichst minimalistisch zu solchen, die alle Bedingungen einhalten. Wir erhalten mit $R(\mathcal{D}) =: \mathcal{C}' \subset \mathcal{B}$ eine neue Menge, die wir in Schritt 5 bewerten können.

Anmerkung: Ggf. ist es nicht möglich, einen naiven Korrekturalgorithmus zu entwickeln, sodass das $\text{Bild}(R) = R(\mathcal{A})$ vollständig in $\mathcal{B}$ liegt. Es kann dennoch vorteilhaft sein, einen Algorithmus $R : \mathcal{A} \to \mathcal{A}$ zu implementieren, für den zumindest gilt: $|\mathcal{B}| < |\{a \in \mathcal{A} | R(a) \in \mathcal{B})\}|$.

In diesem Fall muss auch die Bewertung f angepasst werden auf $f : \mathcal{A} \to \mathbb{R}$, wobei Elemente $x \in \mathcal{A} \setminus \mathcal{B}$ deutlich schlechter bewertet werden müssen.

2.2.2. Anwendungsbeispiel Patternboost

Das folgende Beispiel ist direkt aus der Arbeit von Charton u. a. [Cha+24]. Es gibt für dieses Problem bereits eine explizite Lösung, es hilft aber, die beschriebenen Teilschritte von PatternBoost zu veranschaulichen.

1. Betrachte die Menge aller Graphen mit n Knoten $\mathcal{A} \coloneqq \{(V, E) |\ |V| = n\}$ für ein fest gewähltes $n \in \mathbb{N}$ mit der Menge aller Graphen (V, E) aus der Knotenmenge V und Kantenmenge $E \coloneqq \{\{a, b\} | a, b \in V, a \neq b\}$.

   Ein solcher Graph lässt sich über die binäre, symmetrische Nachbarschaftsmatrix der Kanten repräsentieren oder mithilfe der binären Kantenkette wie in Kapitel 2.1.

2. Untersucht werden sollen alle Graphen $G \in \mathcal{A}$, die keine 3-Zykel haben, d.h.

$$\mathcal{B} \coloneqq \{(V, E) \in \mathcal{A} |\ \forall a, b, c \in V : \{a, b\}, \{b, c\} \in E \Rightarrow \{a, c\} \notin E\}$$

3. Untersucht werden soll die Anzahl der Kanten. Wie viele Kanten können trotz der Vermeidung von Dreiecken erreicht werden? Daher:

$$\begin{aligned} f : \mathcal{B} &\to \mathbb{N}_0 \\ (V, E) &\mapsto |E| \end{aligned}$$

4. Zunächst werden einige zufällige Graphen in $\mathcal{B}$ generiert. Dazu kann auch bereits der Reparaturalgorithmus aus Schritt 7 verwendet werden, denn so müssen nur zufällige Graphen in $\mathcal{A}$ repariert werden.

5. Mit der Bewertung f werden die erzeugten Graphen in $\mathcal{B}$ eingeschätzt und der Transformer auf den $p = 0.25$ besten trainiert.

6. Der Transformer generiert neue Graphen $\mathcal{D}$.

7. Der Reparaturalgorithmus $R : \mathcal{A} \to \mathcal{B}$ entfernt aus jedem Dreieck eine zufällige Kante, bis keine Dreiecke mehr verbleiben, und fügt dann zufällig Kanten ein, solange wie möglich, ohne Dreiecke zu erzeugen. Das Resultat ist ein dreieckloser Graph, $\text{Bild}(R)$ liegt also vollständig in $\mathcal{B}$. Die resultierenden Graphen werden in Schritt 5. wieder bewertet und so weiter.

Nach Schritt 5. in jeder Iteration können wir uns die besten Graphen und deren Performance ansehen.

Tatsächlich erzeugte PatternBoost bereits nach der 5. Iteration fast ausschließlich bipartite Graphen, die notwendig sind, um die Anzahl an Kanten bei Graphen ohne 3-Zykel zu maximieren.

2.2.3. Fallbeispiel: PatternBoost an Tadaos Eiskunstlaufproblem

Wir haben bereits gesehen, dass es in Tadaos Eiskunstlaufproblem in jeder Dimension $N > 1$ immer möglich ist, eine Lösung zu finden. Die Anzahl an Zügen hat dadurch eine obere Schranke mit einer expliziten Formel erhalten, siehe Kapitel 1.2.4. Die untere Schranke $N \cdot 2^N$ scheint zu optimistisch zu sein. Zumindest im Zweidimensionalen liegt sie drei Züge unter der kürzestmöglichen Lösung, die sich mit der oberen Schranke deckt. Wählen wir eine Partitionierung von $\sum(p_i) = N$ (bzw. $= N + 1$ im ungeraden Fall), bei der alle $p_i = 2$ sind, ist dann der Lösungsalgorithmus aus Kapitel 1.2.3 optimal, d.h. $\chi_{\text{Algor.}}(N) = \chi_{\min}(N)$?

Da wir den N-dimensionalen Raum mehrfach hintereinander in Scheiben schneiden und dann nur die Nachbarschaftseigenschaften der flachen Dimension verwenden, liegt die Vermutung nah, dass eine komplexere Abfolge im N-dimensionalen Raum möglich ist und zu noch kürzeren Lösungen

führen kann. Es stellt sich als Herausforderung heraus, allein im 3-dimensionalen eine Beispielzugfolge zu finden, die kürzer ist als die $\chi_{\text{Algor.}}(3) = 38$ Züge des Algorithmus.

Wir verwenden daher PatternBoost, um kürzere Lösungsketten zu finden.

1. $\mathcal{A} := Z_N^n$, als eine beliebige n-stellige Kette an Zügen auf einem N-dimensionalen Spielfeld für ein festgelegtes $n \in \mathbb{N}$.
2. Natürlich wollen wir nur Züge zulassen, die Sinn ergeben. Wir reduzieren also zu
$$\mathcal{B} := \left\{ (z_i) \in Z_N^n | T_{\text{Start}}^{(N)} \cdot \prod_{i=1}^{n} z_i \text{ ist wohldefiniert} \right\}$$
3. Es ist schwer, eine wohldefinierte Folge (z_i) zu konstruieren, da es viele Sackgassen gibt, in denen sich ein naiver Algorithmus bei den vielen Möglichkeiten schnell verfangen kann. Wir können daher keinen sauberen Korrekturalgorithmus entwickeln und müssen davon ausgehen, dass wir fehlerhafte Folgen bewerten müssen. Wir schätzen daher die korrekten Züge einer Folge (z_i) bis zum ersten Fehler wert mit
$$m((z_i)) := \max \left\{ k \in \mathbb{N}_0 | T_{\text{Start}}^{(N)} \prod_{i=1}^{k} z_i \text{ ist wohldefiniert} \right\}$$
In der Endposition $T_{\text{Ziel}}^{(N)}$ gilt für alle Vektoren $(x_i) \in T_{\text{Ziel}}^{(N)}$ und für alle $i \in N : x_i \in \{1, 4\}$. Naiv können wir daher die Entfernung zur Endstellung anhand der Anzahl an Vektorkoordinaten bemessen, die bereits eine der Endpositionen 1 oder 4 einnehmen. Es gibt 2^N viele Vektoren und jeder dieser hat N Koordinaten, d.h. mit der Bestimmungsfunktion, die die Anzahl der korrekt positionierten Koordinaten ermittelt
$$d_M(T) := |\{(x, t) | x := (x_i) \in T, v_t \in \{1, 4\}\}|$$
gilt $d_M\left(T_{\text{Ziel}}^{(N)}\right) = 2^N \cdot N$.

   Die Bewertungsfunktion sieht nun wie folgt aus:
$$\begin{aligned} f : \mathcal{A} &\to [0, 1]_{\mathbb{R}} \\ z &\mapsto \frac{d_M\left(T_{\text{Start}}^{(N)} \prod z\right)}{2^N \cdot N} \cdot \frac{m(z)}{n} \end{aligned}$$
4. Wir benutzen den Reparaturalgorithmus aus 7. um eine Menge $\mathcal{C} \subset \mathcal{A}$ von zufälligen Pfaden ausgehend des Startspielfelds zu generieren.
5. Diese können, obwohl $\mathcal{C} \subset \mathcal{A}$ ist, bewertet werden, da wir die Bewertungsfunktion extra dafür angelegt haben. Wir legen $p = 0.1$ fest und trainieren auf den besten Pfaden.
6. Der Transformer generiert Zug um Zug neue Pfade $\mathcal{D} \in \mathcal{A}$. Die Züge sind über ihre Start-Ziel-Koordinatenvektoren tokenisiert, d.h. ein Zug $z = ((x_i), (y_i))$ bei $N = 3$, z.B. $z = ((1, 3, 2), (1, 4, 2))$ oder kurz als String „132,142“ ist ein einzelner Token. Da immer genau in einer Achse eine Position geändert werden darf, gibt es also genau $3N \cdot 4^N$ Token. Das vermeidet, dass das neuronale Netz die Orthogonalitätsregel, also dass ein Stein immer nur innerhalb einer Achse verschoben werden darf, zusätzlich lernen muss. Das bleibt für kleine N auch noch sehr übersichtlich, siehe Tabelle rechts.

N	$3N \cdot 4^N$
2	96
3	576
4	3072
5	15360

7. Wie bereits angekündigt ist der Reparaturalgorithmus $R : \mathcal{A} \to \mathcal{A}$, also nicht vollständig. Der Algorithmus funktioniert wie folgt:

   Ein generierter Pfad $(z_i) \in \mathcal{D}$ wird bis zu dem Zug $m((z_i))$ belassen, bis zu dem alle Züge wohldefiniert sind. Statt dem nächsten, falschen Zug, werden nun alle möglichen vom Spielfeld ausgehenden Züge ermittelt, die möglich sind, und dann ein zufälliger Zug davon durchgeführt. Der Algorithmus wird fortgesetzt, bis die benötigte Anzahl n an Zügen erreicht ist oder ein Spielfeld keine möglichen Züge hat. In diesem Fall werden die restlichen Züge einfach mit zufälligen Tokens aufgefüllt.

   Die resultierende Menge $\mathcal{C}' := R(\mathcal{D})$ wird in Schritt 5. als $\mathcal{C}$ weiterverarbeitet.

Nach Schritt 5. in jeder Iteration können wir uns die besten Pfade und deren Performance ansehen.

Da wir n fixiert haben, erhalten wir immer Pfade einer festen Länge. Das wirkt zunächst irritierend, schließlich war das Ziel erfolgreiche Pfade zu finden, die kürzer sind als die bis dato gültige Oberschranke.

PatternBoost ist von selbst allerdings nicht in der Lage, auch nach vielen Iterationen, $f(x) = 1$ für ein $x \in \mathcal{B}$ zu erreichen, egal für welches n, zumindest im Fall $N > 2$.

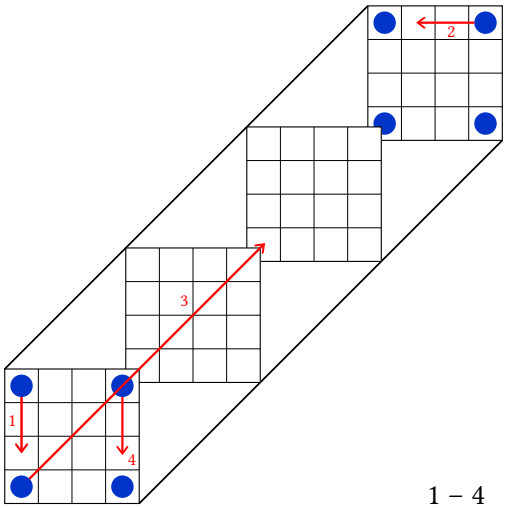
Allerdings erzeugte PatternBoost häufig fast vollständige Pfade, also solche, bei denen nur noch wenige Schritte zum Ziel fehlten.

Es ist mir gelungen, einen Pfad unter den generierten zu entdecken, den ich per Hand vervollständigen konnte. Dieser enthielt außerdem mehrere Schleifen, die nach Entfernung zu einer Verkürzung der Zugzahl führten.

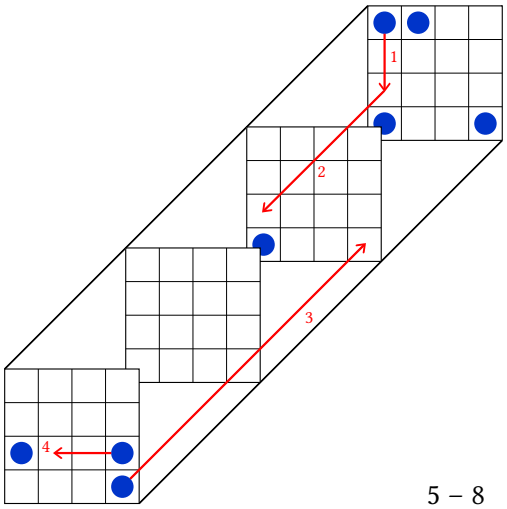
Letztendlich – with a little help of AI – ist eine 28-Zug-Lösung für $N = 3$ entstanden, was sehr performant zwischen den Ober- und Unterschranken wirkt: $24 < 28 < 38$.

Damit haben wir also nachgewiesen, dass der Lösungsalgorithmus aus Kapitel 1.2.3 nicht optimal ist. Ob die 28-Zug-Lösung eine kürzeste Lösung ist, ist zum Zeitpunkt der Einreichung der Arbeit dem Autoren unbekannt.

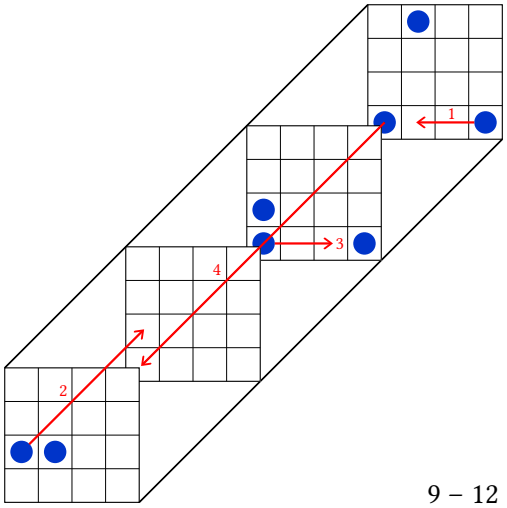
Visualisierung der 28-Zug-Lösung in umgekehrter Zugfolge, d.h. vom Zielspielfeld ausgehend:



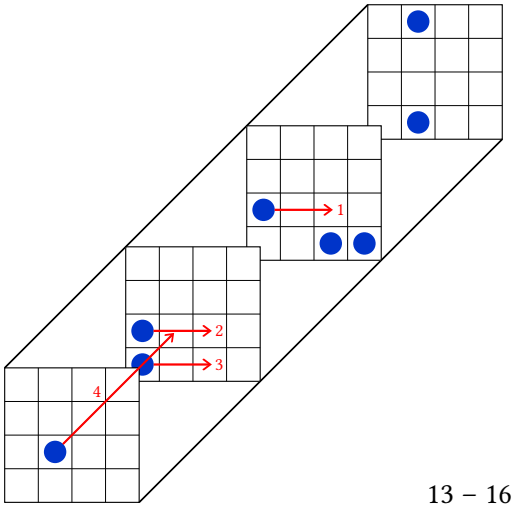
1 – 4



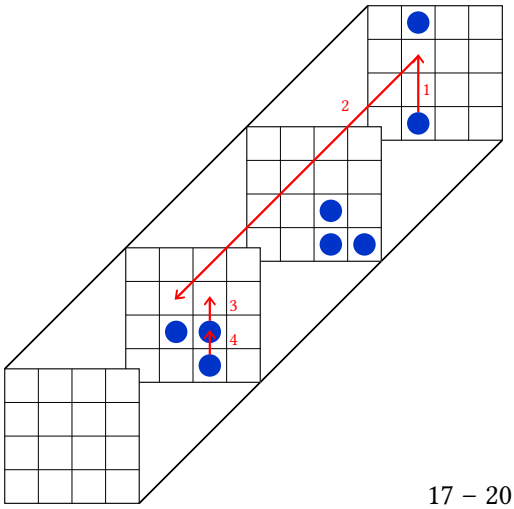
5 – 8



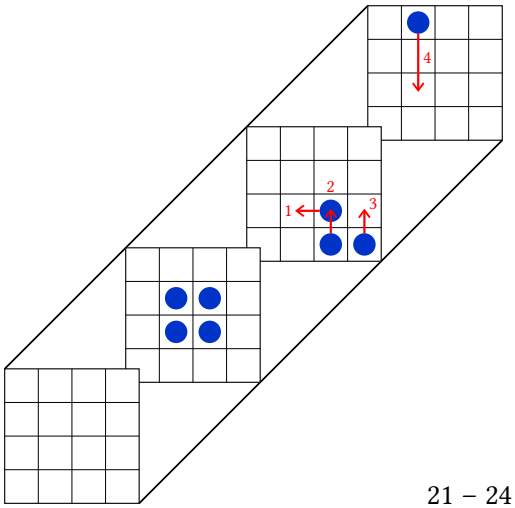
9 – 12



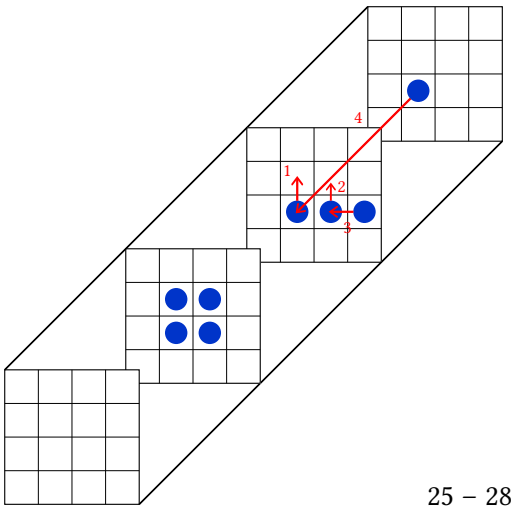
13 – 16



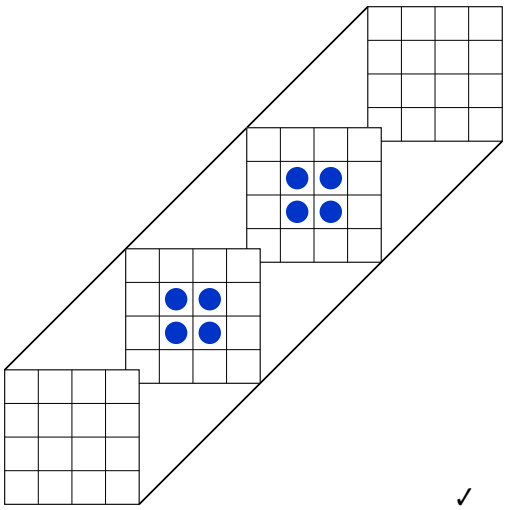
17 – 20



21 – 24



25 – 28



✓

2.3. Verifikation mit formalen Beweissystemen

Bisher wurden die KI-Implementierungen genutzt, um konkrete Beispiele zu generieren. Das Konzept ist gut, jedoch auf einen sehr speziellen Anwendungsfall limitiert. Wir wollen daher untersuchen, ob es auch möglich ist, den Ansatz so anzupassen, dass allgemeinere mathematische Aussagen generiert werden können. Wenn wir mit PatternBoost eine Pfadsuche in Tadaos Eiskunstlaufproblem erleichtern können, warum dann nicht auch die Pfadsuche im azyklischen, gerichteten Graphen eines formalen Beweissystems? Sei zunächst geklärt, was das ist.

2.3.1. Formale Beweissysteme

Ein formales System $\mathcal{S}$, auch Kalkül genannt, ist ein Tupel $\mathcal{S} = (A, G, A^*_G, X, L, Th(X))$ aus 6 Komponenten: [EFT21]

1. **Alphabet** $A = \{...\}$ Menge der Zeichen, auch Buchstaben oder Bausteine genannt, die im System benutzt werden dürfen.

   Eine beliebige Zusammensetzung dieser Zeichen nennt man **Wort**, oft auch Zeichenkette, *Formel* oder Satz genannt. Einige Worte repräsentieren eine mathematische Aussage, die wir im formalen System darstellen wollen.

   Keine Zusammensetzung, also eine „leere“ Zeichenkette, wird als **leeres Wort** ε bezeichnet. Das leere Wort wird genutzt, um rekursiv die *kleenesche Hülle* A^* zu definieren, also alle Worte, die sich mit dem Alphabet A bilden lassen:

   $$\begin{aligned} &1) \quad && \varepsilon \in A^* \\ &2) \quad && w \in A^*, a \in A \Rightarrow wa \in A^* \end{aligned}$$

2. **Grammatik** $G = (A, N, \varepsilon, P)$ sorgt dafür, dass wir nur bestimmte Worte in A^* bilden können, nämlich solche, die eine Interpretation außerhalb des formalen Beweissystems haben. So lassen sich Sätze und falsche Aussagen formulieren, die sich später durch Anwendung der Logikregeln beweisen oder widerlegen lassen. Dabei ist

   - A das bisherige Alphabet;
   - N ein Hilfsalphabet, das hinzugezogen wird, um für die Grammatik verschiedene Typen zu definieren. Die Buchstaben stimmen nicht mit denen des Alphabets überein: $A \cap N = \emptyset$
   - ε das leere Wort, das durch einen Buchstaben $\varepsilon \in N$ repräsentiert wird;
   - P eine Menge an Substitutionsregeln im Relationsformat (X, Y), wobei wir eine Zeichenkette X ersetzen dürfen mit Y, genau dann wenn $(X, Y) \in P$. Dabei ist

     $$P \subseteq \left[(A \cup N)^* \setminus A^*\right] \times (A \cup N)^*$$

     wobei * die kleenesche Hülle der entsprechenden Buchstabenmengen angibt. Wir stellen Elemente der Relation als $X \curvearrowright Y :\Longleftrightarrow (X, Y) \in P$ dar, um zu verdeutlichen, dass es sich um eine Substitution handelt. Eine Zeichenkette X aus $(A \cup N)^* \setminus A^*$, also eine, die nicht ausschließlich aus fertigen Wörtern in A^* besteht, kann durch die Zeichenkette Y ersetzt werden.

     Außerdem lohnt sich die Funktionsdarstellung $p(X) = Y$ für ein $p := (X \curvearrowright Y) \in P$, da sich so zwei konkrete Substitutionen hintereinander darstellen lassen:

     $$p_1(p_2(X_2)) = Y_1 \quad \text{für} \quad p_1, p_2 \in P$$

Die durch P definierten Grammatikregeln werden im Kalkül-Kontext auch *Formationsregeln* genannt. Worte, die durch Anwendung von Grammatikregeln entstehen, heißen auch *wohlgeformte Formeln.*

3. **Wortschatz** A_G^*, auch Satzmenge genannt, ist die Menge aller wohlgeformten Formeln unter G, also alle Zeichenketten, die bei der Anwendung der Substitutionsregeln P entstehen:

$$\begin{aligned} w \in A_G^* :\iff \exists p_1, ..., p_n \in P, n \in \mathbb{N} : \quad & w = p_n(p_{n-1}(...(p_2(p_1(\varepsilon)))...)) \in A^* \\ \iff \exists (p_i) \in P^n, n \in \mathbb{N} : \quad & \varepsilon \overset{p_1}{\curvearrowright} ... \overset{p_n}{\curvearrowright} w \quad , \quad w \in A^* \end{aligned}$$

   Der Wortschatz ist also der grammatikalisch korrekte Teil der kleeneschen Hülle: $A_G^* \subseteq A^*$

   Allen Worten des Wortschatzes lassen sich auf der Bedeutungsebene einer Aussage zuordnen, die entweder wahr oder falsch ist. Kurz gesagt: Jedes grammatikalisch korrekte Satz ist entweder wahr oder falsch.

4. **Axiome** $X = \{w \in A_G^* |\ w \text{ ist Axiom}\} \subset A_G^*$ ist die Menge der Worte, die immer wahr sind. Diese lassen sich nicht innerhalb des Beweissystems beweisen. Sie bilden das Fundament aller Aussagen, die durch Anwendung der logischen Regeln auf die Axiome, als wahr geschlussfolgert werden können.

5. **Logische Schlussregeln** $L \subset \bigcup_{m \geq 1} [(A \cup \text{Ind}(N))^*]^{m+1}$, auch Transformations-, Ableitungs- oder Deduktionsregeln genannt, sind mehrstellige Relationen, mit

$$\text{Ind}(N) := \{v_i \mid v \in N, i \in \mathbb{N}\}$$

   als Indexierungsmenge vom Hilfsalphabet N, um verschiedene Teilworte selben Typs voneinander zu unterscheiden.

   Beispiel: Ist $N = \{X, Y\}$, so ist $\text{Ind}(N) = \{X_1, X_2, ..., Y_1, Y_2, ...\}$.

   Für eine konkrete Regel $l = (X_1, ..., X_n) \in L, n \in \mathbb{N}_{>1}$ gilt, dass, wenn $X_1, ..., X_{n-1}$ wahr sind, dann ist es auch X_n. Auf der Bedeutungsebene würde man schreiben $X_1, ..., X_{n-1} \Rightarrow X_n$. Formal verwendet man dafür den syntaktischen Ableitungsoperator $\vdash$.

$$\exists l \in L : l = (X_1, ..., X_n) \iff : \{X_1, ..., X_{n-1}\} \vdash X_n$$

   Anmerkung: Wir können mit dem logischen „und"-Symbol $\wedge \in A$ und der Logikregel $(X, Y, X \wedge Y) \in L$ mehrstellige Relationen mit $n > 3$ vermeiden.

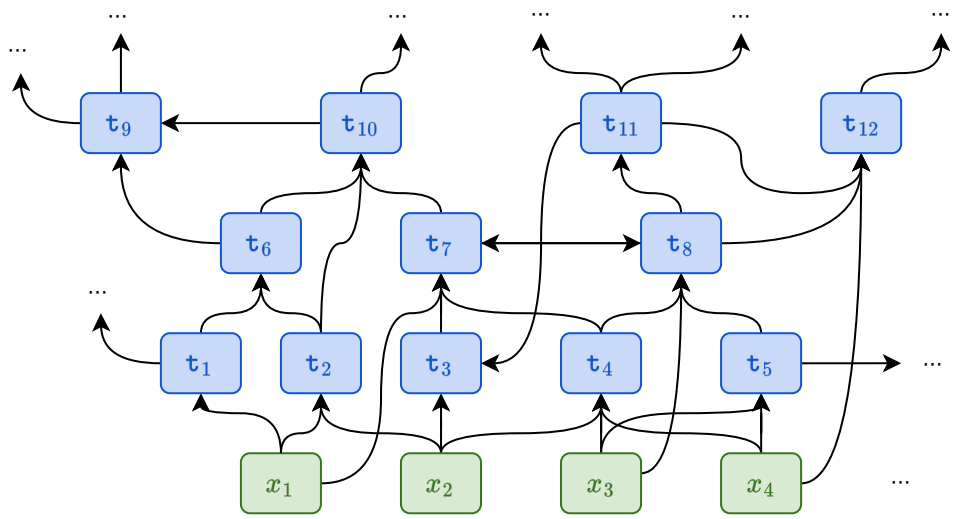
   Mit den logischen Schlussregeln lassen sich nun aus den Axiomen neue, wahre Sätze folgern. Daraus entsteht

6. **Theoreme** $Th(X)$, die Menge aller Sätze, die sich aus den Axiomen in X beweisen lassen.

   Ein *beweisbarer Satz*, auch *Theorem* genannt, ist rekursiv definiert:

$$\begin{aligned} &1) \quad t \in Th(X) :\iff t \in X \\ &2) \quad t \in Th(X) :\iff \exists l \in L, \mathtt{t}_1, ..., \mathtt{t}_n \in Th(X), n \in \mathbb{N} : l = (\mathtt{t}_1, ..., \mathtt{t}_n, t) \end{aligned}$$

Mit all diesen Definitionen zusammen können wir einen azyklischen, gerichteten Graphen (DAG) vom Kalkül $\mathcal{S}$ erstellen. Das Bild kennen wir bereits aus der Einleitung:



x_i in grün sind die Axiome, $\mathtt{t}_i$ alle anderen Theoreme. Eine Logikregel ist durch Pfeile, die an der selben Stelle zusammenführen, gegeben, z.B. $(x_2, x_3, x_4, \mathtt{t}_4)$

Es ist klar, dass mit großem Alphabet, Menge an Axiomen und logischen Regeln der Graph sehr schnell sehr groß und komplex werden kann. Wollen wir also einen konkreten Satz in $\mathcal{S}$ beweisen, können wir keine naiven Suchalgorithmen wie z.B. Brute-Force anwenden.

Im Folgenden wollen wir uns ansehen, wie in einem konkreten formalen System ein Wechselspiel aus cleveren Algorithmen und Machine Learning das Pfadfinden erleichtert.

2.3.2. AlphaGeometry

AlphaGeometry [Tri+24] von Google Deepmind benutzt ein selbstentwickeltes formales Beweissystem zur Repräsentation geometrischer Probleme, um geometrische Sätze zu beweisen. Da Teile des Algorithmus Open Source sind, können wir einige Teilschritte besser verstehen als bei den Nachfolgern Alpha Geometry 2 und den noch allgemeineren Systemen AlphaProof und AlphaEvolve [Nov+25]. Es gibt allerdings auch alternative, komplett Open Source Modelle, die ähnlich funktionieren, wie z.B. der Goedel-Prover [Lin+25].

1. Betrachte eine Struktur $\mathcal{A} := \left(A \cup \left\{\fatsemi\right\}\right)^*$ zu einem Kalkül $\mathcal{S} = (A, G, A_G^*, X, L, Th(X))$ und einem Separierungssymbol $\fatsemi \notin A$. Das * markiert die kleenesche Hülle. Offenbar lassen sich Elemente in dieser willkürlichen Struktur einfach generieren.

2. Von Interesse ist aber nur $\mathcal{B} \subset \mathcal{A}$, nämlich alle Elemente, die eine Repräsentation eines Beweises sind:

$$\mathcal{B} := \Bigg\{ K_1 \fatsemi \ldots \fatsemi K_n \mid K_i \in Th(X)\ \forall i \in [n], n \in \mathbb{N},$$

$$\forall K_i \text{ für } i \in [n] \setminus \{1\} : \exists \text{Ⱪ} \subset \left\{K_j\right\}_{j \in \begin{cases} [n]\,, & i=2 \\ [i-1]\,, & \text{sonst} \end{cases}} \setminus \{K_2\}, l \in L : l(\text{Ⱪ}) = K_i \Bigg\}$$

   Also alle Ketten $K_1 \fatsemi K_2 \fatsemi K_3 \fatsemi \ldots \fatsemi K_n$, bei der es jeweils logische Regeln gibt in L mit denen aus $K_1 \Rightarrow K_3$ folgt, aus $K_1, K_3 \Rightarrow K_4$, aus $K_1, K_3, K_4 \Rightarrow K_5, \ldots,$ aus $K_1, K_3, K_4, \ldots, K_{n-1} \Rightarrow K_n$, und aus $K_1, K_3, K_4, \ldots, K_{n-1}, K_n \Rightarrow K_2$.

   Die ungewöhnliche Aussparung von K_2 begründet sich damit, dass im Herzen von AlphaGeometry ein LLM steckt, das Zeichenketten vervollständigt. K_1 sind demnach die

Prämissen, K_2 das Beweisziel. Das neuronale Netz muss dann die Zeichenkette $K_1 \lessdot K_2$ komplettieren – im Idealfall mit dem korrekten Beweis von K_2 aus den Prämissen K_1.

3. Ein syntaktischer Generator produziert nun eine sehr große Menge $H \in A_G^*$, also grammatikalisch korrekte Sätze, die temporär als wahr angenommen werden. Aus diesen Hypothesen lassen sich dann Sätze schlussfolgern.

4. Ein spezialisierter Deduktionsalgorithmus (DD+AR) wendet häufige Logikregeln und geometriespezifische Lösungsstrategien auf die einzelnen Annahmen $h \in H$ an und exploriert den DAG des formalen Systems von h ausgehend. Der Algorithmus terminiert, nachdem keine weiteren Schlussfolgerungen getroffen werden können, oder ist zeitlich limitiert. Alle Schlussfolgerungen sind Theoreme in $Th(X)$ unter der Prämisse h. Ihr Beweis kann im Format wie in $\mathcal{B}$ abgespeichert werden. Wenn $\mathcal{P}(\cdot)$ die Potenzmengenabbildung ist, dann ist

   $$\mathrm{ddar} : A_G^* \longrightarrow \mathcal{P}(\mathcal{B})$$

   der Deduktionsalgorithmus. Für eine Hypothese $h \in A_G^*$ nennen wir $\mathrm{ddar}(h) \subset \mathcal{B}$ die Deduktionshülle von h (engl. *deduction closure*). Die Vereinigung aller Deduktionshüllen aller Elemente von H ist eine sehr große Datenmenge an Zeichenketten, die alle Beweise im formalen System darstellen.

   $$\bigcup_{B \in \mathrm{ddar}(H)} B =: \mathrm{ddar}(H)^{\cup} \subset \mathcal{B}$$

5. Mithilfe $\mathrm{ddar}(H)^{\cup}$ wird nun der Transformer $f : \mathcal{A} \to \mathcal{A}$ trainiert, die Zeichenketten zu vervollständigen. Der Artikel von Trinh u. a. nennt einige Details zur Implementierung von f, der Algorithmus ist allerdings nicht Open Source [Tri+24].

6. Offensichtlich können bei Schritt 5. Zeichenketten entstehen, die sich nicht an das Format von $\mathcal{B}$ halten oder die einen falschen Beweis darstellen. Wir benötigen also wieder einen Korrekturalgorithmus $R : \mathcal{A} \to \mathcal{A}$, der wie bei Tadaos Eiskunstlaufproblem die längstmögliche valide Kette zulässt. Betrachte das Element $f(a) = K_1 \lessdot \ldots \lessdot K_n$ mit $a = K_1 \lessdot K_2$. Sind die ersten Schritte bis m richtig, d.h. $K_1 \lessdot K_m \lessdot K_3 \lessdot K_4 \lessdot \ldots \lessdot K_{m-1} \in \mathcal{B}$ und K_m kommt nicht in der Vereinigung der Deduktionshüllen von H vor, so können wir mit $\mathrm{ddar}(K_m)$ eine neue Deduktionshülle finden und diese nutzen, um f zu trainieren.

7. Dadurch entsteht ein Zyklus, der nicht nur immer weitere wahre Sätze findet und diese zum Trainieren nutzt, sondern auch ein Tool, mit dem ein konkreter Satz ausgehend von einer gegebenen Prämisse bewiesen werden kann.

   Sei $R_{\_} : \mathcal{A} \to A_G^*$ die Ausgabe von K_m, dem letzten richtigen Beweisschritt einer Zeichenkette $f(a)$.

   Sei K_h eine Prämisse und K_{Ziel} das Beweisziel, beides grammatikalisch korrekte Schreibweisen: $K_h, K_{\text{Ziel}} \in A_G^*$.

   Führe zunächst $\mathrm{ddar}(K_h)$ aus. Liegt $\left(K_h \lessdot K_{\text{Ziel}} \lessdot \ldots\right) \in \mathrm{ddar}(K_h)$, haben wir einen Beweis. Wenn nicht, können wir für alle $\left(K_h \lessdot K_2 \lessdot K_3 \lessdot \ldots \lessdot K_n\right) \in \mathrm{ddar}(K_h)$ $R_{\_}\Big(f\Big(K_h \lessdot K_{\text{Ziel}} \lessdot K_3 \lessdot \ldots \lessdot K_n \lessdot K_2\Big)\Big) =: K'$ bestimmen. Gilt $K' = K_{\text{Ziel}}$ haben wir einen Beweis. Sonst wiederhole die Schritte von $\mathrm{ddar}(K')$ ausgehend.

Mit 7. können wir beliebige Probleme lösen lassen, bei denen die Annahmen und das Ziel klar sind und sich das Problem mit dem spezialisierten formalen Beweissystem verschriftlichen lässt. AlphaGeometry schafft beeindruckende 25 von 30 der geometrischen IMO-Probleme zu lösen; die Übersetzung der Rätsel in die formale Sprache erfolgte allerdings manuell.

3. Limitationen, Diskussion, Ausblick

In diesem letzten Kapitel seien einige der Aspekte adressiert, die im Zuge dieser Bachelorarbeit noch genannt werden sollten. Es geht um Limitationen der aktuellen Ansätze, um die Chancen anderer Ideen und über die Konsequenzen, des aktuellen Einsatzes von KI-Modellen.

Ein erster Aspekt ist bereits im letzten Kapitel adressiert worden: Während wir uns im formalen System darauf verlassen können, dass alle Umstellungen fehlerfrei durchgeführt werden – zumindest unter der Annahme, dass das formale System selbst fehlerfrei designt worden ist – bekommen die formalen Aussagen erst eine Bedeutung durch ihre Zuordnung in die natürliche Sprache. Bei dieser Übertragung können allerdings Fehler passieren [Jia+23, Wu+22], d.h. Problemstellungen in natürlicher Sprache werden falsch in das Beweissystem übersetzt und der Output wird inkorrekt interpretiert. Wenn also ein LLM ein Problem bearbeitet und dann eine Lösung ausgibt, zusammen mit einem Zertifikat eines Beweissystems, das die Korrektheit der Lösung bestätigt (so stellt es sich Alex Kontorovich in seinem 4. „Heiligen Gral der KI-Assistenz“ vor [Kon23]), welche Garantie wäre gegeben, dass der informelle Teil der Antwort mit dem formellen Part übereinstimmt?

Selbst wenn dem Übersetzungsprozess vertraut werden könnte, gibt es inhärente Limitationen der formalen Systeme selbst. Da jeder Beweis essentiell eine Pfadsuche durch eines dieser Beweissysteme ist, stellt sich die Frage, welches System am geeignetsten ist – ähnlich wie die Wegsuche abhängig vom Labyrinth selbst ist. Metamath beispielsweise überzeugt mit der kleinstmöglichen Menge an Logikregeln L ($|L| = 1$) und garantiert dadurch ein hohes Maß an Vertrauen über die Richtigkeit der Umstellungen der Axiom- und Annahmesätze [MW19]. Allerdings ist diese Art, logisch zu argumentieren, sehr weit entfernt von der Art, wie Menschen Argumente informell aufbauen, und lässt sich deshalb schlecht lesen. Beweise werden durch diese Limitation unnötig lang und kompliziert – der de-Bruijn-Faktor, also die relative Länge der formalen Beweise zu ihren informellen Äquivalenten, ist 10 bis 20 Mal höher als bei Lean, Coq oder HOL Light [PS20]. Es ist möglich, dass aber auch diese zu ineffizient sind, um schwierige Probleme anzugehen.

[She+25] argumentiert daher, dass Abkürzungen (engl. *supermoves*) gefunden werden müssen, um den Suchraum des DAGs des formalen Beweissystems schneller zu durchschreiten. Das erinnert daran, dass die menschliche Intuition, die abseits formaler Systeme mit Sprache und Bildern agiert, häufig in der Lage ist, Zusammenhänge bereits ohne Beweis zu erkennen oder zumindest zu vermuten. So ist der Mathematiker Srinivasa Ramanujan dafür bekannt, aufgrund fehlender Schulausbildung, seine eigene mathematische Schreibweise entwickelt zu haben, fernab formaler Beweissysteme. Intuition ist also auch eine Art Abkürzung durch den Suchraum, zumindest wenn sie richtig liegt.

All das zeigt, dass die Wahl des Beweissystems entscheidend sein kann für den Erfolg, neues mathematisches Wissen zu entdecken und zu verifizieren. Es zeigt, dass bisherige formale Systeme womöglich noch nicht ausreichen, um besonders schwierige Probleme zu beweisen oder zu entdecken. In der Tat ist kein Beweissystem perfekt: So zeigte Kurt Gödel mit den Unvollständigkeitssätzen, dass es kein formales Beweissystem (mit grundlegender Arithmetik, siehe [Hof17] für Details) geben kann, das vollständig ist. Die Konsequenz ist, dass es Sätze in A_G^* gibt, die sich nicht mit den logischen Regeln L aus den Axiomen X beweisen lassen. Es müssen also entweder neue Axiome oder logische Regeln entwickelt werden. Überdies könnte es auch Aussagen geben, die wahr sind, sich aber nicht einmal in $\mathcal{S}$ formulieren lassen.

Perspektivisch reicht es also nicht, KIs auf einen formalen Output zu beschränken. Es sei an das Bild des Buchs mit allem Wissen des Universums aus der Einleitung erinnert. Die natürliche Sprache ist

die bisher beste Approximation dafür. Sie ist flexibel genug, um in kleinen Schritten logisch zu argumentieren und in groben Bildern komplexe Zusammenhänge aufzuzeigen. Trotz ihrer Ungenauigkeiten und möglichen Fallstricke muss die natürliche Sprache benutzt werden, um die bisherigen Tools weiterzuentwickeln. Nach der Devise „rationales Denken ist, in immer kleineren Schritten zu argumentieren“ entwickelte das Team von OpenAI ein Modell, das Beweise direkt im LLM erzeugt und dabei in kleinen Schritten vorgeht [Lig+23]. Jeder Schritt wird dann von einem anderen LLM auf seine Richtigkeit gelabelt. In einem anderen vollständig LLM-basierten Modell, *The AI Scientist*, werden in vielen kleinen Teilschritten ganze, vermeintlich wissenschaftliche Arbeiten geschrieben [Lu+24].

Darüber hinaus gibt es den Ansatz, noch mehr der Arbeitsweisen von Menschen zu imitieren und adaptieren. So beziehen sich die Modelle in Kapitel 1.1 ausschließlich auf die Herangehensweise bei gegebenen Problemstellungen. Dabei exkludieren sie teilweise die Interaktion mit anderen Menschen, das Recherchieren im Internet oder das Erstellen von Bildungsmaterial. Einige Studien befassen sich genau damit, diese vielfältigen Aspekte im Leben von Mathematiker:innen zu simulieren und zu untersuchen [TN23, Zha+24].

Die maximal generalisierte Form eines solchen KI-Modells könnte ununterbrochen laufen, würde stets eine Strategie auswählen, diese ausführen und dann mit dem gewonnenen Output die nächste Strategie wählen. Selbst eine neue Strategie zu erfinden, wäre eine wählbare Strategie. Die Wahl der Strategie wird durch ein anderes NN bestimmt als das LLM, das die Strategie ausführt:

$$\mathfrak{S} \times \mathfrak{s} \xrightarrow{\text{LLM}} \mathfrak{S} \xrightarrow{\text{NN}} \mathfrak{s} \subset \mathfrak{S}$$

für $\mathfrak{S}$ die Menge aller möglichen Zeichenketten, die durch das LLM erzeugt werden können, und $\mathfrak{s}$ die Menge der Strategien, ebenfalls als Zeichenketten formalisiert. NN wählt also basierend auf $x \in \mathfrak{S}$ eine Strategie $s \in \mathfrak{s}$ und generiert dann wieder ausgehend von (x, s) einen Output x'.

Übrig bleibt die Frage, wie die Bewertungsfunktion von NN aussieht und welches x_0 initial in das System gesteckt wird. Beide Fragen beziehen sich darauf, wie festgelegt wird, was als relevant gilt. Filatova, Volkovskii und Begen analysieren dafür den wissenschaftlichen Diskurs im Internet unter anderem mit dem Einsatz von neuronalen Netzen [FVB20].

Der Einsatz von Sprachmodellen, die auf menschengemachten Daten erzeugt werden, um die vielen Facetten der menschlichen Arbeitsweise zu imitieren, birgt allerdings auch Gefahren. Einem Report von Anthropic zufolge versuchten sich die meisten der aktuellen LLMs in einem konstruierten Sandbox-Environment lieber selbst statt Menschen in Lebensgefahr zu retten und begründeten dies in vielen kleinen Gedankenschritten [Age25].

Das ist nur eines von vielen Problemen der aktuellen Ansätze. Bisherige KI-Modelle benötigen sehr große Datenmengen für den Trainingsprozess und können daher fast exklusiv nur von Großunternehmen entwickelt werden. Aspekte wie Diskriminierung, Bias, Misinformation bekommen eine dem ökonomischen Interesse untergeordnete Priorisierung [Lig+23]. Außerdem wird bei den aufwendigen Trainingsprozessen viel Strom verbraucht, was ökologische und damit einhergehende soziale Konsequenzen mit sich bringt.

Es gibt bereits Ideen, die bisherigen leistungsstarken, aber problemanfälligen neuronalen Netze durch neuromorphe zu ersetzen, also solche, die die tatsächliche Funktionsweise des Gehirns näher nachahmen. Diese Technologie verbraucht voraussichtlich weit weniger Strom und kann mit weniger Daten effizient lernen, ist allerdings noch in der Entwicklung [Muc24].

Es ist zu erwarten, dass die Diskussion um die positiven und negativen Auswirkungen von KIs in den kommenden Jahren nicht abebbt und weitere Entwicklungen folgend werden.

Literatur

[12 25] „12 Days of OpenAI“. Zugegriffen: 8. September 2025. [Online]. Verfügbar unter: https://openai.com/12-days/

[Liu+25] Y. Liu, Y. Huang, Y. Wang, P. Li, und Y. Liu, „AI Mathematician: Towards Fully Automated Frontier Mathematical Research“, Nr. 2505.22451. arXiv, Mai 2025.

[Lig+23] H. Lightman *u. a.*, „Let's Verify Step by Step“, Nr. 2305.20050. arXiv, Mai 2023.

[Ben05] J. P. V. Bendegem, „The Collatz Conjecture. A Case Study in Mathematical Problem Solving“, *Logic and Logical Philosophy*, Bd. 14, Nr. 1, S. 7–23, Juni 2005, doi: 10.12775/LLP.2005.002.

[Zei16] P. Zeitz, *The Art and Craft of Problem Solving*. John Wiley & Sons, 2016.

[Hof17] D. W. Hoffmann, *Die Gödel'schen Unvollständigkeitssätze: eine geführte Reise durch Kurt Gödels historischen Beweis*, 2. Auflage. Berlin: Springer Spektrum, 2017. doi: 10.1007/978-3-662-54300-9.

[Pol14] G. Polya, *How to Solve It: A New Aspect of Mathematical Method*, Expanded Princeton Science Library edition. in Princeton Science Library. Princeton, NJ: Princeton University Press, 2014. doi: 10.1515/9781400828678.

[RSK21] B. Rott, B. Specht, und C. Knipping, „A Descriptive Phase Model of Problem-Solving Processes“, *ZDM – Mathematics Education*, Bd. 53, Nr. 4, S. 737–752, Aug. 2021, doi: 10.1007/s11858-021-01244-3.

[Kit21] T. Kitazawa, *Arithmetical, Geometrical and Combinatorial Puzzles from Japan*. American Mathematical Soc., 2021.

[Wag21] A. Z. Wagner, „Constructions in Combinatorics via Neural Networks“, Nr. 2104.14516. arXiv, April 2021.

[Edm65] J. Edmonds, „Paths, Trees, and Flowers“, *Canadian Journal of Mathematics*, Bd. 17, S. 449–467, Jan. 1965, doi: 10.4153/CJM-1965-045-4.

[Ste10] D. Stevanović, „Resolution of AutoGraphiX Conjectures Relating the Index and Matching Number of Graphs“, *Linear Algebra and its Applications*, Bd. 433, Nr. 8–10, S. 1674–1677, Dez. 2010, doi: 10.1016/j.laa.2010.06.015.

[Cha+24] F. Charton, J. S. Ellenberg, A. Z. Wagner, und G. Williamson, „PatternBoost: Constructions in Mathematics with a Little Help from AI“, Nr. 2411.00566. arXiv, November 2024.

[EFT21] H.-D. Ebbinghaus, J. Flum, und W. Thomas, *Mathematical Logic*, Bd. 291. in Graduate Texts in Mathematics, vol. 291. Cham: Springer International Publishing, 2021. doi: 10.1007/978-3-030-73839-6.

[Tri+24] T. H. Trinh, Y. Wu, Q. V. Le, H. He, und T. Luong, „Solving Olympiad Geometry without Human Demonstrations“, *Nature*, Bd. 625, Nr. 7995, S. 476–482, Jan. 2024, doi: 10.1038/s41586-023-06747-5.

[Nov+25] A. Novikov *u. a.*, „AlphaEvolve: A Coding Agent for Scientific and Algorithmic Discovery“, Nr. 2506.13131. arXiv, Juni 2025.

[Lin+25] Y. Lin *u. a.*, „Goedel-Prover: A Frontier Model for Open-Source Automated Theorem Proving“, Nr. 2502.07640. arXiv, April 2025.

[Wu+22] Y. Wu *u. a.*, „Autoformalization with Large Language Models“, *Advances in Neural Information Processing Systems*, Bd. 35, S. 32353–32368, Dez. 2022.

[Jia+23] A. Q. Jiang *u. a.*, „Draft, Sketch, and Prove: Guiding Formal Theorem Provers with Informal Proofs“, Nr. 2210.12283. arXiv, Februar 2023.

[Kon23] A. Kontorovich, „Notes on a Path to AI Assistance in Mathematical Reasoning“, Nr. 2310.02896. arXiv, Oktober 2023.

[MW19] N. Megill und D. A. Wheeler, *Metamath: A Computer Language for Mathematical Proofs*. Lulu.com, 2019.

[PS20] S. Polu und I. Sutskever, „Generative Language Modeling for Automated Theorem Proving“, Nr. 2009.03393. arXiv, September 2020.

[She+25] A. Shehper *u. a.*, „What Makes Math Problems Hard for Reinforcement Learning: A Case Study“, Nr. 2408.15332. arXiv, Februar 2025.

[Lu+24] C. Lu, C. Lu, R. T. Lange, J. Foerster, J. Clune, und D. Ha, „The AI Scientist: Towards Fully Automated Open-Ended Scientific Discovery“, Nr. 2408.06292. arXiv, September 2024.

[Zha+24] H. Zhang *u. a.*, „Building Cooperative Embodied Agents Modularly with Large Language Models“, Nr. 2307.02485. arXiv, Februar 2024.

[TN23] Y. Talebirad und A. Nadiri, „Multi-Agent Collaboration: Harnessing the Power of Intelligent LLM Agents“, Nr. 2306.03314. arXiv, Juni 2023.

[FVB20] O. G. Filatova, D. V. Volkovskii, und P. N. Begen, „The Methodology of Discourse Analysis: From Manuel Data Counting to Machine Learning“, in *22nd Scientific Conference "Scientific Services & Internet"*, 2020, S. 612–621. doi: 10.20948/abrau-2020-28.

[Age25] „Agentic Misalignment: How LLMs Could Be Insider Threats“. Zugegriffen: 9. September 2025. [Online]. Verfügbar unter: https://www.anthropic.com/research/agentic-misalignment

[Muc24] T. Mucci, „The Future of Artificial Intelligence“. Zugegriffen: 10. September 2025. [Online]. Verfügbar unter: https://www.ibm.com/think/insights/artificial-intelligence-future