HAW HAMBURG

Masterarbeit

Bjarne Martensen

Entwicklung eines adaptiven Optimierungsverfahrens auf Basis von Large Language-Modellen und evolutionären Algorithmen für Timetabling-Aufgaben.

Betreuung durch: Prof. Dr. Thomas Clemen / Prof. Dr. Christian Lins
Eingereicht am: 18. November 2025

Fakultät Technik und Informatik
Department Informatik

Faculty of Engineering and Computer Science
Department Computer Science

Kurzzusammenfassung

Diese Masterarbeit befasst sich mit der Entwicklung und Untersuchung eines hybriden Optimierungsverfahrens, das Large Language Models mit evolutionären Algorithmen kombiniert, um komplexe Timetabling-Probleme adaptiv zu lösen. Auf Basis eines einheitlichen Daten- und Bewertungsrahmens wurden vier unterschiedliche Implementierungen konzipiert. Diese sind reines Prompting, Evolution of Schedules, Evolution of Heuristics und ein evolutionärer Algorithmus mit LLM-generierten Operatoren. Die Leistungsfähigkeit wurde in drei realitätsnahen Szenarien: Schul-, Zug- und Maschinenbelegungsplanung evaluiert. Die Experimente basieren auf quantitativen Kennzahlen wie Fitnesswert, Constraint-Erfüllung, Konvergenzverhalten und der Anzahl an LLM-Aufrufen. Zur statistischen Auswertung diente der Friedman-Test mit anschließender post-hoc-Analyse. Die Ergebnisse zeigen deutliche Unterschiede zwischen den Verfahren. Reine Prompting-Ansätze liefern zwar textuell plausible, aber strukturell fehlerhafte Lösungen und sind daher für komplexe Planungsaufgaben ungeeignet. Evolution of Schedules und Evolution of Heuristics erzielen durch iterative Einbindung des LLMs eine verbesserte Stabilität, zeigen jedoch erhebliche Schwächen bei der Erfüllung von Constraints und der Reproduzierbarkeit. Der evolutionäre Algorithmus mit gezielter LLM-Integration erzielt in zwei der drei Szenarien signifikant bessere Resultate und ist das einzige Verfahren, das gültige Lösungen hervorbringt, die sowohl intern als auch extern validiert werden können.

Die Arbeit verdeutlicht, dass LLMs keine eigenständigen Optimierungsverfahren ersetzen können, aber einen substantiellen Mehrwert in hybriden Strukturen bieten. Ihr Nutzen liegt insbesondere in der adaptiven Generierung und Anpassung von Heuristiken sowie in der semantischen Interpretation von Problemkontexten. Die Ergebnisse zeigen, dass eine gezielte Kombination von LLMs und evolutionären Algorithmen das Potenzial besitzt, die Generalisierbarkeit und Flexibilität von Planungs- und Scheduling-Systemen deutlich zu erhöhen.

Abstract

This master's thesis addresses the development and investigation of a hybrid optimization approach that combines Large Language Models with evolutionary algorithms to adaptively solve complex timetabling problems. Based on a unified data and evaluation framework, four different implementations were designed. These are pure prompting, Evolution of Schedules, Evolution of Heuristics, and an evolutionary algorithm with LLM-generated operators. Their performance was evaluated in three realistic scenarios: school timetabling, train scheduling, and machine allocation. The experiments are based on quantitative metrics such as fitness value, constraint satisfaction, convergence behavior, and the number of LLM calls. For the statistical evaluation, the Friedman test with a subsequent post-hoc analysis was used. The results show clear differences between the methods. Pure prompting approaches produce textually plausible but structurally flawed solutions and are therefore unsuitable for complex planning tasks. Evolution of Schedules and Evolution of Heuristics achieve improved stability through the iterative involvement of the LLM, but they show substantial weaknesses in satisfying constraints and in reproducibility. The evolutionary algorithm with targeted LLM integration achieves significantly better results in two of the three scenarios and is the only method that produces valid solutions that can be verified both internally and externally.

The study demonstrates that LLMs cannot replace standalone optimization methods, but they can provide substantial value in hybrid structures. Their benefit lies particularly in the adaptive generation and adjustment of heuristics as well as in the semantic interpretation of problem contexts. The findings indicate that a combination of LLMs and evolutionary algorithms has the potential to significantly enhance the generalizability and flexibility of planning and scheduling systems.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

1 Einleitung

1.1 Motivation

Die zunehmende Digitalisierung prägt seit Jahren zahlreiche Lebens- und Arbeitsbereiche und führt zu tiefgreifenden strukturellen Veränderungen („Global Digitalization in 10 Charts“, n. d.; Streim und Heinze, 2021). Neben den damit verbundenen Herausforderungen eröffnen sich erhebliche Chancen. Prozesse können effizienter gestaltet, Entscheidungen datenbasiert getroffen und komplexe Zusammenhänge umfassend analysiert werden (Eleusov et al., 2020; Peng et al., 2023; Todoschak und Frolova, 2023). Ein Bereich, der besonders von dieser Entwicklung profitiert, ist die Planung und Organisation von Aktivitäten. Je komplexer die Anforderungen an die Koordination von Ressourcen, Zeiten und Aufgaben werden, desto wichtiger wird eine optimierte, digital unterstützte Planung. In diesem Kontext gewinnen sogenannte *Timetabling-Probleme* an Bedeutung (Gusti Agung Premananda et al., 2024, S. 15). Dabei handelt es sich um Optimierungsprobleme, die als NP-schwer klassifiziert werden (Pillay, 2014). Dies ist eine Eigenschaft, die nahelegt, dass es kaum möglich ist, innerhalb einer akzeptablen Rechenzeit eine optimale Lösung zu finden. Insbesondere erlaubt sie nicht, dass alle möglichen Lösungen ausprobiert und die Beste verwendet werden kann (Skiena, 2020). Die hohe Komplexität dieser Probleme hat dazu geführt, dass Mathematik und Informatik seit langem intensiv an effizienten Lösungsansätzen arbeiten, um in unterschiedlichsten Anwendungsszenarien tragfähige und leistungsfähige Pläne erstellen zu können (Giffler und Thompson, 1960).

Zur Lösung des Timetabling-Problems wurden in den letzten Jahren viele verschiedene algorithmische Ansätze entworfen. Obwohl diese in der Vergangenheit bereits erhebliche Fortschritte erzielt haben und mittlerweile gute Ergebnisse liefern, bestehen weiterhin Herausforderungen beim praktischen Einsatz (Scully und Harchol-Balter, 2021). Um die Algorithmen an konkrete Anwendungsfälle anzupassen, sind oftmals umfangreiche manuelle Eingriffe erforderlich. Dabei geht es nicht nur darum, spezifische Kontexte wie Produktionsanlagen, Schichtdienste oder Schulen zu berücksichtigen. Vielmehr unterscheiden sich selbst innerhalb dieser Kategorien einzelne Betriebe oder Einrichtungen derart stark, dass allgemeine Lösungen in der Praxis häufig unzureichende Resultate liefern. Dadurch sind erhebliche manuelle Nachbesserungen notwendig, die den erhofften Effizienzgewinn deutlich verringern und das Optimierungspotenzial der eingesetzten Verfahren begrenzen (Jorge Amar et al., 2022).

Die aktuelle Forschung widmet sich daher verstärkt der Entwicklung von Methodiken und Algorithmen, die eine möglichst hohe Allgemeingültigkeit aufweisen oder zumindest flexibel auf verschiedene Anwendungskontexte übertragbar sind. In den vergangenen Jahren wurden zahlreiche neue Ansätze entwickelt, um dieser Herausforderung zu begegnen. Dazu zählen unter anderem genetische Algorithmen, Simulated Annealing, Particle Swarm Optimization sowie Ant Colony Optimization (Blum und Roli, 2003), die in einer Vielzahl von Bereichen, wie der Routenplanung in der Logistik (Osman und Kelly, 1996), der Produktionssteuerung (Ruiz und Maroto, 2005) oder der universitären Stundenplanung (Rhydian Lewis, 2007) erfolgreich eingesetzt werden. Ergänzend dazu gewinnen hyper-heuristische Verfahren zunehmend an Bedeutung. Diese zielen darauf ab, auf einer höheren Abstraktionsebene Strategien zu entwickeln, die nicht nur zwischen bestehenden Heuristiken auswählen, sondern diese auch adaptieren oder neu kombinieren können, beispielsweise unter Einsatz von Machine Learning-Technologien wie Reinforcement Learning (Özcan et al., 2008). Trotz dieser Fortschritte bleibt die Entwicklung von robusten, breit einsetzbaren Verfahren eine zentrale Herausforderung.

Seit dem Jahr 2023 haben generative Formen von Künstlicher Intelligenz (GenAI) sowohl in der breiten Öffentlichkeit als auch in der wissenschaftlichen Forschung zunehmend an Bedeutung gewonnen (Schlagwein und Willcocks, 2023). Inspiriert durch diese Entwicklungen wurden erste Ansätze verfolgt, Large Language Models (LLMs) zur Unterstützung bei der Lösung von Planungsproblemen einzusetzen (Liu et al., 2023). In initialen Testszenarien konnte dabei gezeigt werden, dass LLMs erfolgreich in bestehende Algorithmen integriert werden können und insbesondere bei einfachen Anwendungsfällen praxistaugliche Ergebnisse erzielen (Liu et al., 2024). Diese Entwicklungen eröffnen neue Perspektiven für die Gestaltung adaptiver Planungssysteme, die flexibel auf verschiedene Anforderungen reagieren können. Daher beschäftigt sich diese Arbeit damit, Möglichkeiten für den Einsatz von LLMs zur Lösung von Timetabling-Problemen zu entwickeln.

1.2 Ziel der Arbeit

Diese Arbeit verfolgt das Ziel, ein adaptives Optimierungsverfahren für Timetabling-Aufgaben zu entwickeln, das auf dem Einsatz von Large Language Models basiert. Ausgangspunkt bildet die Beobachtung, dass traditionelle Verfahren häufig Schwierigkeiten aufweisen, sich flexibel an unterschiedliche Anwendungsfälle anzupassen (Jorge Amar et al., 2022). Durch die Integration von LLMs in bestehende algorithmische Muster als

Erweiterung oder Ersetzung von Code, soll die Übersetzung von fachlichen Anforderungen in technische Lösungen erfolgen. Es soll durch den Einsatz von LLMs in mehreren Testszenarien herausgefunden werden, ob und in welchem Rahmen ein Einsatz dieser im Kontext von Planungsproblemen möglich ist. Der Schwerpunkt liegt hierbei darauf sicherzustellen, dass die konkreten Systeme in der Lage sind, sich flexibel an verschiedene Anforderungen anzupassen.

Im Rahmen der Arbeit soll zunächst auf Basis der bestehenden Forschung analysiert werden, welche Möglichkeiten es traditionell zur Lösung solcher Probleme gibt. Zudem soll betrachtet werden, welche Möglichkeiten existieren, LLMs für die Lösung komplexer Probleme einzusetzen. Aufbauend auf den Erkenntnissen werden verschiedene Konzepte entwickelt, welche LLMs hybrid einsetzen, um Lösungen für komplexe Planungsszenarien zu entwickeln.

Das übergeordnete Ziel besteht darin, ein adaptives System zu schaffen, das in der Lage ist, verschiedene Timetabling-Probleme effizient zu lösen und dabei anwendungsspezifische Anforderungen selbstständig zu berücksichtigen. Im Fokus steht die Entwicklung einer Lösung, die auch für Anwenderinnen und Anwender ohne tiefgehende technische Expertise nutzbar ist und damit einen breiten Praxiseinsatz ermöglicht.

Zur Konkretisierung lassen sich die Zielsetzungen der Arbeit wie folgt gliedern:

- Systematische Analyse bestehender Ansätze zur Integration von LLMs in Planungs- und Optimierungsaufgaben, insbesondere im Kontext von Timetabling.
- Konzeption von hybriden Optimierungsverfahren, die Large Language Models mit evolutionären Algorithmen kombinieren.
- Entwicklung der Systeme zur Lösung unterschiedlicher Timetabling-Szenarien.
- Empirische Evaluation der entwickelten Verfahren anhand ausgewählter Anwendungsfälle im Hinblick auf Effizienz, Anpassungsfähigkeit und Ergebnisqualität.

Durch die Bearbeitung dieser Teilziele soll ein Beitrag zur Erweiterung bestehender Optimierungsansätze im Bereich der Timetabling-Probleme geleistet werden. Es wird das Ziel verfolgt, einen Weg aufzuzeigen, wie generative KI und klassische Optimierungsverfahren synergetisch genutzt werden können.

1.3 Themenabgrenzung

Diese Arbeit baut auf aktuellen Veröffentlichungen auf, die erste Ansätze zur Nutzung von LLMs zur Lösung komplexer Optimierungs- und Planungsprobleme untersuchen (Liu et al., 2023, Wu et al., 2024). Diese Arbeiten kombinieren LLMs mit evolutionären oder populationsbasierten Verfahren, um Suchprozesse zu unterstützen oder heuristische Strategien dynamisch anzupassen. Zwar existieren zahlreiche etablierte Verfahren zur Lösung von Timetabling-Problemen (siehe Kapitel 2.2), der Fokus dieser Arbeit liegt jedoch bewusst auf der praktischen Umsetzung eines hybriden Ansatzes aus LLMs und evolutionären Algorithmen.

Der zentrale Grund für diese Fokussierung liegt in der generischen und modularen Struktur evolutionärer Verfahren. Evolutionäre Algorithmen sind darauf ausgelegt, dass ihre interne Logik, wie etwa die Definition von Mutations-, Crossover- oder Selektionsstrategien, leicht angepasst oder vollständig ausgetauscht werden kann (vgl. Kapitel 2.3. Sie benötigen weder Gradienteninformationen noch eine formale Beschreibung des Suchraums, sondern lediglich eine Bewertungsfunktion, die die Qualität eines Individuums bestimmt. Damit eignen sie sich besonders für die Kombination mit LLMs, deren Stärken in der semantischen Wissensrepräsentation und der textuellen Problembeschreibung liegen, die jedoch keine numerischen Gradienten oder exakten Formulierungen liefern.

Durch diese Komplementarität entsteht ein hybrides System, in dem evolutionäre Algorithmen die strukturelle Basis der Optimierung bereitstellen und LLMs adaptive, kontextabhängige Heuristiken oder Lösungsvorschläge generieren können. Die populationsbasierte Natur evolutionärer Verfahren erlaubt es zudem, mehrere LLM-generierte Varianten zu evaluieren, wodurch sowohl Diversität als auch Lernpotenzial im Suchprozess erhöht werden. Im Gegensatz zu deterministischen Verfahren oder klassischen regelbasierten Ansätzen bietet ein evolutionärer Rahmen damit eine flexible Experimentierumgebung, um die Integrationsmöglichkeiten von LLMs systematisch zu untersuchen.

1.4 Forschungsfragen

Ausgehend von der dargestellten Motivation sowie den formulierten Zielsetzungen ergeben sich für die vorliegende Arbeit zentrale Forschungsfragen, die im Verlauf systematisch untersucht werden sollen. Diese Forschungsfragen dienen dazu, die theoretischen und praktischen Aspekte der Entwicklung eines adaptiven Optimierungsverfahrens auf Basis

von Large Language Models und evolutionären Algorithmen für Timetabling-Aufgaben strukturiert zu erfassen und zielgerichtet zu bearbeiten.

Im Einzelnen werden folgende Forschungsfragen adressiert:

1. **Welche Potenziale und Grenzen bestehen bei der Anwendung von Large Language Models für die Lösung von Timetabling-Problemen?**
   Diese Frage zielt darauf ab, die Eignung von LLMs hinsichtlich der Problemerfassung, Lösungsstrukturierung und Adaptivität im spezifischen Kontext von Timetabling-Aufgaben systematisch zu analysieren.

2. **Wie können Large Language Models und evolutionäre Algorithmen synergetisch kombiniert werden, um ein adaptives Optimierungsverfahren für Timetabling-Probleme zu entwickeln?**
   Im Fokus dieser Frage steht die methodische Ausarbeitung eines hybriden Ansatzes, der die Stärken beider Technologien miteinander verbindet.

3. **Welche Anforderungen müssen an ein adaptives Optimierungssystem gestellt werden, damit es in unterschiedlichen Timetabling-Szenarien anwendbar ist?**
   Hierbei wird untersucht, wie es möglich ist, gestellte textuelle Constraints so in das System zu verankern, dass sicherzustellen ist, dass die Antwort den Anforderungen entspricht.

4. **Sind Systeme, welche LLMs verwenden, in der Lage, praktische Timetabling-Probleme zu lösen?**
   Diese Frage dient dazu, den Bezug zu realen Problemen und der Lösung dieser darzustellen, damit kein theoretisches Konstrukt geschaffen wird, welches schlussendlich keinen realen Mehrwert hat.

Durch die Beantwortung dieser Forschungsfragen soll ein fundierter Beitrag zur Weiterentwicklung intelligenter, adaptiver Optimierungsverfahren im Bereich der Timetabling-Probleme geleistet werden.

1.5 Forschungslücke

In der Forschung zu Timetabling-Problemen hat sich in den vergangenen Jahrzehnten ein breites Spektrum an Lösungsansätzen etabliert. Klassische Verfahren haben sich als

leistungsfähige Heuristiken und Metaheuristiken erwiesen, um kombinatorische Optimierungsprobleme effizient zu approximieren (Blum und Roli, 2003; E. Burke und Newall, 1999; Chu und Fang, 1999). Diese Ansätze erzielen für klar definierte Problemklassen häufig gute Ergebnisse, erfordern jedoch eine manuelle und domänenspezifische Anpassung der Parameter, Operatoren und Fitnessfunktionen. Eine Übertragbarkeit auf verschiedene Anwendungskontexte ist daher nur eingeschränkt möglich (Bashab et al., 2022; Pillay, 2014).

Zur Reduktion dieses Anpassungsaufwands wurden generische oder hyper-heuristische Frameworks entwickelt, die auf einer höheren Abstraktionsebene zwischen Heuristiken auswählen oder neue Strategien kombinieren können (E. Burke et al., 2003; Drake et al., 2020; Özcan et al., 2008). Dennoch bleibt die Herausforderung bestehen, geeignete Heuristiken für spezifische Instanzen automatisch zu identifizieren oder zu erzeugen (Gusti Agung Premananda et al., 2024). Selbst moderne generische Evolutionsverfahren verfügen bislang über keine Mechanismen, um Wissen über Problemstrukturen oder Nebenbedingungen adaptiv zu erlernen und zu generalisieren.

Parallel dazu hat der rasante Fortschritt im Bereich der generativen KI, insbesondere durch LLMs, neue Forschungsrichtungen eröffnet. Erste Arbeiten (Liu et al., 2023, 2024) zeigen, dass LLMs genutzt werden können, um algorithmische Bausteine wie Heuristiken, Selektionsstrategien oder Mutationsoperatoren automatisch zu generieren und zu bewerten („Evolution of Heuristics“). Ebenso wird untersucht, wie LLMs zur textbasierten Problembeschreibung und automatisierten Lösungsgenerierung in Scheduling-Kontexten eingesetzt werden können (Jobson und Li, 2024; Szeider, 2024). Trotz vielversprechender Ergebnisse bleibt die Forschung in einem frühen Stadium. Die bislang getesteten Szenarien sind meist stark vereinfacht, nutzen statische Prompt-Strukturen und verzichten auf eine systematische Kopplung mit evolutionären Optimierungsverfahren (Wu et al., 2024).

Daraus ergibt sich eine doppelte Forschungslücke: Erstens fehlt ein systematischer Ansatz, der LLM-basierte Heuristikgenerierung mit klassischen evolutionären Algorithmen integriert, um ein adaptives, domänenübergreifendes Optimierungssystem zu schaffen. Zweitens existieren bislang kaum Untersuchungen, wie textuell formulierte Constraints und Kontextinformationen automatisch, ohne manuelle Regeldefinitionen, in die algorithmische Entscheidungslogik eingebunden werden können.

Diese Arbeit adressiert diese Forschungslücke, indem sie einen hybriden Ansatz entwickelt, der die Prinzipien evolutionärer Algorithmen mit den generativen und adaptiven

Fähigkeiten von LLMs verbindet (Liu et al., 2024). Der entwickelte Ansatz abstrahiert die Kombination zu einem generischen Optimierungssystem und erweitert dieses. Damit leistet die Arbeit einen Beitrag zur Weiterentwicklung generischer, adaptiver Optimierungsverfahren an der Schnittstelle zwischen klassischer evolutionärer Suche und moderner Sprachmodelle.

2 Theoretische Grundlagen

Das folgende Kapitel stellt die theoretischen Grundlagen bereit, die für das Verständnis der späteren Implementierung wesentlich sind. Zunächst wird das grundlegende Konzept von Timetabling-Problemen erläutert und ein Überblick über gängige Lösungsansätze gegeben. Anschließend werden evolutionäre Algorithmen sowie LLMs behandelt, um die Grundlage für deren zielgerichteten Einsatz in der Umsetzung zu schaffen. Im Rahmen dessen werden auch Methoden zur Bewertung der Ergebnisqualität vorgestellt, um eine theoretische Basis für die objektive Beurteilung der Lösungsansätze zu schaffen.

2.1 Timetabling Problem

Das Timetabling Problem, auch als Timetable Scheduling Problem bezeichnet, stellt ein klassisches Optimierungsproblem im Bereich der Informatik dar. Es gehört zur übergeordneten Kategorie der Scheduling- und Planning-Probleme. Während Planning-Probleme sich mit der Frage befassen, *Was* getan werden muss, adressieren Scheduling-Probleme das *Wann*. Timetabling-Probleme konzentrieren sich somit auf die zeitliche Zuordnung bestimmter Aktivitäten zu verfügbaren Ressourcen innerhalb eines festgelegten Zeitrahmens (Vinod Kadam und Samir Yadav, 2016).

Ein Timetabling-Problem besteht typischerweise aus den folgenden Komponenten (E. Burke und Newall, 2004):

- einer endlichen Menge von Zeitfenstern T
- einer endlichen Menge von Ressourcen R
- einer endlichen Menge an Ereignissen M
- einer endlichen Menge an Nebenbedingungen (Constraints) C, unterteilt in harte (hard) $C_h \subseteq C$ und weiche (soft) $C_s \subseteq C$ Constraints.

Das Ziel besteht darin, die Zeitfenster T und Ressourcen R derart auf die Sitzungen M zu verteilen, dass die Nebenbedingungen C möglichst umfassend erfüllt werden (E. Burke und Newall, 2004).

Harte und weiche Nebenbedingungen, im Weiteren Constraints genannt, bilden gemeinsam die Menge der Bedingungen. Harte Constraints stellen Bedingungen dar, die zwingend eingehalten werden müssen. Typische Beispiele umfassen etwa die Anforderung,

dass eine Person nicht gleichzeitig an mehreren Orten sein kann oder dass bestimmte organisatorische Rahmenbedingungen erfüllt sein müssen (Vinod Kadam und Samir Yadav, 2016). Ein Verstoß gegen harte Constraints führt dazu, dass der erstellte Zeitplan nicht qualifiziert genug ist oder nachträglich überarbeitet werden muss.

Weiche Constraints hingegen beschreiben wünschenswerte, jedoch nicht zwingend notwendige Anforderungen. Werden diese teilweise oder gar nicht erfüllt, bleibt der Zeitplan in der Regel weiterhin verwendbar. Aufgrund der oft widersprüchlichen Natur solcher weichen Anforderungen ist es in der Praxis häufig nicht möglich, sämtliche weiche Constraints vollständig zu erfüllen (Vinod Kadam und Samir Yadav, 2016). Ziel ist es daher, eine Lösung zu finden, die möglichst viele dieser Bedingungen berücksichtigt.

Timetabling-Probleme lassen sich aus mathematischer Sicht als kombinatorische Optimierungsprobleme einordnen. Dabei handelt es sich um eine Klasse von Problemen, bei denen eine optimale Lösung aus einer endlichen, aber häufig exponentiell großen Menge möglicher Kombinationen ausgewählt werden muss. Die Schwierigkeit solcher Probleme ergibt sich insbesondere aus der großen Zahl potenzieller Belegungen sowie der zahlreichen Constraints, die berücksichtigt werden müssen (Bashab et al., 2022).

Formal können Timetabling-Probleme häufig als Varianten des Constraint Satisfaction Problems (CSP) oder des Constraint Optimization Problems (COP) modelliert werden. Dabei wird überprüft, ob eine Belegung der Variablen existiert, die alle harten Constraints erfüllt, während weiche Constraints in einer Zielfunktion optimiert werden. Diese Modellierung verdeutlicht die enge Verwandtschaft zu klassischen NP-vollständigen Problemen wie dem Graph-Coloring-Problem. Bei diesem werden die Knoten eines Graphen so eingefärbt, dass benachbarte Knoten unterschiedliche Farben erhalten. Konkret werden Timetabling-Probleme damit als NP-COP klassifiziert – also einem non-deterministic polynomial combinatorial optimization problem (Bashab et al., 2022).

Ein Problem ist als NP-schwer (engl. NP-hard) klassifiziert, wenn jede Instanz eines Problems in der Komplexitätsklasse NP in polynomieller Zeit auf eine Instanz des betrachteten Problems reduziert werden kann. Für NP-schwere Probleme existieren bisher keine Algorithmen, die in polynomieller Zeit für alle Instanzen eine optimale Lösung finden können. Dies gilt auch für Timetabling-Probleme, insbesondere dann, wenn sie reale Rahmenbedingungen mit komplexen harten und weichen Constraints abbilden (Bashab et al., 2022; Pillay, 2014).

In der Praxis bedeutet dies, dass exakte Lösungsverfahren, etwa vollständige Enumeration oder Backtracking, für realistische Problemgrößen in der Regel nicht mehr effizient einsetzbar sind. Stattdessen kommen häufig heuristische oder metaheuristische Verfahren zum Einsatz, die innerhalb akzeptabler Rechenzeit gute, wenn auch nicht notwendigerweise optimale, Lösungen finden können. Beispiele hierfür sind Simulated Annealing, Genetische Algorithmen oder Tabu Search (Augugliaro et al., 1999; Chu und Fang, 1999).

Die NP-Schwere von Timetabling-Problemen stellt somit eine zentrale Herausforderung dar und ist maßgeblich dafür verantwortlich, dass in der Forschung und Praxis nach Lösungen für diese Problemstellung gesucht wird.

2.2 Lösungsansätze für Timetabling-Probleme

Aufgrund der nachgewiesenen NP-Schwere vieler Varianten von Timetabling-Problemen ist es in der Praxis häufig nicht möglich, exakte Lösungsverfahren effizient einzusetzen (Rhydian Lewis, 2007). Das folgende Kapitel stellt eine Auswahl verschiedener heuristischer und metaheuristischer Verfahren vor, welche in der Literatur existieren, mit denen Optimierungsprobleme gelöst werden können.

2.2.1 Heuristische Verfahren

Heuristiken basieren auf Problemspezifika und nutzen Domänenwissen, um gezielt Suchpfade zu verfolgen. Sie liefern meist schnell brauchbare Lösungen, ohne den gesamten Lösungsraum zu durchsuchen (Rhydian Lewis, 2007). Bekannte heuristische Strategien im Timetabling-Kontext sind beispielsweise:

Greedy-Algorithmen: Der Algorithmus 1 arbeitet iterativ und wählen in jedem Schritt die aus aktueller Sicht „beste“ Entscheidung aus. Vorab wird die Liste aller Elemente (in diesem Fall Prüfungen bzw. *Exams*) anhand der Schwere der Einsortierung der Elemente in den Zeitplan sortiert (E. Burke und Newall, 2004). Die konkrete Implementierung der Bewertung ist abhängig von dem Anwendungsfall. In der darauffolgenden Schleife werden die Elemente in den Plan so einsortiert, dass möglichst wenig Fehler entstehen.

Algorithm 1: Greedy Algorithm nach E. Burke und Newall, 2004

$exams \leftarrow difficulty_{desc}(exams)$;
for *e in exam* **do**
  if *e can be scheduled in timetable* **then**
    schedule e in period with least penalty
  else
    Leave e unscheduled

Saturation Degree Heuristic: Der Algorithmus 2 priorisiert Veranstaltungen, die in besonders wenigen Zeitslots untergebracht werden können (Brélaz, 1979). In dieser Variante des *Dsatur*-Algorithmus nach Brélaz, 1979 wird jede Unterrichtseinheit l_i als Knoten im Graphen modelliert, wobei eine Kante (l_i, l_j) einen Konflikt beschreibt (z. B. gleicher Lehrer, gleiche Klasse oder gleicher Raum). Der Algorithmus wählt schrittweise die Einheit mit dem höchsten Sättigungsgrad $\text{sat}(l_i)$, also der Anzahl verschiedener bereits belegter Zeitfenster ihrer Nachbarn. Diesem weist er das kleinste verfügbare Zeitfenster $c(l_i)$ zu, das keinen Konflikt verursacht. Die Auswahlreihenfolge orientiert sich zusätzlich am Grad $\deg(l_i)$, also der Anzahl direkter Konflikte eines Knotens. Dieser Prozess wird fortgesetzt, bis alle Unterrichtseinheiten eingeplant sind und ein konfliktfreier Stundenplan mit minimaler Anzahl an Zeitperioden entsteht.

Algorithm 2: Dsatur Algorithm for School Lesson Scheduling nach Brélaz, 1979

Data: Graph $G = (L, E)$, lessons $L = \{l_1, \ldots, l_n\}$,
edges $(l_i, l_j) \in E$ if l_i and l_j cannot be scheduled simultaneously (same teacher, class, or room)

Result: Conflict-free timetable $c : L \to \mathbb{N}$ minimizing the number of periods

foreach $l_i \in L$ **do**
    $\deg(l_i) \leftarrow |\{l_j \mid (l_i, l_j) \in E\}|$;
    $\text{sat}(l_i) \leftarrow 0$;
    $c(l_i) \leftarrow 0$
end
$l^* \leftarrow \arg\max_{l_i \in L} \deg(l_i)$;
$c(l^*) \leftarrow 1$;
while *some* l_i *remains unassigned* **do**
    foreach *unassigned* l_i **do**
        $\text{sat}(l_i) \leftarrow |\{c(l_j) \mid (l_i, l_j) \in E,\ c(l_j) \neq 0\}|$;
    end
    $l^* \leftarrow \arg\max_{l_i \in L \text{ unassigned}} (\text{sat}(l_i), \deg(l_i))$;
    $C_{\text{conf}} \leftarrow \{c(l_j) \mid (l^*, l_j) \in E,\ c(l_j) \neq 0\}$;
    $c(l^*) \leftarrow \min\{p \in \mathbb{N} \mid p \notin C_{\text{conf}}\}$;
end
return c

Heuristische Verfahren sind leicht implementierbar und bieten einen guten Ausgangspunkt für komplexere Strategien, stoßen jedoch bei stark konträren bzw. im Konflikt stehenden Anforderungen häufig an ihre Grenzen (E. Burke und Newall, 2004).

2.2.2 Metaheuristische Verfahren

Metaheuristiken sind allgemeiner formulierte Optimierungsverfahren, die unabhängig vom spezifischen Problemtyp eingesetzt werden können. Sie kombinieren systematische und stochastische Elemente, um lokale Optima zu vermeiden und eine bessere Durchmischung des Suchraums zu erreichen (E. Burke et al., 2003). Zu den am häufigsten eingesetzten Metaheuristiken im Kontext von Timetabling zählen:

Simulated Annealing (SA): Der Algorithmus 3 (nach Dowsland, 1993) basiert auf einem stochastischen Optimierungsverfahren, das von physikalischen Abkühlungsprozes-

sen inspiriert ist. Ausgangspunkt ist eine anfängliche Lösung, deren Qualität durch eine Kostenfunktion bewertet wird. In jeder Iteration wird eine leicht veränderte Nachbarlösung erzeugt und deren Kostenunterschied zur aktuellen Lösung berechnet. Verbesserte Lösungen werden stets akzeptiert, während schlechtere Lösungen mit einer gewissen Wahrscheinlichkeit übernommen werden, die von der Temperatur abhängt. Diese Wahrscheinlichkeit nimmt im Verlauf des Algorithmus mit sinkender Temperatur ab. Dadurch wird eine anfängliche Exploration des Suchraums ermöglicht, die sich schrittweise zu einer gezielten Exploitation entwickelt. Das Verfahren endet, sobald ein Abbruchkriterium erreicht ist, und liefert die beste gefundene Lösung.

Algorithm 3: Simulated Annealing nach Dowsland, 1993

Input: Initial solution S, cost function $f(S)$, initial temperature T
Output: Improved solution S_{best}
$S_{\text{best}} \leftarrow S$;
while *stopping condition not met* **do**
    Generate a neighbour S' of S;
    $\Delta \leftarrow f(S') - f(S)$;
    if $\Delta < 0$ **then**
        $S \leftarrow S'$
    else if $random(0,1) < e^{-\Delta/T}$ **then**
        $S \leftarrow S'$
    Update temperature T;
    Update S_{best} if $f(S) < f(S_{\text{best}})$;
return S_{best};

Tabu Search (TS): Der Algorithmus 4 (nach E. Burke und Newall, 1999) erweitert lokale Suchverfahren um ein Gedächtnisprinzip, um das wiederholte Durchsuchen bereits besuchter Lösungsräume zu vermeiden. Ausgangspunkt ist eine anfängliche Lösung, aus der durch kleine Änderungen eine Nachbarschaft möglicher Alternativen erzeugt wird. Die jeweils beste Nachbarlösung wird gewählt, sofern sie nicht in der sogenannten Tabuliste vermerkt ist oder ein definiertes Aspirationskriterium erfüllt. Die Tabuliste speichert kürzlich durchgeführte Änderungen und verhindert dadurch kurzfristige Rücksprünge in bereits untersuchte Zustände. Nach jeder Iteration wird die Liste aktualisiert und ältere Einträge verfallen. Auf diese Weise kombiniert der Algorithmus zielgerichtete Suche mit kontrollierter Diversifikation und liefert nach Ablauf der Abbruchbedingung die beste gefundene Lösung.

Algorithm 4: Tabu Search nach E. Burke und Newall, 1999

Input: Initial solution S, cost function $f(S)$, tabu tenure t_{max}

Output: Best timetable S_{best}

$S_{\text{best}} \leftarrow S$;

Initialize an empty tabu list T;

while *stopping condition not met* **do**

    Generate a set of neighbour solutions $N(S)$ by small timetable modifications;

    Select the best candidate S' from $N(S)$ that is not tabu or satisfies the aspiration criterion;

    $S \leftarrow S'$;

    Update T with the move leading to S' (expire oldest entries when $|T| > t_{max}$);

    if $f(S) < f(S_{best})$ **then**

        $S_{\text{best}} \leftarrow S$;

return S_{best};

Particle Swarm Optimization (PSO): Der Algorithmus 5 (nach Tassopoulos und Beligiannis, 2012) ist ein populationsbasiertes Optimierungsverfahren, das vom Schwarmverhalten in der Natur inspiriert ist. Eine Menge von Partikeln bewegt sich dabei durch den Lösungsraum, wobei jedes Partikel eine potenzielle Lösung darstellt. Jedes Partikel merkt sich seine bisher beste Position und orientiert sich zusätzlich am global besten gefundenen Zustand aller Partikel. Durch die Kombination von Trägheit, individueller Lernerfahrung und kollektiver Orientierung wird die Geschwindigkeit und Position jedes Partikels schrittweise angepasst. Dieser Mechanismus ermöglicht sowohl Exploration als auch Konvergenz hin zu vielversprechenden Regionen im Suchraum. Am Ende liefert der Algorithmus die beste gefundene Lösung.

Algorithm 5: Particle Swarm Optimization nach Tassopoulos und Beligiannis, 2012

Input: Population of particles P, cost function $f(S)$, inertia weight w, learning factors c_1, c_2

Output: Best timetable S_{gbest}

Initialize each particle i with a random timetable (position) and velocity;

Evaluate fitness $f(S_i)$ for all particles;

Set each particle's best-known position $S_{\text{pbest},i}$ and determine global best S_{gbest};

while *stopping condition not met* **do**

  foreach *particle* i **do**

    Update velocity:

    $v_i \leftarrow w \cdot v_i + c_1 \cdot rand() \cdot (S_{\text{pbest},i} - S_i) + c_2 \cdot rand() \cdot (S_{\text{gbest}} - S_i)$;

    Update position S_i based on new velocity (apply discrete mapping to feasible timetable);

    Evaluate $f(S_i)$ and update $S_{\text{pbest},i}$ if improved;

  Update S_{gbest} if a better solution is found;

return S_{gbest};

Die Wahl des Verfahrens hängt stark von den konkreten Problemparametern, der Struktur der Constraints und den Anforderungen an Lösungsqualität und Rechenzeit ab. Häufig werden auch hybride Ansätze verfolgt, bei denen mehrere Methoden kombiniert werden, etwa durch die Verwendung einer Heuristik zur Initialisierung und einer Metaheuristik zur Optimierung (E. K. Burke et al., 2013).

2.3 Evolutionäre Algorithmen

Evolutionäre Algorithmen (EAs) sind stochastische, populationsbasierte Optimierungsverfahren, deren Funktionsweise von der natürlichen Evolution inspiriert ist. Wie die in Kapitel 2.2.2 aufgeführten Algorithmen gehören sie zu den metaheuristischen Verfahren. Ziel von evolutionären Algorithmen ist es, durch wiederholte Anwendung von Selektions- und Variationsmechanismen die Qualität der Lösungskandidaten innerhalb eines gegebenen Suchraums schrittweise zu verbessern (Michalewicz et al., 1997; Simon, 2013). Das Vorgehen eines evolutionären Algorithmus lässt sich schematisch wie in Algorithmus 6 darstellen.

Algorithm 6: Evolutionärer Algorithmus nach Michalewicz et al., 1997

```
1 t ← 0;
2 initialize P(t);
3 evaluate P(t);
4 while not termination-condition do
5 |   t ← t + 1;
6 |   select P(t) from P(t − 1);
7 |   alter P(t);
8 |   evaluate P(t);
```

Wie in Algorithmus 6 beschrieben, operiert ein evolutionärer Algorithmus auf einer Population $P(t) = \{x_1^t, \ldots, x_N^t\}$ von N Individuen (Lösungskandidaten) in der Generation t, wobei jedes Individuum $x \in S$ einem Punkt im Suchraum S entspricht. Jedem Individuum ist ein Fitnesswert $f(x)$ zugeordnet, der die Qualität der jeweiligen Lösung quantifiziert. Der Algorithmus verfolgt das Ziel, über mehrere Generationen hinweg Individuen mit möglichst guter Fitness zu identifizieren und so eine optimale oder hinreichend gute Lösung des zugrunde liegenden Problems zu finden.

Ein evolutionärer Algorithmus ist zyklisch aufgebaut. Die Phasen, die dabei durchlaufen werden, sind zunächst die Initialisierung und die Evaluierung. Im Detail lassen sich diese nach Michalewicz, 1996 wie folgt beschreiben:

- **Initialisierung:** Zu Beginn wird eine initiale Population $P(0)$ erzeugt, wobei die Individuen in der Regel zufällig und möglichst gleichmäßig über den Suchraum verteilt werden. Dies fördert eine hohe genetische Diversität und reduziert die Gefahr einer frühzeitigen Konvergenz auf suboptimale Lösungen.
- **Evaluation:** Jedes Individuum $x \in P(t)$ wird anhand einer vordefinierten Fitnessfunktion bewertet. Der resultierende Fitnesswert gibt Aufschluss über die Qualität der Lösung und dient als Grundlage für den Auswahlprozess.

Nachdem diese initiale Population und die dazugehörige Bewertung durchgeführt wurden, geht der Algorithmus in einen zyklischen Prozess zur Verbesserung der Lösungen über (vgl. Algorithmus 6). Hierfür ist eine Abbruchbedingung (*termination condition*) vordefiniert. Diese besteht in der Regel aus einer maximalen Anzahl von Generationen t oder einem Schwellenwert der Fitnessfunktion. Wird der Schwellenwert erreicht, wird

das Resultat des Algorithmus als hinreichend gut angesehen. Die im Zyklus durchzuführenden Schritte der Selektion, Variation und Evaluation lassen sich im Detail wie folgt beschreiben (Michalewicz et al., 1997):

- **Selektion:** In dieser Phase werden Individuen auf Basis ihrer Fitnesswerte für die Reproduktion ausgewählt. Ziel ist es, fittere Individuen mit höherer Wahrscheinlichkeit als Eltern für die nächste Generation heranzuziehen. Gängige Selektionsverfahren sind (Michalewicz et al., 1997):
  - *Fitness-proportionale Selektion:* Die Auswahlwahrscheinlichkeit eines Individuums x_i ist proportional zu dessen Fitnesswert: $P(x_i) = \frac{f(x_i)}{\sum_{j=1}^{N} f(x_j)}$.
  - *Rangbasierte Selektion:* Individuen werden entsprechend ihrer Fitnesswerte geordnet, und die Auswahl erfolgt basierend auf ihrer Rangposition.
  - *Turnierselektion:* Eine zufällige Teilmenge der Population wird ausgewählt, und das Individuum mit der höchsten Fitness innerhalb dieser Teilmenge wird als Elternteil bestimmt.
- **Variation (Reproduktion):** Die ausgewählten Individuen der Vorgängergeneration werden verändert, um Nachkommen für die neue Generation zu erzeugen. Dieses Vorgehen erfolgt zumeist nach den Prinzipien genetischer Algorithmen (Michalewicz et al., 1997). Dabei werden allgemein zwei Mechanismen eingesetzt, um neue Individuen für die Population $P(t)$ zu erzeugen:
  - **Rekombination (Crossover):** Es werden aus der Vorgängergeneration eine Anzahl von n (meist zwei) Eltern ausgewählt. Die ausgewählten Eltern werden mittels Crossover-Operatoren kombiniert, um neue Individuen zu erzeugen. Ein Crossover-Operator $c : S \times S \to S$ erzeugt aus zwei Elternteilen einen oder mehrere Nachkommen, die genetische Merkmale beider Elternteile enthalten.
  - **Mutation:** Um die genetische Diversität zu erhalten und das Suchverhalten zu verbessern, werden die durch Crossover erzeugten Nachkommen zufällig verändert. Eine Mutation tritt mit einer bestimmten Wahrscheinlichkeit $m \in [0, 1]$ auf. Dabei wird ein Individuum in einzelnen Komponenten variiert, was neue Suchrichtungen eröffnet und eine mögliche lokale Stagnation verhindert.
- Die anschließende **Evaluation** erfolgt analog zur zuvor beschriebenen Bewertung während der Initialisierungsphase.

Die beschriebenen Schritte des Zyklus wiederholen sich, bis eine der Abbruchbedingungen erfüllt ist. Der Algorithmus liefert dann das zu diesem Zeitpunkt als am besten bewertete Ergebnis zurück.

Dieser iterative Ablauf ist charakteristisch für evolutionäre Algorithmen und bildet ihre zentrale Funktionsweise: die kontinuierliche Verbesserung von Lösungskandidaten durch Variation und Selektion nach dem Prinzip der natürlichen Selektion („Survival of the Fittest“). Die zyklische Struktur ermöglicht es dem Algorithmus, adaptiv auf komplexe, nichtlineare und hochdimensionale Optimierungsprobleme zu reagieren und dabei verschiedene Suchstrategien zu vereinen (Michalewicz, 1996).

Beispielhaft ist der Ablauf eines evolutionären Algorithmus in Abbildung 1 dargestellt. f steht hierbei kurz für die Fitnessfunktion $f(x)$. Die Fitness Funktion zielt in diesem Fall darauf ab, möglichst niedrig zu sein (möglichst nah an 0) und so die Fehlerzahl zu minimieren. Als maximale Anzahl an Generation T ist 10 gesetzt. Das Beispiel ist angelehnt an die Beschreibung durch den Algorithmus 1.

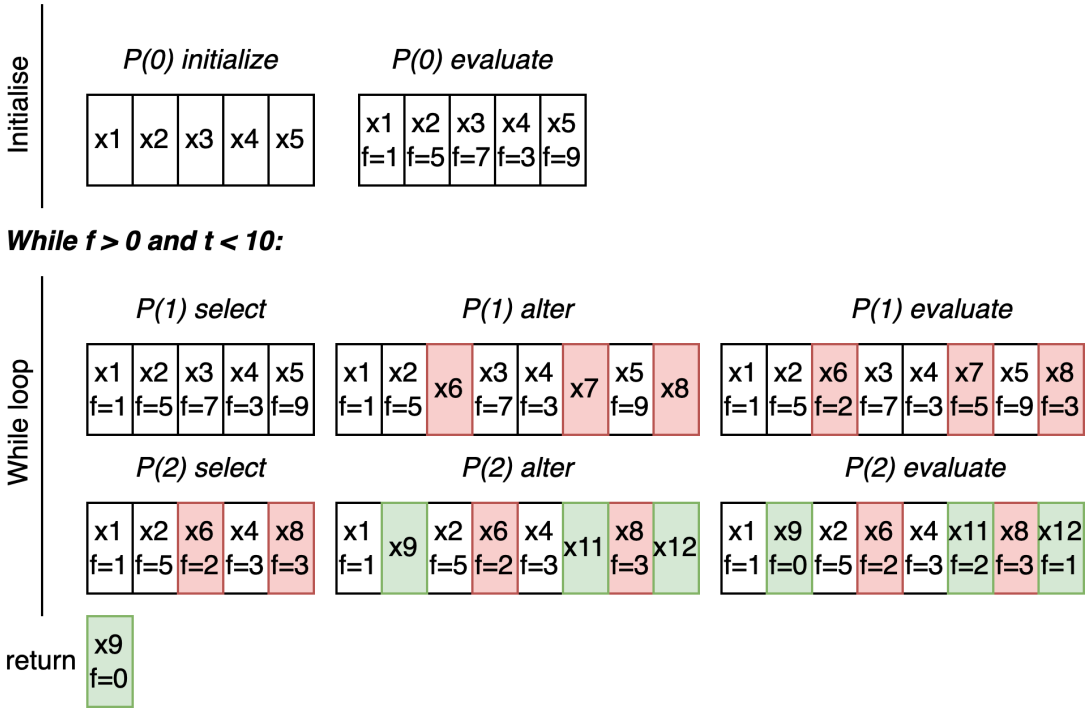

Abbildung 1: Beispielhafter Ablauf eines evolutionären Algorithmus (eigene Darstellung)

Ein Framework zur Implementierung evolutionärer Algorithmen bilden die sogenannten generischen evolutionären Algorithmen (GEAs). Der generische Algorithmus abstrahiert dabei die Kernkomponenten des evolutionären Algorithmus – also Initialisierung, Evaluation, Selektion und Variation – und parametrisiert diese über eine Problemdefinition (Gusti Agung Premananda et al., 2024). Ziel ist es, eine universell einsetzbare Rahmenstruktur bereitzustellen, die für verschiedene Anwendungsszenarien spezifisch implementiert werden kann. Die genaue Implementierung der anwendungsspezifischen Logiken, die neben dem Rahmen erforderlich sind, bleibt jedoch auch bei generischen Algorithmen eine Herausforderung, die ihre universelle Einsetzbarkeit über verschiedene Anwendungsszenarien hinweg einschränkt (Drake et al., 2020).

2.4 Bewerten von Ergebnissen evolutionärer Algorithmen

Die systematische Bewertung der Ergebnisse evolutionärer Algorithmen ist ein zentrales Element jeder Optimierungsstudie. Sie liefert die Grundlage für methodische Vergleiche, die Auswahl geeigneter Konfigurationen und die Validierung neuer Ansätze. Insbesondere im Kontext komplexer Zeitplanungsprobleme, bei denen mehrere konkurrierende Ziele und vielfältige Nebenbedingungen berücksichtigt werden müssen, ist eine differenzierte Beurteilung unerlässlich.

Dieses Kapitel gliedert sich in zwei Teilbereiche. Der erste Abschnitt widmet sich der Bewertung der algorithmischen Leistungsfähigkeit, also der Analyse von Konvergenzverhalten und Lösungsqualität anhand quantitativer Metriken. Ziel ist es, die Effektivität und Robustheit evolutionärer Algorithmen im Hinblick auf definierte Planungsziele zu untersuchen und statistisch abzusichern.

Der zweite Abschnitt behandelt die Gestaltung geeigneter Fitnessfunktionen - also der internen Bewertung eines evolutionären Algorithmus von sich selbst. Diese fungieren als zentrale Steuergröße des Optimierungsprozesses und beeinflussen maßgeblich, ob und wie gut zulässige sowie qualitativ hochwertige Lösungen gefunden werden können.

2.4.1 Bewerten der Güte von Evolutionären Algorithmen

Die Bewertung der Güte evolutionärer Algorithmen (EA) bei Optimierungsproblemen erfolgt anhand mehrerer messbarer Kriterien. Da es sich bei EAs um stochastische Verfahren handelt, müssen Aussagen über ihre Leistungsfähigkeit stets statistisch abgesichert

werden. Üblicherweise erfolgt dies durch mehrfache unabhängige Läufe und die Auswertung gemittelter Kennzahlen (Eiben und Smith, 2015). Wesentliche Indikatoren sind das Konvergenzverhalten, die Lösungsqualität, der Rechenaufwand, die Robustheit sowie die Populationsdiversität (Jacquelin, 2019). Diese Metriken ermöglichen eine fundierte Beurteilung der Effektivität und Zuverlässigkeit des Algorithmus in Bezug auf die gestellte Optimierungsaufgabe.

Konvergenzverhalten Das Konvergenzverhalten beschreibt den Verlauf der Lösungsverbesserung über die Iterationen hinweg. Häufig wird hierzu eine sogenannte *Progress Curve* aufgezeichnet, die den besten oder durchschnittlichen Fitnesswert über die Generationen darstellt (Eiben und Smith, 2015). Eine steil ansteigende Kurve signalisiert eine schnelle Verbesserung, während eine früh stagnierende Kurve auf das Festlaufen in einem lokalen Optimum hindeutet[1]. Die Analyse der Konvergenzkurve erlaubt Rückschlüsse auf das Gleichgewicht zwischen Exploration und Exploitation des Algorithmus. Ergänzend wird die Populationsdiversität betrachtet, da eine zu frühe Konvergenz häufig auf Kosten der Vielfalt geschieht und suboptimale Lösungen begünstigt (Eiben und Smith, 2015).

Lösungsqualität Die Qualität der gefundenen Lösungen wird in der Regel über den erreichten Fitnesswert bewertet. Hierbei werden sowohl der beste als auch der durchschnittlich beste Fitnesswert (Mean Best Fitness, MBF) über mehrere Läufe berücksichtigt (Eiben und Smith, 2015). Ist ein Optimalwert bekannt, kann der Abstand zur optimalen Lösung über das sogenannte *Optimality Gap* oder die *Success Rate (SR)* quantifiziert werden. Liegt kein globales Optimum vor kann eine praxisnahe Referenz, etwa ein manuell erstellter Plan, als Vergleichsmaßstab dienen. Neben optimalitätsbezogenen Kriterien spielt die Einhaltung harter Constraints eine zentrale Rolle, da unzulässige Lösungen trotz guter Zielfunktion keine praktische Relevanz besitzen.

Laufzeit und Effizienz Die Effizienz eines EAs wird häufig über die benötigte Rechenzeit oder über hardwareunabhängige Kennzahlen, wie die durchschnittliche Anzahl an Fitnessauswertungen, gemessen (Eiben und Smith, 2015). In der Praxis ist die Laufzeit, insbesondere in zeitkritischen Anwendungen, ein entscheidendes Bewertungskriterium. Die Performanz wird daher als Trade-off zwischen Lösungsqualität und Rechenaufwand

[1] Ansteigend gilt hierbei, wenn die erreichte Fitness möglichst hoch sein soll. Bei einer Fitness, welche gegen null konvergiert, also bei null am besten ist, gilt die Richtung genau umgekehrt.

betrachtet. Ein gutes Anytime-Verhalten, also die Fähigkeit, bereits nach kurzer Laufzeit qualitativ hochwertige Lösungen zu liefern, ist von besonders hoher Bedeutung.

Robustheit der Ergebnisse Die Robustheit beschreibt die Zuverlässigkeit des Algorithmus, bei wiederholten Ausführungen konsistente Ergebnisse zu erzielen. Aufgrund der stochastischen Natur evolutionärer Verfahren können unterschiedliche Läufe variierende Resultate liefern. Eine geringe Standardabweichung der Fitnesswerte über mehrere Läufe weist auf eine hohe Robustheit hin (Eiben und Smith, 2015). Darüber hinaus kann Robustheit als Stabilität gegenüber Parameteränderungen oder unterschiedlichen Instanzen interpretiert werden. Ein robuster Algorithmus erzielt somit auch ohne erneute Feinabstimmung vergleichbare Ergebnisse.

Populationsdiversität Die Diversität innerhalb der Population beschreibt die Vielfalt der Lösungen zu einem gegebenen Zeitpunkt. Eine hohe Diversität ist notwendig, um den Suchraum breit zu explorieren, das Risiko einer vorzeitigen Konvergenz zu verringern und nicht in einem lokalen Optimum zu verfangen (Jacquelin, 2019). Typische Metriken sind die mittlere Hamming-Distanz bei binären Kodierungen oder allgemeine Entropie-Indikatoren. Ein Rückgang der Diversität kann auf das Zusammenfallen der Population und somit auf eine eingeschränkte Suchfähigkeit hinweisen. Mechanismen wie Mutationen dienen gezielt der Aufrechterhaltung dieser Vielfalt.

In praxisnahen Szenarien, wie der Zeitplanoptimierung, müssen die genannten Kriterien gemeinsam betrachtet werden. Ein evolutionärer Algorithmus gilt als erfolgreich, wenn er in akzeptabler Zeit einen gültigen, qualitativ hochwertigen Zeitplan erzeugt, der harte und weiche Constraints berücksichtigt. Zur objektiven Beurteilung der Leistungsfähigkeit kommen statistische Testverfahren, wie der Wilcoxon-Vorzeichen-Rang-Test oder der Friedman-Test, zum Einsatz (Brest und Sepesy Maučec, 2025; Eiben und Smith, 2015). Diese Verfahren prüfen, ob beobachtete Unterschiede in den Fitnesswerten signifikant sind. Für den statistischen Vergleich zweier Algorithmen wird in der Regel der Wilcoxon-Vorzeichen-Rang-Test herangezogen, während der Friedman-Test bei mehr als zwei Verfahren Anwendung findet (Kim, 2014). Da im Rahmen dieser Arbeit mehr als zwei Algorithmen miteinander verglichen werden, wird im Folgenden ausschließlich der Friedman-Test näher erläutert:

Der Friedman-Test ist ein nichtparametrisches Verfahren zur Analyse von Varianz in verbundenen Stichproben und eignet sich insbesondere für den Vergleich mehrerer Algo-

rithmen über verschiedene Datensätze hinweg (Demšar, 2006). Er basiert auf Rangwerten anstelle der Rohdaten und prüft, ob alle betrachteten Verfahren im Mittel dieselbe Leistung zeigen.

Für die Durchführung werden zunächst für jeden Datensatz $j \in \{1, \ldots, N\}$ eines Algorithmus i die Kennzahlen in Ränge R_{ij} umgewandelt. Dafür wird für jede Messgröße der Wert eines Datensatzes mit denen der anderen Algorithmen verglichen. Die beste Lösung erhält den Rang eins. Anschließend wird für jeden Algorithmus der mittlere Rang über alle Datensätze berechnet:

$$\bar{R}_i = \frac{1}{N} \sum_{j=1}^{N} R_{ij}. \tag{1}$$

Dies wird dann die Formel der Teststatistik des Friedman-Tests überführt:

$$\chi_F^2 = \frac{12N}{k(k+1)} \left(\sum_{i=1}^{k} \bar{R}_i^2 - \frac{k(k+1)^2}{4} \right) \tag{2}$$

wobei N die Anzahl der Datensätze und k die Anzahl der verglichenen Algorithmen ist. Der daraus resultierende χ_F^2-Wert kann bei kleinen Stichproben (N $<$ 10 und k $<$ 5) zu ungenau sein. Daher wird der von Iman und Davenport, 1980 vorgeschlagene, auf der F-Verteilung basierende Korrekturterm verwendet (Demšar, 2006):

$$F_F = \frac{(N-1)\chi_F^2}{N(k-1) - \chi_F^2}, \tag{3}$$

Der daraus resultierende Wert F_F muss für einen signifikanten Unterschied der Werte oberhalb der F-Verteilung liegen[2]. In dieser Arbeit wird, wie zumeist in der Literatur, als Signifikanzniveau α in Höhe von 5% angewendet. Damit wird die F-Verteilung wie folgt ermittelt:

$$F_{k-1,(k-1)(N-1),\alpha} \tag{4}$$

[2]Die F-Verteilung kann aus unterschiedlichen Quellen nachgelesen oder mit gängigen Tools ermittelt werden. Eine umfangreiche Tabelle ist beispielsweise: https://www.saskoer.ca/app/uploads/sites/313/2020/05/F-Distribution-Table.pdf

Wenn daraus resultierend $F_F > F_{k-1,(k-1)(N-1),\alpha}$ ist, dann ist der Unterschied der Algorithmen signifikant (Demšar, 2006).

Als Nächstes wird der Nemenyi-post-hoc Test angewendet. Dieser wird verwendet, um bei einem signifikanten Unterschied zu ermitteln, zwischen welchen Algorithmen ein signifikanter Unterschied besteht (Demšar, 2006; Nemenyi, 1963).

Hierfür wird zunächst die Critical Difference (CD) berechnet, welche den Schwellenwert für den Unterschied darstellt, der für einen signifikanten Unterschied mindestens gegeben sein muss (Demšar, 2006):

$$CD = q_\alpha \sqrt{\frac{k(k+1)}{6N}} \tag{5}$$

Der hier angegebene q_α Wert ist ebenfalls ein je nach gegebenem α fester Wert, welche als Tabelle unter anderem bei Demšar, 2006 aufgeführt werden.

Daraufhin wird die Differenz zwischen den jeweiligen durchschnittlichen Rängen mit der Critical Difference verglichen. Zwischen allen Algorithmenpaaren, wo die Differenz größer als CD ist, besteht ein signifikanter Unterschied. Algorithmen-Paare, welche einen niedrigeren Unterschied aufweisen, liegen nicht signifikant weit auseinander. Dieser Vergleich wird meist tabellarisch dargestellt.

Diese Vorgehensweise gewährleistet eine faire und nachvollziehbare Beurteilung der Leistungsfähigkeit unterschiedlicher Optimierungsverfahren in multidimensionalen Evaluationsszenarien.

2.4.2 Berechnen von Fitnessmethoden

Die Bewertung der Qualität eines Individuums innerhalb eines evolutionären Algorithmus erfolgt über eine Fitnessfunktion, welche die Zielfunktion sowie mögliche Nebenbedingungen eines Optimierungsproblems abbildet. Insbesondere bei kombinatorischen Optimierungsproblemen mit harten und weichen Constraints, wie sie typischerweise in der Planungsoptimierung auftreten, spielt die Konstruktion einer geeigneten Fitnessfunktion eine zentrale Rolle für die Performanz des Algorithmus (Coello Coello, 2002). Im Folgenden werden fünf etablierte Ansätze zur Berechnung der Fitness vorgestellt, die unterschiedliche Strategien im Umgang mit Nebenbedingungen und Mehrzielproblemen abbilden.

Strafterm-Methoden (Penalized Cost Functions) Eine der ältesten und am weitesten verbreiteten Methoden zur Behandlung von Nebenbedingungen in Optimierungsproblemen ist die Berechnung der Zielfunktion durch Strafkosten für Constraint-Verletzungen. Bei einem Optimierungsproblem mit Zielfunktion $f(x)$, K Ungleichungsbedingungen[3] $g_j(x) \leq 0$ und L Gleichungsbedingungen[4] $h_k(x) = 0$ ergibt sich die Fehler bestrafende Zielfunktion typischerweise zu:

$$f(x) = \sum_{j=1}^{K} \lambda_j \max(0, g_j(x)) + \sum_{k=1}^{L} \lambda'_k |h_k(x)|, \tag{6}$$

wobei $\lambda_j, \lambda'_k > 0$ Strafgewichte darstellen. Zulässige Lösungen verursachen keine Strafe, während Verletzungen mit einem Kostenzuschlag belegt werden. Alternativ können quadratische Strafterme wie $\max(0, g_j(x))^2$ oder $|h_k(x)|^2$ verwendet werden (Jorge Nocedal und Stephen J. Wright, 2006).

Für evolutionäre Algorithmen wird die Fitnessbewertung häufig in einer gewichteten Form umgesetzt, die harte (C_H) und weiche (C_S) Constraints berücksichtigt:

$$f(x_i) = \alpha \sum_{c_h \in C_H} \omega_h \, v(c_h, x_i) + \beta \sum_{c_s \in C_S} \omega_s \, v(c_s, x_i), \tag{7}$$

wobei $v(c, x_i)$ die Stärke der Constraintverletzung beschreibt und $\alpha \gg \beta$ sicherstellt, dass harte Bedingungen stärker bestraft werden als weiche. Dadurch können auch leicht falsche (wenige hart Constraints verletzende), aber potenziell vielversprechende Individuen in den Suchprozess einfließen. ω kann in dieser Funktion optional mit angegeben werden und steht für die Gewichtung eines Constraints in der jeweiligen Constraint-Gruppe (Coello Coello, 2002). Die Wahl geeigneter Strafkoeffizienten ist dabei entscheidend: Zu hohe Werte führen zu Suchstagnation, zu niedrige zu einer Vernachlässigung der Constraints. Zur Lösung dieses Problems wurden dynamische und adaptive Varianten entwickelt, die Strafgewichte im Verlauf der Optimierung automatisch anpassen (Takahama und Sakai, 2006).

Machbarkeitsregeln (Feasibility Rules) Eine weitere parameterfreie Methode wurde von Deb, 2000 vorgeschlagen. Die sogenannten Feasibility Rules ersetzen Strafwerte durch deterministische Vergleichsregeln:

[3] Ungleichungsbedingungen erwarten, dass ein bestimmter Wert kleiner als eine vorgegebene Größe ist.
[4] Gleichungsbedingungen erwarten, dass ein Wert genau dem Wert der Bedingung entspricht.

I. Sind beide Lösungen zulässig, gewinnt die mit besserem Zielfunktionswert.

II. Ist nur eine Lösung zulässig, gewinnt die diese.

III. Sind beide unzulässig, gewinnt die Lösung mit der kleineren Summe der Constraint-Verletzungen.

Diese Regeln gewährleisten, dass sich die Population schrittweise in den zulässigen Raum bewegt. Es wurde gezeigt, dass dieser Ansatz robust gegenüber verschiedenen Problemtypen ist und in vielen späteren EA-Implementierungen übernommen wurde (Deb, 2000). Da keine Strafparameter erforderlich sind, eignet sich der Ansatz insbesondere für Probleme mit unbekannter oder dynamischer Constraints-Struktur.

Constraint-Domination-Prinzip Ein erweiterter Ansatz der Machbarkeitsregel ist das sogenannte Constraint-Domination-Prinzip. Das von Deb, 2004 vorgestellte Konzept vergleicht ebenfalls zwei Lösungen x und y und gibt keine numerische Bewertung:

I. Ist x zulässig und y nicht, dominiert x.

II. Sind beide unzulässig, so wird die Lösung mit der geringeren Gesamtverletzung bevorzugt.

III. Sind beide zulässig, so entscheidet der Zielfunktionswert.

Dieses prioritätsbasierte Verfahren stellt sicher, dass zulässige Individuen im Selektionsprozess bevorzugt werden, ohne dass Strafparameter erforderlich sind. Es wird insbesondere in Mehrziel-Evolutionären Algorithmen eingesetzt, um hart Constraints verletzende Lösungen effizient auszusortieren (Deb, 2004). Der Unterschied zur Machbarkeitsregel besteht im Kern in dem Vergleich zwischen den einzelnen Lösungen in den Stufen. So werden in der zweiten Stufe harte Constraints zur Selektion betrachtet und in der dritten Stufe die bessere Bewertung der weichen Constraints. Bei der Machbarkeitsregel wird primär auf die Summe der Constraintsverletzungen geschaut, weshalb hier keine Anpassung oder Gewichtung möglich ist.

Gewichtete Summenaggregation Bei Optimierungsproblemen mit mehreren Kriterien werden mehrere Zielgrößen häufig zu einem skalaren Fitnesswert aggregiert. Die gewichtete Summenaggregation definiert die kombinierte Fitnessfunktion als:

$$F(x) = \sum_{i=1}^{M} w_i f_i(x), \tag{8}$$

wobei $w_i > 0$ die relative Bedeutung der Zielkriterien ausdrücken. Diese Methode ist einfach zu implementieren und erlaubt eine flexible Gewichtung von Teilzielen, beispielsweise zur Unterscheidung harter und weicher Constraints (Coello Coello, 2002).

Adaptive Strafwertverfahren Zur Überwindung der Limitierungen fixer Strafparameter wurden dynamische bzw. adaptive Penalty-Methoden entwickelt (Coello Coello, 2002). Diese Verfahren passen die Strafgewichte $\lambda(t)$ während der Evolution automatisch an den aktuellen Suchverlauf an, um ein Gleichgewicht zwischen Exploration und Exploitation zu gewährleisten. Anstatt einen festen Parameter zu wählen, wird $\lambda(t)$ in Abhängigkeit des Anteils zulässiger Individuen innerhalb der Population angepasst (Lemonge und Barbosa, 2004; Tessema und Yen, 2006). Ist der Anteil zulässiger Lösungen hoch, wird die Strafstärke reduziert, um den Suchraum breiter zu explorieren. Bei einer geringen Zahl zulässiger Individuen wird sie erhöht, um die Suche stärker in Richtung der zulässigen Regionen zu lenken. Ein solches adaptives Verfahren lässt sich formal wie folgt darstellen:

$$\lambda(t{+}1) = \begin{cases} \dfrac{1}{\beta_1}\lambda(t), & \text{wenn der Anteil zulässiger Individuen größer als ein Schwellwert } \tau_{\text{high}} \text{ ist,} \\ \beta_2\lambda(t), & \text{wenn der Anteil zulässiger Individuen kleiner als } \tau_{\text{low}} \text{ ist,} \\ \lambda(t), & \text{wenn keine der oberen Bedingungen eingetreten ist.} \end{cases} \tag{9}$$

Die Faktoren β_1 und β_2 dienen als Skalierungsparameter zur Steuerung der Anpassungsrate des Strafgewichts $\lambda(t)$. Während $\beta_1 > 1$ die Verringerung der Strafstärke bewirkt, wenn ein ausreichend hoher Anteil zulässiger Individuen vorliegt, wird $\beta_2 > 1$ genutzt, um die Strafstärke zu erhöhen, falls die Population zu viele unzulässige Individuen enthält. Je größer die gewählten Werte von β_1 und β_2, desto stärker fällt die Anpassung von $\lambda(t)$ zwischen zwei Generationen aus. Damit bilden die Faktoren ein Regulationsinstrument, dass das Verhältnis zwischen der Suche in zulässigen und unzulässigen Regionen beeinflusst. Zu große Werte können jedoch zu Instabilität im Suchprozess führen, während zu

kleine Werte den Anpassungseffekt abschwächen (Coello Coello, 2002; Tessema und Yen, 2006). Diese adaptiven Verfahren gelten als robust gegenüber Parametrierungsfehlern und haben sich insbesondere bei komplexen Constraint-Systemen, wie sie in realitätsnahen Optimierungs- und Planungsproblemen auftreten, bewährt (Coello Coello, 2002).

2.5 LLMs

Große Sprachmodelle (LLMs) sind eine Klasse künstlicher neuronaler Netze, die auf umfangreichen Textkorpora trainiert werden, um menschenähnliche Sprache zu erzeugen (Minaee et al., 2025). Sie dienen als Basismodelle, sind also Modelle, die zunächst auf einer breiten Datenbasis trainiert und später an spezifische Aufgaben angepasst werden (Mienye et al., 2025). Im Gegensatz zu traditionellen aufgabenspezifischen Modellen können LLMs eine breite Palette von Sprachaufgaben ohne aufgabenspezifisches Training durchführen, insbesondere wenn sie durch entsprechende Aufforderungen angeleitet werden (Amatriain et al., 2024). Dieses Kapitel bietet einen technischen Überblick über die Funktionalität von LLMs, die Prinzipien der Prompttechnik und eine Analyse der Zuverlässigkeit und Schwächen von LLMs. Das Verständnis dieser Aspekte ist entscheidend für die Integration und Verwendung von LLMs in Systemen zur Lösung komplexer Optimierungsprobleme, wie in dieser Arbeit vorgesehen.

2.5.1 Funktionsweise

LLMs bestehen im Wesentlichen aus drei Komponenten: einer Modellarchitektur, großen Mengen an Trainingsdaten und einem darauf aufbauenden Lernprozess. Für das Training werden meist unstrukturierte Textdaten in Milliardenhöhe verwendet. Diese Daten sind nicht manuell annotiert. Das Training erfolgt daher im sogenannten unsupervised learning-Modus, also ohne explizite Zielvorgaben oder externe Korrektur. Später verwendete Modelle sind pre-trained, also bereits trainiert und werden dann in verschiedenster Form verwendet (Minaee et al., 2025).

Ziel des Trainings ist die Vorhersage des nächsten Wortes innerhalb eines gegebenen Textkontexts (Minaee et al., 2025). Bei einem Eingabetext wie „The sky is ...“ soll das Modell etwa „blue“ vorhersagen. Zu Beginn sind die Modellgewichte zufällig. Das Modell generiert daher zunächst wenig sinnvolle Vorschläge, z. B. „Home“. Durch den Abgleich mit den tatsächlichen Textverläufen und die anschließende Optimierung der Gewichte verbessert sich die Vorhersagegenauigkeit schrittweise (Vaswani et al., 2017).

Als Architektur kommt bei modernen LLMs meist der sogenannte Transformer zum Einsatz. Dieses Modell wurde von Vaswani et al., 2017 eingeführt und hat sich als Standard für Aufgaben der Sprachverarbeitung etabliert (Amatriain et al., 2024). Die Architektur erlaubt es, kontextuelle Zusammenhänge über große Textbereiche hinweg effizient zu erfassen. Das ist zentral für komplexe Anwendungen wie die Interpretation textbasierter Optimierungsprobleme.

Die Verarbeitung eines Eingabetexts durch ein Transformer-basiertes Sprachmodell beginnt mit der sogenannten Tokenisierung. Dabei wird der Text in eine Sequenz von Token überführt, die in der Regel aus einzelnen Wörtern oder Wortteilen bestehen. Dies ermöglicht eine flexible Handhabung unbekannter oder zusammengesetzter Wörter und wird typischerweise durch Verfahren wie Byte Pair Encoding (BPE) umgesetzt (Amatriain et al., 2024). Aus dem Satz *„Data visualization empowers users to...“* entstehen auf diese Weise beispielsweise die Token `[Data, visualization, em, powers, users, to]`.

Im Anschluss werden diese Token in numerische IDs umgewandelt, die auf einem vordefinierten Vokabular basieren. So könnte etwa `Data` der ID `3145`, `visualization` der ID `8123` und `powers` der ID `4321` zugeordnet sein (Cho et al., 2024). Um die Reihenfolge der Token im Satz abzubilden, wird zu jeder Token-ID eine Positionscodierung addiert. Dadurch entsteht ein sogenanntes Token-Embedding, das sowohl semantische Informationen als auch Positionsinformationen beinhaltet (Vaswani et al., 2017).

Für das erste Token ergibt sich beispielsweise:

$$\text{Embedding} = ID(\texttt{3145}) + \text{PositionEncoding}(0) \tag{10}$$

Diese Embeddings bilden den eigentlichen Input für das Modell und werden nacheinander durch mehrere identisch strukturierte Transformatorschichten geleitet. Jede dieser Schichten besteht im Wesentlichen aus zwei aufeinanderfolgenden Komponenten: einer mehrköpfigen Selbstaufmerksamkeit und einem Feedforward-Netzwerk (Cho et al., 2024).

Der Mechanismus der Multi-Head Self-Attention erlaubt es dem Modell, Beziehungen zwischen unterschiedlichen Positionen innerhalb der Eingabesequenz zu erkennen (Vaswani et al., 2017). So kann das Token `em` beispielsweise den Kontext von `visualization` oder `users` berücksichtigen. Jeder Aufmerksamkeitskopf (Head) lernt dabei eigenständig, welche anderen Tokens für ein bestimmtes Token besonders relevant sind. Dies geschieht durch die Berechnung gewichteter Mittelwerte, der sogenannten Value-Vektoren, auf Basis von Abgleichungen zwischen Query- und Key-Vektoren (Vaswani et al., 2017):

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^\top}{\sqrt{d_k}}\right)V \tag{11}$$

Im genannten Beispiel könnte ein Aufmerksamkeitskopf etwa lernen, dass `powers` stark mit `users` korreliert, während ein anderer Kopf die semantische Verbindung zwischen `Data` und `visualization` modelliert.

Nach dem Selbstaufmerksamkeitsmechanismus wird die Repräsentation jedes Tokens durch ein positionsunabhängiges Feedforward-Netzwerk weiterverarbeitet. Dieses Netzwerk besteht aus zwei vollständig verbundenen Schichten mit einer nichtlinearen Aktivierungsfunktion dazwischen (Vaswani et al., 2017; Wolf et al., 2020):

$$\text{FFN}(x) = \max(0, xW_1 + b_1)W_2 + b_2 \tag{12}$$

Um die Stabilität während des Trainings zu erhöhen, werden sowohl die Selbstaufmerksamkeit als auch das Feedforward-Netzwerk jeweils mit einer Residualverbindung (auch Skip-Verbindungen genannt) und einer Layer-Normalisierung kombiniert. Diese architektonischen Komponenten tragen wesentlich zur Konvergenzgeschwindigkeit und Generalisierungsfähigkeit des Modells bei (Ba et al., 2016; Vaswani et al., 2017). Die Transformation innerhalb einer Schicht lässt sich somit wie folgt formal beschreiben(Vaswani et al., 2017):

$$\text{Output}_1 = \text{LayerNorm}(x + \text{MultiHeadAttention}(x)) \tag{13}$$

$$\text{Output}_2 = \text{LayerNorm}(\text{Output}_1 + \text{FFN}(\text{Output}_1)) \tag{14}$$

Nach dem Durchlaufen mehrerer dieser Schichten liegt für jedes Token eine kontextualisierte Repräsentation vor, die sowohl semantische als auch syntaktische Abhängigkeiten aus dem gesamten Satz berücksichtigt. So enthält beispielsweise die finale Repräsentation von `Data` nun nicht nur Informationen über das Wort selbst, sondern auch über dessen Beziehung zu `visualization` und `empowers`.

Im letzten Schritt werden diese Repräsentationen über einen linearen Projektionslayer auf den Vokabularraum abgebildet. Durch Anwendung einer Softmax-Funktion kann das Modell Wahrscheinlichkeiten für jedes mögliche Folgetoken berechnen (Vaswani et al., 2017):

$$P(w_i) = \frac{\exp(\mathbf{h}_i^\top \mathbf{W}_{\text{vocab}})}{\sum_j \exp(\mathbf{h}_i^\top \mathbf{W}_{\text{vocab}}^{(j)})} \tag{15}$$

Hierbei steht $\mathbf{h}_i$ für die finale Repräsentation des i-ten Tokens und $\mathbf{W}_{\text{vocab}}$ für die Gewichtsmatrix des Output-Vokabulars. Dieses Verfahren bildet die Grundlage für die Textgenerierung, wie sie in GPT, LLaMA oder BERT-Modellen umgesetzt wird (Radford et al., 2019). Dabei wird ein Token nach dem anderen erzeugt, indem jeweils eine Liste an wahrscheinlichen Folgetoken erzeugt und einer daraus gewählt wird. Für das oben genannte Beispiel könnte so die Fortsetzung `Data visualization empowers users to create` generiert werden[5].

2.5.2 Prompt Engineering

Prompt Engineering bezeichnet die gezielte Gestaltung und Optimierung von Eingabetexten, die ein Large Language Model zu einer bestimmten Ausgabe veranlassen sollen (Amatriain, 2024; Schulhoff et al., 2025). Da moderne LLMs in der Inferenzphase (in diesem Fall der Verwendung) mit fixierten Modellgewichten arbeiten, fungiert der Prompt als zentrale Schnittstelle zwischen menschlicher Intention und maschineller Generierung (Sahoo et al., 2025; Vatsal und Dubey, 2024). Die Qualität dieses Prompts beeinflusst maßgeblich die Genauigkeit, Interpretierbarkeit und Konsistenz der Ergebnisse (Chen et al., 2025; Zhou et al., 2023). Dies gilt auch für hybride Systeme, wie in diesem Anwendungsfall, in dem das LLM sowohl natürlichsprachliche Nebenbedingungen interpretieren, als auch strukturierte Ausgabeformate erzeugen muss.

[5] Eine grafische Repräsentation von Cho et al., 2024 über dieses Verfahrens lässt sich unter folgender Webadresse finden: https://poloclub.github.io/transformer-explainer/

Während frühe Arbeiten das Erstellen von Prompts primär als eine kreative Aufgabe beschrieben, wurde in der jüngeren Forschung eine systematische Einordnung entwickelt. Vatsal und Dubey, 2024 sowie Sahoo et al., 2025 unterscheiden moderne Prompting-Strategien anhand dreier Kriterien: dem Format der Information (z.B. natürliche Sprache oder eingebettete Vektoren), der Granularität des Kontexts (z.B. Einzelsätze oder längere Beispiele) und der Steuerung des Denkprozesses (implizit vs. explizit).

Manuell gestaltete bzw. textuelle Prompts lassen sich in verschiedene Kategorien unterteilen, die folgend vorgestellt werden. Die erste ist das Zero-Shot Prompting. Hierbei reicht eine einfache Anweisung aus, um das Modell auf eine Aufgabe zu fokussieren (vgl. Abbildung 2).

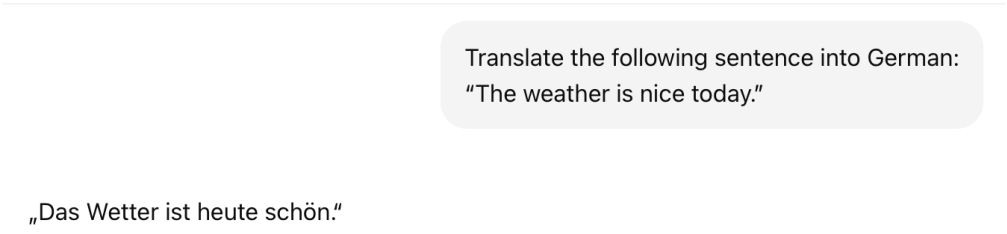

Abbildung 2: Beispielhafter Aufruf Zero-Shot Prompting (Screenshot ChatGPT)

Beim Few-Shot Prompting werden zusätzlich Input-Output-Beispiele angegeben, die das gewünschte Format und Verhalten verdeutlichen (vgl. Abbildung 3).

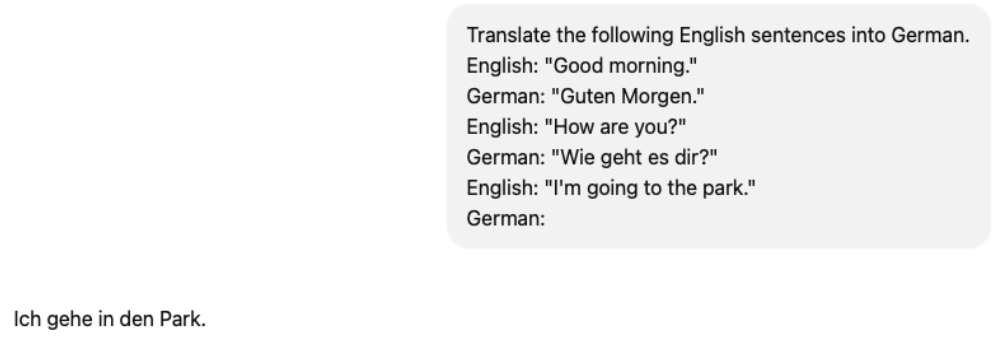

Abbildung 3: Beispielhafter Aufruf Few-Shot Prompting (Screenshot ChatGPT)

Eine Erweiterung stellt das sogenannte Chain-of-Thought (CoT) Prompting dar, bei dem beispielhafte Zwischenüberlegungen explizit im Prompt enthalten sind, um logische Schlussfolgerungen zu fördern. Wei et al., 2023 konnten zeigen, dass dadurch vor allem bei

symbolischen oder arithmetischen Aufgaben signifikante Leistungssteigerungen möglich sind (Beispiel in Abbildung 4).

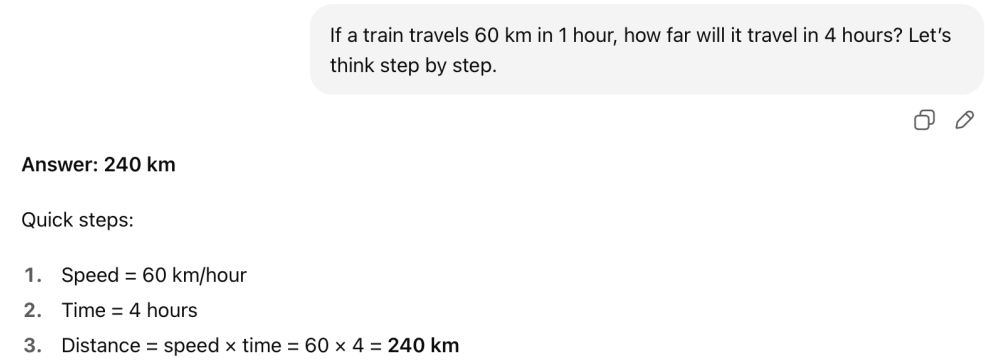

Abbildung 4: Beispielhafter Aufruf CoT Prompting (Screenshot ChatGPT)

In Kombination mit Self-Consistency (X. Wang et al., 2023), bei dem mehrere Antwortpfade generiert und die häufigste Lösung gewählt wird, lassen sich zudem Fehler durch Halluzinationen (vgl. Kapitel 2.5.3) oder inkonsistente Ausgaben reduzieren.

Grenzen der manuellen Gestaltung zeigen sich allerdings bei komplexen Aufgabenstellungen mit vielen Bedingungen – wie im Timetabling. Hier bietet sich der Einsatz automatischer Methoden an. Automatic Prompt Engineering (APE) ist einer dieser Methoden. Es formuliert den Prompt als optimierbares Objekt. In einem iterativen Verfahren werden verschiedene Prompts erzeugt, anhand einer Validierungsmenge bewertet und weiterentwickelt. Zhou et al., 2023 zeigten, dass dieses Verfahren in der Lage ist, manuell erstellte Prompts in vielen NLP-Aufgaben zu übertreffen, ohne Zugriff auf Gradienten des Zielmodells zu benötigen. Hsieh et al., 2023 erweiterten diesen Ansatz um evolutionäre Operatoren und Suchstrategien, welche die Erzeugung und Bewertung längerer, strukturierter Prompts ermöglichen.

Trotz dieser Fortschritte bestehen weiterhin Einschränkungen. Prompts reagieren oft empfindlich auf kleine Änderungen im Wortlaut oder in der Reihenfolge, was zu deutlichen Schwankungen in der Modellleistung führen kann (Schulhoff et al., 2025; Zhou et al., 2023). Zudem steigen die Rechenkosten mit der Länge des Prompts überproportional an, da sowohl die Anzahl der zu verarbeitenden Token als auch der Suchraum für mögliche Eingabevarianten exponentiell zunimmt (Hsieh et al., 2023). Eine zuverlässige Bewertung von Prompting-Strategien, insbesondere unter Aspekten wie Fairness, Robustheit

und Übertragbarkeit, bleibt bislang ungelöst (Chen et al., 2025; Sahoo et al., 2025; Schulhoff et al., 2025). Dadurch ist eine Bewertung, welche Strategie in welchem fachlichen Kontext optimal ist, kaum möglich.

2.5.3 Verlässlichkeit von Resultaten

LLMs werden zunehmend für evaluative Aufgaben eingesetzt. Dazu zählt etwa die Überprüfung, ob strukturierte Daten komplexen Anforderungen in natürlicher Sprache genügen. Diese Nutzung wirft zentrale Fragen zur Zuverlässigkeit und Konsistenz der Modelle auf. Das gilt insbesondere dann, wenn strukturierte Formate (z.B. JSON) mit offen formulierten Regeln kombiniert werden (Bubeck et al., 2023).

Während moderne LLMs in der Verarbeitung natürlicher Sprache große Fortschritte zeigen, bestehen weiterhin Schwächen bei der Anwendung formaler Logik und der konsistenten Umsetzung mehrstufiger Regeln (Creswell et al., 2022; Holtzman et al., 2020). Szeider, 2024 zeigt, dass Modelle in manchen Fällen Anforderungen als erfüllt bewerten, obwohl diese nicht in den strukturierten Daten belegt sind. Dieses Verhalten wird unter anderem auf das sogenannte „Halluzinieren“ zurückgeführt – die Tendenz, plausibel klingende, aber faktisch unzutreffende Aussagen zu generieren (Ji et al., 2023). Hinzu kommt, wie zuvor aufgeführt, ein Mangel an Konsistenz. Schon kleinere Variationen in der Formulierung eines Prompts können zu unterschiedlichen Bewertungen führen, was die Reproduzierbarkeit einschränkt (Szeider, 2024).

Die Herausforderung verschärft sich, wenn Anforderungen aus mehreren teils konkurrierenden Regeln bestehen. LLMs generieren ihre Ausgaben nicht durch systematische Exploration, sondern auf Basis erlernter Muster (F. Wang et al., 2024). Daraus ergibt sich das Risiko, dass Randfälle nicht erkannt oder Zielkonflikte nicht korrekt aufgelöst werden. Selbst feinjustierte Modelle zeigen in empirischen Untersuchungen regelmäßig Regelverletzungen, insbesondere bei wachsender Komplexität der Aufgabe (Jobson und Li, 2024; Szeider, 2024). Die Neigung zu affirmativen oder „wohlklingenden“ Antworten kann dabei zusätzlich zu falsch-positiven Bewertungen führen (Madmoni et al., 2024).

Weitere bekannte Schwächen betreffen kognitive Verzerrungen (Bias) und Bewertungsskalen. Modelle zeigen etwa Familiarity Bias. Gemeint ist damit eine Präferenz für vertraut wirkende Muster, sowie eine ungleichmäßige Nutzung von Skalenwerten (Stureborg et al., 2024). So tendieren LLMs bei einer Bewertung auf einer 1–100-Skala zu runden Zahlen und vermeiden niedrige Werte, wodurch die Aussagekraft solcher Bewertungen

eingeschränkt ist (Jobson und Li, 2024). Zudem besteht geringe Übereinstimmung der Modelle mit sich selbst („low self-consistency“). Ergebnisse können sich bei identischen Aufgaben mit minimalen oder auch keinen Promptänderungen signifikant unterscheiden (Megahed et al., 2025). Gleichzeitig ist die Entscheidungslogik von LLMs häufig intransparent. Generierte Begründungen sind oft nicht mit der tatsächlichen internen Bewertung verknüpft und können selbst fehlerhaft sein (P. Wang et al., 2023).

In der Anwendung auf strukturierte Entscheidungsprozesse zeigen sich Stärken in der semantischen Interpretation sprachlicher Anforderungen sowie bei der Generierung erster Bewertungsvorschläge. Diese Vorschläge können als Ausgangspunkt für weitere Analysen dienen, sind jedoch selten formal korrekt oder vollständig (Jobson und Li, 2024). Sowohl Szeider, 2024, als auch Jobson und Li, 2024 zeigen, dass LLMs zwar problemangemessene Bewertungen erzeugen können, dabei aber häufig Regeln verletzen oder wichtige Details übersehen. Ein vielversprechender Ansatz besteht darin, LLMs mit formalen Verifikationssystemen zu kombinieren. In solchen hybriden Architekturen wird das Modell zur sprachlichen Erfassung und Strukturierung eingesetzt, während ein dediziertes System die formale Validierung übernimmt (Szeider, 2024). Hierdurch lassen sich verlässlichere Systeme erzeugen.

Allgemein gesagt, bestehen also begründete Vorbehalte gegen die Resultate von LLMs. Dies ist insbesondere in dem Kontext von hybriden Systemen relevant, wo das Resultat der Verarbeitung durch ein LLM in einem statischen System weiterverwendet wird. Hier können dadurch diverse Risiken und Fehlinterpretationen entstehen.

3 Experimentalumgebung

Das folgende Kapitel beschreibt die experimentelle Umgebung, in der die in Kapitel 2 vorgestellten theoretischen Konzepte praktisch umgesetzt und evaluiert werden. Es bildet damit die Brücke zwischen den theoretischen Grundlagen und der empirischen Untersuchung der entwickelten Verfahren. Ziel ist es, die Rahmenbedingungen, Anwendungsfälle, Datenstrukturen und Implementierungen so präzise zu beschreiben, dass alle im weiteren Verlauf präsentierten Ergebnisse reproduzierbar und nachvollziehbar sind.

Zunächst werden in Abschnitt 3.1 die verschiedenen Anwendungsfälle vorgestellt, die als Grundlage für die Experimente dienen. Diese umfassen unterschiedliche Varianten von Scheduling-Problemen, die sich zwar in ihrem fachlichen Kontext unterscheiden, strukturell jedoch ein gemeinsames Muster aufweisen: Sie sind alle typische Probleme des Timetabling Problems (Kapitel 2.1) und müssen über die Zuordnung zweier Ressourcentypen unter Berücksichtigung harter und weicher Nebenbedingungen gelöst werden. Die Szenarien decken dabei sowohl klassische Bildungs- als auch industrielle und logistische Planungsdomänen ab (z. B. Schul-, Zug- und Maschinenbelegungsplanung). Durch die Auswahl dieser unterschiedlichen, aber formal verwandten Problemstellungen kann überprüft werden, inwieweit die entwickelten Systeme domänenunabhängig anwendbar sind.

Darauf aufbauend werden in Abschnitt 3.2 die verschiedenen Implementierungen beschrieben, die im Rahmen dieser Arbeit entwickelt und getestet wurden. Jede dieser Implementierungen repräsentiert einen unterschiedlichen methodischen Ansatz zur Generierung und Optimierung von Zeitplänen. Die vier untersuchten Varianten – Prompting, Evolution of Heuristics (EoH), Evolution of Schedules (EoS) und ein hybrider Evolutionärer Algorithmus (EA) – unterscheiden sich insbesondere hinsichtlich des Grades der LLM-Integration, des Automatisierungsniveaus und der Art der Suchraumexploration. Hierbei wird systematisch erläutert, wie die Verfahren technisch umgesetzt wurden, welche Architekturen ihnen zugrunde liegen und welche Eingabe- und Ausgabestrukturen verwendet werden. Die konkreten Implementierungen sind auf GitHub bereitgestellt. Die dazugehörigen Verlinkungen finden sich im Anhang A.3 wieder.

Zusätzlich werden die verwendeten Datenmodelle und Validierungsmechanismen vorgestellt, die eine einheitliche Vergleichbarkeit der Ansätze gewährleisten. Ein besonderes Augenmerk liegt auf der Definition der Eingabedaten, welche die Struktur der informationsarmen (IA) und informationsreichen (IR) Seiten sowie die globalen Constraints und

Bewertungsmetriken spezifizieren. Diese Elemente bilden die Grundlage für die Kommunikation zwischen den Systemkomponenten und dem eingebetteten LLM.

3.1 Anwendungsfälle

Zur Beurteilung der Leistungsfähigkeit und Generalisierbarkeit der entwickelten Verfahren wurden mehrere Anwendungsfälle definiert, die unterschiedliche Arten von Timetabling-Problemen repräsentieren. Alle Szenarien folgen demselben strukturellen Grundprinzip von Timetabling-Problemen (vgl. Kapitel 2.1) Sie beinhalten die zeitliche und ressourcenbezogene Zuordnung einer Menge von Aufgaben oder Ereignissen unter Beachtung definierter harter und weicher Constraints. Trotz dieser gemeinsamen Grundstruktur unterscheiden sich die Anwendungsfälle in ihrem Kontext, ihren Randbedingungen sowie in der Art und Anzahl der beteiligten Ressourcen. Insbesondere das Optimierungsobjekt, also die Variable, welche in dem jeweiligen Kontext verschiedene Werte annehmen kann, um ein optimiertes Ergebnis zu erzielen, unterscheidet sich in allen Kontexten voneinander. Die Probleme sind domänenspezifische Varianten des Timetablings, um die Übertragbarkeit des entwickelten Systems auf unterschiedliche Problemtypen zu prüfen.

3.1.1 Allgemeine Beschreibung

Die in dieser Arbeit definierten Anwendungsfälle bilden die Grundlage für die empirische Evaluation der entwickelten Optimierungsverfahren. Sie wurden so konzipiert, dass sie einerseits realitätsnahe Anforderungen aus unterschiedlichen Anwendungsdomänen widerspiegeln und andererseits auf einer gemeinsamen abstrakten Struktur basieren. Diese Abstraktion gewährleistet, dass alle Verfahren nach demselben formalen Schema beschrieben und verarbeitet werden können.

Formell lässt sich jedes Szenario durch eine gemeinsame Problemdefinition darstellen:

$$P = (T, R_L, R_R, S, C)$$

mit den folgenden Bestandteilen:

- T: Menge der verfügbaren Zeitslots oder Zeiteinheiten,
- R_L: Menge der linken bzw. informationsreichen Ressourcen (z. B. Klassen, Züge, Maschinen),

- R_R: Menge der rechten bzw. informationsarmen Ressourcen (z. B. Lehrkräfte, Zugpersonal, Aufträge),
- S: Menge der möglichen Zuordnungen zwischen R_L und R_R innerhalb eines Zeitraums T,
- C: Menge der Constraints, unterteilt in harte (C_h) und weiche (C_s) Bedingungen.

Das Ziel der Optimierung besteht darin, eine Abbildung

$$f : (R_L \times R_R \times T) \rightarrow S$$

zu finden, die sämtliche harte Constraints erfüllt und gleichzeitig die Summe der Verletzungen weicher Constraints minimiert. Diese allgemeine Struktur ist domänenübergreifend anwendbar und dient als konzeptionelle Grundlage für alle im weiteren Verlauf beschriebenen Planungsprobleme.

Für die Implementierung der Experimente werden die Szenarien in einem generischen, JSON-basierten Format abgebildet. Jede Domäne folgt dabei einem einheitlichen Schema, das aus Entitäten, Attributen und Constraints besteht. Ein vereinfachtes Beispiel eines Eingabeobjekts lautet:

```
{
    "left_id": {
        "Monday": {
            "1": {
                "right_id": 0
            },
            "2": {
                "right_id": 2
            }
        },
        "Tuesday": {
            "2": {
                "right_id": 1
            }
        }
    }
}
```

JSON-Darstellung 1: Allgemeine Beschreibung eines left objects

Dieses Schema erlaubt eine flexible Erweiterung um zusätzliche Informationen oder die Reduktion einzelner Ebenen, ohne dass der zugrunde liegende Algorithmus angepasst werden muss. Entscheidend ist die Abbildung zwischen `left_id` und `right_id`. Dabei repräsentiert `left_id` stets die übergeordnete, informationsreiche Instanz (IR), während `right_id` als informationsarme Instanz (IA) betrachtet wird. Die Bedingungen der Verbindungen zwischen beiden Ressourcen werden in der IR-Instanz definiert.

Wie bereits im Kapitel 2.4.2 erläutert, werden auch in diesen Szenarien Constraints in harte und weiche Bedingungen unterteilt. Beispiele hierfür sind das Verbot der mehrfachen Belegung einer Ressource (harter Constraint) sowie die Minimierung von Leerlaufzeiten (weicher Constraint).

Zur Bewertung der erzeugten Zeitpläne kommt die in Kapitel 2.4.2 beschriebene *Strafterm-Methode* (Penalized Cost Functions) zum Einsatz. Sie wird in diesem Kontext in vereinfachter Form verwendet, da auf eine Gewichtung innerhalb der Gruppen harter bzw. weicher Constraints verzichtet wird. Die Fitness eines Zeitplans x_i ergibt sich somit zu:

$$f(x_i) = \alpha \sum_{c_h \in C_H} v(c_h, x_i) + \beta \sum_{c_s \in C_S} v(c_s, x_i), \tag{16}$$

wobei $v(c_h, x_i)$ und $v(c_s, x_i)$ die berechneten Verletzungswerte je Constraint darstellen. Durch die Gewichtungsfaktoren $\alpha \gg \beta$ wird sichergestellt, dass harte Verstöße deutlich stärker penalisiert werden als weiche.

Die abstrakte Formulierung dieses Rahmens ermöglicht es, verschiedene Szenarien in einem gemeinsamen Algorithmus abzubilden. Während sich die inhaltliche Ausgestaltung durch domänenspezifische Erweiterungen unterscheidet, bleibt die strukturelle Grundlage identisch. Dadurch können sowohl unterschiedliche Varianten evolutionärer Algorithmen konsistent verglichen als auch neue Problemstellungen mit minimalem Anpassungsaufwand integriert werden.

Der allgemeine Rahmen erfüllt somit zwei zentrale Ziele:

1. **Vergleichbarkeit:** Alle Optimierungsverfahren greifen auf identische Datenstrukturen, Constraints und Bewertungsmethoden zurück. Unterschiede in den Ergeb-

nissen lassen sich dadurch ausschließlich auf die jeweilige Implementierung zurückführen und nicht auf Unterschiede in den Eingangsdaten oder der Domäne.

2. **Adaptivität:** Die Abstraktion gewährleistet, dass verschiedene Implementierungen auf unterschiedliche Anwendungsfälle übertragbar sind, ohne dass das zugrunde liegende Szenario verändert werden muss. Domänenspezifische Besonderheiten werden lediglich ergänzt oder modifiziert, nicht jedoch in der Struktur des Problems angepasst.

3.1.2 Schulplanung

Das Szenario der Schulplanung bildet einen klassischen Vertreter der Timetabling-Probleme im Bildungsbereich. Ziel ist es, für jede Klasse einer Schule einen Stundenplan zu erzeugen, der sämtliche fachliche, personelle und zeitliche Anforderungen erfüllt. Die Problemstellung lässt sich formal als Zuordnungsproblem zwischen den informationsreichen Ressourcen (IR) – den Klassen – und den informationsarmen Ressourcen (IA) – den Lehrkräften – modellieren. Die Abbildung erfolgt über ein Mapping, das definiert, welche Lehrkraft ein bestimmtes Fach in einer Klasse unterrichtet. Die zu lösende Fragestellung ist, wie diese Mappings auf die Zeiträume verteilt werden können.

Analog zur allgemeinen Problemdefinition (Kapitel 3.1.1) werden die Eingabedaten im JSON-Format bereitgestellt. Jede Klasse enthält eine Liste ihrer zu unterrichtenden Fächer, der erforderlichen Wochenstunden und der jeweiligen Lehrkraft, die das Fach unterrichtet. Zusätzlich ist für jede Klasse definiert, welche Zeitpunkte es für den Unterricht gibt, was also der verfügbare Rahmen für die Unterrichtsstunden ist. Für die Lehrkräfte werden die individuellen Verfügbarkeiten und Constraints (z. B. Teilzeitmodelle oder Präferenzen für bestimmte Wochentage) in den individuellen Anforderungen definiert.

Ein abstrahiertes Beispiel eines IR-Objekts (Klasse) lautet:

```
{
  "id": "1A",
  "time-description": "Monday to Friday the hours 1, 2, 3, 4.",
  "subjects": [
    {
        "subject": "Musik",
        "number": 1,
        "right_id": 100
```

```
    }
  ]
}
```

JSON-Darstellung 2: Eine Klasse als IR-Objekt

Das zugehörige IA-Objekt (Lehrkraft) wird in einer separaten Liste gespeichert und enthält Attribute wie:

```
{
  "id": 100,
  "constraints_list": [
    {"text": "Does not work on Monday", "priority":
       "hard-constraint"},
  ]
}
```

JSON-Darstellung 3: Eine Lehrkraft als IR-Objekt

Das Mapping zwischen Klassen und Lehrkräften erfolgt über das Attribut `right_id` innerhalb des Stundenplans. Die Kombination von `left_id` (Klasse), `right_id` (Lehrkraft) und `subject` bildet eine elementare Einheit im Zeitplan, die wiederum in Zeitslots (z. B. Stunden oder Doppelstunden) eingeordnet wird.

Die globalen Anforderungen, welche vom Algorithmus beachtet werden sind für diese Szenario folgende:

- Eine Lehrkraft darf zu keinem Zeitpunkt zwei Klassen parallel unterrichten (hard Constraint).
- Jede Klasse darf das gleiche Fach maximal zwei Stunden am Tag haben (hard Constraint).
- Klassen dürfen keine Freistunden zwischen den Fächern haben (hard Constraint)

Die durch den Algorithmus erzeugten Zeitpläne werden ebenfalls im JSON-Format erzeugt. Das folgende Beispiel zeigt die Struktur eines Wochenplans für die Klasse `1A`:

```
{
    "1A": {
        "Monday": {
```

```
            "1": {
                "subject": "German",
                "right_id": 100
            },
            "2": {
                "subject": "German",
                "right_id": 101
            }
        },
        "Tuesday": {
            "2": {
                "subject": "German",
                "right_id": 100
            }
        }
    }
}
```

JSON-Darstellung 4: Sochenplan der Klasse 1A

Dieses Format erlaubt es, pro Klasse und Zeitslot die zugewiesene Lehrkraft und das unterrichtete Fach eindeutig zu identifizieren. Auf diesem Ergebnis werden die Anforderungen der Lehrer, Klassen und die globalen Constraints überprüft und daraus die Fitness-Bewertung errechnet.

Das reale Testdatenset basiert auf einer Grundschule in Schleswig-Holstein und wurde zu Forschungszwecken abstrahiert und anonymisiert. Es umfasst die Klassenstufen eins bis vier mit jeweils zwei Parallelklassen sowie das vollständige Kollegium. Dadurch entsteht ein praxisnahes, aber technisch kontrollierbares Szenario, welches in sich geschlossen ist und lösbare Anforderungen enthält.

3.1.3 Zugpersonalplanung

Das Szenario der Zugpersonalplanung bildet das zweite Beispiel für ein domänenspezifisches Timetabling-Problem. Analog zur Schulplanung besteht das Ziel darin, eine konsistente Zuordnung zwischen den informationsreichen Ressourcen (IR) – in diesem Fall

den Zügen – und den informationsarmen Ressourcen (IA) – dem Zugpersonal – zu erzeugen. Dabei muss gewährleistet sein, dass sämtliche zeitliche, räumliche und logische Constraints eingehalten werden.

Jeder Zug besitzt einen definierten Fahrplan, der festlegt, zu welchen Zeiten und zwischen welchen Bahnhöfen er verkehrt. Dieser Plan kann für einen einzelnen Tag, für wiederkehrende Fahrten oder für eine definierte Menge abweichender Tage angegeben sein. Die angegebene Liste an `right_id`s beschreibt, welches Zugpersonal für diese Fahrt potenziell eingesetzt werden könnte. Das folgende Beispiel zeigt ein abstrahiertes IR-Objekt, das eine Zugfahrt beschreibt:

```
{
    "id": "Train-1",
    "time-description": "Monday from 6 to 18",
    "stops": [
        {
            "from": "Hamburg",
            "to": "Berlin",
            "departure": "6",
            "arrival": "8",
            "right_id:" [101, 102, 103]
        },
        {
            "from": "Berlin",
            "to": "Hamburg",
            "departure": "9",
            "arrival": "11",
            "right_id": [101, 102, 103]
        }
    ]
}
```

JSON-Darstellung 5: Zug als IR-Objekt

Das korrespondierende IA-Objekt beschreibt das Zugpersonal, beispielsweise Lokführerinnen und Lokführer, mit ihren individuellen zeitlichen und räumlichen Einschränkungen:

```
{
    "id": 101,
    "constraints_list": [
        {"text": "Has Tuesday off", "priority":
           "hard-constraint"},
        {"text": "Starts and end every day in hamburg",
        "priority": "soft-constraint"}
    ]
}
```

JSON-Darstellung 6: Zugpersonal als IA-Objekt

Die Aufgabe des Algorithmus besteht darin, die gegebenen Zugfahrten mit dem verfügbaren Personal zu verknüpfen, sodass alle operativen und logischen Anforderungen erfüllt werden. Es wird jeweils immer nur eine `right_id` je Fahrt gebraucht.

Die globalen Constraints sind an das Schul-Szenario angelehnt, jedoch domänenspezifisch erweitert:

- Eine Person darf zu keinem Zeitpunkt gleichzeitig mehreren Zügen zugeordnet sein (hard Constraint).
- Ein Wechsel zwischen Zügen ist nur möglich, wenn der neue Zug am gleichen Bahnhof startet, an dem der vorherige endet (hard Constraint).
- Zu jeder Zugfahrt ist genau eine `right_id` zugeordnet.

Das durch den Algorithmus erzeugte Ergebnis wird im gleichen JSON-Format wie bei den anderen Szenarien ausgegeben. Auch hier wird der Plan für die informationsreiche Seite (die Züge) ausgegeben:

```
{
    "Train-1": {
        "Monday": {
            "1": {
                "departure": 6,
                "from": "Hamburg",
                "arrival": 8,
                "to": "Berlin",
```

```
                "right_id": 101,
                "possible_right_id": [101, 102, 103]
            },
            "2": {
                "departure": 9,
                "from": "Berlin",
                "arrival": 11,
                "to": "Hamburg",
                "right_id": 102,
                "possible_right_id": [101, 102, 103]
            }
        }
    }
}
```

JSON-Darstellung 7: Belegungsplan für Zug-1

Dieses Format erlaubt eine eindeutige Identifikation aller Fahrten, inklusive Start- und Zielbahnhof sowie der zugehörigen Personal-IDs. Auf Basis dieses Plans erfolgt die Validierung, ob die Constraints erfüllt sind und ob die zeitliche Abfolge eingehalten wurde.

Der Zweck dieses Szenarios liegt darin, das zuvor beschriebene Schulszenario zu ergänzen und die Übertragbarkeit der entwickelten Verfahren auf weitere Domänen zu demonstrieren. Während im Schulkontext die Aufgabe darin besteht, Unterrichtseinheiten zeitlich zu planen, basiert die Zugpersonalplanung auf einem bereits festen Fahrplan. Die Optimierung betrifft hier also nicht die Erzeugung des Zeitplans selbst, sondern die Zuweisung von Personal zu bestehenden Zugverbindungen. Damit stellt dieses Szenario eine komplementäre Herausforderung dar, bei der die Leistungsfähigkeit der Algorithmen insbesondere in der logischen und zeitlichen Konsistenzbewertung geprüft wird.

3.1.4 Maschinenbelegungsplanung

Das Szenario der Maschinenbelegungsplanung bildet den industriellen Gegenpart zu den zuvor beschriebenen domänenspezifischen Anwendungsfällen und erweitert die experimentelle Umgebung um eine produktionsnahe Problemstellung. Ziel ist die Zuordnung von Arbeitskräften (IR) zu Maschinen (IA) innerhalb eines gegebenen Zeithorizonts,

sodass alle technischen, organisatorischen und personellen Anforderungen eingehalten werden.

Analog zur allgemeinen Problemdefinition (Kapitel 3.1.1) besteht die Aufgabe darin, für jede Maschine eine zeitlich konsistente und ressourcenschonende Belegung zu erzeugen. Die Zuordnung zwischen Maschinen und Mitarbeitenden erfolgt über ein Mapping, das angibt, welche Person(en) zu welchem Zeitpunkt an welcher Maschine tätig ist (sind). Die Unterscheidung dieses Szenarios zu den anderen ist, dass zu jedem Zeitpunkt eine Menge an Mitarbeitenden ≥ 1 eingesetzt wird. Dies bringt eine neue Form der Komplexität für das Programm mit sich, da der Lösungshorizont dadurch exponentiell steigt.

Die Eingabedaten werden im gleichen strukturellen JSON-Format wie bei den anderen Szenarien abgebildet. Eine Maschine enthält ihren individuellen Zeitplan, Angaben zu Schichten sowie die erforderliche Anzahl an gleichzeitig tätigen Mitarbeitenden. Ein Beispiel für die Beschreibung einer Maschine lautet:

```
{
  "id": "Machine-1",
  "time-description": "Monday to Friday, shifts 1 to 3",
  "number-workers": 3,
  "right_id": [11, 12, 13, 14]
}
```

JSON-Darstellung 8: Beispielhafte Maschinenbeschreibung

Die informationsarme Seite beschreibt die verfügbaren Arbeitskräfte inklusive individueller Verfügbarkeiten:

```
{
  "id": 201,
  "constraints_list": [
    {"text": "Works only morning shifts", "priority":
       "hard-constraint"},
    {"text": "Prefers Monday to Thursday", "priority":
       "soft-constraint"}
  ]
}
```

JSON-Darstellung 9: Beispielhafte Beschreibung einer Arbeitskraft

Das vom Algorithmus erzeugte Ergebnis wird ebenfalls im JSON-Format ausgegeben und enthält für jede Maschine eine Zuordnung der Mitarbeitenden zu Zeitpunkten bzw. Schichten:

```
{
  "Machine-1": {
    "Monday": {
      "1": {
        "right_id": [201, 202],
        "possible_right_id": [201, 202, 203]
      },
      "2": {
        "right_id": [204, 205],
        "possible_right_id": [201, 202, 203]
      }
    },
    "Tuesday": {
      "1": {
        "right_id": [202, 204],
         "possible_right_id": [201, 202, 203]
      }
    }
  }
}
```

JSON-Darstellung 10: Beispielhafte Ausgabe eines Maschinenbelegungsplans

Dieses Format erlaubt die eindeutige Zuordnung mehrerer Arbeitskräfte zu einem Zeitslot pro Maschine. Zusätzlich werden potenziell zulässige Alternativen über das Feld `possible_right_id` abgebildet, um spätere Mutations- und Optimierungsschritte innerhalb des evolutionären Algorithmus zu erleichtern.

Zur Sicherstellung der Realitätsnähe gelten eine Reihe domänenübergreifender und spezifischer Constraints:

- Eine Arbeitskraft darf zu keinem Zeitpunkt an mehreren Maschinen gleichzeitig eingeplant sein.

- Die maximale wöchentliche Arbeitszeit von fünf Schichten je Person darf nicht überschritten werden.
- Die Arbeitskräfte müssen nach jeder Schicht mindestens zwei aufeinanderfolgende Schichten freihaben.

Die Validierung der Maschinenbelegungspläne erfolgt analog zu den zuvor in den anderen Szenarien beschriebenen Prüfmechanismen. Dabei werden pro Plan sowohl harte als auch weiche Constraints quantitativ ausgewertet. Verstöße gegen harte Bedingungen führen zu einer starken Verschlechterung eines Planes, während Verstöße gegen softe Constraints diesen weniger negativ bewerten.

Dieses Szenario dient der Demonstration der Übertragbarkeit der entwickelten Optimierungsverfahren auf industriell geprägte Planungsprobleme. Während in der Schulplanung die Unterrichtsverteilung und in der Zugpersonalplanung die logische Abfolge von Einsätzen im Vordergrund steht, liegt der Fokus hier auf Kapazitäts- und Ressourcenkonsistenz. Durch die Mehrfachzuweisung von Arbeitskräften pro Maschine entsteht eine erhöhte kombinatorische Komplexität, welche die Robustheit und Adaptivität der getesteten Verfahren weiter herausfordert.

3.2 Vorstellung der Implementierungsoptionen

Dieses Kapitel führt in die verschiedenen Implementierungsansätze ein, die zur technischen Umsetzung der zuvor beschriebenen Anwendungsfälle verwendet werden. Ziel ist es, einen strukturierten Überblick über die zugrunde liegenden Konzepte und architektonischen Unterschiede zu geben, bevor die jeweiligen Verfahren in den folgenden Unterkapiteln detailliert beschrieben werden.

Alle Ansätze basieren auf einem gemeinsamen abstrakten Datenmodell, das Ressourcen, Zeitslots und Constraints in einer einheitlichen JSON-Struktur abbildet (vgl. Kapitel 3.1.1). Dadurch wird eine konsistente Vergleichbarkeit zwischen den unterschiedlichen Implementierungen sichergestellt. Die Verfahren unterscheiden sich im Wesentlichen darin, in welchem Umfang und auf welche Weise LLMs in den Optimierungsprozess integriert werden und welche Rolle sie im jeweiligen Lösungszyklus übernehmen.

Im Rahmen dieser Arbeit werden vier Implementierungsoptionen betrachtet, die sich entlang des Grades der LLM-Integration und ihrer funktionalen Einbettung unterscheiden:

- **Prompting:** Ein direkter, generativer Ansatz, bei dem das LLM auf Basis einer natürlichsprachlichen Problembeschreibung vollständige Lösungsvorschläge generiert.
- **Evolution of Schedules (EoS):** Ein hybrides Verfahren, das LLMs zur Erzeugung und Bewertung kompletter Zeitpläne innerhalb eines evolutionären Algorithmus einsetzt.
- **Evolution of Heuristics (EoH):** Eine erweiterte Variante des EoS, bei der das LLM nicht direkt Zeitpläne generiert, sondern heuristische Strategien entwirft und adaptiv anpasst.
- **Evolutionärer Algorithmus (EA):** Eine klassische Implementierung evolutionärer Verfahren, bei der das LLM lediglich zur einmaligen Generierung spezifischer Codebestandteile verwendet wird.

Zur Sicherstellung einer objektiven Vergleichbarkeit zwischen den generierten Ergebnissen wurden zudem Validierungsskripte für jedes Szenario entwickelt. Diese prüfen anhand der jeweiligen Constraint-Definitionen, ob und an welchen Zeitpunkten Abweichungen auftreten. Die Skripte selbst sind nicht Teil der Optimierungsverfahren, sondern dienen ausschließlich der nachträglichen Validierung der erzeugten Pläne. Jede Überprüfung ist als eigenständige Methode implementiert und überprüft einen definierten Sachverhalt, wodurch die Anzahl und Art der Constraint-Verletzungen als quantitative Bewertungsgröße des Ergebnisses herangezogen werden kann.

3.2.1 Prompting

Das *Prompting*-Verfahren stellt die konzeptionell einfachste Implementierungsvariante dar und dient als Referenz für die Beurteilung der nachfolgenden hybriden und evolutionären Ansätze. Im Zentrum steht die Untersuchung, inwieweit LLMs in der Lage sind, kombinatorische Planungsprobleme allein auf Basis natürlichsprachlicher Instruktionen zu lösen. Die Methode basiert auf einer einzelnen textuellen Eingabeaufforderung, welche das gesamte Problem in strukturierter Form beschreibt. Das LLM wird dabei angewiesen, eine vollständige Lösung in einem vorgegebenen JSON-Format zu erzeugen, die alle relevanten Ressourcen, Zeiträume und Zuordnungen abbildet. Der Ansatz folgt einem

Zero-Shot-Prinzip (vgl. Kapitel 3.2.1), bei dem keine zusätzlichen Trainingsdaten, vollständige Beispiele oder iterative Korrekturen eingesetzt werden. Das Modell arbeitet somit vollständig autonom, auf Grundlage der bereitgestellten semantischen Beschreibung des Szenarios.

Der generierte Prompt ist so konzipiert, dass er für unterschiedliche Anwendungsfälle verwendet werden kann. Dazu zählen Schulplanungs-, Zug- und Maschinenbelegungsszenarien, die alle auf demselben abstrakten Datenmodell (vgl. Kapitel 3.1.1) basieren. Dieses Modell beschreibt eine Menge von linken und rechten Ressourcen (R_L, R_R), eine Menge von Zeitslots (T) sowie die zugehörigen Constraints (C), welche die Regeln der Zuordnung definieren. In der Promptbeschreibung werden diese Bestandteile in natürlicher Sprache spezifiziert und anschließend um konkrete Rahmenbedingungen erweitert. Dazu gehören beispielsweise Restriktionen hinsichtlich der zeitlichen Verfügbarkeit von Ressourcen, Kapazitätsgrenzen, Reihenfolgenabhängigkeiten oder Ausschlussbedingungen. Die Problemdefinition wird dem LLM in textueller Form übermittelt und durch ein Referenzbeispiel im JSON-Format ergänzt, das die gewünschte Struktur der Lösung vorgibt. Diese Kombination ermöglicht es, dass das Modell sowohl die semantische als auch die formale Ebene der Aufgabenstellung versteht und eine strukturell konsistente Antwort generiert.

Das Ergebnis der Generierung ist ein JSON-Objekt, das eine Zuordnung der Ressourcen über die Zeit hinweg beschreibt. Dieses wird anschließend einer Validierungsstufe unterzogen, in der das ursprünglich verwendete LLM oder ein zweites Modell denselben Prompt zusammen mit der erzeugten Lösung erneut erhält. Ziel dieser zweiten Abfrage ist die autonome Überprüfung der erstellten Lösung anhand der zuvor definierten Constraints. Zusätzlich soll hierüber ermittelt werden, wie gut LLMs in der Lage sind, komplexe Datenstrukturen anhand textueller Anforderungen zu bewerten. Dabei wird geprüft, ob die im Prompt spezifizierten Bedingungen verletzt wurden und in welchem Ausmaß. Die Bewertung erfolgt nicht nur binär, sondern durch eine aggregierte Fehlerbewertung, die der in Kapitel 3.1.1 beschriebenen Constraint-Bewertung entspricht. Hierzu führt das Modell schrittweise eine interne logische Begründung durch, ähnlich einem *Chain-of-Thought*-Prozess (vgl. Kapitel 3.2.1) und bewertet anschließend die Gesamtlösung auf Basis der gefundenen Abweichungen.

Der Prompting-Ansatz ist die Baseline für die Bewertung der Leistungsfähigkeit von LLMs bei der Lösung und Bewertung strukturierter Optimierungsprobleme. Er ermöglicht eine Einschätzung, in welchem Maße ein Sprachmodell ohne algorithmische Unter-

stützung in der Lage ist, Constraints korrekt zu interpretieren, zeitliche und ressourcenbezogene Abhängigkeiten zu erfassen und konsistente Lösungen zu erzeugen. Die gewonnenen Ergebnisse dienen als Vergleichsmaßstab für die später vorgestellten hybriden Implementierungen, bei denen LLMs in einen evolutionären Optimierungsprozess integriert werden. Durch den Vergleich der Resultate kann bewertet werden, ob die zusätzliche algorithmische Steuerung zu einer signifikanten Verbesserung der Lösungsqualität oder der Constraint-Erfüllung führt.

3.2.2 Evolution of Schedules

Die Evolution of Schedules (EoS) stellt die erste Erweiterung des reinen Prompting-Verfahrens dar und bildet damit den Übergang von einem rein generativen Ansatz hin zu einem iterativ lernenden Optimierungsprozess. Das Ziel des Verfahrens ist es, die generativen Fähigkeiten von LLMs mit der Suchlogik evolutionärer Algorithmen zu kombinieren. Auf diese Weise werden über mehrere Generationen hinweg Pläne erzeugt, bewertet und schrittweise verbessert. Während das einfache Prompting nur einen einzelnen Plan erzeugt, integriert EoS das LLM in einen zyklischen Prozess, der eine fortlaufende Weiterentwicklung der Lösungen ermöglicht.

Das Konzept orientiert sich am Evolution of Heuristics (EoH)-Framework nach Liu et al., 2024. Der entscheidende Unterschied liegt jedoch darin, dass das LLM hier keinen Algorithmuscode oder Heuristikoperatoren generiert, sondern direkt vollständige Pläne als Optimierungsobjekte. Der Ansatz ähnelt damit stärker der klassischen Struktur evolutionärer Algorithmen nach Michalewicz et al., 1997, wobei die einzelnen Funktionen durch LLM-Aufrufe ersetzt werden.

Jedes Individuum s_i innerhalb der Population stellt eine mögliche Lösung des zugrunde liegenden Optimierungsproblems dar (vgl. Kapitel 2.3:

$$P = (T, R_L, R_R, S, C)$$

Dabei beschreibt T die Menge der Zeitslots, R_L die informationsreichen Ressourcen (z. B. Klassen, Maschinen oder Züge) und R_R die informationsarmen Ressourcen (z. B. Lehrkräfte, Aufträge oder Personal). S steht für die erzeugte Zuordnung und C für die Menge der Constraints. Das LLM generiert aus diesen Eingaben vollständige JSON-Strukturen, die die Verteilung der Ressourcen über die Zeit abbilden. Im Gegensatz zu klassischen

evolutionären Algorithmen, die auf numerischen Vektoren operieren, arbeitet EoS direkt auf semantisch strukturierten Objekten. Der Ablauf ist in Algorithmus 7 dargestellt.

Algorithm 7: Evolution of Schedules (EoS)

Input: Number of generations N; population size L; mutation probability m

Data: Left objects R_L; right objects R_R; result schema S_{schema}; global parameters G

Result: Best schedule s^*

$t \leftarrow \text{initPrompt}(R_L, R_R, S_{schema})$;
$P \leftarrow []$;
while $|P| < L$ **do**
    $p \leftarrow \text{generateSchedule}(t)$;
    $f(p), V_{desc} \leftarrow \text{validateSchedule}(p)$;
    $P \leftarrow P \cup \{(p, f(p), V_{desc})\}$;
for $i = 1, \ldots, N$ **do**
    while $|P| < 2L$ **do**
        $p_1, p_2 \leftarrow \text{randomParents}(P)$;
        $t \leftarrow \text{crossOverPrompt}(t, p_1, p_2)$;
        $p' \leftarrow \text{generateSchedule}(t)$;
        $f(p'), V_{desc} \leftarrow \text{validateSchedule}(p')$;
        if *random*() $< m$ **then**
            $p' \leftarrow \text{mutateSchedule}(p')$;
        $P \leftarrow P \cup \{(p', f(p'), V_{desc})\}$;
    $P \leftarrow \text{selectTop}(P, L)$;
    $s^* \leftarrow \text{best}(P)$;
    if $f(s^*) = 0$ **then**
        return s^*;
return s^*

Die Initialisierung erfolgt über die Funktion `initPrompt()`, welche die Eingabedaten in eine natürlichsprachliche Beschreibung überführt. Der Prompt enthält eine semantische Aufgabenbeschreibung, textuelle Definitionen der Ressourcen R_L und R_R sowie ein JSON-Beispielschema für die erwartete Ausgabe. Explizite Constraints werden zunächst nicht übergeben, sondern später während der Validierung berücksichtigt. Das LLM gene-

riert aus diesen Eingaben eine erste Population von L Plänen, die jeweils eine potenzielle Lösung des Problems darstellen.

Jeder Plan p wird anschließend in zwei Schritten bewertet. Zuerst überprüft ein deterministisches Validierungsmodul grundlegende Strukturbedingungen. Dazu gehört, dass alle `left_ids` und `right_ids` vorhanden sind, keine Pläne leer bleiben, keine Ressource doppelt verplant ist und die maximale Arbeitszeit der `right_ids` eingehalten wird. Die Summe der festgestellten Verstöße ergibt den Basiswert $f_1(p)$.

Im zweiten Schritt bewertet ein LLM die Lösung aus semantischer Sicht. Es analysiert die Gesamtheit der Constraints C, ordnet sie als harte oder weiche Anforderungen ein und weist ihnen Gewichtungen zu. Anschließend gibt es die Anzahl der Verstöße pro Constraint sowie eine kurze Begründung zurück. Damit diese Gewichtungen konsistent bleiben, werden sie normalisiert, sodass ihre Summe stets eins ergibt. Die Fitness eines Plans ergibt sich somit zu:

$$f(p) = f_1(p) + \sum_{c \in C} v_{num}(c) \cdot w(c) \tag{17}$$

wobei $v_{num}(c)$ die Anzahl der Verstöße und $w(c)$ die Gewichtung des jeweiligen Constraints darstellt. Zusätzlich erzeugt das LLM eine kurze textuelle Rückmeldung, die Schwachstellen und mögliche Verbesserungen beschreibt. In dem Algorithmus 7 ist all dieses unter der Methode `validateSchedule(p)` abgebildet. Diese dient in der nächsten Generation als Grundlage für Mutation und Crossover.

Die Variation der Population erfolgt ebenfalls promptbasiert. Bei der Mutation wird das LLM gezielt angewiesen, einen bestehenden Plan zu verändern. Dabei erhält es den Plan, den Fitnesswert und die zugehörige Bewertung. Auf dieser Basis soll es gezielt Schwachstellen korrigieren, ohne die gesamte Struktur zu verändern. Es ist trotzdem angewiesen, eine gewisse Abweichung von dem aktuellen Plan zu erzielen. Dadurch wird der Suchraum erweitert, ohne dass wertvolle Informationen verloren gehen.

Beim Crossover werden zwei Elternpläne p_1 und p_2 zufällig ausgewählt. Das LLM erhält beide Planstrukturen, ihre Fitnesswerte und die Begründungen und soll daraus eine neue Lösung entwickeln. Diese soll Merkmale beider Eltern kombinieren und möglichst weniger Constraint-Verletzungen enthalten. Das Verfahren folgt damit dem klassischen Prinzip der Rekombination, erweitert es jedoch um eine semantische Ebene, die durch das LLM realisiert wird.

Sobald die neue Generation erzeugt wurde, wird die Population mithilfe der *Survivor Selection* wieder auf die ursprüngliche Größe L reduziert. Nur die Individuen mit der besten Fitness werden in die nächste Generation übernommen. Der Algorithmus endet, wenn entweder die maximale Generation N erreicht oder eine fehlerfreie Lösung gefunden wurde.

Um die Anwendbarkeit über verschiedene Domänen hinweg sicherzustellen, verwendet EoS ein generisches JSON-Schema. Es erlaubt domänenspezifische Erweiterungen, ohne dass die zugrunde liegende Systemlogik angepasst werden muss. Für die informationsreichen Ressourcen (R_L) ist das Basisschema wie folgt definiert:

```
{
  "id": 0,
  "constraints_list": [],
  "individual": {}
}
```

JSON-Darstellung 11: Schema für left_id

Das Feld `individual` enthält die konkrete Planung und muss mindestens die Zuordnung zu `right_id` enthalten. Zusätzliche Informationen wie Tage oder Schichten können je nach Domäne ergänzt werden. Die Benennung *individual* ist hier nur ein Platzhalter und wird während des Überschreibens mit dem konkreten Schema ausgetauscht. Für die informationsarmen Ressourcen (R_R) gilt ein restriktiveres Schema, das die Konsistenz der Bewertung sicherstellt:

```
{
  "id": 0,
  "constraints_list": [],
  "max_work_hours": 0
}
```

JSON-Darstellung 12: Schema für right_id

Dieses Schema kann nicht abgeändert bzw. individualisiert werden. Alle notwendigen Informationen für das Mapping oder andere Restriktionen müssen über die informationsreiche Seite oder die Constraints dargestellt werden. Das resultierende Schema für die erzeugten Pläne folgt demselben Aufbau eines grundlegenden Schemas und bildet die zeitliche Belegung der linken Ressourcen ab:

```
{
  "left_id": {
    "Montag": {
      "1": {
        "endzeit": 2,
        "right_id": 0
      }
    },
    "Dienstag": {
      "2": {
        "endzeit": 4,
        "right_id": 0
      }
    }
  }
}
```

JSON-Darstellung 13: Schema für result

Auf unterster Ebene können hier Informationen ergänzt werden, die relevant für den resultierenden Plan sind.

Insgesamt überführt EoS die Prinzipien klassischer evolutionärer Algorithmen in ein textbasiertes Framework. Das LLM übernimmt dabei die Rolle eines aktiven Operators, der Pläne generiert, bewertet und modifiziert. Dadurch entsteht ein Verfahren, das adaptiv auf domänenspezifische Anforderungen reagieren kann. Durch die generische JSON-Struktur bleibt das System universell einsetzbar, ohne dass Codeanpassungen erforderlich sind. EoS bildet somit die konzeptionelle Brücke zwischen der einmaligen Planerzeugung des Prompting-Ansatzes und der in Kapitel 3.2.3 beschriebenen heuristischen Evolution.

3.2.3 Evolution of Heuristics

Die grundlegende Idee der *Evolution of Heuristics* (EoH) geht auf Liu et al., 2024 zurück. Ziel ist es, durch die evolutionäre Weiterentwicklung von Heuristiken, die von Large Language Models generiert werden, eine für das jeweilige Problem optimale Entscheidungs-

strategie zu identifizieren (Liu et al., 2023). Im Gegensatz zur EoS, bei der das LLM unmittelbar vollständige Pläne erzeugt und diese im Verlauf der Evolution verbessert, wird im EoH-Ansatz die Heuristik selbst als evolvierbares Objekt betrachtet. Das bedeutet, dass das LLM Strategien generiert, die beschreiben, wie ein Zeitplan konstruiert werden soll – etwa durch priorisierte Zuweisungen, Greedy-Verfahren oder Konfliktauflösungen. Diese Strategien werden in einem evolutionären Algorithmus als Individuen interpretiert und nach den Prinzipien von Selektion, Rekombination und Mutation verarbeitet, um iterativ verbesserte Heuristiken zu entwickeln (Liu et al., 2024).

Die zugrunde liegende Annahme lautet, dass durch die Kombination und sukzessive Weiterentwicklung mehrerer Heuristiken bessere Ergebnisse erzielt werden können als durch einmaliges Zero-Shot-Prompting. Dies folgt den Grundannahmen, dass LLMs bei der schrittweisen Aufschlüsselung von Lösungsansätzen und Zwischenüberlegungen qualitativ hochwertigere Ergebnisse liefern. Das Prinzip der EoH überträgt dieses Verhalten auf die strategische Ebene. Über wiederholte Iterationen werden verschiedene Lösungsansätze miteinander kombiniert, sodass die resultierenden Heuristiken zunehmend leistungsfähiger und robuster werden.

Der Aufbau des Algorithmus ist grundlegend gleich wie beim EoS (Kapitel 3.2.2). Der Unterschied hier ist lediglich, dass vom LLM Heuristiken und nicht die finalen Pläne erstellt werden, weshalb der Code erst noch ausgeführt wird. Daher wird im folgenden auf redundante Beschreibungen des Ablaufes verzichtet und der Fokus auf die Eingaben in den Algorithmus gelegt.

Die Eingaben des Algorithmus lassen sich in zwei Hauptkategorien unterteilen: *Input* und *Data*. Unter Input werden die für evolutionäre Algorithmen typischen Konfigurationsparameter zusammengefasst, darunter die Anzahl der Generationen, die Populationsgröße, die Mutationswahrscheinlichkeit und die Anzahl der Eltern pro Individuum. Der Bereich Data beschreibt die fachlichen Eingangsdaten des Optimierungsproblems. Dazu gehören die Listen der informationsreichen (IR bzw. left_id) und informationsarmen (IA bzw. right_id) Objekte, das Ergebnis-Schema (result schema) sowie eine Menge globaler Parameter. Diese globalen Parameter stellen einen zentralen Bestandteil der semantischen Kontextbildung dar und erweitern die klassische Konfiguration eines evolutionären Algorithmus um fachliche Zusatzinformationen. Hierzu zählen insbesondere:

- **Days:** Eine Liste der relevanten Tage (z. B. Montag bis Freitag), die im Planungsprozess berücksichtigt werden.

- **Global-Constraints:** Übergreifende Bedingungen, die für alle generierten Pläne gleichermaßen gelten, etwa Verfügbarkeiten, Kapazitätsgrenzen oder Ruhezeiten.
- **Use-Case-Description:** Eine textuelle Beschreibung des Anwendungsfalls, die dem LLM zusätzlichen Kontext über den fachlichen Hintergrund der Planung liefert.
- **Description IA/IR:** Semantische Beschreibungen der beteiligten Objekte, um dem Modell ein besseres Verständnis über deren Rolle und Abhängigkeiten zu ermöglichen.

Die Datenbereitstellung erfolgt in einem zweistufigen Verfahren, das in Abbildung 5 schematisch dargestellt ist.

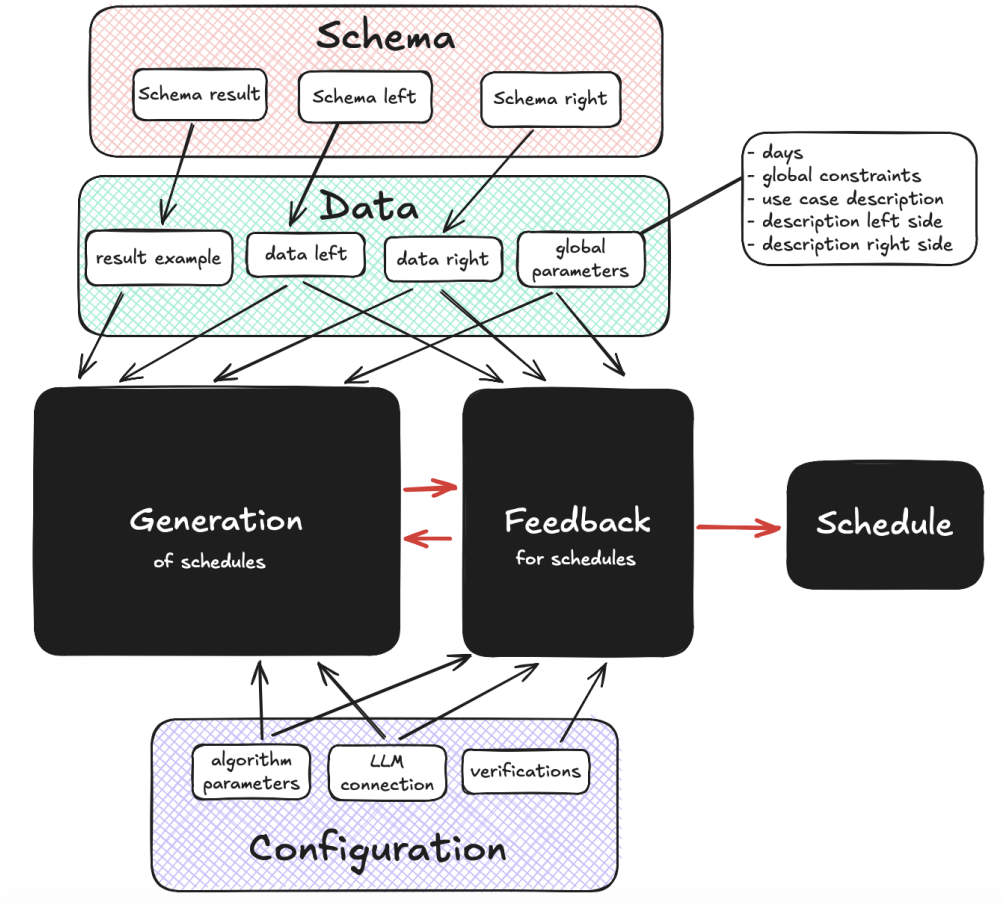

Abbildung 5: Schematischer Aufbau der Eingabedaten

Zunächst werden die Objekte nach einem vordefinierten Schema formatiert (vgl. Kapitel 3.2.2), um eine konsistente Übergabe an das LLM zu gewährleisten. Die Struktur kann flexibel erweitert werden, um domänenspezifische Anforderungen abzubilden, wie etwa Fachbelegungen in der Schulplanung oder Maschinenaufträge in der Produktionsplanung. Das result schema ist analog aufgebaut und definiert die Struktur der erwarteten Ergebnisdaten.

Die algorithmische Architektur des Verfahrens ist im Algorithmus 8 dargestellt. Der Ablauf folgt dabei der Grundstruktur aus EoS, mit der Veränderung, dass durch das LLM eine Heuristik erstellt wird, die erst für die Generierung eines Plans angewendet werden muss.

Algorithm 8: Evolution of Heuristics EoH nach Liu et al., 2024

Input: Number of generations N; population size L; mutation probability m

Data: Left objects R_L; right objects R_R; result schema S_{schema}; global parameters G

Result: Best schedule s^*

```
t ← initPrompt(R_L, R_R, S_schema) ;
P ← [];
while |P| < L do
    h ← generateHeuristic(t);
    p ← executeHeuristic(h);
    f(p), V_desc ← validateSchedule(p);
    P ← P ∪ {(p, f(p), V_desc, h)};
for i = 1, ..., N do
    while |P| < 2L do
        p_1, p_2 ← randomParents(P);
        t ← crossOverPrompt(t, p_1, p_2);
        h' ← generateHeuristic(t);
        p' ← executeHeuristic(h');
        f(p'), V_desc ← validateSchedule(p');
        if random() < m then
            p' ← mutateSchedule(p');
        P ← P ∪ {(p', f(p'), V_desc, h')};
    P ← selectTop(P, L);
    s* ← best(P);
    if f(s*) = 0 then
        return s*;
return s*
```

Auf Grundlage der zuvor genannten Eingaben wird bei EoH ein initialer Prompt t erzeugt, der alle relevanten Daten in einen textuellen Kontext überführt. Dieser Prompt wird an das LLM übergeben, das daraus eine Heuristik h generiert. Jede erzeugte Heuristik beschreibt, wie der Plan erstellt werden soll, und wird anschließend innerhalb des Systems ausgeführt. Das resultierende Planungsergebnis p wird anschließend einer Validierung unterzogen, die wie bei der Implementierung von EoS funktioniert. Zunächst überprüft ein fester Validierungscode die harten logischen Bedingungen (z. B. alle IR-Objekte sind

zugeordnet, kein Objekt ist doppelt geplant). In der zweiten Stufe werden die textuellen Constraints an das LLM übergeben, das diese hinsichtlich ihrer Gewichtung (in Prozent) und der Anzahl der Fehler bewertet. Die daraus resultierenden Bewertungen werden als Tupel (Punktzahl, Gewichtung) aggregiert und in die Fitnessberechnung einbezogen.

Auf dieser Grundlage erfolgt die evolutionäre Weiterentwicklung der Population. Nach der Bildung der initialen Menge an Heuristiken wird in den Folgegenerationen der Cross-Over-Prompt verwendet, welcher zufällige Elternheuristiken sowie deren Bewertungsinformationen kombiniert, um daraus neue Varianten abzuleiten. Das LLM agiert hierbei als semantisch informierte Rekombinationsinstanz: Es wird gezielt angewiesen, die vorteilhaften Elemente mehrerer Heuristiken zu kombinieren und gleichzeitig Varianten einzubringen, um eine ausreichende Diversität sicherzustellen. Darüber hinaus kann das LLM angewiesen werden, bestehende Heuristiken gezielt zu mutieren, wenn die Wahrscheinlichkeit m einer Mutation eintritt. Auf diese Weise wird verhindert, dass die Population in lokalen Optima stagniert.

Das Zusammenspiel aus Heuristik, Plan, Validierung und Prompt bildet das Individuum innerhalb der Population. Die Evolution endet, sobald eine Heuristik eine Fitness von null erreicht oder die maximale Anzahl N an Generationen durchlaufen wurde. In beiden Fällen wird die aktuell beste Heuristik s^* als Ergebnis zurückgegeben.

Durch die Kopplung der generativen Fähigkeiten von LLMs mit den Mechanismen evolutionärer Optimierung entsteht ein System, das nicht nur einzelne Pläne, sondern allgemeine Strategien zur Planerzeugung lernen und verfeinern kann.

3.2.4 Evolutionärer Algorithmus

Die Implementierung des Evolutionären Algorithmus orientiert sich an der in Kapitel 2.3 dargestellten theoretischen Grundlage. Die Struktur ist in Algorithmus 9 beschrieben und folgt einem generischen Ablauf, der vom konkreten Problem abstrahiert. Diese generischen Komponenten lassen sich auf eine Vielzahl ähnlicher Optimierungsprobleme übertragen. Die anwendungsspezifischen Anpassungen erfolgen an den im Algorithmus hervorgehobenen Stellen.

Algorithm 9: Evolutionary Algorithm nach Grundlagen von Colorni et al., 1991

Input: Number of Population: N; Population size: L, Probability of mutation: m; Soft constraints weight: w; spot identifier definition: id_s

Data: IA object list: R_R; IR object list: R_L

Result: Best schedule s^*

```
for i=1,...,L do
    s ← individual(R_L, R_R) Create individual: A basis schedule s_i that randomly
      distributes the events across the timetable. The frame already fulfills the
      requirements regarding the timeframe. Each timetable gets a fitness value
      f(s_i, r, l);
Assemble initial population P = s_1, ..., s_L;
for i=1,...,N do
    while |P| < L*2 do
        s_g1 ← random(P);
        s_g2 ← random(P);
        s_c ← mate(s_g1, s_g2) Mating: Crossover of both parents to create a new
          schedule that includes characteristics of both parents;
        if random < m then
            s_c ← mutate(s_c) Mutating: Creates a new schedule based on the given
              schedule;
        f(s_c), slots(s_c) ← fitness(s_c, w, r, l) Calculate fitness: Calculates the fitness
          of the given schedule ans the slots with mistakes;
        I_c ← (s_c, f(s_c), slots(s_c));
        P = (I_1, I_2, ..., I_|P|, I_c)
    P ← sort(P, by f(s) ascending);
    P ← { s ∈ P | s is among the L fittest individuals };
    if f(first element of P) = 0 then
        return first element of P as s*
s* = first { s ∈ P | noHardConstraintsViolations(s) };
return s*
```

Algorithmus 9 verdeutlicht die übergeordnete Struktur des Verfahrens. Der Ablauf umfasst die Initialisierung einer Population, die wiederholte Erzeugung neuer Individuen durch Rekombination und Mutation sowie die Bewertung und Selektion der jeweils bes-

ten Lösungen. Am Ende wird ein Plan s^* bestimmt, der keine Hard-Constraints verletzt und eine möglichst gute Fitness (niedrige Fehleranzahl) aufweist.

Die Eingaben des Algorithmus lassen sich in zwei Kategorien gliedern: *Input* und *Data*. Sie ähneln sich damit stark jenen von EoS und EoH. Unter Input sind die algorithmischen Parameter zusammengefasst, die den Ablauf steuern, etwa Populationsgröße, Anzahl der Iterationen oder Mutationswahrscheinlichkeit. Diese Parameter definieren das Rahmenwerk des Suchprozesses. Ergänzend dazu wurde der Parameter *spot identifier definition* eingeführt, der festlegt, wie eine Position oder ein Zeitslot innerhalb der JSON-Struktur eindeutig identifiziert werden kann. Dadurch wird ermöglicht, dass Constraint-Validierungen nicht nur die Anzahl der Verstöße, sondern auch deren konkrete Positionen im Plan zurückgeben können. Der Benutzer muss hierzu definieren, welche JSON-Felder (z. B. `day`, `slot`, `left_id` oder `right_id`) notwendig sind, um einen spezifischen Punkt im Plan eindeutig zu bestimmen.

Unter Data werden die konkreten Eingabedaten verstanden, die aus einer informationsarmen (R_R) und einer informationsreichen (R_L) Seite bestehen.

Die informationsreiche Seite enthält verschiedene Bestandteile. Jedes Element verfügt über eine Liste von Constraints, die in hard und soft in der Eingabe zu unterteilen sind. Zusätzlich ist eine time description hinterlegt, die den relevanten Zeitrahmen beschreibt, beispielsweise bestimmte Tage oder Zeitintervalle. Darüber hinaus besitzt jedes Element eine mapping description, welche die Zuordnungsmöglichkeiten zur informationsarmen Seite beschreibt. Hier kann entweder der Informationsgehalt einzelner Zeitslots angegeben werden, oder es wird definiert, welche informationsarmen Elemente für bestimmte Zustände infrage kommen.

Das Ergebnis (*Result*) ist stets ein Plan, der anhand der Fitnessbewertung als bestgeeignete Lösung identifiziert wird. Dabei wird der beste gültige Plan zurückgegeben, der keine Hard-Constraints verletzt (vgl. Algorithmus 9, Zeile 18).

Neben den explizit angegebenen Parametern existieren weitere zu spezifizierende Funktionen, die im Algorithmus 9 dick gedruckt hervorgehoben sind. Sie stellen die anpassbaren Bestandteile des generischen Algorithmus dar. Während die generische Struktur fest implementiert ist, werden diese spezifischen Funktionen dynamisch durch ein LLM erzeugt. Dies betrifft insbesondere die Initialisierung der Individuen (Zeile 2), die Definition der Rekombinationsstrategie (*Mating*, Zeile 8), die Beschreibung von Mutationsoperationen (Zeile 10) sowie die Umsetzung der Constraint-Prüfungen (Zeile 11). Auf diese Weise

wird der generische Algorithmus adaptiv an unterschiedliche Problemstellungen angepasst, ohne dass eine manuelle Programmierung erforderlich ist.

Damit das LLM die relevanten Code-Snippets erzeugen kann, benötigt es eine Reihe an vorbereitenden Informationen. Diese dienen als Grundlage für die Generierung der im Pseudocode hervorgehobenen Methoden und setzen die bereitgestellten Daten in den notwendigen Kontext. Die Eingaben lassen sich wie folgt strukturieren:

- **Global Constraints**: Liste der globalen Anforderungen an die erstellten Pläne. Dazu zählen Bedingungen, die für jedes Element einer Seite gelten, oder Anforderungen, die zwar intuitiv plausibel erscheinen, für die automatische Validierung jedoch explizit aufgeführt werden müssen. Alle Constraints sind als *hard* oder *soft* gekennzeichnet. Über den Parameter Soft constraints weight im Algorithmus 9 wird bestimmt, in welchem Maße die soft Constraints in die Fitnessbewertung einfließen.
- **Mapping Description Example**: Ein exemplarisches Schema für die Verbindung zwischen IA- und IR-Seite. Es definiert, welche Felder hinterlegt werden müssen und welche Art von Informationen darin gespeichert werden kann. Alle Mappings der informationsreichen Seite müssen diesem Schema entsprechen.
- **Mapping Description**: Eine textuelle Beschreibung der Verwendung des Mapping-Beispiels. Sie legt fest, wie die bereitgestellten Werte zu interpretieren sind. Beispielsweise kann ein Attribut wie *Anzahl* die Häufigkeit eines Datensatzes beschreiben oder es kann eine Optionsliste für die Verknüpfung zur informationsarmen Seite enthalten sein, aus der ein Wert zufällig ausgewählt wird. Auf diese Weise wird präzisiert, wie mit den Daten umzugehen ist und welche Informationen aus ihnen abzuleiten sind.
- **Optimization Perspective**: Beschreibung der Vorgehensweise beim Crossover bzw. Mating der Elternpläne. Hier wird festgelegt, welches Optimierungsziel verfolgt wird, etwa die Kombination einzelner Zeitelemente oder die Beibehaltung bestimmter Strukturen. Es wird damit definiert, welche Veränderungen an den Plänen erlaubt sind – beispielsweise ob lediglich die IDs der informationsarmen Seite ausgetauscht werden dürfen. Alternativ kann auch konkret hinterlegt werden, welche Schritte der Algorithmus beim Crossover durchlaufen soll, um eine verbesserte Lösung zu erlangen.

- **Mutation Perspective**: Ergänzend zur *Optimization Perspective* beschreibt dieser Teil, wie ein Plan zufällig verändert werden darf. Dies reicht von einer vollständigen Durchmischung aller Elemente bis zum gezielten Austausch einzelner IDs. Damit wird die Bandbreite zulässiger Mutationen bestimmt. Entscheidend ist hier, dass durch die Randomisierung nicht die Struktur des Ergebnisses verletzt wird oder Zustände entstehen, welche nach den Rahmenbedingungen nicht möglich sind und daher gar nicht erst eintreten sollen.
- **Spot Identifier Definition**: Diese Komponente legt fest, welche JSON-Felder notwendig sind, um eine bestimmte Position innerhalb eines Plans eindeutig zu adressieren. Auf Basis dieser Definition können die Constraint-Prüfungen die exakten Positionen fehlerhafter Einträge zurückgeben, sodass das LLM im Mating-Schritt gezielt an diesen Stellen ansetzen kann, um die Fehler zu beheben oder zu minimieren.
- **Example Result**: Abschließend wird ein JSON-Beispiel für das Resultat angegeben, das die Struktur des finalen Plans vorgibt. In diesem Format werden die einzelnen Elemente s erstellt und optimiert, sodass das Ergebnis konsistent zurückgegeben werden kann.

Wie in Algorithmus 9 (Zeile 2) beschrieben, erhält jedes Individuum einen Fitness-Wert. Die Berechnung erfolgt ebenfalls nach der in Kapitel 2.4.2 aufgeführten Penalized Cost Function und liefert als bestmöglichen Wert null, sofern keine Constraints verletzt werden. Je höher der Wert, desto mehr Fehler enthält der Plan. Der Einfluss der Soft-Constraints wird über den Parameter w gesteuert: je nach Gewichtung fließen Verstöße gegen Soft-Constraints mit reduziertem Wert in die Berechnung ein, während Hard-Constraints stets mit einem Wert von eins berücksichtigt werden. Formal lässt sich die Fitness-Funktion wie folgt darstellen:

$$f(p) = \sum_{c \in C_{\text{hard}}} \delta_c(p) + \sum_{c \in C_{\text{soft}}} \delta_c(p) \cdot w \tag{18}$$

Dabei beschreibt $f(p)$ die Fitness eines Plans, C_{hard} und C_{soft} die Mengen der Hard- bzw. Soft-Constraints, und $\delta_c(p)$ die von einem LLM implementierte Prüf-Funktion. Wichtig hierbei zu betonen ist, dass die Validierung des Resultates in dieser Implementierung ebenfalls durch vom LLM einmalig generierten Code durchgeführt wird. Diese Methoden werden für jedes definierte Constraint implementiert und für jede Validierung wiederholt

aufgerufen. Die einzelnen Methoden geben dann nicht nur die Anzahl der Verstöße zurück, sondern auch die konkreten Positionen der betroffenen Elemente im Plan. Diese Positionsinformationen werden direkt in den nächsten Mating-Schritt überführt, sodass das LLM gezielt jene Bereiche priorisieren kann, in denen Verstöße aufgetreten sind.

Neben der Initialisierung ist das LLM auch bei den Schritten *Mating* und *Mutation* eingebunden. In beiden Fällen ist entscheidend, dass die bereitgestellten Eingaben festlegen, welche Veränderungen erlaubt sind. Das LLM erzeugt auf dieser Basis die entsprechenden Methoden, die anschließend wiederholt aufgerufen werden. Beim Mating wird ein neuer Kind-Plan erzeugt, der im besten Fall die positiven Eigenschaften beider Elternpläne kombiniert. Dabei kann das LLM mithilfe der von der Fitnessfunktion übermittelten Positionsinformationen gezielt jene Slots verändern, an denen zuvor Fehler erkannt wurden, wodurch der Suchprozess effizienter wird. Bei der Mutation hingegen wird ein bestehender Plan zufällig verändert, um die Suche zu diversifizieren und ein Festfahren im lokalen Optimum zu vermeiden. Beide Funktionen werden einmalig generiert und anschließend für alle Iterationen genutzt.

Die Kommunikation mit dem LLM zur Generierung der benötigten Code-Snippets erfolgt vollständig über Prompting. Dabei wird auf die in Kapitel 3.2.1 beschriebenen Erkenntnisse über Prompting zurückgegriffen. Durch die Aufteilung in klar abgegrenzte Arbeitsschritte handelt es sich jeweils um überschaubare Anforderungen, die keine komplexen mehrstufigen Anweisungen erfordern. Aus diesem Grund wurde die Methode des Zero-Shot-Promptings gewählt.

Ergänzend zur beschriebenen Funktionsweise ist im Code fest verankert, dass nach zwanzig aufeinanderfolgenden Iterationen ohne messbare Verbesserung[6] die Mating-Funktion automatisch neu generiert wird. Hintergrund dieser Anpassung ist das stochastische Antwortverhalten von LLMs, die auch bei identischer Eingabe leicht unterschiedliche Ergebnisse liefern können (vgl. Kapitel 2.5.1). In ersten Tests zeigte sich, dass dadurch vereinzelt suboptimale Mating-Strategien erzeugt wurden, welche die Weiterentwicklung der Population behinderten. Die automatische Neugenerierung nach wiederholtem Misserfolg erhöht daher die Robustheit des Systems gegenüber zufälligen Ausreißern und trägt zu einer stabileren Optimierung bei.

[6] Als erfolglose Iterationen werden jene verstanden, in denen der Algorithmus das jeweils beste Individuum der Population nicht weiter optimieren konnte.

4 Experimente

Das folgende Kapitel beschreibt den empirischen Teil der Arbeit, in dem die zuvor vorgestellten Implementierungen unter kontrollierten Bedingungen getestet und systematisch miteinander verglichen werden. Ziel dieser Experimente ist es, die Leistungsfähigkeit der entwickelten Verfahren im Hinblick auf ihre Genauigkeit, Stabilität und Effizienz zu bewerten und daraus Rückschlüsse auf ihre praktische Anwendbarkeit in unterschiedlichen Planungsdomänen zu ziehen.

Während Kapitel 3 die konzeptionellen Grundlagen, Datenmodelle und technischen Implementierungen vorgestellt hat, liegt der Fokus nun auf der methodischen Ausgestaltung und Durchführung der Experimente. Die Untersuchungen dienen insbesondere dazu, empirisch zu belegen, in welchem Maße LLMs die Lösungsfindung in evolutionären Prozessen unterstützen und ob sich durch die Kombination beider Ansätze leistungsfähige, adaptive Algorithmen ergeben. Damit bildet dieses Kapitel die Grundlage für die spätere Ergebnisanalyse und Bewertung in Kapitel 5 und 6.

Zur Sicherstellung der Vergleichbarkeit wurden alle Tests unter identischen Rahmenbedingungen durchgeführt. Für jedes Szenario - Schulplanung, Zugpersonalplanung und Maschinenbelegungsplanung - wurden sämtliche Implementierungen (Prompting, Evolution of Schedules, Evolution of Heuristics und Evolutionärer Algorithmus) mehrfach ausgeführt. Auf diese Weise wird sichergestellt, dass zufällige Einflüsse, etwa durch stochastisches Antwortverhalten des LLMs, statistisch ausgeglichen werden und robuste Durchschnittswerte vorliegen. Die gewonnenen Daten liefern damit eine objektive Grundlage, um Stärken, Schwächen und typische Verhaltensmuster der unterschiedlichen Ansätze zu identifizieren.

Um realistische und zugleich kontrollierbare Testbedingungen zu schaffen, wurden für jedes der betrachteten Szenarien im Vorfeld Referenzpläne definiert, welche jeweils eine konsistente und gültige Lösung des zugrunde liegenden Problems darstellen (vgl. Anhang A.1. Für die Schulplanung diente ein reales Szenario einer bestehenden Schule als Ausgangspunkt. In den Fällen der Maschinen- und Zugpersonalplanung wurde hingegen künstlich ein strukturierter Plan erstellt. Bei diesem war das Ziel, die informationsarmen Ressourcen gering zu halten, um eine Ressourcenknappheit darzustellen. Auf Basis der jeweiligen Zuordnungen wurden die zugehörigen Constraints abgeleitet und so gestaltet, dass eine anspruchsvolle, aber prinzipiell vollständig erfüllbare Ressourcenlage entsteht. Ziel dieser Konstruktion war es, den Schwierigkeitsgrad der Szenarien so zu wählen, dass

keine triviale Lösung existiert, gleichzeitig aber ein theoretisch optimaler Plan möglich bleibt.

Für alle definierten Szenarien wurden ergänzend Validierungsskripte implementiert, die eine automatisierte Prüfung der erzeugten Lösungen ermöglichen. Diese Skripte überprüfen, ob sämtliche harte und weiche Constraints erfüllt sind und dienen somit als objektiver Maßstab zur Bewertung der Ergebnisqualität. Durch ihre Anwendung kann sichergestellt werden, dass auftretende Fehler eindeutig identifiziert werden. Auf diese Weise wird die methodische Güte der Experimente gewahrt und eine nachvollziehbare Beurteilung der erzeugten Resultate gewährleistet.

In Abschnitt 4.1 wird zunächst erläutert, welche quantitativen und qualitativen Kennwerte erhoben wurden, um die Ergebnisse systematisch zu bewerten. Diese Metriken erfassen sowohl ressourcenbezogene Faktoren, wie die Anzahl der LLM-Aufrufe, als auch leistungsorientierte Größen, wie die Erfüllung von Hard- und Soft-Constraints, die Konvergenzgeschwindigkeit und die Vielfalt der gefundenen Lösungen. Zudem wird begründet, weshalb die jeweiligen Messgrößen gewählt wurden und wie sie zur Beantwortung der Forschungsfragen dieser Arbeit beitragen.

Darauf aufbauend beschreibt Abschnitt 4.2 den konkreten Aufbau der Experimente. Hier werden die Parameter, Wiederholungen und Evaluationsmechanismen im Detail erläutert, um die Reproduzierbarkeit und Nachvollziehbarkeit der Ergebnisse sicherzustellen.

Insgesamt verfolgt dieses Kapitel das Ziel, ein transparentes und konsistentes Fundament für die Ergebnisanalyse zu schaffen, das sowohl den methodischen Anspruch der Arbeit erfüllt als auch eine objektive Bewertung der entwickelten Systeme ermöglicht.

4.1 Erhobene Datenpunkte

Zur Beurteilung der Leistungsfähigkeit der entwickelten Verfahren wurden verschiedene quantitative und qualitative Kennwerte erhoben, die unterschiedliche Dimensionen des Systemverhaltens abbilden. Die Auswahl dieser Datenpunkte orientiert sich unmittelbar an den Forschungsfragen dieser Arbeit und an den theoretischen Grundlagen zur Bewertung von evolutionären Algorithmen in Kapitel 2.4.1. Die erhobenen Kennzahlen dienen somit als Grundlage für eine objektive und nachvollziehbare Bewertung der Ansätze hinsichtlich ihrer Effizienz, Genauigkeit und Adaptivität.

Anzahl der LLM-Aufrufe Die Anzahl der Aufrufe eines LLMs während eines Optimierungslaufs stellt einen zentralen Indikator für den Ressourcenverbrauch und die Interaktionsintensität zwischen Algorithmus und Modell dar. Ein hoher Wert deutet darauf hin, dass das LLM stark in den iterativen Prozess eingebunden ist, während eine geringere Anzahl auf eine effizientere Nutzung oder eine stabilere Lösungsfindung hinweist. Dieser Kennwert ist insbesondere relevant, um die ökonomische und technische Skalierbarkeit des Ansatzes zu bewerten. In direktem Bezug zu den Forschungsfragen erlaubt er Rückschlüsse darauf, ob der Einsatz des LLMs tatsächlich zu einer Verbesserung der Lösungsqualität führt oder lediglich zu einem erhöhten Rechen- und Kommunikationsaufwand. Zudem soll er in diesem Fall auch repräsentativ für die Laufzeit der Algorithmen stehen. Durch Verfügbarkeiten, Internetgeschwindigkeit und Rechenleistung der LLMs sind die Laufzeiten der Algorithmen in Teilen sehr unterschiedlich. Die größte Zeit wird für die Verarbeitung durch das LLM aufgewendet. Dadurch kann mittels dieser Messgröße der Verarbeitungsaufwand, der Ressourcenverbrauch und implizit auch die Laufzeit mit abgebildet werden.

Durchschnittliche Fitness zwischen verschiedenen Durchläufen Für jede getestete Implementierung wurden mehrere unabhängige Läufe durchgeführt, um stochastische Effekte auszugleichen. Die durchschnittliche Fitness beschreibt dabei den Mittelwert der finalen Fitnesswerte der jeweils besten gefundenen Lösungen über alle Wiederholungen hinweg. Diese Kennzahl gibt Aufschluss darüber, wie konsistent und zuverlässig ein Verfahren dazu in der Lage ist, qualitativ hochwertige Ergebnisse zu liefern. Im Kontext der Forschungsfragen ermöglicht sie den Vergleich der generellen Leistungsfähigkeit der unterschiedlichen Ansätze und zeigt, ob die Einbindung von LLMs zu einer signifikanten Verbesserung der durchschnittlichen Ergebnisqualität führt.

Durchschnittliche Anzahl der Fehler laut Validierungsskripten Die externen Validierungsskripte dienen als objektiver Maßstab für die Korrektheit der erzeugten Lösungen. Für jeden Lauf wurde die Anzahl an erkannten Fehlern — also der verletzten harten oder weichen Constraints — ermittelt und anschließend über alle Wiederholungen gemittelt. Diese Kennzahl misst, in welchem Ausmaß ein Verfahren in der Praxis gültige und konsistente Pläne erzeugt. Sie trägt damit direkt zur Beantwortung der Frage bei, ob die entwickelten Systeme in der Lage sind, die Planungsprobleme fehlerfrei oder zumindest weitgehend constraint-konform zu lösen.

Differenz zwischen interner Fitness und externer Bewertung Die Differenz zwischen der durch das System berechneten Fitness und der durch die Validierungsskripte ermittelten Fehlerzahl wird als Maß für die Übereinstimmung interner und externer Qualitätsbewertung verwendet. Dabei wird für jeden Lauf der Unterschied der finalen Werte bestimmt und anschließend über alle Wiederholungen gemittelt. Eine geringe Differenz weist darauf hin, dass die Implementierung ihre eigene Lösungsqualität realistisch einschätzt, während größere Abweichungen auf eine unzureichende interne Bewertung hindeuten. Diese Metrik ist somit entscheidend, um die Verlässlichkeit der internen Bewertungsmechanismen zu prüfen und zu bewerten, inwieweit LLMs zur objektiven Selbstevaluierung eines Systems beitragen können. Dies ist insbesondere in dem Kontext spannend, dass das LLM in der Implementierung des Evolutionären Algorithmus (Kapitel 3.2.4) für die Bewertung des Resultats Code generiert, welcher wiederverwendet wird und nicht wie in den anderen Implementierungen direkt gefragt wird.

Konvergenzverhalten des Systems Das Konvergenzverhalten beschreibt, wie sich die Fitnesswerte im Verlauf der Iterationen verändern und ob eine Stabilisierung in Richtung einer optimalen oder zumindest stabilen Lösung erfolgt. Diese Analyse ermöglicht Rückschlüsse auf die Lern- und Anpassungsfähigkeit der jeweiligen Methode. Ein rasches, aber stabiles Konvergenzverhalten weist auf eine effektive Suchstrategie hin, während starke Schwankungen oder ein frühzeitiges Festlaufen auf lokale Optima hindeuten. Damit steht diese Metrik im direkten Zusammenhang mit der Forschungsfrage, ob die Integration von LLMs zu einer schnellen und qualitativ hochwertigen Lösung führt oder ob sie sich in lokale Optima verrennen. Anders als bei den anderen Metriken lässt sich das Konvergenzverhalten nicht bei der Implementierung des Promptings messen (Kapitel 3.2.1). Daher wird bei der abschließenden Bewertung der Systeme diese Metrik auch nur für den Vergleich jener verwendet, bei denen eine Bewertung dieser Art möglich ist.

Diversität der finalen Lösungen Neben der Qualität der Ergebnisse ist auch die Vielfalt der finalen Lösungen ein wichtiger Aspekt. Eine hohe Diversität zeigt, dass das Verfahren unterschiedliche Lösungsstrategien erkundet und damit robust gegenüber Zufallseinflüssen oder abweichenden Szenarien ist. Eine geringe Diversität hingegen kann auf Überanpassung oder eine starke Abhängigkeit vom initialen Suchzustand hindeuten. Diese Kennzahl trägt somit zur Beurteilung der Adaptivität und Generalisierungsfähigkeit der Verfahren bei – also der Frage, ob die Systeme in der Lage sind, unter wechselnden Bedingungen verschiedene, gleichwertig gute Lösungen zu finden, anstatt stets

einem festen Muster zu folgen. Die Diversität der Lösungen wird ermittelt, indem jede Lösung einmal mit allen anderen verglichen wird. Dabei wird jeder Zeitpunkt mit dem Äquivalent der anderen Lösung verglichen. Wenn ein Wert abweicht, wird der Zeitpunkt als ungleich deklariert; wenn alle Werte übereinstimmen, dann ist der Zeitpunkt gleich. Am Ende wird die Anzahl der ungleichen Zeitpunkte durch die Anzahl aller verglichenen Zeitpunkte geteilt. Dadurch entsteht für jedes Paar ein Wert zwischen null und eins, der bei einer hohen Ungleichheit dicht an eins ist. Schlussendlich wird aus allen ermittelten Werten der ungewichtete Durchschnitt gebildet. Dieser Durchschnittswert ist dann der aufgeführte Wert für den Unterschiedsgrad der Resultate.

False Positives Ein weiterer relevanter Kennwert ist der sogenannte *False-Positive*-Anteil. Er beschreibt den Prozentsatz jener Fälle, in denen das System eine erzeugte Lösung intern als korrekt oder fehlerfrei bewertet, während die externen Validierungsskripte dennoch Verstöße gegen die definierten Constraints identifizieren. Dieser Wert ist von zentraler Bedeutung, da er die Zuverlässigkeit der internen Selbstbewertung widerspiegelt. Insbesondere in einem realen Anwendungskontext, in dem keine externen Skripte zur Verfügung stehen, würde ein hoher False-Positive-Anteil dazu führen, dass fehlerhafte Pläne als gültig interpretiert und möglicherweise produktiv eingesetzt werden. Ein solcher Fall hätte gravierende Auswirkungen auf die praktische Nutzbarkeit der Systeme, da er das Vertrauen in die autonomen Entscheidungsmechanismen des Algorithmus untergräbt. Die Berechnung des False-Positive-Anteils erfolgt, indem die Anzahl der Fälle, in denen ein System seine eigene Lösung als fehlerfrei einstuft, obwohl externe Prüfungen noch Verstöße feststellen, ins Verhältnis zur Gesamtzahl der erzeugten Lösungen gesetzt wird. Ein niedriger Wert zeigt an, dass die interne Bewertungslogik zuverlässig mit der objektiven externen Beurteilung übereinstimmt, während ein hoher Wert auf eine Diskrepanz zwischen beiden Bewertungsmetriken hinweist.

Tatsächlich fehlerfreie Lösungen (True Positives) Neben der Analyse fehlerhafter oder inkonsistent bewerteter Ergebnisse ist auch die Betrachtung jener Fälle relevant, in denen eine erzeugte Lösung sowohl vom Algorithmus selbst als auch vom externen Validierungsskript als korrekt eingestuft wird. Diese sogenannten *True Positives* bezeichnen valide Lösungen, die keine harten Constraints verletzen und somit als vollständig zulässig gelten. Sie stellen den zentralen Erfolgsindikator eines Verfahrens dar, da sie zeigen, dass sowohl die interne Bewertungslogik als auch die operative Lösungsfindung effektiv funktionieren. Ein hoher Anteil tatsächlich fehlerfreier Lösungen deutet darauf hin, dass

der Algorithmus in der Lage ist, konsistente, realisierbare Pläne zu erzeugen und diese zugleich intern als solche zu identifizieren. Dies ist insbesondere für den Einsatz in autonomen Planungsumgebungen von Bedeutung, in denen keine externe Verifikation mehr erfolgt und die Qualität der internen Bewertung unmittelbar über die Praxistauglichkeit entscheidet. Darüber hinaus lassen sich aus dem Verhältnis zwischen True Positives und False Positives Rückschlüsse auf die Sensitivität und Spezifität der internen Bewertungsmechanismen ziehen: Ein hoher True-Positive-Wert bei gleichzeitig niedrigem False-Positive-Anteil zeigt, dass das System zuverlässig zwischen gültigen und fehlerhaften Lösungen differenziert und damit ein hohes Maß an interner Kohärenz besitzt. Die Berechnung dieser Kennzahl erfolgt analog zur False-Positive-Rate, indem alle Fälle, in denen eine Lösung intern als fehlerfrei (nicht hard-Constraint verletzend) und extern ebenfalls als constraint-konform bewertet wird, ins Verhältnis zur Gesamtzahl der erzeugten Lösungen gesetzt werden. Dieser Wert dient damit nicht nur als Qualitätsmaß für die Lösungsfindung selbst, sondern auch als Indikator für die Robustheit der internen Bewertung und die tatsächliche Anwendungsreife des Systems.

Insgesamt bilden diese Kennwerte eine umfassende Datengrundlage zur Analyse der entwickelten Optimierungsverfahren. Durch die Kombination von internen und externen Bewertungen wird eine mehrdimensionale Betrachtung ermöglicht, die sowohl die technische Effizienz als auch die inhaltliche Korrektheit der Lösungen berücksichtigt. Die Auswertung dieser Daten liefert damit den empirischen Rahmen, um in Kapitel 6 die Stärken, Schwächen und Grenzen der einzelnen Ansätze differenziert zu diskutieren.

4.2 Experimentaufbau

Der Aufbau der Experimente erfolgte mit dem Ziel, die in Kapitel 4.1 beschriebenen Metriken unter kontrollierten und reproduzierbaren Bedingungen zu erheben. Dabei wurde ein einheitlicher Rahmen geschaffen, der es ermöglicht, die unterschiedlichen Implementierungen systematisch miteinander zu vergleichen. Die Experimente sind so konzipiert, dass sowohl die Effizienz als auch die Stabilität und Ergebnisqualität der Verfahren quantifiziert werden können. Grundlage für alle Versuche bildet eine identische Problemstruktur in Form der drei zuvor aufgeführten Szenarien. Diese Szenarien wurden als Eingabeobjekte definiert und spiegeln jeweils die Eigenschaften der Domäne wider, etwa Ressourcenbeschränkungen, Zuordnungsregeln und spezifische harte und weiche Constraints.

Vorbereitung der Szenarien Für jedes Szenario wurden exemplarische Musterpläne erstellt (siehe Anhang A.1), die entweder reale Strukturen mit fiktiven erweiterten Einschränkungen oder aber komplett fiktive Szenarien abbilden. Ziel dieser zweigleisigen Konstruktion war es, einerseits eine praxisnahe Ausgangslage zu schaffen und andererseits die Verfahren unter Bedingungen mit hoher Constraint-Dichte und einer niedrigen Anzahl möglicher Lösungen zu testen. Die fachliche Beschreibung der einzelnen Szenarien erfolgte bereits in Kapitel 3.1 und wird daher hier nicht nochmal beschreibend ergänzt.

Implementierungsseitige Eingaben Die vier Implementierungen erhielten unterschiedliche, auf ihre Funktionsweise abgestimmte Eingabeformate:

- Das **Prompting**-Verfahren nutzte ausschließlich natürlichsprachliche Texteingaben, welche das Problem, die relevanten Constraints sowie die gewünschte JSON-Ausgabeform strukturiert beschrieben. Hier wurden die Mapping-Beschreibungen in textueller Form oder in vereinfachten JSONS übergeben, damit diese in natürlicher Sprache lesbar sind.
- Bei **EoS** und **EoH** wurden die Szenarien in maschinenlesbarer Form (JSON) übergeben, welche sich an den vordefinierten Rahmen der Implementierung halten. Ergänzend enthielten die Prompts textuelle Instruktionen zur jeweiligen Aufgabenstellung, Beschreibung der Eingabewerte oder der globalen Constraints.
- Der **Evolutionäre Algorithmus** nutzte abgeänderte Definitionen, welche nicht die gesamte Struktur, sondern nur das Mapping beschreiben. Zusätzlich wurden hier alle zuvor aufgeführten (Kapitel 3.2.4) textuellen Beschreibungen ergänzt.

Die vollständigen Konfigurationsdateien und Quellcodes sind im zugehörigen GitHub-Repository dokumentiert. Dort finden sich auch die in dieser Arbeit verwendeten Testdaten und die Rohresultate der Messungen.

Parametrisierung der Verfahren Ein wesentlicher Bestandteil des Experimentaufbaus war die konsistente Parametrisierung aller Verfahren. Da insbesondere bei evolutionären Verfahren Parameter wie Populationsgröße, Mutationswahrscheinlichkeit und Generationenlimit maßgeblich das Konvergenzverhalten und die Ergebnisqualität beeinflussen, wurde eine ausgewogene Einstellung gewählt, die den Charakter und die Laufzeitbeschränkungen der jeweiligen Methode berücksichtigt. Für **EoS** und **EoH** wurden identische Parameter verwendet, um eine direkte Vergleichbarkeit zu gewährleisten:

- Maximale Generationenanzahl: 5
- Populationsgröße: 6
- Anzahl der Elternchromosomen: 2
- Mutationswahrscheinlichkeit: 0.3 (30%)

Darüber hinaus wurde bei EoH ein Zeitlimit von sechs Sekunden pro generierter Funktion eingeführt. Diese Beschränkung verhindert, dass vom LLM erzeugte Codefragmente endlose Berechnungen verursachen und stellt sicher, dass alle Ansätze in vergleichbaren Laufzeitgrenzen operieren. Für den Evolutionären Algorithmus wurden abweichende Parameter gewählt, da hier deutlich seltener LLM-Aufrufe stattfinden und häufige Aufrufe des generierten Codes vertretbar sind:

- Maximale Generationenanzahl: 250
- Populationsgröße: 30
- Anzahl der Elternchromosomen: 2 (Ist in der Implementierung nicht zu parametrisieren, da einfachheitshalber immer von zwei Elternchromosomen ausgegangen wird)
- Mutationswahrscheinlichkeit: 0.3 (30%)

Zusätzlich wurde hier ein Gewichtungsfaktor eingeführt, der bestimmt, in welchem Verhältnis weiche Constraints in die Fitnessbewertung einfließen. Während bei EoS und EoH die Gewichtung der Constraints dem LLM überlassen bleibt, basiert sie beim EA auf vordefinierten festen Werten. Dieser Wert ist auf 0.3 (30%) gesetzt und besagt, dass die Anzahl der Fehler für weiche Constraints nur mit einer 30% Gewichtung in die Fitness-Funktion mit eingeht. Ein weiterer Parameter betrifft den sogenannten *Offspring-Faktor*, welcher festlegt, wie viele Nachkommen pro Generation erzeugt werden. Im EA kann dieser Wert dynamisch angepasst werden, während EoS und EoH ein fixes Verhältnis von zwei Nachkommen pro Elternpaar verwenden. Die geringere Anzahl resultiert hier aus der hohen Kostenstruktur der LLM-Aufrufe, da bei jeder Kreuzungsoperation ein neuer LLM-Request ausgelöst wird. Für den EA ist dieser Wert für die Tests auf 7 gesetzt.

Durchführung der Experimente Jede Implementierung wurde mehrfach pro Szenario ausgeführt, um zufallsbedingte Schwankungen statistisch zu glätten. Die Anzahl der Wiederholungen wurde in Abhängigkeit von der Komplexität und Laufzeit der jeweiligen Methode gewählt. Tabelle 1 zeigt die ausgeführten Läufe im Überblick.

Tabelle 1: Anzahl der ausgeführten Runs pro Szenario und Implementierung

Implementierung	**Schule**	**Zugplanung**	**Maschinen**
Prompting	12	12	10
Evolution of Schedules (EoS)	6	6	6
Evolution of Heuristics (EoH)	6	6	6
Evolutionärer Algorithmus (EA)	17	17	15

Die geringere Anzahl an Läufen bei EoS und EoH ergibt sich aus den hohen Rechenkosten der LLM-Aufrufe innerhalb der Iterationen, während der EA aufgrund seiner gleichbleibenden Code-Bestandteile und kürzeren Laufzeiten eine höhere Anzahl an Wiederholungen erlaubt. Alle Tests wurden unter identischen Rahmenbedingungen durchgeführt, um externe Einflüsse auszuschließen. Dazu zählen sowohl die verwendete Hardware als auch die API-Version und die Temperatureinstellung des eingesetzten LLMs. Für die Tests wurde das LLM von OpenAI mit der GPT-Version *gpt-5-mini-2025-08-07* verwendet. Der genutzte REST-Call war folgender: *https://api.openai.com/v1/chat/completions*.

Vergleich der Implementierungen mittels Friedman-Test Zur statistischen Bewertung der Unterschiede zwischen den Implementierungen wird im Anschluss an die Experimente der in Kapitel 2.4.1 beschriebene Friedman-Test angewendet. Dieser nichtparametrische Rangtest eignet sich besonders für den Vergleich mehrerer Algorithmen über verschiedene Szenarien und Metriken hinweg. Für jede der in Kapitel 4.1 definierten Kennzahlen (z.B. Fitness, Fehleranzahl, Konvergenzgeschwindigkeit und Lösungsdiversität) werden die Resultate pro Implementierung in Ränge überführt und anschließend mittlere Rangwerte berechnet. Die Teststatistik des Friedman-Verfahrens überprüft dann die Nullhypothese, dass alle Implementierungen im Mittel dieselbe Leistung zeigen. Ein signifikant hoher Testwert weist darauf hin, dass mindestens eine Implementierung in einer oder mehreren Metriken statistisch überlegen ist. Damit liefert der Friedman-Test

eine objektive Grundlage für die Bewertung der in dieser Arbeit entwickelten Ansätze und ermöglicht eine quantitative Beantwortung der Forschungsfragen im Hinblick auf Effizienz und Ergebnisqualität.

Der beschriebene Aufbau ermöglicht eine methodisch fundierte Gegenüberstellung der vier Ansätze. Durch die standardisierte Parametrisierung und identische Eingabeszenarien kann analysiert werden, wie die LLM-basierten Algorithmen für komplexe Optimierungsprobleme Lösungen finden und bewerten können. Die Ergebnisse der Experimente werden im folgenden Kapitel 5 dargestellt.

5 Ergebnisse

Dieses Kapitel fasst die empirischen Resultate der durchgeführten Experimente zusammen und bewertet sie im Hinblick auf die in Kapitel 4.1 beschriebenen Kennzahlen. Dazu wird in Abschnitt 5.1 zunächst die Datenerhebung vorgestellt, welche die gemessenen Werte für die verschiedenen Szenarien enthält. In Abschnitt 5.2 erfolgt anschließend die statistische Auswertung der Ergebnisse anhand des Friedman-Tests.

5.1 Datenerhebung

Aus den zuvor beschriebenen Testläufen wurden die in Kapitel 4.1 definierten Kennzahlen aggregiert und als Mittelwerte in den folgenden Tabellen zusammengefasst. Die zugrundeliegenden Einzelergebnisse sind in den jeweiligen Anhängen zu finden (Anhang A.2.1, A.2.2, A.2.3).

Tabelle 2: Ergebnisse Szenario Schulplanung (Mittelwerte über alle durchgeführten Testläufe)

Szenario: Schulplanung				
Kennzahl	**Prompting**	**EoS**	**EoH**	**EA**
Anzahl LLM-Aufrufe	2	90,33	99	31,06
Avg. Fitness	0,38	0,08	0,05	0,37
Avg. Fehler nach Skript	260,75	61,72	64,88	0,72
Diff. intern/extern Fitness	260,38	61,64	64,83	0,35
False Positive	0,92	0,33	0,83	0,29
True Positive	0,0	0,0	0,0	0,41
Diversität der Lösungen	0,75	0,72	0,91	0,83

Die Werte in Tabelle 2 zeigen die durchschnittlich erreichte Fitness und Fehlerrate über alle Iterationen der jeweiligen Verfahren. Bei der Implementierung des Prompting-Verfahrens traten vermehrt Validierungsfehler auf, die durch unvollständige oder fehlerhafte Planstrukturen im Ergebnis verursacht wurden. Um die Vergleichbarkeit der Datensätze dennoch sicherzustellen, wurde in diesen Fällen ein pauschaler Fehlerwert von 300 als Platzhalter hinterlegt.

Tabelle 3: Ergebnisse Szenario Zugplanung (Mittelwerte über alle durchgeführten Testläufe)

Szenario: Zugplanung				
Kennzahl	**Prompting**	**EoS**	**EoH**	**EA**
Anzahl LLM-Aufrufe	2	69,33	92	47,24
Avg. Fitness	2,61	2,64	0,76	0,39
Avg. Fehler nach Skript	54,82	2,92	3,52	1,36
Diff. intern/extern Fitness	52,21	2,69	2,76	1,05
False Positive	0,5	0,17	0	0,18
True Positive	0,0	0,0	0,0	0,59
Diversität der Lösungen	0,76	0,69	0,84	0,80

Die in Tabelle 3 aufgeführten Durchschnittswerte zeigen die konsolidierten Resultate aller Versuche im Kontext der Zugplanung. Die Erhebung erfolgte auf Grundlage der in Anhang A.2.2 dargestellten Einzeltabellen. Die Kennzahlen spiegeln die durchschnittliche Leistung der vier Implementierungen über die durchgeführten Läufe wider.

Tabelle 4: Ergebnisse Szenario Maschinenplan (Mittelwerte über alle durchgeführten Testläufe)

Szenario: Maschinenplan				
Kennzahl	**Prompting**	**EoS**	**EoH**	**EA**
Anzahl LLM-Aufrufe	2	74,67	90	31,07
Avg. Fitness	3,16	1,45	0,88	6,17
Avg. Fehler nach Skript	35,1	33,7	16,27	10,23
Diff. intern/extern Fitness	31,94	32,25	15,39	7,73
False Positive	0,4	0	0	0,07
True Positive	0,0	0,0	0,0	0,2
Diversität der Lösungen	0,48	0,72	0,87	0,88

Die Werte in Tabelle 4 basieren auf der im Anhang A.2.3 dokumentierten Datengrundlage. Für jede Implementierung wurden die Resultate der einzelnen Testläufe gemittelt, um ein konsistentes Bild der durchschnittlichen Leistungsfähigkeit zu erhalten.

Abschließend wurden die Mittelwerte aller Kennzahlen über alle drei Szenarien hinweg aggregiert. Dies ermöglicht einen übergeordneten Vergleich der vier Verfahren hinsichtlich ihres allgemeinen Verhaltens.

Tabelle 5: Aggregierte Durchschnittswerte je Implementierung über alle Szenarien

Aggregierte Werte				
Kennzahl	**Prompting**	**EoS**	**EoH**	**EA**
Anzahl LLM-Aufrufe	2	78,11	93,67	36,46
Avg. Fitness	2,2	1,39	0,56	6,93
Avg. Fehler nach Skript	116,89	32,78	28,22	4,1
Diff. intern/extern Fitness	114,84	32,19	27,66	3,04
False Positive	0,61	0,17	0,28	0,18
True Positive	0,0	0,0	0,0	0,4
Diversität der Lösungen	0,67	0,71	0,87	0,84

Die aggregierten Kennzahlen stellen eine zusammenfassende Übersicht der über alle Szenarien gemittelten Werte dar und dienen als Grundlage für einen bewertenden Vergleich der verschiedenen Verfahren.

Zusätzlich wurde für jeden Algorithmus (EoS, EoH, EA) ein Konvergenzdiagramm erstellt. Dieses zeigt übergreifend über alle Szenarien den durchschnittlichen Fitnesswert je Generation. Zu beachten ist hierbei, dass die Konvergenz für verschiedenste Durchläufe zählt. Wenn ein Durchlauf zum Zeitpunkt einer Generation bereits ein vermeitlich optimales Ergebnis gefunden hat, so wird der Durchlauf ab der Generation in den Durchschnitt nicht mehr mit eingerechnet. So kann es potenziell wieder einen höheren Schnitt in der nächsten Generation geben. Die Graphen sind in Abbildung 6, 7 und 8 dargestellt:

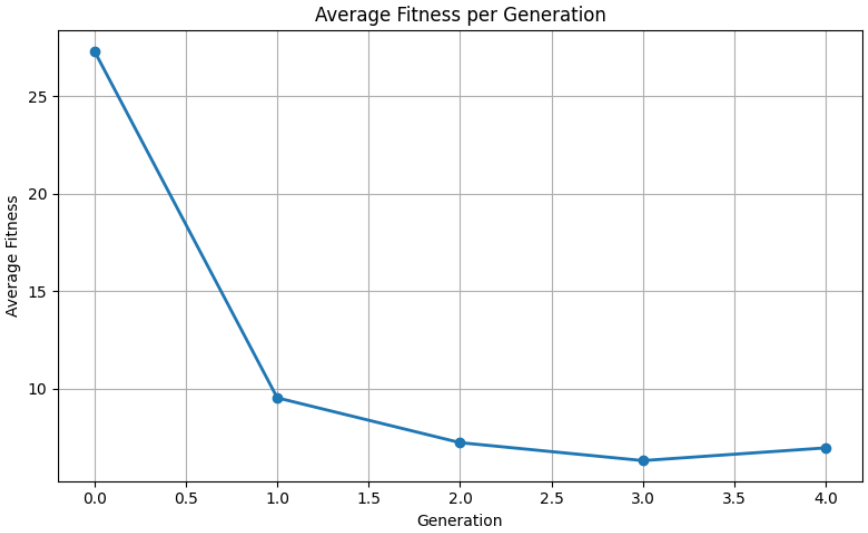

Abbildung 6: Konvergenzdiagramm für EoS

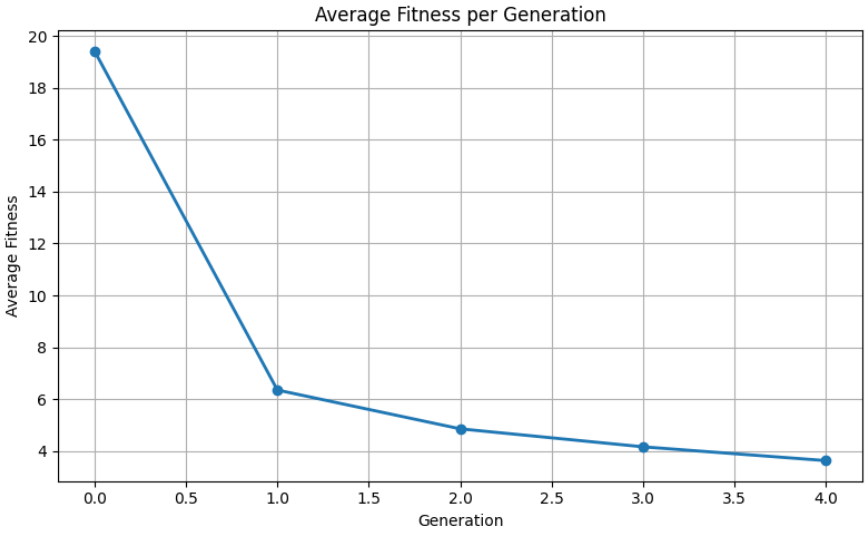

Abbildung 7: Konvergenzdiagramm für EoH

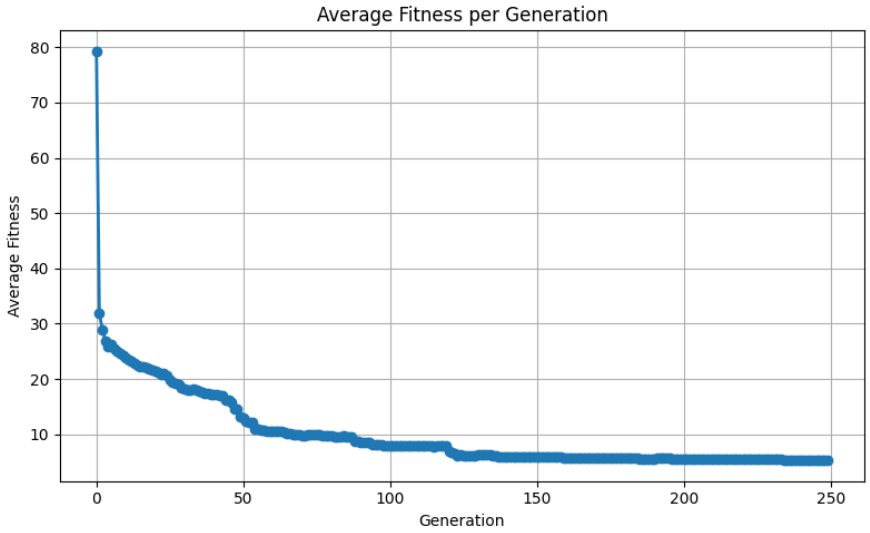

Abbildung 8: Konvergenzdiagramm für EA

5.2 Datenauswertung

Zur quantitativen Auswertung der erhobenen Ergebnisse wurde der in Kapitel 2.4.1 vorgestellte Friedman-Test verwendet. Dieser dient als nichtparametrisches Verfahren zum Vergleich mehrerer verbundener Stichproben und ermöglicht eine Beurteilung, ob sich die betrachteten Implementierungsansätze in ihrer Leistungsfähigkeit signifikant voneinander unterscheiden. Da im Rahmen dieser Arbeit vier verschiedene Verfahren untersucht wurden (*Prompting*, *EoS*, *EoH*, *EA*), eignet sich der Friedman-Test besonders, um Unterschiede zwischen diesen Gruppen statistisch zu prüfen, ohne Normalverteilungsannahmen treffen zu müssen.

Der Test wurde für jedes Szenario separat durchgeführt, um Unterschiede innerhalb der jeweiligen Anwendungsdomäne zu identifizieren. Die einzelnen Berechnungen basieren jeweils auf den in Abschnitt 5.1 dargestellten Mittelwerten der Kennzahlen. Für jede Kennzahl wurde pro Szenario eine Rangordnung erstellt, wobei der beste Wert den Rang 1 erhält. Anschließend wurde aus den Rangwerten der Mittelrang je Verfahren bestimmt, um die Teststatistik gemäß Kapitel 2.4.1 zu berechnen.

Bei der Berechnung der durchschnittlichen Ränge wurde das Bewertungskriterium der Anzahl der LLM-Aufrufe nicht mit berücksichtigt. Der Grund hierfür ist, dass durch die Gegebenheit des Promptings diese Implementierung immer den besten Wert in dieser

Kategorie hat. Dadurch verzerrt die Darstellung dieses Wertes das Ergebnis in Bezug auf die tatsächliche Leistungsfähigkeit. Bei der späteren Analyse der Ergebnisse werden die dort angegebenen Messwerte jedoch mit berücksichtigt.

Friedman-Test Szenario Schulplanung Im Szenario der Schulplanung wird mithilfe des Friedman-Tests überprüft, ob zwischen den vier untersuchten Verfahren signifikante Unterschiede in der Leistungsfähigkeit bestehen (vgl. Kapitel 2.4.1). Grundlage der Berechnung bilden die in Tabelle 2 dargestellten Durchschnittswerte der zentralen Kennzahlen. Die daraus abgeleiteten Rangwerte sind in Tabelle 6 zusammengefasst. Wie bereits in Kapitel 2.4.1 erläutert, gilt ein niedriger Rang als Indikator für eine bessere Leistung des jeweiligen Verfahrens – unabhängig davon, ob eine Kennzahl minimiert oder maximiert wird.

Tabelle 6: Friedman-Test Rangwerte Szenario Schulplanung

Kennzahl	**Prompting**	**EoS**	**EoH**	**EA**
Avg. Fitness	4	2	1	3
Avg. Fehler nach Skript	4	2	3	1
Diff. intern/extern Fitness	4	2	3	1
False Positive	4	2	3	1
True Positive	3	3	3	1
Diversität der Lösungen	3	4	1	2
Mittlerer Rang	3,67	2,5	2,33	1,5

Die mittleren Ränge ergeben sich aus dem arithmetischen Mittel der einzelnen Rangwerte und dienen als Grundlage für die anschließende Berechnung der Teststatistik, die im Friedman-Test die zentrale Entscheidungsgröße bildet. Zur Ermittlung dieser Teststatistik wird folgende Gleichung verwendet:

$$\chi_F^2 = \frac{12N}{k(k+1)} \left(\sum_{i=1}^{k} \bar{R}_i^2 - \frac{k(k+1)^2}{4} \right)$$

Hierbei bezeichnet $N = 6$ die Anzahl der betrachteten Kennzahlen und $k = 4$ die Anzahl der verglichenen Verfahren. Die Teststatistik erfasst somit die Varianz der Rangwerte zwischen den Verfahren. Ein hoher χ_F^2-Wert weist auf deutliche Unterschiede in

den Rangverteilungen hin, während ein niedriger Wert eine ähnliche Leistungsfähigkeit signalisiert.

Für die Summe der quadrierten mittleren Ränge ergibt sich:

$$\sum R_j^2 = 3{,}67^2 + 2{,}5^2 + 2{,}33^2 + 1{,}5^2 = 27{,}4$$

Eingesetzt in die Formel ergibt sich damit:

$$\chi_F^2 = \frac{12 \cdot 6}{4 \cdot 5} \cdot \left(27{,}4 - \frac{4 \cdot (4+1)^2}{4}\right) = 3{,}6 \cdot (27{,}4 - 25) = 8{,}64.$$

Damit beträgt die berechnete Teststatistik $\chi_F^2 = 8{,}64$. Um diese in einen F-Wert zu überführen und mit den kritischen Werten der F-Verteilung vergleichen zu können, wird folgende Umrechnungsformel angewandt:

$$F_F = \frac{(N-1)\chi_F^2}{N(k-1) - \chi_F^2} = \frac{5 \cdot 8{,}64}{18 - 8{,}64} = 4{,}62.$$

Die Freiheitsgrade betragen $df_1 = k - 1 = 3$ und $df_2 = (k-1)(N-1) = 12$. Für ein Signifikanzniveau von $\alpha = 0{,}05$ ergibt sich daraus der kritische Vergleichswert:

$$F_{krit} = F_{(3;15;0{,}05)} = 3{,}29.$$

Da $F_F = 4{,}62 > F_{krit} = 3{,}29$, ist das Ergebnis signifikant. Somit bestehen statistisch messbare Unterschiede zwischen den getesteten Verfahren im Szenario der Schulplanung.

Im Anschluss wird der Nemenyi-Post-hoc-Test durchgeführt, um zu bestimmen, zwischen welchen Verfahren diese signifikanten Unterschiede bestehen. Die Berechnung der sogenannten *Critical Difference (CD)* erfolgt nach folgender Gleichung:

$$CD = q_\alpha \sqrt{\frac{k(k+1)}{6N}} = 2{,}569 \sqrt{\frac{4 \cdot 5}{6 \cdot 6}} = 2{,}569 \cdot 0,75 \approx 1{,}91$$

Für ein Signifikanzniveau von 5 % wurde $q_\alpha = 2{,}569$ herangezogen. Die Differenzen der mittleren Ränge der Verfahren werden nun mit dem errechneten CD-Wert verglichen. Liegt die Differenz über dieser Schwelle, besteht ein signifikanter Unterschied. Liegt sie darunter, ist dieser nicht signifikant. Die Ergebnisse sind in Tabelle 7 dargestellt.

Tabelle 7: Vergleich der Algorithmen im Schulplanungs-Szenario

	Prompting	**EoS**	**EoH**	**EA**
Prompting	x	1,17	1,34	**2,17**
EoS	x	x	0,17	1
EoH	x	x	x	0,83

Das Ergebnis zeigt, dass lediglich der evolutionäre Algorithmus (EA) signifikant besser abschneidet als der Prompting-Ansatz. Zwischen den übrigen Verfahren bestehen zwar Unterschiede, diese liegen jedoch unterhalb der berechneten CD-Schwelle und sind somit statistisch nicht signifikant. Dieses Ergebnis bestätigt, dass der EA im Schulplanungs-Szenario die robusteste und konsistenteste Leistung erbringt, während die übrigen Verfahren trotz teils ähnlicher Rangwerte keine signifikant unterschiedlichen Resultate liefern.

Friedman-Test Szenario Zugplanung Analog zur Schulplanung wird auch für das Zugplanungsszenario ein Friedman-Test durchgeführt, um die Leistungsunterschiede der vier betrachteten Verfahren zu überprüfen (vgl. Kapitel 2.4.1). Grundlage sind die in Tabelle 3 aufgeführten Durchschnittswerte der zentralen Kennzahlen. Die daraus abgeleiteten Rangwerte sind in Tabelle 8 dargestellt. Auch hier gilt, dass ein niedriger Rang für eine bessere Bewertung des jeweiligen Verfahrens steht.

Tabelle 8: Friedman-Test Rangwerte Szenario Zugplanung

Kennzahl	**Prompting**	**EoS**	**EoH**	**EA**
Avg. Fitness	3	4	2	1
Avg. Fehler nach Skript	4	2	3	1
Diff. intern/extern Fitness	4	2	3	1
False Positive	4	3	1	2
True Positive	3	3	3	1
Diversität der Lösungen	3	4	1	2
Mittlerer Rang	3,5	3,0	2,17	1,33

Die mittleren Ränge ergeben sich aus dem arithmetischen Mittel der Rangwerte je Verfahren und dienen als Basis für die Berechnung der Friedman-Teststatistik. Diese bewertet, ob zwischen den Verfahren eine signifikante Varianz in den Rängen besteht. Die Berechnung erfolgt analog zur vorherigen Auswertung nach folgender Formel:

$$\chi_F^2 = \frac{12N}{k(k+1)} \left(\sum_{i=1}^{k} \bar{R}_i^2 - \frac{k(k+1)^2}{4} \right)$$

Mit $N = 6$ Kennzahlen und $k = 4$ Verfahren ergibt sich für die Summe der quadrierten mittleren Ränge:

$$\sum R_j^2 = 3{,}5^2 + 3{,}0^2 + 2{,}17^2 + 1{,}33^2 = 27{,}73$$

Eingesetzt in die Formel ergibt sich:

$$\chi_F^2 = \frac{12 \cdot 6}{4 \cdot 5} \cdot \left(27{,}73 - \frac{4 \cdot (4+1)^2}{4} \right) = 3{,}6 \cdot (27{,}73 - 25) = 9{,}83.$$

Damit beträgt die berechnete Teststatistik $\chi_F^2 = 9{,}83$. Zur Umrechnung in einen F-Wert, der den Vergleich mit den kritischen F-Verteilungswerten erlaubt, wird folgende Gleichung verwendet:

$$F_F = \frac{(N-1)\chi_F^2}{N(k-1)-\chi_F^2} = \frac{5 \cdot 9{,}83}{18-9{,}83} = 6{,}02.$$

Mit den Freiheitsgraden $df_1 = k-1 = 3$ und $df_2 = (k-1)(N-1) = 15$ ergibt sich für ein Signifikanzniveau von $\alpha = 0{,}05$:

$$F_{krit} = F_{(3;15;0{,}05)} = 3{,}29.$$

Da $F_F = 6{,}02 > F_{krit} = 3{,}29$, besteht ein signifikanter Unterschied zwischen den getesteten Verfahren im Zugplanungsszenario. Dieses Ergebnis unterstreicht, dass die Leistungsunterschiede hier leicht deutlicher als im Schulplanungsfall ausfallen.

Zur Bestimmung, welche Verfahren sich signifikant voneinander unterscheiden, wird anschließend der Nemenyi-Post-hoc-Test angewendet. Die Critical Difference ist hier analog zu jener im Schulplanungsszenario:

$$CD \approx 1{,}91$$

Die Differenzen der mittleren Ränge werden mit dieser Schwelle verglichen. Übersteigt die Differenz den CD-Wert, liegt ein signifikanter Unterschied vor. Bleibt sie darunter, ist der Unterschied statistisch nicht signifikant. Tabelle 9 zeigt die entsprechenden Werte.

Tabelle 9: Vergleich der Algorithmen im Zugplanungs-Szenario

	Prompting	**EoS**	**EoH**	**EA**
Prompting	x	0,5	1,33	**2,17**
EoS	x	x	0,83	1,67
EoH	x	x	x	0,84

Die Ergebnisse zeigen, dass der evolutionäre Algorithmus (EA) im Zugplanungsszenario abermals besser abschneidet als alle anderen Verfahren, insbesondere im Vergleich zum Prompting-Ansatz. Damit bestätigt sich das bereits im vorangegangenen Szenario beobachtete Muster. Der EA liefert konsistent die besten und stabilsten Ergebnisse, während die anderen Verfahren nicht gleichwertig in der Leistung sind.

Friedman-Test Szenario Maschinenplanung Für das Szenario der Maschinenplanung wurde ebenfalls ein Friedman-Test durchgeführt, um mögliche Unterschiede in der Leistungsfähigkeit der vier Verfahren zu identifizieren (vgl. Kapitel 2.4.1). Grundlage der Berechnung bilden die in Tabelle 4 dargestellten Durchschnittswerte der zentralen Kennzahlen. Die daraus abgeleiteten Rangwerte sind in Tabelle 10 aufgeführt. Auch hier gilt, dass ein niedriger Rang für eine bessere Bewertung des jeweiligen Verfahrens steht.

Tabelle 10: Friedman-Test Rangwerte Szenario Maschinenplanung

Kennzahl	**Prompting**	**EoS**	**EoH**	**EA**
Avg. Fitness	3	2	1	4
Avg. Fehler nach Skript	4	3	2	1
Diff. intern/extern Fitness	3	4	2	1
False Positive	4	1	1	3
True Positive	3	3	3	1
Diversität der Lösungen	4	3	2	1
Mittlerer Rang	3,5	2,67	1,83	1,83

Die mittleren Ränge ergeben sich aus dem arithmetischen Mittel der Rangwerte je Verfahren. Wie in den vorherigen Szenarien dient diese Zusammenfassung als Grundlage für die Berechnung der Teststatistik, mit der geprüft wird, ob zwischen den Verfahren signifikante Unterschiede bestehen. Die Berechnung erfolgt nach folgender Formel:

$$\chi_F^2 = \frac{12N}{k(k+1)} \left(\sum_{i=1}^{k} \bar{R}_i^2 - \frac{k(k+1)^2}{4} \right)$$

Mit $N = 5$ Kennzahlen und $k = 4$ Verfahren ergibt sich für die Summe der quadrierten mittleren Ränge:

$$\sum R_j^2 = 3{,}5^2 + 2{,}67^2 + 1{,}83^2 + 1{,}83^2 = 26{,}08$$

Eingesetzt in die Formel ergibt sich:

$$\chi_F^2 = \frac{12 \cdot 6}{4 \cdot 5} \cdot \left(26{,}08 - \frac{4 \cdot (4+1)^2}{4} \right) = 3{,}6 \cdot 26{,}08 - 25 = 3{,}89.$$

Damit beträgt die berechnete Teststatistik $\chi_F^2 = 3{,}89$. Zur Umrechnung in einen F-Wert, der den Vergleich mit den kritischen F-Verteilungswerten erlaubt, wird folgende Formel verwendet:

$$F_F = \frac{(N-1)\chi_F^2}{N(k-1) - \chi_F^2} = \frac{5 \cdot 3{,}89}{18 - 3{,}89} = 1{,}38.$$

Für ein Signifikanzniveau von $\alpha = 0{,}05$ und die Freiheitsgrade $df_1 = 3$ sowie $df_2 = 15$ ergibt sich der kritische Vergleichswert:

$$F_{krit} = F_{(3;15;0,05)} = 3{,}29.$$

Da $F_F = 1{,}38 < F_{krit} = 3{,}29$ besteht zwischen den einzelnen Algorithmen kein signifikanter Unterschied. Damit kann in diesem Szenario nicht gesagt werden, dass eine der Implementierungen statistisch signifikant besser als andere ist.

Die weiteren Schritte bezüglich Nemenyi-Post-hoc-Tests zum direkten Vergleich der einzelnen Algorithmen können damit ausgelassen werden. Das Resultat wäre hier deckungsgleich und würde darauf hinauslaufen, dass keine signifikanten Unterschiede zwischen einzelnen Lösungen existieren.

Insgesamt ergibt sich aus den Friedman-Tests somit, dass im Kontext der Schulplanung und der Zugplanung der EA deutlich besser ist als die reine Prompting Variante. EoH und EoS hingegen weisen keine signifikante Verbesserung der Resultate auf. Bei dem Maschinenszenario ist der Unterschied nach den durchgeführten Tests nicht signifikant genug, als dass eine Implementierung als ausreichend besser deklariert werden könnte. Die daraus resultierenden Schlüsse und Implikationen für den Praktischen Einsatz werden in dem kommenden Kapitel 6 aufgeführt.

6 Bewertung und Interpretation der Ergebnisse

Das folgende Kapitel ordnet die in Kapitel 5 dargestellten Resultate inhaltlich ein und interpretiert sie im Hinblick auf die Forschungsfragen dieser Arbeit. Ziel ist es, die beobachteten Leistungsunterschiede zwischen den Implementierungen zu erklären, übergreifende Muster zu identifizieren und mögliche Ursachen für Abweichungen zu diskutieren. Dabei wird auch auf Limitationen des Studiendesigns sowie auf Implikationen für zukünftige Forschung eingegangen.

6.1 Überblick über die zentralen Befunde

Die in Kapitel 5.2 dargestellten Ergebnisse des Friedman-Tests verdeutlichen, dass sich die vier untersuchten Verfahren in zwei der drei getesteten Szenarien statistisch signifikant voneinander unterscheiden. Sowohl in der Schul- als auch in der Zugplanung liegen die berechneten F_F-Werte über dem kritischen Grenzwert ($F_{\text{krit}} = 3{,}29$), während das Maschinenszenario abweichend keinen signifikanten Unterschied aufweist ($F_F = 1{,}38 < F_{\text{krit}}$). Insgesamt ergibt sich in nahezu allen Metriken und Szenarien dasselbe relative Leistungsranking: *Prompting* < *EoS* < *EoH* < *EA*. Lediglich im Maschinenszenario zeigen *EoH* und *EA* identische mittlere Ränge, was die insgesamt hohe Stabilität dieser Rangfolge unterstreicht.

Übergreifend zeigt sich, dass der evolutionäre Algorithmus in zwei der drei Szenarien signifikant bessere Ergebnisse als das reine Prompting erzielt. Darüber hinaus ist er das einzige Verfahren, das in der Lage ist, tatsächlich gültige Lösungen zu generieren, die sowohl vom Algorithmus selbst als auch vom externen Prüfsystem als korrekt erkannt werden (True Positives, vgl. Kapitel 5.1). Dies verdeutlicht, dass der EA eine deutlich höhere Problemlösungsfähigkeit aufweisen als das reine Prompting oder EoH bzw. EoS.

Das reine Prompting erweist sich dagegen als grundsätzlich ungeeignet zur Lösung komplexer Planungsprobleme. Die erzeugten JSON-Strukturen enthalten in allen Szenarien eine hohe Fehlerquote und weisen erhebliche Abweichungen zwischen interner (LLM-basierter) und externer Bewertung auf. Diese Diskrepanz zeigt sich auch bei den hybriden Varianten EoS und EoH, die sich in vielen Fällen selbst signifikant überschätzen. Während die vom LLM generierten Selbsteinschätzungen gute Fitnesswerte suggerieren, offenbart die objektive Bewertung durch das Skript deutliche Struktur- und Konsistenzfehler. Dies verdeutlicht, dass diese Verfahren zwar lernfähiger und dynamischer sind als

reines Prompting, jedoch noch keine zuverlässige interne Validierungskompetenz besitzen.

Trotz dieser Schwächen zeigen EoS und EoH in allen Metriken eine deutliche Verbesserung gegenüber dem reinen Prompting, auch wenn diese Unterschiede statistisch nicht signifikant ausfallen. Sie profitieren insbesondere von der iterativen Struktur und der Nutzung evolutionärer Prinzipien, wodurch die Lösungsqualität und Plausibilität zunimmt. Gleichzeitig verursachen sie durch ihre hohe Zahl an LLM-Aufrufen (zwischen 70 und 100 pro Lauf) einen signifikant höheren Ressourcenaufwand im Vergleich zu EA, der mit deutlich weniger Aufrufen auskommt.

Ein weiterer zentraler Befund betrifft die Lösungsdiversität. Über alle Verfahren hinweg zeigt sich eine hohe Varianz zwischen den generierten Antworten, was auf eine ausgeprägte kreative Streuung der LLM-Ausgaben hinweist. Diese Vielfalt kann einerseits als Zeichen explorativer Suchfähigkeit interpretiert werden, erschwert andererseits jedoch die Konsistenzbewertung und die Identifikation tatsächlich valider Lösungen.

Insgesamt lässt sich festhalten, dass LLM-basiertes Prompting allein keine geeignete Strategie für die Lösung komplexer Planungsprobleme darstellt. Erst durch die Kombination mit evolutionären Verfahren entstehen robuste und reproduzierbare Optimierungsergebnisse. Während EoS und EoH durch ihre iterativen Feedbackmechanismen zwar Verbesserungen gegenüber dem Prompting erzielen, ist die Implementierung des EAs hinsichtlich Signifikanz, Stabilität und Lösungsvalidität das überlegene Verfahren.

6.2 Interpretation nach Verfahren

In den folgenden Abschnitten werden die Ergebnisse der einzelnen Implementierungsansätze qualitativ interpretiert. Dabei steht jeweils im Fokus, welche Stärken und Schwächen sich aus den gemessenen Leistungskennzahlen, dem beobachteten Verhalten in den Experimenten sowie den Ressourcenanforderungen der Verfahren ableiten lassen.

Prompting (Baseline) Das reine *Prompting* dient in dieser Arbeit als Baseline und Referenzpunkt für die Bewertung der weiterentwickelten hybriden Verfahren. Wie erwartet, schneidet dieser Ansatz in nahezu allen Kennzahlen deutlich schwächer ab. Die generierten Pläne weisen häufig strukturelle Mängel auf: So fehlen in den JSON-Ausgaben teilweise ganze Zeitslots, Zuordnungen oder Pflichtfelder, was zu unvollständigen oder

nicht interpretierbaren Plänen führt. Darüber hinaus treten regelmäßig Schemaabweichungen auf, bei denen beispielsweise die erwartete Verschachtelungstiefe nicht eingehalten oder Schlüssel falsch benannt werden. Solche Fehler deuten darauf hin, dass das LLM zwar in der Lage ist, formale Strukturen zu reproduzieren, aber keine konsistente Kontrolle über komplexe und verschachtelte Objektbeziehungen besitzt.

Die hohe Differenz zwischen der internen, vom LLM selbst eingeschätzten Fitness und der externen, durch das Bewertungsskript bestimmten Fitness bestätigt diese Beobachtung. Das Modell tendiert dazu, eigene Ausgaben systematisch zu überschätzen, was in einer großen Anzahl an *False Positives* resultiert. Diese Diskrepanz zeigt, dass ein LLM ohne rekursive Rückkopplung oder formales Validierungssystem nicht in der Lage ist, Optimierungsprobleme mit diversen Constraints zuverlässig zu lösen. Das reine Prompting kann daher allenfalls als explorativer Ansatz zur Initialisierung oder für einfache, schwach constraint-basierte Probleme herangezogen werden, nicht jedoch für komplexe Timetabling-Aufgaben.

Evolution of Schedules (EoS) Der Ansatz Evolution of Schedules stellt die erste iterative Erweiterung des Prompting-Verfahrens dar. Hier wird die generative Fähigkeit des LLMs genutzt, um in mehreren Iterationen aus bestehenden JSON-Plänen neue, verbesserte Varianten zu erzeugen. Die Ergebnisse zeigen, dass EoS in nahezu allen Szenarien eine deutliche Leistungssteigerung gegenüber der Baseline erzielt. Die durchschnittliche Fitness verbessert sich und auch die extern gemessenen Fehlerwerte sinken spürbar, auch wenn dadurch keine signifikante Verbesserung gegenüber dem Prompting entstehen.

Gleichzeitig steigt der Ressourcenaufwand erheblich. Mit durchschnittlich rund 70 bis 80 LLM-Aufrufen pro Lauf zählt EoS zu den kostenintensivsten Verfahren dieser Arbeit. Ursache hierfür ist, dass in jeder Generation mehrere Varianten eines Plans erzeugt, bewertet und anschließend kombiniert werden. Durch den großen Kontextumfang der Prompts – bestehend aus zwei kompletten JSON-Plänen, Bewertungen und der Problemdefinition – verlängert sich die Laufzeit pro Iteration deutlich.

EoS beweist damit, dass eine iterative Generierung mit Rückkopplung durch Bewertungen das Optimierungspotenzial von LLMs steigert. Trotz höherer Kosten und erheblich geminderter Effizienz ist es jedoch in keinem Szenario in der Lage, ein signifikant besseres Ergebnis zu erzielen.

Evolution of Heuristics (EoH) Der Ansatz Evolution of Heuristics verfolgt im gleichen Verfahren einen grundsätzlich anderen Ansatz: Statt konkrete Pläne zu erzeugen, entwickelt das LLM Heuristiken, die wiederum zur Planerstellung angewendet werden. Dadurch wird die Lernfähigkeit des Modells stärker genutzt, um strategische Entscheidungsregeln anstelle vollständiger Lösungen zu optimieren. Die Ergebnisse zeigen, dass EoH im Durchschnitt bessere Fitnesswerte erzielt als EoS, insbesondere in der externen Bewertung und im mittleren Ranking über alle Szenarien.

Trotz einer allgemein besseren Fitness und einer besseren Einordnung in den Friedman-Tests hat EoH ein immer noch hohen Wert an False Positives. Dies ist insbesondere im Kontext des Schulszenarios auffällig. Dies legt die Schlussfolgerung nahe, dass das LLM nicht in der Lage ist, alle komplexen Anforderungen korrekt zu ergreifen und in einem verschachtelten Plan zu evaluieren.

Allgemein zeigt der Ansatz, dass LLMs nicht nur konkrete Lösungen, sondern auch Meta-Regeln generieren können. Diese tragen zur autonomen Optimierung bei und ermöglichen das Erzeugen von bessere Lösungen als mit dem direkten Generieren von Resultaten. Damit markiert EoH den Übergang von rein generativen hin zu lernfähigen, adaptiven Optimierungssystemen.

Evolutionärer Algorithmus (EA) Der Evolutionäre Algorithmus bildet den methodischen Endpunkt der in dieser Arbeit entwickelten Ansätze. Er kombiniert einen generischen Algorithmus mit gezieltem LLM-Einsatz zur Bewertung und Heuristikgenerierung. Die Ergebnisse zeigen, dass der EA in allen getesteten Szenarien die beste Gesamtleistung erzielt. Er weist den niedrigsten mittleren Rang, die geringste durchschnittliche Fehlerzahl nach externer Bewertung und die höchste Anzahl richtiger Lösungen auf. Zudem ist er deutlich ressourcenschonender als EoS und EoH, da die LLM-Interaktion nur punktuell zur initialen Heuristikgenerierung oder Validierung erfolgt.

Die Kombination aus der Grundstruktur generischer Algorithmen und dem einmaligen Generieren der adaptiven Code-Bestandteile führt zu einer effizienten Lösung, welche über viele große Generationen im Suchraum Lösungen identifizieren kann. Während das LLM als Wissenskomponente genutzt wird, um qualitativ hochwertige Ausgangspunkte oder Bewertungen bereitzustellen, sorgt der evolutionäre Mechanismus für die gezielte Verbesserung dieser Kandidaten. Dadurch lassen sich sowohl die Stärken klassischer EAs

(Robustheit, Stabilität, Effizienz) als auch die semantischen Fähigkeiten von LLMs (Verständnis komplexer Constraints) nutzen, ohne deren Schwächen (fehlende Konsistenz, hohe Kosten) zu übernehmen.

Insgesamt zeigt der EA, dass eine wohldosierte Integration von LLMs in bestehende evolutionäre Frameworks eine deutliche Leistungssteigerung ermöglichen kann, ohne die Reproduzierbarkeit und Effizienz der klassischen Verfahren zu gefährden. Er bildet damit den einzigen hybriden Ansatz mit einer signifikanten Leistungssteigerung dieser Arbeit und liefert zugleich den methodischen Beleg dafür, dass LLMs den größten Nutzen entfalten, wenn sie als komplementäre Intelligenzkomponente, nicht als alleiniger Problemlöser eingesetzt werden. Zudem belegt die unterschiedliche Qualität der Lösungen, dass es LLMs deutlich leichter fällt, durch reduzierte Komplexität und fest definierte Kontexte Code zu generieren, welcher konstant verwendet werden kann.

Für weitere Forschung gibt es verschiedene Verbesserungsoptionen, mit denen die Resultate des Algorithmus potenziell noch weiter verbessert werden können. Ein Aspekt, welcher dabei heraussticht, ist das Problem der konsistenten LLM Antworten. Diese beinhalten teilweise Fehler oder sind ungenügend. In der aktuellen Implementierung kann dadurch das Resultat eines Durchlaufes erheblich beeinträchtigt werden. Empfehlenswert ist daher, auf Basis der ermittelten Daten einen Grad der Generierungs-Duplikation festzulegen. Gemeint ist damit, dass Mating- und Bewertungsfunktionen mehrfach generiert werden. Beim Mating kann dann beispielsweise per Zufallsprinzip eine mögliche Funktion gewählt werden und bei der Bewertung der Schnitt der Rückmeldungen gebildet werden. Dadurch wird das Risiko durch potenzielle fehlerhafte Generierungen durch das LLM weiter reduziert und potenziell kann die False Positive Rate gesenkt werden. In der aktuellen Arbeit wurden solche Erweiterungen zunächst jedoch bewusst ausgelassen, um eine Vergleichbarkeit zwischen den Verfahren zu gewährleisten.

6.3 Szenarienübergreifende Muster und Vergleiche

Wie bereits in Kapitel 6.1 gezeigt, eignet sich ausschließlich das EA-Verfahren für die Generierung von Plänen in allen drei Szenarien. Auch die Abweichung, dass kein signifikanter Unterschied in dem Szenario der Maschinenplanung besteht, ändert an diesem Sachverhalt nichts. Die anderen Implementierungen kommen teils dicht an akzeptable Lösungen dran, waren jedoch per se nicht in der Lage, die Optimierungsprobleme hinreichend zu lösen.

Auffällig ist, dass die Fehleranfälligkeit, gemessen als Anzahl der durch die externen Skripte identifizierten Verstöße, je nach Verfahren und Szenario stark variiert. Während in der Zugplanung insbesondere die Verfahren EoS, EoH und der EA stabile und qualitativ hochwertige Ergebnisse erzielen, zeigt sich beim Prompting-Ansatz eine deutlich höhere Fehlerrate. So auch bei der Schulplanung. Hier erzielt der EA weiterhin solide Ergebnisse, während EoS und EoH mit zunehmender Komplexität und der Vielzahl an gleichzeitig zu berücksichtigenden Constraints sichtbar an Grenzen stoßen. Die Maschinenplanung ist im Vergleich dazu deutlich ähnlicher. Die Fehleranzahl bei EoH und EoS ist niedriger, während die von EA höher ist.

Diese Unterschiede zwischen den Szenarien lassen sich auf zwei zentrale Ursachen zurückführen. Zum einen unterscheiden sich die textuellen Formulierungen der Aufgabenstellung und der Bewertungsanweisungen zwischen den Szenarien. Wie in Kapitel 3.2.1 erläutert, reagiert das LLM empfindlich auf minimale Veränderungen in der Formulierung. Da die Prompts für die verschiedenen Szenarien unterschiedlich formuliert sind, kann das Prompting in einem Szenario besser geglückt sein als in einem anderen. Dieser Effekt erklärt nicht die Unterschiede zwischen den einzelnen Implementierungen, kann jedoch ein Indiz für die unterschiedliche Qualität der jeweiligen Implementierung in den einzelnen Szenarien sein.

Zum anderen spielt die interne Selbstbewertung der Systeme eine wesentliche Rolle. In der Zugplanung, in der die Aufgabenstruktur vergleichsweise klar und repetitiv ist, fällt die Diskrepanz zwischen interner und externer Bewertung am geringsten aus. Dies deutet darauf hin, dass das LLM in diesem Szenario die zugrundeliegenden Regeln gut erfassen und seine eigenen Ergebnisse relativ zuverlässig evaluieren konnte. In der Schulplanung hingegen zeigen EoS und EoH deutliche Abweichungen zwischen interner und externer Fitnessbewertung. Das LLM war nicht in der Lage, Fehler in den generierten Plänen korrekt zu identifizieren oder zu gewichten – beispielsweise bei fehlerhaften Lehrerzuordnungen oder unvollständigen Stundenverteilungen. Diese Fehlinterpretationen führten dazu, dass Pläne, die aus externer Sicht deutliche Regelverletzungen aufwiesen, intern dennoch als nahezu fehlerfrei bewertet wurden. Damit wird deutlich, dass die Komplexität der relationalen Abhängigkeiten im Schulkontext die Kapazität des LLMs zur strukturellen Validierung übersteigt. Anders ist dies beim EA. Hier werden einzelne Constraints vom LLM via einmal generiertem Code überprüft. Dadurch ist die Differenz zwischen den Szenarien (insbesondere im Maschinenplan) zwar unterschiedlich, jedoch in Relation zu der Fehleranzahl nicht erheblich anders. Was hier zudem auffällt ist, dass bei der EA Implementierung im Szeanrio Maschinenplan die Abweichung zwischen interner und

externer Fitnessbewertung höher ist, als der durchschnittliche Unterschied. Dies lässt darauf schließen, dass die Implementierung sich zumindest teilweise schlechter eingeschätzt hat (vgl. Tabelle 19). Die hier sehr niedrige False-Positive Rate weist ebenfalls auf eine kritische Selbsteinschätzung hin.

Über die Szenarien hinweg lässt sich somit ein klares Muster ableiten:

- Der reine Prompting-Ansatz stößt bereits bei allen Problemkomplexität an seine Grenzen. Die Anzahl der Constraints und die Abhängigkeit zwischen Entitäten überfordern die reine Sprachgenerierung, sodass keine konsistenten, fehlerfreien Lösungen erzielt werden können.
- Die hybriden Verfahren EoS und EoH zeigen eine verbesserte Anpassungsfähigkeit, bleiben jedoch von der Formulierungsqualität der LLM-Prompts und der internen Bewertungslogik abhängig. Sie liefern tendenziell stabile Resultate, sind aber nicht in der Lage, korrekte Ergebnisse zu generieren. Anhand des Friedman-Tests konnte zudem gezeigt werden, dass die Ergebnisse nicht signifikant besser werden als das Prompting.
- Der EA als weiterentwickelter Ansatz demonstriert die größte Robustheit. Er kann in mehreren Szenarien fehlerfreie oder nahezu perfekte Pläne erzeugen, allerdings nicht in jeder Iteration konsistent. Die Varianz zwischen den Läufen zeigt, dass das System zwar über ein hohes Optimierungspotenzial verfügt, die LLM-Interaktion aber weiterhin Schwankungen in der Qualität verursacht.

Diese Muster bestätigen, dass die Integration von LLMs in evolutionäre Strukturen zwar domänenübergreifend funktioniert, die Leistungsfähigkeit jedoch stark von der Art der textuellen Interaktion und der Szenariokomplexität abhängt. Daraus lässt sich auch schließen, dass die Implementierung des EAs grundsätzlich auch für weitere fachliche Anwendungsfälle geeignet ist. Die Qualität innerhalb der Domänen hängt primär von der Qualität der Problembeschreibung ab. Für den tatsächlichen praktischen Einsatz müsste jedoch noch die False Positive Rate optimiert werden. Durch das Fehlen von Validierungsskripten in der Praxis würde das System hier an Grenzen stoßen. Erste Ansätze wurden bereits in Kapitel 6.2 vorgeschlagen.

6.4 Schwächen und methodische Limitationen

Trotz der insgesamt überzeugenden und konsistenten Ergebnisse weist das durchgeführte Forschungsdesign einige methodische Einschränkungen auf, die bei der Interpretation der Resultate berücksichtigt werden müssen. Diese betreffen sowohl die Gestaltung der Prompts und die sprachliche Konsistenz zwischen den Verfahren als auch die Stichprobengröße, den Ressourceneinsatz und die Vergleichbarkeit der Systeme. Hinzu kommen grundsätzliche Limitationen, die aus der Verwendung von LLMs in experimentellen Umgebungen resultieren.

Sprachliche und strukturelle Inkonsistenzen der Prompts Ein wesentlicher methodischer Einflussfaktor liegt in der nicht einheitlichen Gestaltung der verwendeten Prompts. Während die Prompts der Verfahren EoS und EoH in deutscher Sprache formuliert wurden, erfolgte das Prompting für den EA sowie das Prompting auf Englisch. Diese Entscheidung wurde bewusst getroffen, da sich in den Voruntersuchungen zeigte, dass deutschsprachige Prompts, insbesondere in heuristischen Kontexten, eine präzisere Kontextinterpretation ermöglichten, während bei abstrakteren, generischen Anweisungen englische Prompts konsistentere Strukturen erzeugten. Dennoch führt diese Differenzierung zu einem eingeschränkten Grad an Vergleichbarkeit. LLMs zeigen eine deutliche Abhängigkeit von sprachlichen Nuancen und der semantischen Rahmung einer Anfrage. Selbst geringfügige Abweichungen im Wortlaut oder in der Aufgabenbeschreibung können zu divergierenden Interpretationen und Ausgaben führen. Auch wenn bei allen Implementierungen darauf geachtet wurde, die Intention der Anfrage möglichst konstant zu halten, kann nicht ausgeschlossen werden, dass die Formulierungsunterschiede einen messbaren Einfluss auf die Ergebnisse hatten.

Begrenzte Testgröße und Laufzeitrestriktionen Ein weiterer limitierender Faktor ergibt sich aus der relativ geringen Anzahl an Durchläufen, insbesondere bei den komplexeren Verfahren EoS und EoH. Die Ausführung dieser Systeme ist sowohl zeitlich als auch ressourcentechnisch stark aufwändig. Ein einzelner Durchlauf kann mehrere Stunden beanspruchen, wobei zusätzlich eine hohe Zahl an LLM-Requests anfällt. Diese führen zu signifikanten Rechen- und Nutzungskosten, die eine umfangreiche Wiederholung der Experimente begrenzen. Während beim Prompting- und beim EA-Ansatz eine höhere Wiederholungszahl erreicht wurde, bleibt die Aussagekraft der stärker LLM-abhängigen

Verfahren aufgrund der geringeren Stichprobenanzahl eingeschränkt. Diese Einschränkung betrifft insbesondere die Robustheitsanalyse und die statistische Absicherung der Ergebnisse. Eine größere Zahl an unabhängigen Läufen könnte hier zur Glättung zufälliger Ausreißer beitragen und die Varianz innerhalb der Resultate präziser quantifizieren. Die einzelnen Durchläufe bei EoS und EoH erzielten jedoch schon ähnliche Ergebnisse, weshalb nicht von einem signifikant anderem Ergebnis bei höherer Durchlaufzahl auszugehen ist.

Einsatz eines allgemeinen LLMs Ein weiterer methodischer Aspekt betrifft die Wahl des Sprachmodells. Für die vorliegende Arbeit wurde bewusst ein allgemeines, nicht domänenspezifisch optimiertes LLM verwendet. Dieses Vorgehen hatte das Ziel, die Übertragbarkeit der Ergebnisse zu maximieren und eine Reproduzierbarkeit der Experimente ohne Spezialmodelle zu gewährleisten. Gleichzeitig führt dies jedoch dazu, dass mögliche Leistungssteigerungen durch Feintuning, Reinforcement Learning from Human Feedback (RLHF) oder domänenspezifisches Prompt-Training unberücksichtigt bleiben. Die Ergebnisse spiegeln somit primär das Verhalten eines generischen Modells wider, nicht das theoretisch erreichbare Maximum eines spezialisierten Systems. Damit beschränkt sich die Aussagekraft der Resultate auf die grundsätzliche Machbarkeit und weniger auf die absolute Leistungsfähigkeit der LLMs. In zukünftigen Studien sollte geprüft werden, inwiefern ein speziell trainiertes Modell eine höhere Präzision oder Robustheit in der Planerzeugung und Bewertung erzielen kann.

Fehlender Vergleich zu klassischen Heuristiken Eine weitere Einschränkung ergibt sich aus dem fehlenden direkten Vergleich mit etablierten heuristischen oder metaheuristischen Verfahren. Der Fokus dieser Arbeit lag auf der Entwicklung und Evaluation neuer hybrider Systeme und nicht auf einer Reproduktion bekannter Benchmark-Verfahren. Zwar erlaubt die gewählte Herangehensweise eine klare Analyse der relativen Unterschiede zwischen den vorgestellten LLM-basierten Implementierungen, sie liefert jedoch keine Einordnung der erzielten Ergebnisse im Verhältnis zu klassischen Lösungsstrategien. Eine solche Vergleichsbasis wäre insbesondere für eine quantitative Einschätzung der relativen Leistungsfähigkeit relevant. Allerdings ist zu berücksichtigen, dass keine der gängigen heuristischen Verfahren gleichermaßen auf alle untersuchten Szenarien (Schul-, Zug- und Maschinenplanung) übertragbar gewesen wäre. Für eine weiterführende Forschung empfiehlt sich dennoch, zumindest für Teilprobleme standardisierte Benchmark-

Heuristiken einzubeziehen, um die ermittelten Ergebnisse stärker in den bestehenden Forschungsstand einzuordnen.

Einschränkungen der Validierungsskripte Die Qualität der generierten Pläne wurde in allen Verfahren anhand validierender Skripte überprüft, die spezifische Regelverletzungen und Schemaabweichungen identifizieren. Obwohl diese Validierungen umfangreich konzipiert und auf verschiedene Fehlertypen abgestimmt sind, kann nicht vollständig ausgeschlossen werden, dass sie in Einzelfällen Fehler übersehen oder falsch klassifizieren. Eine absolute Vollständigkeit der Prüfskripte lässt sich nicht garantieren, insbesondere bei komplexen, mehrdimensionalen Constraint-Verletzungen. Da jedoch alle Verfahren denselben Prüfmechanismus durchlaufen, wirkt sich eine potenzielle Unschärfe der Validierungssystematik gleichmäßig auf alle Implementierungen aus. Somit bleibt die interne Vergleichbarkeit gewahrt, auch wenn die absolute Fehlerquote möglicherweise verzerrt sein könnte.

Erhobene Messgrößen Ein weiterer Kritikpunkt sind die erhobenen Datenpunkte bzw. Messgrößen, mittels welcher die Implementierungen bewertet wurden. Hier wurden die in Kapitel 2.4.1 vorgestellten Ansätze für das Bewerten von Algorithmen verwendet und auf den Anwendungsfall hin modifiziert. Jedoch lässt sich nicht ausschließen, dass durch weitere Messwerte die Ergebnisse präziser und signifikanter geworden wären. Insbesondere die Abhängigkeit einzelner Datenpunkte (False / True Positives, Differenz Selbsteinschätzung und Fremdeinschätzung) kann das Ergebnis potenziell falsch beeinflussen. Für diesen Anwendungsfall waren dies die Datenpunkte, welche als am passendsten eingeschätzt wurden. Im Rahmen weiterer Forschung können sich jedoch auch andere präzisere Vergleichswerte ergeben.

Die genannten Limitationen relativieren die Generalisierbarkeit der Ergebnisse, ohne jedoch deren grundsätzliche Aussagekraft zu mindern. Für zukünftige Arbeiten ergeben sich daraus mehrere Ansatzpunkte: Eine Vereinheitlichung der Sprachwahl und Prompt-Struktur, eine signifikant größere Testbasis, der Vergleich mit klassischen Heuristiken sowie der Einsatz optimierter, domänenspezifischer LLMs könnten die Aussagekraft und Reproduzierbarkeit deutlich erhöhen. Auf dieser Grundlage ließe sich die in dieser Arbeit gezeigte Machbarkeit weiter empirisch absichern und methodisch vertiefen.

6.5 Implikationen für die Forschungsfragen

Das folgende Kapitel greift die in Kapitel 1.4 formulierten Forschungsfragen erneut auf und beantwortet diese auf Grundlage der in den vorangegangenen Kapiteln dargestellten Ergebnisse, Experimente und Interpretationen. Dabei werden die zentralen Erkenntnisse im Hinblick auf ihren Beitrag zur Zielsetzung dieser Arbeit betrachtet.

F1 – Beitrag von LLMs zur Lösung komplexer Planungsprobleme Die Ergebnisse zeigen deutlich, dass LLMs zwar ein grundlegendes Verständnis komplexer Zusammenhänge und textuell beschriebener Anforderungen besitzen, jedoch nicht autonom in der Lage sind, Timetabling-Probleme zu lösen. Reine Prompting-Ansätze führen zwar zu strukturell plausiblen, aber inhaltlich häufig unvollständigen oder fehlerhaften Lösungen. Dies deckt sich mit den theoretischen Erkenntnissen aus Kapitel 2.5.1, wonach LLMs keine algorithmische Suche oder formale Constraint-Erfüllung implementieren, sondern Wahrscheinlichkeitsverteilungen über mögliche Textfortsetzungen erzeugen.

Die empirischen Ergebnisse bestätigen, dass LLMs ohne Rückkopplungsschleifen schnell an ihre Grenzen stoßen, sobald ein Lösungsraum durch harte Constraints eingeschränkt ist. Erst durch die Integration eines Bewertungs- und Korrekturmechanismus, wie er in EoS oder EoH implementiert wurde, entstehen Lösungsprozesse, die zu stabileren und konsistenteren Ergebnissen führen. Die Unterschiede zu reinem Prompten sind jedoch nicht signifikant und auch hier war das LLM nicht in der Lage, für die definierten Fälle akzeptable Lösungen zu erzeugen. Besonders effektiv erwies sich die Nutzung von LLMs innerhalb klar abgegrenzter Kontexte mit wohldefinierten Aufgabenstellungen (z. B. Generierung einzelner Teilpläne oder Code-Segmente) wie es bei EA implementiert wurde. Werden die daraus entstehenden Teillösungen anschließend zu einem Gesamtergebnis kombiniert, lässt sich die inhärente Stärkenverteilung der Modelle gezielt nutzen und es werden signifikant bessere Ergebnisse erzielt. Diese waren in der Lage, für alle gegebenen Szenarien optimale Lösungen zu erzeugen. Damit bestätigt sich, dass LLMs einen wertvollen Beitrag zur Lösungsunterstützung leisten können, jedoch nicht als alleinstehende Problemlöser fungieren können.

F2 – Kombination von LLMs und evolutionären Algorithmen Die durchgeführten Experimente zeigen, dass die Kopplung von LLMs mit evolutionären Prinzipien substanzielle Leistungsgewinne ermöglichen kann, sofern die Interaktion zwischen beiden

Komponenten klar definiert ist. In dieser Arbeit wurden drei Varianten einer solchen Integration untersucht: Evolution of Schedules, Evolution of Heuristics und der Evolutionäre Algorithmus mit LLM-generierten Funktionskomponenten.

Während EoS das LLM in jeder Iteration zur direkten Lösungsgenerierung und Bewertung nutzt, überträgt EoH die Aufgabe auf eine Metaebene, indem das LLM Heuristiken oder Operatoren generiert, welche anschließend algorithmisch ausgeführt werden. Dadurch wird der Suchprozess stabiler, da das LLM nicht fortlaufend denselben Kontext neu interpretieren muss, sondern generalisierbare Strategien produziert. Der generische EA-Ansatz nutzt LLMs ausschließlich initial zur Erstellung des Algorithmus-„Baukastens“ (z. B. Crossover-, Mutations- oder Evaluationsfunktionen) und führt danach klassische evolutionäre Zyklen aus. Dieses Vorgehen erweist sich als einzige der Implementierungsoptionen, die eine ausreichend gute Lösungen erzeugen kann.

Die Kombination beider Technologien erweist sich somit als komplementär. Während evolutionäre Algorithmen explorative Suchmechanismen bereitstellen, tragen LLMs adaptives Wissen über Strukturen, Syntax und semantische Zusammenhänge bei. Diese Synergie ermöglicht ein adaptives Optimierungssystem, das kontextabhängig zwischen explorativer Suche und inhaltlicher Generalisierung balanciert.

F3 – Anforderungen an adaptive Systeme Die Analyse der entwickelten Ansätze verdeutlicht, dass adaptive Systeme in der Lage sein müssen, domänenspezifische Strukturen zu erfassen und flexibel darauf zu reagieren. Voraussetzung dafür ist ein einheitliches, maschinenlesbares Repräsentationsformat, das über textuelle Beschreibungen befüllt, angepasst und evaluiert werden kann. Das in Kapitel 3.1.1 vorgestellte generische Schema hat sich hierfür als geeignet erwiesen, da es die flexible Abbildung verschiedener Ressourcen- und Constraint-Typen erlaubt.

Zentral ist dabei die dynamische Anpassbarkeit der Operatoren. Sowohl die Mutations- als auch die Mating-Strategien müssen kontextspezifisch variierbar sein. Während in der Schulplanung beispielsweise vollständige Zuordnungen zwischen Klassen und Unterrichtseinheiten neu kombiniert werden dürfen, darf im Zugplanungsszenario ausschließlich das Personal getauscht werden, nicht jedoch der Fahrplan selbst. LLMs können hierbei als semantische Schnittstelle dienen, indem sie textuelle Anforderungen in domänenspezifischen Code übersetzen. Dadurch entsteht eine flexible Verbindung zwischen natürlichsprachlicher Problemdefinition und algorithmischer Ausführung. Adaptive Systeme erfordern folglich eine modulare Architektur, die semantische Generalisierung (durch LLMs)

und formale Konsistenz (durch algorithmische Steuerung) vereint. Möglich ist es auch, dass diese Kombination in andere metaheuristischen Verfahren zu guten Ergebnissen führen kann. Andere Algorithmen als der evolutionäre Algoithmus wurden in dieser Arbeit jedoch nicht vertiefend untersucht.

F4 – Praktische Anwendbarkeit In Bezug auf die praktische Nutzbarkeit zeigen die experimentellen Ergebnisse, dass lediglich der EA-Ansatz (Kapitel 6.1) die Anforderungen an Stabilität, Ergebnisqualität und Rechenaufwand erfüllt, die für reale Timetabling-Probleme erforderlich sind. Die Varianten EoS und EoH erzielen keine ausreichend guten Resultate und sind durch hohe Laufzeiten und inkonsistente LLM-Ausgaben limitiert. Die Kombination aus einem metaheuristischen Framework und LLM-generierten Operatoren stellt dagegen einen tragfähigen Kompromiss zwischen Automatisierung, Anpassungsfähigkeit und Effizienz dar.

Das Ergebnis unterstreicht, dass LLMs in der praktischen Optimierung nicht als Ersatz, sondern als Erweiterung traditioneller Verfahren zu verstehen sind. Der Einsatz lohnt sich insbesondere dort, wo komplexe Anforderungen automatisiert in Programmcode überführt werden sollen oder wo menschliche Expertise in regelbasiertes Wissen transformiert werden muss. In diesem Sinne bietet der hier entwickelte Ansatz eine Grundlage für zukünftige Forschung zur Entwicklung generischer LLM-gestützter Optimierungssysteme, die für unterschiedliche Anwendungsfelder eingesetzt werden können. Weitere Forschung sollte hierauf aufbauen, sodass durch gleichartige Systeme konstant Ergebnisse geliefert werden, welche sich insbesondere verlässlich selbst evaluieren können.

7 Zusammenfassung und Ausblick

Die vorliegende Arbeit untersuchte das Potenzial und die Grenzen von Large Language Models in der Lösung komplexer Planungs- und Optimierungsprobleme. Im Zentrum stand die Frage, inwieweit LLMs durch die Kombination mit evolutionären Verfahren befähigt werden können, adaptive, konsistente und qualitativ hochwertige Lösungen für verschiedene Timetabling-Domänen zu erzeugen. Aufbauend auf den theoretischen Grundlagen, der Implementierung von vier unterschiedlichen Ansätzen (Prompting, Evolution of Schedules, Evolution of Heuristics und Evolutionärer Algorithmus) sowie deren empirischer Evaluation, lassen sich mehrere zentrale Erkenntnisse ableiten.

Die Ergebnisse der Experimente zeigen deutlich, dass reine LLM-basierte Ansätze durch typisches Prompten nicht in der Lage sind, komplexe Planungsprobleme zuverlässig zu lösen. Obwohl LLMs ein semantisches Verständnis aufweisen und in der Lage sind, strukturell plausible Pläne zu erzeugen, scheitern sie an der Einhaltung harter Constraints und der Bewertung der eigenen Ergebnisse. Erst die Integration in metaheuristische Strukturen führte zu einer signifikanten Leistungssteigerung.

Besonders die Integration eines LLMs in einem evolutionären Algorithmus erwies sich als stabilstes und leistungsfähigstes Verfahren, welches signifikant besser als reines Prompten ist. Es erreichte über alle Szenarien hinweg die besten Fitnesswerte, die niedrigste Fehlerquote und die geringste Abweichung zwischen interner und externer Bewertung. Der EA nutzte LLMs gezielt für semantische Teilaufgaben (z. B. Generierung von Operatoren oder Heuristiken) und kombinierte damit die Stärken beider Paradigmen: die Suchstabilität evolutionärer Verfahren mit der Generalisierungsfähigkeit generativer Modelle.

Der Ansatz Evolution of Heuristics zeigte, dass LLMs durchaus in der Lage sind, domänenspezifische Heuristiken zu formulieren, die sich innerhalb der Optimierung adaptiv anwenden lassen. Dieses Verfahren bietet Potenzial, hat jedoch auch gezeigt, dass ein LLM alleine nicht in der Lage ist, innerhalb eines iterativen Verfahrens, seine eigenen Ergebnisse signifikant zu verbessern. Evolution of Schedules hat die gleichen Probleme, ist jedoch noch weniger in der Lage passende Pläne zu erzeugen, als das davorgestellte Generieren von Heuristiken. Beide Verfahren waren nicht in der Lage, für die definierten Problemstellungen ausreichend gute Lösungen zu generieren.

Über alle Verfahren hinweg wurde deutlich, dass hybride Systeme mit expliziten Rückkopplungsschleifen zwischen algorithmischer Bewertung und LLM-Generierung die besten

Resultate liefern. Der kombinierte Ansatz aus algorithmischer Kontrolle und sprachbasierter Generalisierung erwies sich als entscheidend für die Übertragbarkeit auf unterschiedliche Problemkontexte (z. B. Schul-, Zug- und Maschinenplanung).

Die Arbeit zeigt, dass LLMs nicht als Ersatz, sondern als Erweiterung klassischer metaheuristischer Optimierungsverfahren verstanden werden sollten. Ihre Stärke liegt in der Fähigkeit, textuelle, semantische oder schwach formal definierte Probleme in algorithmisch verwertbare Strukturen zu übersetzen. Durch die Einbettung in Frameworks entstehen adaptive Systeme, die explorative und deterministische Komponenten vereinen.

Methodisch verdeutlicht die Arbeit zugleich die Herausforderungen dieser Integration. Die stochastische Natur der LLMs führt zu Varianzen in den Ergebnissen, die Reproduzierbarkeit und Vergleichbarkeit erschweren. Dies ist insbesondere in der internen Bewertung des Resultats problematisch, da dadurch fälschlicherweise das System die eigenen Resultate als optimal passend einstufen kann. Ebenso wurde deutlich, dass kleine Abweichungen im Prompting erheblichen Einfluss auf die Performanz haben. Dennoch konnte gezeigt werden, dass auch durch LLM-generierte Funktionen innerhalb der Heuristiken konstant Optimierungsverläufe realisiert werden können.

Für die Praxis ergeben sich mehrere Konsequenzen: LLMs sollten gezielt in Optimierungsumgebungen eingesetzt werden, insbesondere in Teilaufgaben, die semantische Interpretationen oder flexible Regelableitungen erfordern. Der vorgestellte EA-Ansatz bietet hier eine robuste Basis, um komplexe Optimierungsprobleme mit überschaubarem Ressourcenaufwand zu adressieren. Zudem wurde gezeigt, dass das System in der Lage ist, verschiedenste Planungsprobleme mit unterschiedlichen Optimierungsobjekten lösen zu können. Damit wird eine Brücke zwischen generativer KI und klassischer Metaheuristik geschaffen, die neue Möglichkeiten für adaptive Planungs- und Entscheidungssysteme eröffnet.

Auf Grundlage der gewonnenen Erkenntnisse ergeben sich verschiedene Perspektiven für zukünftige Forschung. Erstens sollte anhand größerer Datenmengen und komplexerer Szenarien weiter geprüft werden, wie gut die implementierten Lösungen, insbesondere der EA, sind. Zweitens sollte der Vergleich zwischen den adaptiven Lösungen und klassischen Heuristiken hergestellt werden. Dadurch kann im größeren Kontext bestehender Forschung die Leistungsfähigkeit der vorgestellten Algorithmen in Relation gesetzt werden. Drittens sollte geprüft werden, ob durch einen Ausbau des EA-Ansatzes bessere

Ergebnisse erzielt werden können. Ansätze hierfür wurden bereits in Kapitel 6.2 aufgezeigt.

Langfristig eröffnet sich ein Forschungsfeld, das klassische Optimierungsverfahren und moderne Sprachmodelle enger miteinander verzahnt. LLMs bieten eine gute Möglichkeit, bestehende Verfahren für verschiedenste Anwendungsfälle zu öffnen, ohne dass jedes Mal ein neuer Algorithmus programmiert werden muss. Diese Möglichkeiten sollten genutzt werden, um zukünftig generalisierte, adaptive und leistungsstarke Systeme zu entwickeln, welche einen parktischen Mehrwert für verschiedenste Optimierungsprobleme liefern.

Literatur

Amatriain, X. (2024, 4. Mai). *Prompt Design and Engineering: Introduction and Advanced Methods*. arXiv: 2401.14423 [cs]. https://doi.org/10.48550/arXiv.2401.14423

Amatriain, X., Sankar, A., Bing, J., Bodigutla, P. K., Hazen, T. J., & Kazi, M. (2024, 31. März). *Transformer Models: An Introduction and Catalog*. arXiv: 2302.07730 [cs]. https://doi.org/10.48550/arXiv.2302.07730

Augugliaro, A., Dusonchet, L., & Sanseverino, E. R. (1999). Genetic, simulated annealing and tabu search algorithms: Three heuristic methods for optimal reconfiguration and compensation of distribution networks. *European Transactions on Electrical Power*, *9*(1), 35–41. https://doi.org/10.1002/etep.4450090104

Ba, J. L., Kiros, J. R., & Hinton, G. E. (2016, 21. Juli). *Layer Normalization*. arXiv: 1607.06450 [stat]. https://doi.org/10.48550/arXiv.1607.06450

Bashab, A., Ibrahim, A., Abakar, I., Aggarwal, K., Mukhlif, F., Ghaleb, F., & Abdelmaboud, A. (2022). Optimization Techniques in University Timetabling Problem: Constraints, Methodologies, Benchmarks, and Open Issues. *Computers, Materials & Continua*, *74*(3), 6461–6484. https://doi.org/10.32604/cmc.2023.034051

Blum, C., & Roli, A. (2003). Metaheuristics in Combinatorial Optimization: Overview and Conceptual Comparison. *ACM Comput. Surv.*, *35*(3), 268–308. https://doi.org/10.1145/937503.937505

Brélaz, D. (1979). New methods to color the vertices of a graph. *Commun. ACM*, *22*(4), 251–256. https://doi.org/10.1145/359094.359101

Brest, J., & Sepesy Maučec, M. (2025). Comparative Study of Modern Differential Evolution Algorithms: Perspectives on Mechanisms and Performance. *Mathematics*, *13*(10), 1556. https://doi.org/10.3390/math13101556

Bubeck, S., Chandrasekaran, V., Eldan, R., Gehrke, J., Horvitz, E., Kamar, E., Lee, P., Lee, Y. T., Li, Y., Lundberg, S., Nori, H., Palangi, H., Ribeiro, M. T., & Zhang, Y. (2023, April). Sparks of Artificial General Intelligence: Early experiments with GPT-4. https://doi.org/10.48550/arXiv.2303.12712

Burke, E., Kendall, G., Newall, J., Hart, E., Ross, P., & Schulenburg, S. (2003). Hyper-Heuristics: An Emerging Direction in Modern Search Technology. In F. Glover & G. A. Kochenberger (Hrsg.), *Handbook of Metaheuristics* (S. 457–474). Springer US. https://doi.org/10.1007/0-306-48056-5_16

Burke, E. K., Gendreau, M., Hyde, M., Kendall, G., Ochoa, G., Özcan, E., & Qu, R. (2013). Hyper-heuristics: a survey of the state of the art. *Journal of the Operational Research Society*, *64*(12), 1695–1724. https://doi.org/10.1057/jors.2013.71

Burke, E., & Newall, J. (1999). A multistage evolutionary algorithm for the timetable problem. *IEEE Transactions on Evolutionary Computation*, *3*(1), 63–74. https://doi.org/10.1109/4235.752921

Burke, E., & Newall, J. (2004). Solving Examination Timetabling Problems through Adaption of Heuristic Orderings. *Annals of Operations Research*, *129*(1), 107–134. https://doi.org/10.1023/B:ANOR.0000030684.30824.08

Chen, B., Zhang, Z., Langrené, N., & Zhu, S. (2025). Unleashing the potential of prompt engineering for large language models. *Patterns*, 101260. https://doi.org/10.1016/j.patter.2025.101260

Cho, A., Kim, G. C., Karpekov, A., Helbling, A., Wang, Z. J., Lee, S., Hoover, B., & Chau, D. H. (2024, 8. August). *Transformer Explainer: Interactive Learning of Text-Generative Models*. arXiv: 2408.04619 [cs]. https://doi.org/10.48550/arXiv.2408.04619

Chu, S., & Fang, H. (1999). Genetic Algorithms vs. Tabu Search in Timetable Scheduling. *1999 Third International Conference on Knowledge-Based Intelligent Information Engineering Systems. Proceedings (Cat. No.99TH8410)*, 492–495. https://doi.org/10.1109/KES.1999.820230

Coello Coello, C. A. (2002). Theoretical and numerical constraint-handling techniques used with evolutionary algorithms: a survey of the state of the art. *Computer Methods in Applied Mechanics and Engineering*, *191*(11), 1245–1287. https://doi.org/10.1016/S0045-7825(01)00323-1

Colorni, A., Dorigo, M., & Maniezzo, V. (1991). Genetic algorithms and highly constrained problems: The time-table case. In H.-P. Schwefel & R. Männer (Hrsg.), *Parallel Problem Solving from Nature* (S. 55–59). Springer. https://doi.org/10.1007/BFb0029731

Creswell, A., Shanahan, M., & Higgins, I. (2022). Selection-Inference: Exploiting Large Language Models for Interpretable Logical Reasoning. Verfügbar 27. Mai 2025 unter https://openreview.net/forum?id=3Pf3Wg6o-A4

Deb, K. (2000). An efficient constraint handling method for genetic algorithms. *Computer Methods in Applied Mechanics and Engineering*, *186*(2), 311–338. https://doi.org/10.1016/S0045-7825(99)00389-8

Deb, K. (2004). *Multi-objective optimization using evolutionary algorithms* (Repr). Wiley.

Demšar, J. (2006). Statistical Comparisons of Classifiers over Multiple Data Sets. *J. Mach. Learn. Res.*, *7*, 1–30.

Dowsland, K. A. (1993). Some experiments with simulated annealing techniques for packing problems. *European Journal of Operational Research*, *68*(3), 389–399. https://doi.org/10.1016/0377-2217(93)90195-S

Drake, J. H., Kheiri, A., Özcan, E., & Burke, E. K. (2020). Recent advances in selection hyper-heuristics. *European Journal of Operational Research*, *285*(2), 405–428. https://doi.org/10.1016/j.ejor.2019.07.073

Eiben, A. E., & Smith, J. E. (2015). Working with Evolutionary Algorithms. In A. Eiben & J. Smith (Hrsg.), *Introduction to Evolutionary Computing* (S. 147–163). Springer. https://doi.org/10.1007/978-3-662-44874-8_9

Eleusov, A., Saparbayev, A., & Makulova, A. (2020). Application of Digital Technologies for Solving Optimization Tasks: *Proceedings of the 2nd International Scientific and Practical Conference on Digital Economy (ISCDE 2020)*. https://doi.org/10.2991/aebmr.k.201205.043

Giffler, B., & Thompson, G. L. (1960). Algorithms for Solving Production-Scheduling Problems. *Operations Research*, *8*(4), 487–503. https://doi.org/10.1287/opre.8.4.487

Global Digitalization in 10 Charts. (n. d.). World Bank. Verfügbar 19. Januar 2025 unter https://www.worldbank.org/en/news/immersive-story/2024/03/05/global-digitalization-in-10-charts

Gusti Agung Premananda, I., Tjahyanto, A., & Muklason, A. (2024). Timetabling Problems and the Effort Toward Generic Algorithms: A Comprehensive Survey. *IEEE Access*, *12*, 143854–143868. https://doi.org/10.1109/ACCESS.2024.3463721

Holtzman, A., Buys, J., Du, L., Forbes, M., & Choi, Y. (2020, Februar). The Curious Case of Neural Text Degeneration. https://doi.org/10.48550/arXiv.1904.09751

Hsieh, C.-J., Si, S., Yu, F. X., & Dhillon, I. S. (2023, November). Automatic Engineering of Long Prompts. https://doi.org/10.48550/arXiv.2311.10117

Iman, R. L., & Davenport, J. M. (1980). Approximations of the critical region of the fbietkan statistic. *Communications in Statistics - Theory and Methods*, *9*(6), 571–595. https://doi.org/10.1080/03610928008827904

Jacquelin, D. (2019). *Entwicklung eines hybriden Algorithmus für die sequenzielle Optimierung der Reihenfolgen- und Maschinenbelegungsplanung* [Thesis]. Technische Universität Wien [Accepted: 2020-06-28T06:32:55Z]. https://doi.org/10.34726/hss.2019.51822

Ji, Z., Lee, N., Frieske, R., Yu, T., Su, D., Xu, Y., Ishii, E., Bang, Y., Chen, D., Dai, W., Chan, H. S., Madotto, A., & Fung, P. (2023). Survey of Hallucination in

Natural Language Generation. *ACM Computing Surveys*, *55*(12), 1–38. https://doi.org/10.1145/3571730

Jobson, D., & Li, Y. (2024, Juni). Investigating the Potential of Using Large Language Models for Scheduling. https://doi.org/10.48550/arXiv.2406.07573

Jorge Amar, Sohrab Rahimi, Nicolai von Bismarck & Akshar Wunnava. (2022, Januar). Workforce optimization: Staff scheduling with AI. Verfügbar 15. Oktober 2025 unter https://www.mckinsey.com/capabilities/operations/our-insights/smart-scheduling-how-to-solve-workforce-planning-challenges-with-ai?utm_source=chatgpt.com

Jorge Nocedal & Stephen J. Wright. (2006). *Numerical Optimization*. Springer New York. https://doi.org/10.1007/978-0-387-40065-5

Kim, H.-Y. (2014). Statistical notes for clinical researchers: Nonparametric statistical methods: 2. Nonparametric methods for comparing three or more groups and repeated measures. *Restorative Dentistry & Endodontics*, *39*(4), 329–332. https://doi.org/10.5395/rde.2014.39.4.329

Lemonge, A. C. C., & Barbosa, H. J. C. (2004). An adaptive penalty scheme for genetic algorithms in structural optimization. *International Journal for Numerical Methods in Engineering*, *59*(5), 703–736. https://doi.org/10.1002/nme.899

Liu, F., Tong, X., Yuan, M., Lin, X., Luo, F., Wang, Z., Lu, Z., & Zhang, Q. (2024, 1. Juni). *Evolution of Heuristics: Towards Efficient Automatic Algorithm Design Using Large Language Model*. arXiv: 2401.02051 [cs]. https://doi.org/10.48550/arXiv.2401.02051

Liu, F., Tong, X., Yuan, M., & Zhang, Q. (2023, 26. November). *Algorithm Evolution Using Large Language Model*. arXiv: 2311.15249 [cs]. https://doi.org/10.48550/arXiv.2311.15249

Madmoni, L., Zait, A., Labzovsky, I., & Karmon, D. (2024, September). The Ability of Large Language Models to Evaluate Constraint-satisfaction in Agent Responses to Open-ended Requests. https://doi.org/10.48550/arXiv.2409.14371

Megahed, F. M., Chen, Y.-J., Jones-Farmer, L. A., Lee, Y., Wang, J. B., & Zwetsloot, I. M. (2025, Mai). Reliable Decision Support with LLMs: A Framework for Evaluating Consistency in Binary Text Classification Applications. https://doi.org/10.48550/arXiv.2505.14918

Michalewicz, Z. (1996). Heuristic methods for evolutionary computation techniques. *Journal of Heuristics*, *1*(2), 177–206. https://doi.org/10.1007/BF00127077

Michalewicz, Z., Hinterding, R., & Michalewicz, M. (1997). Evolutionary Algorithms. In W. Pedrycz (Hrsg.), *Fuzzy Evolutionary Computation* (S. 3–31). Springer US. https://doi.org/10.1007/978-1-4615-6135-4_1

Mienye, I. D., Jere, N., Obaido, G., Ogunruku, O. O., Esenogho, E., & Modisane, C. (2025). Large language models: an overview of foundational architectures, recent trends, and a new taxonomy. *Discover Applied Sciences*, *7*(9), 1027. https://doi.org/10.1007/s42452-025-07668-w

Minaee, S., Mikolov, T., Nikzad, N., Chenaghlu, M., Socher, R., Amatriain, X., & Gao, J. (2025, März). Large Language Models: A Survey. https://doi.org/10.48550/arXiv.2402.06196

Nemenyi, P. (1963). *Distribution-free Multiple Comparisons*. Princeton University.

Osman, I. H., & Kelly, J. P. (1996). Meta-Heuristics: An Overview. In I. H. Osman & J. P. Kelly (Hrsg.), *Meta-Heuristics: Theory and Applications* (S. 1–21). Springer US. https://doi.org/10.1007/978-1-4613-1361-8_1

Özcan, E., Bilgin, B., & Korkmaz, E. E. (2008). A comprehensive analysis of hyper-heuristics. *Intelligent Data Analysis*, *12*(1), 3–23. https://doi.org/10.3233/IDA-2008-12102

Peng, Y., Ahmad, S. F., Irshad, M., Al-Razgan, M., Ali, Y. A., & Awwad, E. M. (2023). Impact of Digitalization on Process Optimization and Decision-Making towards Sustainability: The Moderating Role of Environmental Regulation. *Sustainability*, *15*(20), 15156. https://doi.org/10.3390/su152015156

Pillay, N. (2014). A survey of school timetabling research. *Annals of Operations Research*, *218*(1), 261–293. https://doi.org/10.1007/s10479-013-1321-8

Radford, A., Wu, J., Child, R., Luan, D., Amodei, D., & Sutskever, I. (2019). Language Models Are Unsupervised Multitask Learners. Verfügbar 21. Mai 2025 unter https://www.semanticscholar.org/paper/Language-Models-are-Unsupervised-Multitask-Learners-Radford-Wu/9405cc0d6169988371b2755e573cc28650d14dfe

Rhydian Lewis. (2007). A Survey of Metaheuristic-Based Techniques for University Timetabling Problems. *Cardiff University*, 27.

Ruiz, R., & Maroto, C. (2005). A Comprehensive Review and Evaluation of Permutation Flowshop Heuristics. *European Journal of Operational Research*, *165*(2), 479–494. https://doi.org/10.1016/j.ejor.2004.04.017

Sahoo, P., Singh, A. K., Saha, S., Jain, V., Mondal, S., & Chadha, A. (2025, März). A Systematic Survey of Prompt Engineering in Large Language Models: Techniques and Applications. https://doi.org/10.48550/arXiv.2402.07927

Schlagwein, D., & Willcocks, L. (2023). 'ChatGPT et al.': The ethics of using (generative) artificial intelligence in research and science. *Journal of Information Technology*, *38*(3), 232–238. https://doi.org/10.1177/02683962231200411

Schulhoff, S., Ilie, M., Balepur, N., Kahadze, K., Liu, A., Si, C., Li, Y., Gupta, A., Han, H., Schulhoff, S., Dulepet, P. S., Vidyadhara, S., Ki, D., Agrawal, S., Pham, C., Kroiz, G., Li, F., Tao, H., Srivastava, A., ... Resnik, P. (2025, 26. Februar). *The Prompt Report: A Systematic Survey of Prompt Engineering Techniques*. arXiv: 2406.06608 [cs]. https://doi.org/10.48550/arXiv.2406.06608

Scully, Z., & Harchol-Balter, M. (2021, 22. Oktober). *How to Schedule Near-Optimally under Real-World Constraints*. arXiv: 2110.11579 [cs]. https://doi.org/10.48550/arXiv.2110.11579

Simon, D. (2013). *Evolutionary Optimization Algorithms* (1. Aufl.). Wiley.

Skiena, S. S. (2020). Graph Problems: NP-Hard. In S. S. Skiena (Hrsg.), *The Algorithm Design Manual* (S. 585–620). Springer International Publishing. https://doi.org/10.1007/978-3-030-54256-6_19

Streim, A., & Heinze, D. (2021, 4. Mai). *Corona: Unternehmen spüren wirtschaftlichen Nutzen der Digitalisierung | Presseinformation | Bitkom e. V.* (Questionair). Bitkom e.V. Verfügbar 19. Januar 2025 unter https://www.bitkom.org/Presse/Presseinformation/Corona-Unternehmen-spueren-wirtschaftlichen-Nutzen-der-Digitalisierung

Stureborg, R., Alikaniotis, D., & Suhara, Y. (2024, Mai). Large Language Models are Inconsistent and Biased Evaluators. https://doi.org/10.48550/arXiv.2405.01724

Szeider, S. (2024, Dezember). MCP-Solver: Integrating Language Models with Constraint Programming Systems. https://doi.org/10.48550/arXiv.2501.00539

Takahama, T., & Sakai, S. (2006). Constrained Optimization by the Constrained Differential Evolution with Gradient-Based Mutation and Feasible Elites. *2006 IEEE International Conference on Evolutionary Computation*, 1–8. https://doi.org/10.1109/CEC.2006.1688283

Tassopoulos, I. X., & Beligiannis, G. N. (2012). Solving effectively the school timetabling problem using particle swarm optimization. *Expert Systems with Applications*, *39*(5), 6029–6040. https://doi.org/10.1016/j.eswa.2011.12.013

Tessema, B., & Yen, G. (2006). A Self Adaptive Penalty Function Based Algorithm for Constrained Optimization. *2006 IEEE International Conference on Evolutionary Computation*, 246–253. https://doi.org/10.1109/CEC.2006.1688315

Todoschak, O., & Frolova, Ye. I. (2023). Optimization of Public Administration Methods in the Conditions of Digitalization. *Uzhhorod National University Herald. Series: Law*, *2*(76), 79–84. https://doi.org/10.24144/2307-3322.2022.76.2.13

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł. ukasz, & Polosukhin, I. (2017). Attention Is All You Need. *Advances in Neural Information Processing Systems*, *30*. Verfügbar 21. Mai 2025 unter https://papers.nips.cc/paper_files/paper/2017/hash/3f5ee243547dee91fbd053c1c4a845aa-Abstract.html

Vatsal, S., & Dubey, H. (2024, Juli). A Survey of Prompt Engineering Methods in Large Language Models for Different NLP Tasks. https://doi.org/10.48550/arXiv.2407.12994

Vinod Kadam & Samir Yadav. (2016). ACADEMIC TIMETABLE SCHEDULING : REVISITED. *ResearchGate*. Verfügbar 30. April 2025 unter https://www.researchgate.net/publication/301564510_ACADEMIC_TIMETABLE_SCHEDULING_REVISITED

Wang, F., Shang, C., Jain, S., Wang, S., Ning, Q., Min, B., Castelli, V., Benajiba, Y., & Roth, D. (2024, März). From Instructions to Constraints: Language Model Alignment with Automatic Constraint Verification. https://doi.org/10.48550/arXiv.2403.06326

Wang, P., Li, L., Chen, L., Cai, Z., Zhu, D., Lin, B., Cao, Y., Liu, Q., Liu, T., & Sui, Z. (2023, August). Large Language Models are not Fair Evaluators. https://doi.org/10.48550/arXiv.2305.17926

Wang, X., Wei, J., Schuurmans, D., Le, Q., Chi, E., Narang, S., Chowdhery, A., & Zhou, D. (2023, März). Self-Consistency Improves Chain of Thought Reasoning in Language Models. https://doi.org/10.48550/arXiv.2203.11171

Wei, J., Wang, X., Schuurmans, D., Bosma, M., Ichter, B., Xia, F., Chi, E., Le, Q., & Zhou, D. (2023, Januar). Chain-of-Thought Prompting Elicits Reasoning in Large Language Models. https://doi.org/10.48550/arXiv.2201.11903

Wolf, T., Debut, L., Sanh, V., Chaumond, J., Delangue, C., Moi, A., Cistac, P., Rault, T., Louf, R., Funtowicz, M., Davison, J., Shleifer, S., Platen, P. von, Ma, C., Jernite, Y., Plu, J., Xu, C., Scao, T. L., Gugger, S., … Rush, A. M. (2020, 14. Juli). *HuggingFace's Transformers: State-of-the-art Natural Language Processing*. arXiv: 1910.03771 [cs]. https://doi.org/10.48550/arXiv.1910.03771

Wu, X., Wu, S.-h., Wu, J., Feng, L., & Tan, K. C. (2024, 29. Mai). *Evolutionary Computation in the Era of Large Language Model: Survey and Roadmap*. arXiv: 2401.10034 [cs]. https://doi.org/10.48550/arXiv.2401.10034

Zhou, Y., Muresanu, A. I., Han, Z., Paster, K., Pitis, S., Chan, H., & Ba, J. (2023, März). Large Language Models Are Human-Level Prompt Engineers. https://doi.org/10.48550/arXiv.2211.01910

A Anhang

A.1 Musterpläne

Die folgend angehängten Pläne sind die Referenz für die Darstellung der Machbarkeit der einzelnen Szenarien. Die daraus abgeleiteten Anforderungen wurden für die Algorithmen verwendet, zum Erstellen der Pläne. Die Pläne sind in folgender Reihenfolge angefügt:

1. Belegungsplan für Züge
2. Belegungsplan für Maschinen
3. Stundenplan für eine Schule

Tagesplan Züge und Personal

	0	1	2	3	4	5
Zug-1 (4)	Start Lübeck	Hamburg	Lübeck	Hamburg	Lübeck	Hamburg
Zug-11 (4)		Start Lübeck	Hamburg	Lübeck	Hamburg	Lübeck
Zug-2 (1)		Start Kiel	Neumünster	Hamburg	Neumünster	Kiel
Zug-22 (1)	Start Hamburg	Neumünster	Kiel	Neumünster	Hamburg	
Zug-3 (2)		Start Flensburg	Neumünster	Hamburg	Neumünster	Flensburg
Zug-33 (2)	Start Hamburg	Neumünster	Flensburg	Neumünster	Hamburg	
Zug-4 (3)	Start Büchen	Lübeck	Kiel	Lübeck	Büchen	Lübeck
Zug-5 (6)					Lübeck	Schwerin
Zug-6 (5)	Start Rostock	Schwerin	Büchen	Hamburg	Büchen	Schwerin
Zug-66 (5)	Start Hamburg	Büchen	Schwerin	Rostock	Schwerin	Büchen
Team-1	Start Zug-1	Zug-1	Zug-1	Zug-1	Zug-1	Zug-1
Team-2				Start Zug-11	Zug-11	Zug-11
Team-3		Start Zug-11	Zug-11	Zug-11	Zug-5	Zug-5
Team-4						
Team-5						
Team-6		Start Zug-2	Zug-2	Zug-2	Zug-2	Zug-2
Team-7	Start Zug-22	Zug-22	Zug-22	Zug-22	Zug-22	
Team-8		Start Zug-3	Zug-3	Zug-3	Zug-3	Zug-3
Team-9	Start Zug-33	Zug-33	Zug-33	Zug-33	Zug-33	
Team-10	Start Zug-6	Zug-6	Zug-6	Zug-6	Zug-6	Zug-4
Team-11	Start Zug-4	Zug-4	Zug-4	Zug-4	Zug-4	Zug-6
Team-12	Start Zug-66	Zug-66	Zug-66	Zug-66	Zug-66	Zug-66
Team-13						

	6	7	8	9	10	11
Zug-1 (4)	Lübeck	Hamburg	Lübeck	Hamburg	Lübeck	Hamburg
Zug-11 (4)	Hamburg	Lübeck	Hamburg	Lübeck	Hamburg	Lübeck
Zug-2 (1)		Neumünster	Hamburg	Neumünster	Kiel	
Zug-22 (1)	Neumünster	Kiel	Neumünster	Hamburg		
Zug-3 (2)		Neumünster	Hamburg	Neumünster	Flensburg	
Zug-33 (2)	Neumünster	Flensburg	Neumünster	Hamburg		
Zug-4 (3)	Kiel	Lübeck	Büchen	Lübeck	Kiel	Lübeck
Zug-5 (6)	Lübeck	Schwerin	Lübeck			
Zug-6 (5)	Rostock	Schwerin	Büchen	Hamburg	Büchen	Schwerin
Zug-66 (5)	Hamburg	Büchen	Schwerin	Rostock	Schwerin	Büchen
Team-1	Zug-1	Zug-1	Zug-1			
Team-2	Zug-11	Zug-11		Zug-1	Zug-1	Zug-1
Team-3	Zug-5	Zug-5	Zug-5			
Team-4		Start Zug-11	Zug-11	Zug-11	Zug-11	Zug-11
Team-5			Start Zug-5	Zug-5	Zug-5	Zug-5
Team-6		Zug-2	Zug-2	Zug-2	Zug-2	
Team-7	Zug-22	Zug-22	Zug-22	Zug-22		
Team-8		Zug-3	Zug-3	Zug-3	Zug-3	
Team-9	Zug-33	Zug-33	Zug-33	Zug-33		
Team-10	Zug-4	Zug-4	Zug-4			Zug-6
Team-11	Zug-6	Zug-6	Zug-6	Zug-6	Zug-6	
Team-12	Zug-66					
Team-13	Start Zug-66	Zug-66	Zug-66	Zug-66	Zug-66	Zug-66

	12	
Zug-1 (4)	Lübeck	
Zug-11 (4)		
Zug-2 (1)		
Zug-22 (1)		
Zug-3 (2)		
Zug-33 (2)		
Zug-4 (3)	Büchen	
Zug-5 (6)		
Zug-6 (5)	Rostock	
Zug-66 (5)	Hamburg	
		Stunden
Team-1		9
Team-2	Zug-1	9
Team-3		8
Team-4		5
Team-5	Zug-5	5
Team-6		9
Team-7		9
Team-8		9
Team-9		9
Team-10	Zug-6	11
Team-11		11
Team-12		7
Team-13	Zug-66	7
		108

Maschinen - Beispielplan

Tage	Monday			Tuesday			Wednesday			Thursday			Friday			Saturday			Sunday		
Schichten	**S-1**	**S-2**	**S-3**	**S-1**	**S-2**	**S-3**	**S-1**	**S-2**	**S-3**	**S-1**	**S-2**	**S-3**	**S-1**	**S-2**	**S-3**	**S-1**	**S-2**	**S-3**	**S-1**	**S-2**	**S-3**
Maschine-1	1	2	3	4	2	3	1	5	3	4	2	3	1	5	3	1	4	2	1	4	2
Maschine-2	4	6		8	6		8	6		9	8		7	8		7	5		7	5	
Maschine-3			7			7			10			6			6			9			9
Maschine-4	10, 11	13, 15		10, 11	13, 14		11, 12	13, 15		11, 14	12, 15		11, 13	9, 12		10, 13	12, 14		10, 15	12, 14	
Maschine-5				16, 17, 18	15, 19, 20	21, 22, 23															
Maschine-6													16, 19, 20	17, 21, 22	18, 23, 24	16, 19, 20	17, 21, 22	18, 23, 24	16, 19, 20	17, 21, 22	18, 23, 24
Maschine-7										16, 19, 20	17, 21, 22	18, 23, 24									

Maschinen

	Laufzeit	Belegung	Mitarbeiter
Maschine-1	Mo-So 3 Schichten	1	1, 2, 3, 4, 5
Maschine-2	Mo-So 1. + 2. Schicht	1	4, 5, 6, 7, 8, 9
Maschine-3	Mo-So 3. Schicht	1	6, 7, 9
Maschine-4	Mo-So 1. + 2. Schicht	2	9, 10, 11, 12, 13, 14, 15
Maschine-5	Tue 3 Schichten	3	15, 16, 17, 18, 19, 20, 21, 22, 23
Maschine-6	Fr-Su 3 Schichten	3	16, 17, 18, 19, 20, 21, 22, 23, 24
Maschine-7	Thu 3 Schichten	3	16, 17, 18, 19, 20, 21, 22, 23, 24

Mitarbeiter

	Anzahl Schichten	Zeitwunsch
M-1	5	Schicht 1 oder 2 (Optional)
M-2	5	-
M-3	5	Schicht 3 (Optional)
M-4	5	
M-5	4	
M-6	5	Nicht am Wochenende
M-7	5	Nicht Mittwochs
M-8	4	
M-9	4	
M-10	5	
M-11	5	Nicht am Wochenende
M-12	5	
M-13	5	
M-14	4	
M-15	5	
M-16	5	
M-17	5	
M-18	5	
M-19	5	
M-20	5	
M-21	5	
M-22	5	
M-23	5	
M-24	4	

Schule - Monday

	1	2	3	4	5	6
11		Mus - 101	M - 102	Rel - 103	D - 103	
12		Rel - 107	Ku - 107	D - 110	M - 111	
21		D - 108	D - 108	M - 105	SU- 109	
22		SU - 109	SU - 109	M - 102	D - 107	
31	KL - 105	M - 105	M - 105	D - 111	Mus - 101	Eng - 110
32	M - 102	D - 112	Mus - 101	SU - 104	Eng - 104	Sp - 112
41	Sp - 103	Sp - 103	D - 103	M - 109	Eng - 110	Mus - 101
42	SU - 111	M - 102	Eng - 104	D - 107	Statt AG - 112	Rel - 105

Schule - Tuesday

	1	2	3	4	5	6
11		D - 103	M - 102	SU - 104	Ku - 103	
12		SU - 111	Mus - 101	D - 110	D - 110	
21		SU - 109	M - 105	M - 105	Rel - 107	
22		M - 102	D - 107	M - 102	Ku - 112	
31		M - 105	SU - 112	D - 111	El - 105	Eng - 110
32		SU - 104	SU - 104	D - 112	M - 102	Rel - 105
41		Ku - 112	D - 103	SU - 103	M - 109	Klas - 103
42		D - 107	SU - 111	D - 107	Mus - 101	M - 111

Schule - Wednesday

	1	2	3	4	5	6
11		Sp - 106	D - 103	M - 102	D - 103	
12		SU - 111	Sp - 107	Ku - 107	M - 111	
21		Mus - 101	Klas - 108	Ku - 108	D - 108	
22		Ku - 112	M - 102	Mus - 101	Sp - 107	
31	Mus - 101	Ku - 105	M - 105	D - 111	SU - 112	Rel - 105
32	D - 112	M - 102	D - 112	Rel - 105	Ku - 102	Ku - 102
41	D - 103	Pat - 103	Mus - 101	SU - 103	Rel - 105	D - 103
42	D - 107	D - 107	M - 111	SU - 112	Mus - 101	M - 111

Schule - Thursday

	1	2	3	4	5	6
11		M - 102	D - 103	Rel - 105	Sp - 106	
12		D - 110	M - 111	Ku - 111	Rel - 111	
21		Sp - 106	M - 105	D - 108	D - 108	
22		Sp - 107	D - 107	D - 107	Rel - 107	
31		D - 111	SU - 112	SU - 112	Sp - 105	M - 105
32		D - 112	Mus - 101	Eng - 102	M - 102	Statt AG - 102
41		D - 103	M - 109	SU - 103	Ku - 112	Eng - 108
42		Rel - 105	Sp - 106	Sp - 106	Eng - 104	M - 111

Schule - Friday

	1	2	3	4	5
11		M - 102	SU - 104	D - 103	Ku - 103
12		M - 111	D - 110	D - 110	Sp - 111
21		D - 108	Sp - 108	M - 105	Ku - 108
22		D - 107	D - 107	M - 102	Klas - 107
31	Sp - 105	M - 105	D - 111	D - 111	M - 105

	1	2	3	4	5
32	Sp - 112	SU - 104	M - 102	D - 112	Klas - 102
41	D - 103	SU - 103	Rel - 105	M - 109	M - 109
42	M - 111	Ku - 112	Ku - 112	D - 107	SU - 112

A.2 Berechnungen Auswertung

A.2.1 Schulplan

- EA: Unterschied der Resultate: 0,83

- EoH: Unterschied der Resultate: 0,91

- EoS: Unterschied der Resultate: 0,72

- Prompting: Unterschied der Resultate: 0,75

- Beim Prompting wurde für Skriptdurchläufe, wo das System auf einen Error lief, der Wert 300 eingetragen als Ersatz. Das System läuft bei Plänen, welche in keinster Weise zu den Anforderungen passen, auf einen Error. Das ist in den meisten Fällen, wenn eine Klasse Unterrichtsstunden hat, die sie nicht benötigt.

Tabelle 11: Schule: Berechnung der Werte für EA

LLM-Calls	Avg-Fitness	Fehler Skript	Differenz	False Positive	True Positives
33	0	0	0	0	1
27	0	0	0	0	1
35	1	2	1	0	0
36	2	2	0	0	0
27	0	0	0	0	1
34	0	0	0	0	1
26	0	1	1	1	0
27	0	1	1	1	0
34	0,3	0,3	0	0	1
26	0	0	0	0	1
29	0	1	1	1	0
26	0	1	1	1	0
38	1	1	0	0	0
26	0	1	1	1	0
38	1	1	0	0	0
37	1	1	0	0	0
29	0	0	0	0	1
31,06	0,37	0,72	0,35	0,29	0,41

Tabelle 12: Schule: Berechnung der Werte für EoH

LLM-Calls	Avg-Fitness	Fehler Skript	Differenz	False Positive	True Positives
218	0	29,3	29,3	1	0
76	0	69	69	1	0
90	0	49	49	1	0
94	0,3	8	7,7	0	0
83	0	217	217	1	0
33	0	17	17	1	0
99	0,05	64,88	64,83	0,83	0

Tabelle 13: Schule: Berechnung der Werte für EoS

LLM-Calls	Avg-Fitness	Fehler Skript	Differenz	False Positive	True Positives
80	0,01	68	67,99	0	0
117	0,08	63	62,92	0	0
103	0,12	52,3	52,18	0	0
68	0	68	68	1	0
79	0	54	54	1	0
95	0,24	65	64,76	0	0
90,33	0,08	61,72	61,64	0,33	0

Tabelle 14: Schule: Berechnung der Werte für Prompting

LLM-Calls	Avg-Fitness	Fehler Skript	Differenz	False Positive	True Positives
2	0	125	125	1	0
2	0	300	300	1	0
2	0	300	300	1	0
2	0	300	300	1	0
2	0	300	300	1	0
2	0	300	300	1	0
2	4,5	118	113,5	0	0
2	0	300	300	1	0
2	0	300	300	1	0
2	0	300	300	1	0
2	0	186	186	1	0
2	0	300	300	1	0
2	0,38	260,75	260,38	0,92	0

A.2.2 Zugplan

EA: Unterschied der Resultate: 0,80

EoH: Unterschied der Resultate: 0,84

EoS: Unterschied der Resultate: 0,69

Prompting: Unterschied der Resultate: 0,76

Tabelle 15: Zugplan: Berechnung der Werte für EA

LLM-Calls	Avg-Fitness	Fehler Skript	Differenz	False Positive	True Positives
43	0	0	0	0	1
40	0	5	5	1	0
44	0	1	1	1	0
40	0	1	1	1	0
48	0,3	0,3	0	0	1
47	0,3	0,3	0	0	1
46	0,6	0,6	0	0	1
50	0,6	0,3	0,3	0	1
49	0,9	1	0,1	0	0
47	0,9	1,6	0,7	0	0
51	0,6	9,6	9	0	0
49	0	0	0	0	1
50	0,6	0,6	0	0	1
50	0,6	0,6	0	0	1
49	0,3	0	0,3	0	1
50	0,3	0,3	0	0	1
50	0,6	1	0,4	0	0
47,24	0,39	1,36	1,05	0,18	0,59

Tabelle 16: Zugplan: Berechnung der Werte für EoH

LLM-Calls	Avg-Fitness	Fehler Skript	Differenz	False Positive	True Positives
119	0,02	1,9	1,88	0	0
85	0,2	3,6	3,4	0	0
86	3,87	7,2	3,33	0	0
87	0,1	3,9	3,8	0	0
86	0,02	2,9	2,88	0	0
89	0,35	1,6	1,25	0	0
92	0,76	3,52	2,76	0	0

Tabelle 17: Zugplan: Berechnung der Werte für EoS

LLM-Calls	Avg-Fitness	Fehler Skript	Differenz	False Positive	True Positives
68	0,1	3,2	3,1	0	0
81	6,04	2,2	3,84	0	0
68	4,04	2,6	1,44	0	0
79	5,55	3,6	1,95	0	0
44	0	2,6	2,6	1	0
76	0,1	3,3	3,2	0	0
69,33	2,64	2,92	2,69	0,17	0

Tabelle 18: Zugplan: Berechnung der Werte für Prompting

LLM-Calls	Avg-Fitness	Fehler Skript	Differenz	False Positive	True Positives
2	3,7	71,2	67,5	0	0
2	2,5	35,2	32,7	0	0
2	4,2	75,8	71,6	0	0
2	0	29,8	29,8	1	0
2	2,2	53,6	51,4	0	0
2	0	55,8	55,8	1	0
2	0	45,2	45,2	1	0
2	6,4	70,6	64,2	0	0
2	0	64,6	64,6	1	0
2	0	46,4	46,4	1	0
2	0	48,8	48,8	1	0
2	12,3	60,8	48,5	0	0
2	2,61	54,82	52,21	0,5	0

A.2.3 Maschinenplan

EA: Unterschied der Resultate: 0,88

EoH: Unterschied der Resultate: 0,87

EoS: Unterschied der Resultate: 0,72

Prompting: Unterschied der Resultate: 0,48

Tabelle 19: Maschinen: Berechnung der Werte für EA

LLM-Calls	Avg-Fitness	Fehler Skript	Differenz	False Positive	True Positives
33	7,3	3,3	4	0	0
24	0	0	0	0	1
31	6,3	6,3	0	0	0
31	10,3	14,3	4	0	0
34	2	2	0	0	0
33	17	6	11	0	0
34	2	5	3	0	0
29	0	3	3	1	0
27	0	0	0	0	1
33	0	0	0	0	1
31	7,3	2,3	5	0	0
33	20	47	27	0	0
31	13	34	21	0	0
30	3	4	1	0	0
32	4,3	26,3	22	0	0
31,07	6,17	10,23	6,73	0,07	0,2

Tabelle 20: Maschinen: Berechnung der Werte für EoH

LLM-Calls	Avg-Fitness	Fehler Skript	Differenz	False Positive	True Positives
87	0,84	15,3	14,46	0	0
99	1,22	33	31,78	0	0
91	0,64	9	8,36	0	0
88	1,13	14,3	13,17	0	0
84	0,75	13	12,25	0	0
91	0,68	13	12,32	0	0
90	0,88	16,27	15,39	0	0

Tabelle 21: Maschinen: Berechnung der Werte für EoS

LLM-Calls	Avg-Fitness	Fehler Skript	Differenz	False Positive	True Positives
72	0,44	56	55,56	0	0
70	1,48	33,3	31,82	0	0
77	2,2	27,3	25,1	0	0
79	1,76	28	26,24	0	0
77	1,15	25,3	24,15	0	0
73	1,68	32,3	30,62	0	0
74,67	1,45	33,7	32,25	0	0

Tabelle 22: Maschinen: Berechnung der Werte für Prompting

LLM-Calls	Avg-Fitness	Fehler Skript	Differenz	False Positive	True Positives
2	0	33,3	33,3	1	0
2	2,7	34,3	31,6	0	0
2	0	20,3	20,3	1	0
2	2,4	52,3	49,9	0	0
2	13,4	50,3	36,9	0	0
2	0,3	30,3	30	0	0
2	0	50,3	50,3	1	0
2	8,4	25,3	16,9	0	0
2	4,4	27,3	22,9	0	0
2	0	27,3	27,3	1	0
2	3,16	35,1	31,94	0,4	0

A.3 Liste der Repositorys

Im Rahmen dieser Masterarbeit wurde für die verschiedenen Verfahren jeweils ein Repository angelegt, in welchem die Implementierung, die Verwendeten Testdaten und eine Beschreibung der Verwendung ist. Folgend sind die Links zu den Repositorys gelistet:

- **Prompting-Ansatz:** https://github.com/bjarne999/Timetabling_Prompting
- **Evolution of Schedules-Ansatz:** https://github.com/bjarne999/Timetabling_EoS

- **Evolution of Heuristics-Ansatz:** https://github.com/bjarne999/Timetabling_EoH
- **Evolutionary Algorithm-Ansatz:** https://github.com/bjarne999/Timetabling_Evolutionary_Algorithm

Alle Implementierungen sind jeweils im *main* Branch enthalten. Zusätzlich gibt es noch ein Repository, mit welchem die Tests durchgeführt wurden und die Konvergenzdiagramme erstellt wurde. Dieses ist das folgende:

Test-Repository: https://github.com/bjarne999/Master-Thesis-Reports

Erklärung zur selbstständigen Bearbeitung

Hiermit versichere ich, dass ich die vorliegende Arbeit ohne fremde Hilfe selbständig verfasst und nur die angegebenen Hilfsmittel benutzt habe. Wörtlich oder dem Sinn nach aus anderen Werken entnommene Stellen sind unter Angabe der Quellen kenntlich gemacht.

Ort	Datum	Unterschrift im Original