Generierung, Schwierigkeitsbewertung und Lösung von Buraitoraito-Puzzles mittels deduktiver Suche: Entwicklung und systematischer Vergleich

Bachelorarbeit

Hoang Long Vu Luong
0460185

6. April 2026

Betreuer: Prof. Dr. Benjamin Blankertz
Dr.-Ing. Stefan Fricke

Technische Universität Berlin
Fakultät IV – Elektrotechnik und Informatik
Institut für Softwaretechnik und Theoretische Informatik
Fachgebiet Neurotechnologie

Kurzfassung

Diese Arbeit untersucht die algorithmische Schwierigkeitsbewertung von Buraitoraito-Instanzen. Ziel ist es, die Schwierigkeit eines Rätsels nicht subjektiv einzuschätzen, sondern anhand reproduzierbarer Kennzahlen aus einem deduktionsgeleiteten Lösungsprozess abzuleiten. Zu diesem Zweck wird ein deduktionsbasierter Solver implementiert, der aus einem gegebenen Spielzustand logisch ableitbare Konsequenzen nutzt und nur bei Bedarf durch begrenzte Suche ergänzt.

Auf Basis des Lösungsprozesses werden verschiedene Kennzahlen zur Schwierigkeitsquantifizierung bestimmt. Dazu zählen insbesondere der deduktive Schwierigkeitswert (DS), der die Struktur und Anzahl der notwendigen Deduktionsschritte berücksichtigt, sowie das Level of Embedding (LoE), das die maximale Verschachtelung von Deduktionen beschreibt. Diese Metriken werden auf mehreren Instanzkorpora systematisch ausgewertet und hinsichtlich ihrer Verteilung und Aussagekraft analysiert.

Zusätzlich wird ein Generierungsverfahren implementiert, das neue Buraitoraito-Instanzen erzeugt und diese mithilfe des Solvers automatisch bewertet. Auf dieser Grundlage werden Instanzen nach Schwierigkeitsklassen eingeteilt und die Generierbarkeit verschiedener Schwierigkeitsgrade empirisch untersucht.

Darüber hinaus werden die vom Solver berechneten Schwierigkeitskennzahlen mit extern vergebenen Schwierigkeitsstufen aus Datensätzen der Plattformen GrandGames und Janko verglichen, um den Zusammenhang zwischen algorithmischer Bewertung und menschlicher Schwierigkeitseinschätzung zu analysieren.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

1. Einleitung

1.1. Motivation

Logikrätsel wie Sudoku, Nonogramme oder Slitherlink wurden in der Forschung bereits intensiv untersucht. Dabei stehen insbesondere die algorithmische Lösbarkeit sowie die Modellierung als Constraint-Satisfaction-Problem im Fokus [1]. Darüber hinaus wurden Ansätze zur Bewertung der Schwierigkeit entwickelt, die auf der Analyse deduktiver Lösungsprozesse basieren, wie sie im Deductive-Search-Ansatz von Browne beschrieben werden [2].
Ein zentraler Gedanke dieser Arbeiten ist, dass die Schwierigkeit eines Puzzles nicht ausschließlich von der Größe oder der Anzahl möglicher Lösungen abhängt, sondern maßgeblich durch die Struktur der während des Lösungsprozesses erforderlichen Deduktionsschritte bestimmt wird. Insbesondere die Einbettungstiefe von Hypothesen sowie die Art der angewendeten Regeln liefern Hinweise auf die kognitive Komplexität eines Rätsels.
Im Gegensatz zu diesen gut untersuchten Puzzleklassen existieren für das Logikrätsel Buraitoraito bislang nur wenige wissenschaftliche Arbeiten. Insbesondere fehlen systematische Untersuchungen zur algorithmischen Schwierigkeitsbewertung sowie zur automatischen Generierung von Instanzen.
Diese Forschungslücke bildet die Motivation der vorliegenden Arbeit. Ziel ist es, einen deduktionsgeleiteten Solver für Buraitoraito zu entwickeln und darauf aufbauend eine quantitative Schwierigkeitsmetrik abzuleiten, die auf der Struktur des Lösungsprozesses basiert. Dabei wird der Deductive-Search-Ansatz von Browne auf die spezifischen Eigenschaften von Buraitoraito übertragen und um domänenspezifische Deduktionsregeln erweitert.
Darüber hinaus wird ein Verfahren zur automatischen Instanzgenerierung entwickelt, das es ermöglicht, Puzzle gezielt nach Schwierigkeitsklassen zu erzeugen. Die Kombination aus Solver, Schwierigkeitsbewertung und Generierung erlaubt somit eine systematische Analyse der Zusammenhänge zwischen Puzzle-Struktur und Schwierigkeit.

1.2. Algorithmische Komplexität

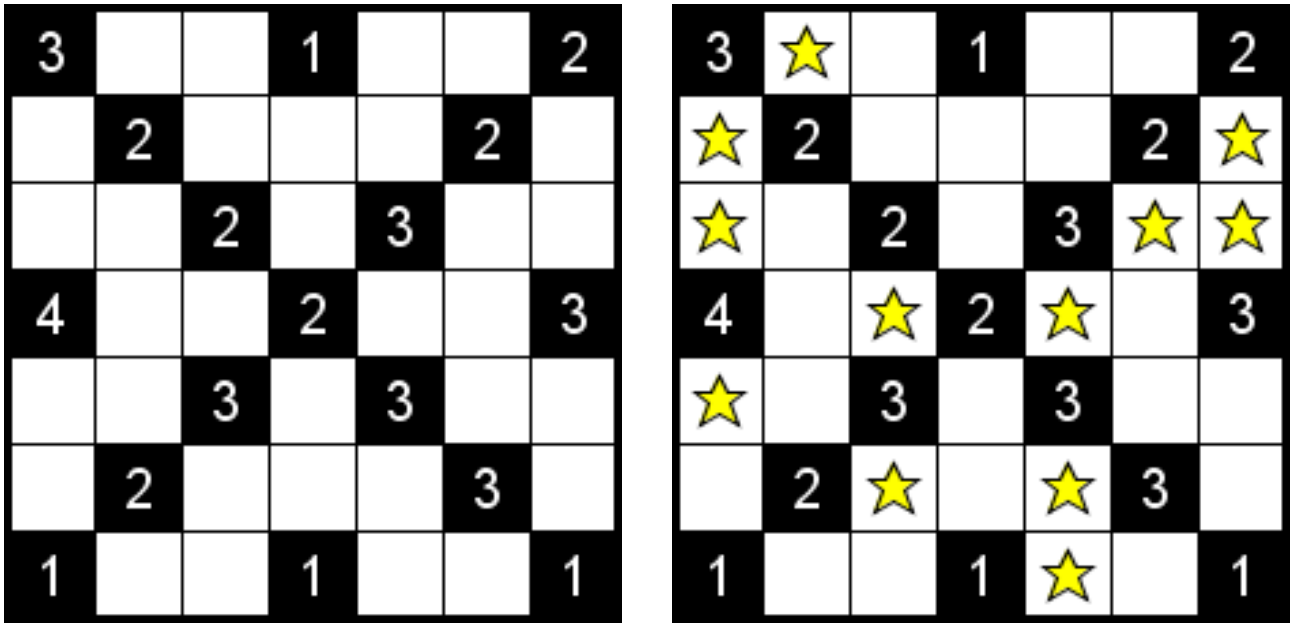
Viele Logikrätsel erzeugen aufgrund der großen Anzahl möglicher Variablenbelegungen sehr große Suchräume. Daher wurden verschiedene Rätseltypen auch aus Sicht der theoretischen Informatik untersucht.
Für mehrere Puzzle mit ähnlichen Strukturmerkmalen wurde gezeigt, dass ihre Entscheidungsvariante NP-vollständig ist. Ein bekanntes Beispiel ist das Rätsel Akari (auch *Light Up*), das ebenfalls auf Sichtlinienregeln und nummerierten Blockfeldern basiert. Für dieses Puzzle wurde gezeigt, dass die Entscheidungsvariante NP-vollständig ist [3, S. 1–3]. Das Resultat wird auch in späteren veröffentlichten Arbeiten zur Komplexität von Akari explizit aufgegriffen [4].
Buraitoraito weist eine sehr ähnliche Struktur auf. Sterne werden in ein Gitter gesetzt und müssen numerische Sichtlinienbedingungen erfüllen. Aufgrund dieser strukturellen Ähnlichkeit ist davon aus-

zugehen, dass auch Buraitoraito eine vergleichbare kombinatorische Komplexität besitzt.

1.3. Spielprinzip von Buraitoraito

Buraitoraito (englisch *Bright Light*) ist ein Logikrätsel auf einem rechteckigen Gitter aus weißen und schwarzen Feldern. Einige schwarze Felder enthalten Zahlenhinweise. Ziel ist es, Sterne in ausgewählte weiße Felder einzutragen, sodass alle Zahlenbedingungen erfüllt sind.[1]
Die Regeln des Spiels lauten:

1. Sterne dürfen nur in weiße Felder eingetragen werden.
2. Die Zahl in einem schwarzen Feld gibt an, wie viele Sterne insgesamt sichtbar sind, wenn man von diesem Feld aus nach Norden, Süden, Osten oder Westen blickt.
3. Die Sichtlinie endet am Rand des Gitters oder am nächsten schwarzen Feld.
4. Sterne blockieren die Sicht nicht.



(a) Instanz (b) Lösung

Abbildung 1.1.: Beispiel einer Buraitoraito-Instanz (links) und eine Lösung (rechts).

1.4. Zielsetzung der Arbeit

Ziel dieser Arbeit ist die algorithmische Analyse und Bewertung von Buraitoraito-Instanzen auf Basis eines deduktionsgeleiteten Lösungsansatzes. Hierzu wird ein Solver entwickelt, der deterministische Deduktionsregeln mit systematischer Suche kombiniert.

[1] https://www.janko.at/Raetsel/Buraitoraito/index.htm, Zugriff am 27.02.2026.

Auf Grundlage des Lösungsprozesses werden Kennzahlen zur Quantifizierung der Schwierigkeit eines Rätsels definiert. Diese Metriken basieren unter anderem auf der Anzahl und Struktur der angewendeten Deduktionsschritte sowie auf der Tiefe notwendiger Suchentscheidungen.
Zusätzlich wird ein Verfahren zur automatischen Generierung von Buraitoraito-Instanzen entwickelt. Die generierten Rätsel werden mithilfe des Solvers analysiert und anschließend nach Schwierigkeitsbereichen klassifiziert.
Der Beitrag dieser Arbeit besteht in der Entwicklung eines deduktionsgeleiteten Solvers für Buraitoraito, der Definition und Implementierung einer darauf aufbauenden Schwierigkeitsmetrik sowie in der systematischen Analyse und Generierung von Instanzen unterschiedlicher Schwierigkeit.

2. Methoden

Dieses Kapitel beschreibt die in dieser Arbeit verwendeten Methoden zur Analyse und Generierung von Buraitoraito-Instanzen. Zunächst wird das zugrunde liegende Problem formal beschrieben. Anschließend wird die interne Repräsentation des Spielfelds erläutert sowie der implementierte deduktionsgeleitete Solver vorgestellt. Darauf aufbauend werden die verwendeten Deduktionsregeln, die Berechnung der Schwierigkeitsmetriken sowie das Verfahren zur Instanzgenerierung beschrieben.

2.1. Problemformulierung

Ein Buraitoraito-Puzzle wird als rechteckiges Gitter G mit Höhe H und Breite W modelliert. Jede Zelle $(r, c) \in G$ ist entweder weiß oder schwarz, wobei ein Teil der schwarzen Zellen einen Zahlenhinweis enthält.
Für jede weiße Zelle wird eine binäre Variable

$$x_{r,c} \in \{0, 1\}$$

eingeführt, wobei

$$x_{r,c} = \begin{cases} 1 & \text{falls ein Stern gesetzt wird} \\ 0 & \text{sonst.} \end{cases}$$

Für eine nummerierte schwarze Zelle b mit Wert $k(b)$ gibt $k(b)$ die Anzahl der Sterne an, die in den von b sichtbaren weißen Zellen platziert werden müssen.
Die Menge $vis(b)$ bezeichnet alle weißen Zellen, die von b aus in orthogonalen Richtungen sichtbar sind. Die Sichtlinie endet jeweils am Rand des Gitters oder an der nächsten schwarzen Zelle.
Damit beschreibt $vis(b)$ genau diejenigen Variablen, die durch die Nebenbedingung von b eingeschränkt werden.
Die zentrale Bedingung des Puzzles lautet:

$$\sum_{(r,c) \in vis(b)} x_{r,c} = k(b)$$

für jede Hinweiszelle b.
Damit lässt sich Buraitoraito als Constraint-Satisfaction-Problem (CSP) formulieren, bei dem die Variablen durch die weißen Zellen, die Domänen durch $\{0, 1\}$ und die Constraints durch die Sichtlinienbedingungen gegeben sind [1, vgl. Kap. 6].

2.1.1. Wahl der Programmiersprachen

Die Implementierung des Solvers erfolgte in C++. Diese Wahl wurde vor allem durch die Anforderungen an Effizienz und Kontrolle über die verwendeten Datenstrukturen motiviert. C++ erlaubt eine speichereffiziente Implementierung sowie direkte Kontrolle über Speicherlayout und bitweise Operationen, was insbesondere für zustandsintensive Algorithmen und Suchverfahren von Vorteil ist. Der entwickelte Solver verwendet eine Bitboard-basierte Repräsentation des Spielfelds sowie zahlreiche bitweise Operationen zur Aktualisierung von Zuständen und Kandidatenmengen. Solche Operationen lassen sich in C++ effizient implementieren und sind daher für performante Puzzle-Solver geeignet.
Die Instanzgenerierung wurde hingegen in Python umgesetzt. Ein wesentlicher Grund hierfür ist die Verfügbarkeit der Optimierungsbibliothek OR-Tools, die eine komfortable Python-Schnittstelle bereitstellt und häufig zur Modellierung kombinatorischer Optimierungsprobleme eingesetzt wird.[1] Python eignet sich zudem gut für experimentelle Auswertungen und Skripting, da komplexe Workflows mit vergleichsweise geringem Implementierungsaufwand realisiert werden können.
Die Kombination beider Sprachen ermöglicht somit eine effiziente Solverimplementierung in C++ bei gleichzeitig flexibler Experiment- und Generierungsumgebung in Python.

2.2. Infrastruktur

Die Implementierung des Solvers basiert auf einer effizienten Repräsentation des Spielfelds sowie auf spezialisierten Datenstrukturen zur Verwaltung des aktuellen Lösungszustands.
Zur Darstellung des Gitters wird eine Bitboard-basierte Repräsentation verwendet, bei der jede Zelle durch ein Bit kodiert wird. Dadurch können Mengen von Zellen, etwa gesetzte Sterne, ausgeschlossene Positionen oder Kandidatenmengen, durch Bitmasken dargestellt werden. Mengenoperationen wie Vereinigung, Schnitt und Differenz lassen sich so effizient mittels bitweiser Operationen durchführen [5].
Zusätzlich werden für jede Hinweiszelle die zugehörigen orthogonalen Sichtsegmente vorab berechnet und gespeichert. Dadurch können Deduktionsregeln gezielt auf relevanten Teilstrukturen des Gitters arbeiten, ohne in jedem Schritt das gesamte Spielfeld erneut analysieren zu müssen.
Diese Infrastruktur bildet die Grundlage für den in den folgenden Abschnitten beschriebenen deduktionsgeleiteten Lösungsansatz.

2.3. Deduktive Suche

Der in dieser Arbeit verwendete Solver orientiert sich am Deductive-Search-Ansatz von Browne, bei dem Deduktion als Kombination aus *Simplification*, *Shaving* und *Agreement* modelliert wird. Aus diesem Ansatz werden in der vorliegenden Arbeit insbesondere die Einteilung in LoE-Stufen sowie die Grundidee des Deductive Score übernommen. Die konkrete Ausgestaltung der Deduktionsregeln und die genaue Zählweise der Updates werden jedoch an die Buraitoraito-Modellierung und die implementierte Solverlogik angepasst.

[1] https://developers.google.com/optimization, Zugriff am 18.03.2026.

Ausgangspunkt ist ein aktueller Zustand S, der die noch nicht festgelegten Variablen und deren zulässige Werte beschreibt. Der Solver arbeitet mit drei Arten von Deduktionsschritten:

- **Simplify:** deterministische Propagation lokaler und globaler Konsistenzbedingungen,
- **Shaving:** testweises Annehmen eines Werts mit anschließender Widerspruchsprüfung,
- **Agreement:** Übernahme von Konsequenzen, die unter allen betrachteten Hypothesen auftreten.

Der Lösungsprozess ist in verschiedene *Level of Embedding* (LoE) unterteilt. Diese Ebenen unterscheiden sich darin, wie tief hypothetische Annahmen verschachtelt werden. In der vorliegenden Implementierung werden LoE0, LoE1 und LoE2 betrachtet.

2.3.1. LoE0: Simplify bis zum Fixpunkt

Auf der Ebene LoE0 werden ausschließlich deterministische Deduktionsregeln angewendet. Diese Regeln werden iterativ wiederholt, bis ein Fixpunkt erreicht ist, also wenn keine weitere Regel mehr den Zustand verändert. In der Implementierung entspricht dies der Funktion `simplify_0LoE()`.
Falls dabei ein Widerspruch auftritt, ist der aktuelle Zustand inkonsistent. Falls alle Variablen festgelegt sind, ist das Puzzle gelöst. LoE0 beschreibt damit einen rein deterministischen Lösungsprozess ohne hypothetische Annahmen.

2.3.2. LoE1: Eine Hypothese mit Simplifizierung

Falls LoE0 keinen vollständigen Lösungszustand liefert, betrachtet der Solver auf LoE1 für eine noch nicht festgelegte weiße Zelle die beiden komplementären Hypothesen, dass sich auf dieser Zelle ein Stern befindet beziehungsweise dass sich dort kein Stern befindet. Jede dieser Annahmen wird anschließend durch die deterministischen Regeln aus LoE0 weiterverarbeitet. In der Implementierung entspricht dies der Funktion `step_1LoE()`, die intern Hypothesen bildet und nach jeder Annahme erneut `simplify_0LoE()` ausführt.
Da die zugrunde liegende Modellierung binär ist, können die beiden zentralen Deduktionsprinzipien Shaving und Agreement in einer vereinfachten Form beschrieben werden.

Shaving und Agreement auf LoE1

Shaving. Beim Shaving wird eine Hypothese testweise angenommen und anschließend durch deterministische Deduktion weiterverarbeitet. Führt eine der beiden Hypothesen zu einem Widerspruch, kann sie ausgeschlossen werden. In der binären Modellierung bedeutet dies unmittelbar, dass die jeweils komplementäre Belegung gelten muss.
Formal sei $X_i \in \{0, 1\}$ eine binäre Variable. Dann gilt:

$$X_i = v \Rightarrow \bot \quad \Rightarrow \quad X_i = 1 - v.$$

Im Kontext von Buraitoraito entspricht dies den beiden Fällen:

$$X_i = \text{Stern} \Rightarrow \bot \quad \Rightarrow \quad X_i = \text{kein Stern},$$

beziehungsweise

$$X_i = \text{kein Stern} \Rightarrow \bot \quad \Rightarrow \quad X_i = \text{Stern}.$$

Agreement. Beim Agreement werden beide konsistenten Hypothesen für dieselbe binäre Variable betrachtet. Stimmen die aus ihnen abgeleiteten Konsequenzen für andere Variablen überein, können diese gemeinsamen Konsequenzen direkt in den globalen Zustand übernommen werden.
Formal gelte für $X_i \in \{0, 1\}$ und eine weitere Variable X_j:

$$(X_i = 0 \Rightarrow C_j) \ \wedge \ (X_i = 1 \Rightarrow C_j) \quad \Rightarrow \quad C_j,$$

wobei C_j eine unter beiden Hypothesen identische Konsequenz über X_j bezeichnet.
In der vorliegenden Modellierung sind solche Konsequenzen insbesondere Festlegungen der Form „X_j ist Stern“ oder „X_j ist kein Stern“.
Da die Variablen binär sind, genügt hier die Betrachtung genau zweier komplementärer Hypothesen. In allgemeineren, mehrwertigen Domänen müsste Shaving als Eliminierung einzelner Domänenwerte und Agreement als Übernahme von Folgerungen beschrieben werden, die unter allen noch zulässigen Werten einer Variablen auftreten.

2.3.3. LoE2: Zwei verschachtelte Hypothesen

Falls auf LoE1 weiterhin keine vollständige Lösung erreicht wird, kann der Solver auf LoE2 übergehen. Dabei werden wiederum für eine noch nicht festgelegte Zelle die beiden Hypothesen „Stern“ und „kein Stern“ betrachtet. Der Unterschied zu LoE1 besteht darin, dass innerhalb jeder dieser Hypothesen nicht nur eine reine Simplifizierung auf LoE0 ausgeführt wird, sondern zusätzlich eine Deduktion auf LoE1. Damit entsteht eine zweite Einbettungsebene hypothetischer Schlüsse.
In der Implementierung entspricht dies der Funktion `step_2LoE()`. Diese verwendet für jede Hypothese eine rekursive Auswertung, in der zunächst `simplify_0LoE()` und anschließend LoE1-Deduktion angewendet wird. Shaving und Agreement werden auf LoE2 daher nach demselben Prinzip wie auf LoE1 ausgeführt, jedoch auf Grundlage stärker ausgeschöpfter Hypothesenräume. Dadurch lassen sich zusätzliche indirekte Folgerungen ableiten, die auf LoE1 noch nicht zugänglich sind.

2.3.4. Einordnung der vorliegenden Implementierung

Die vorliegende Implementierung verwendet zunächst LoE0 zur deterministischen Vereinfachung des Zustands. Falls der Fixpunkt noch keine vollständige Lösung liefert, werden auf LoE1 Shaving- und Agreement-Schritte ausgeführt. Reicht auch dies nicht aus, wird LoE2 verwendet. Die während dieses Prozesses protokollierten Deduktionsschritte bilden die Grundlage für die spätere Schwierigkeitsbewertung.

2.4. Deduktionsregeln

Vor dem Einsatz hypothetischer Annahmen werden im Solver zunächst deterministische Deduktionsregeln angewendet, um eine Instanz möglichst weit ohne Verzweigung zu vereinfachen. Diese Regeln werden in der Simplify-Phase iterativ bis zum Fixpunkt wiederholt. In der vorliegenden Implementierung sind die Regeln `step_fill_all_needed()`, `step_block_when_complete()` und `step_global_segment_bounds()` implementiert. Im aktuellen Auswertungsstand erfolgt die Simplify-Phase in `simplify_0LoE()` über `step_global_segment_bounds()`.
Jede Anwendung einer Regel wird protokolliert. Diese Protokolle werden später zur Berechnung der Schwierigkeitsmetriken verwendet.

2.4.1. Alle Kandidaten sind notwendig

(`step_fill_all_needed`)

Intuition. Wenn genau so viele mögliche Positionen vorhanden sind, wie noch Sterne benötigt werden, müssen alle diese Positionen Sterne enthalten.

Formale Bedingung. Sei b eine Hinweiszelle mit Hinweiswert $k(b)$. Sei $S(b)$ die Menge bereits gesetzter Sterne und $A(b)$ die Menge zulässiger Kandidaten.
Gilt

$$|A(b)| = k(b) - |S(b)|,$$

so müssen alle Zellen in $A(b)$ Sterne enthalten.

Wirkung. Alle Kandidatenpositionen in $A(b)$ werden zu Sternen gesetzt.

(a) Vor Anwendung (b) Nach Anwendung

Abbildung 2.1.: Schematische Darstellung der Regel Alle Kandidaten sind notwendig.

Erklärung: Im Beispiel verlangt die Hinweiszelle insgesamt zwei Sterne. Da genau zwei mögliche Positionen verbleiben und noch keine Sterne gesetzt sind, müssen beide Kandidaten Sterne enthalten.

2.4.2. Keine weiteren Sterne erlaubt

(`step_block_when_complete`)

Intuition. Sobald eine Hinweiszelle bereits genügend Sterne besitzt, dürfen keine weiteren Sterne in ihren Sichtlinien platziert werden.

Formale Bedingung. Sei b eine Hinweiszelle mit Hinweiswert $k(b)$. Sei $S(b)$ die Menge bereits gesetzter Sterne in den Sichtlinien von b.
Gilt

$$|S(b)| = k(b),$$

so dürfen keine weiteren Sterne in den Sichtlinien von b gesetzt werden.

Wirkung. Alle verbleibenden Kandidaten in den Sichtlinien von b werden ausgeschlossen.

(a) Vor Anwendung (b) Nach Anwendung

Abbildung 2.2.: Schematische Darstellung der Regel Keine weiteren Sterne erlaubt.

Erklärung: Da die Hinweiszelle bereits die erforderliche Anzahl an Sternen besitzt, können alle übrigen Kandidaten in ihren Sichtlinien ausgeschlossen werden.

2.4.3. Globale Segmentbeschränkung

Die globale Segmentbeschränkung ist in der Implementierung als `step_global_segment_bounds` realisiert.
Im Unterschied zu rein lokalen Regeln betrachtet diese Regel nicht einzelne weiße Felder, sondern zusammenhängende weiße Segmente einer Zeile oder Spalte, die durch schwarze Felder begrenzt werden. Jedes solche Segment wird als gemeinsame Variable aufgefasst, deren Wert der Anzahl der in diesem Segment liegenden Sterne entspricht.
Für jede nummerierte schwarze Zelle entsteht damit eine Gleichung über die an sie angrenzenden Segmente. Ist c eine Hinweiszelle mit Hinweiswert $k(c)$ und sind $S_1, \ldots, S_m$ die von ihr aus sichtbaren Segmente, so gilt

$$S_1 + \cdots + S_m = k(c).$$

Für jedes Segment S wird im aktuellen Zustand eine untere Schranke $L(S)$ und eine obere Schranke $U(S)$ geführt. Dabei entsprechen diese Schranken der minimal beziehungsweise maximal noch möglichen Anzahl von Sternen in diesem Segment. Initial ergeben sie sich aus dem aktuellen Boardzustand als

$$L(S) = \text{have}(S), \qquad U(S) = \text{have}(S) + \text{free}(S),$$

wobei $\text{have}(S)$ die Zahl bereits festgelegter Sterne im Segment und $\text{free}(S)$ die Zahl der noch verfügbaren Zellen bezeichnet.
Aus einer Hinweisgleichung lassen sich diese Schranken für jedes beteiligte Segment verschärfen. Sei dazu

$$S_1 + \cdots + S_m = k(c)$$

eine solche Gleichung und S_i eines der beteiligten Segmente. Dann müssen die übrigen Segmente zusammen mindestens

$$\sum_{j \neq i} L(S_j)$$

und höchstens

$$\sum_{j \neq i} U(S_j)$$

Sterne beitragen. Daraus folgen für S_i die verschärften Schranken

$$L'(S_i) = \max\Big(L(S_i),\ k(c) - \sum_{j \neq i} U(S_j)\Big),$$

$$U'(S_i) = \min\Big(U(S_i),\ k(c) - \sum_{j \neq i} L(S_j)\Big).$$

Diese Verschärfung wird über alle Hinweisgleichungen iterativ wiederholt, bis ein Fixpunkt erreicht ist. Die globale Segmentbeschränkung besteht also nicht in einer einzelnen lokalen Setzregel, sondern in einer wiederholten Bounds-Propagation über ein System von Segmentgleichungen.
Erst nach Abschluss dieser Schrankenpropagation werden harte Konsequenzen in Zell-Updates übersetzt. Für ein Segment S ergeben sich insbesondere die folgenden beiden Fälle:

- Gilt
$$L(S) = \text{have}(S) + \text{free}(S),$$
so müssen alle noch verfügbaren Zellen des Segments Sterne sein.

- Gilt
$$U(S) = \text{have}(S),$$
so kann keine der noch verfügbaren Zellen des Segments einen Stern enthalten.

Der Kern der Regel besteht somit darin, dass Informationen mehrerer Hinweiszellen über gemeinsame Segmente zusammengeführt werden. Dadurch können Folgerungen entstehen, die aus keiner einzelnen Hinweiszelle allein bereits lokal ableitbar wären.

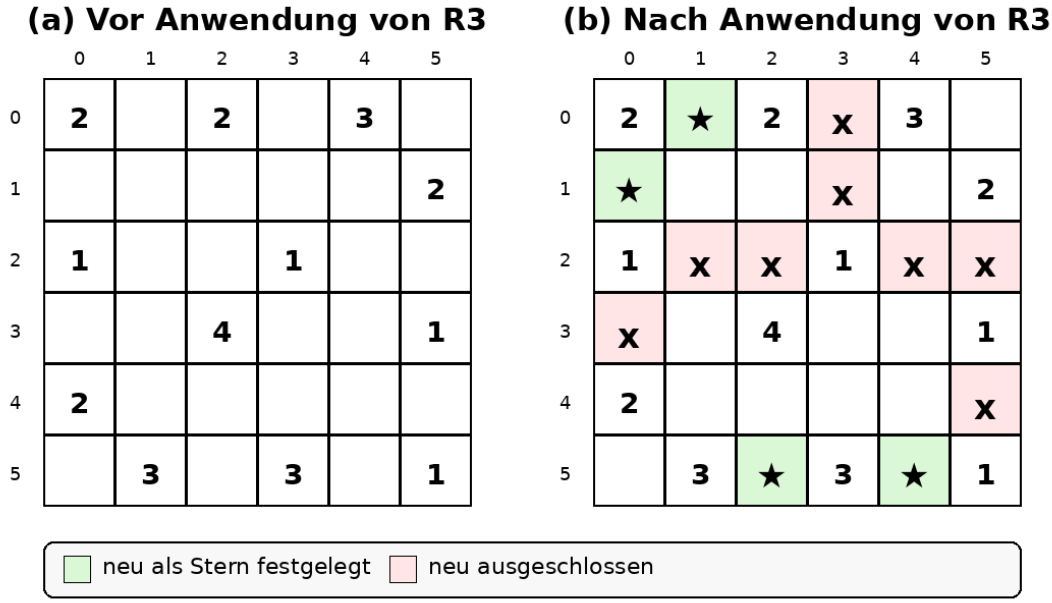


Abbildung 2.3.: Beispiel eines realen Solverzustands vor dem ersten Schritt von `SIMPLIFY/R3`. Rechts sind ausschließlich die in diesem Schritt neu erzeugten Änderungen dargestellt: Grün markiert sind neu als Sterne festgelegte Zellen, rot markiert neu ausgeschlossene Zellen. Die Abbildung illustriert die Wirkung der globalen Segmentbeschränkung im aktuellen Implementierungsstand.

Abbildung 2.3 zeigt einen realen Solverzustand vor dem ersten Schritt von `SIMPLIFY/R3` sowie die dadurch erzeugten Aktualisierungen. Die Abbildung dient dabei nur der Illustration der Wirkung der Regel. Der eigentliche Mechanismus besteht in der zuvor beschriebenen iterativen Verschärfung der unteren und oberen Schranken der Segmente über mehrere Hinweisgleichungen hinweg.
Im gezeigten Schritt werden die Zellen $(0, 1)$, $(1, 0)$, $(2, 5)$ und $(4, 5)$ als Sterne festgelegt, während die Zellen $(0, 3)$, $(1, 2)$, $(2, 2)$, $(3, 0)$, $(3, 1)$, $(4, 2)$, $(5, 2)$ und $(5, 4)$ ausgeschlossen werden. Im aktuellen Implementierungsstand werden diese Aktualisierungen durch `SIMPLIFY/R3` erzeugt und veranschaulichen damit die Wirkung der globalen Segmentbeschränkung in der vorliegenden Solverimplementierung.

2.5. Schwierigkeitsmetriken

Zur Bewertung der Schwierigkeit eines Buraitoraito-Puzzles werden Kennzahlen aus dem Lösungsprozess des deduktionsgeleiteten Solvers abgeleitet. Ziel ist es, die Schwierigkeit einer Instanz nicht subjektiv zu bestimmen, sondern aus der Struktur des algorithmischen Lösungsprozesses zu berechnen.
Die Bewertung orientiert sich am Konzept des *Deductive Score* (DS), das von Browne für deduktive Puzzle-Solver vorgeschlagen wurde [2]. Der Deductive Score ist ein quantitatives Maß für die

Schwierigkeit einer Instanz bei deduktiver Lösung. Er basiert auf den Zustandsaktualisierungen, die der Solver während des Lösungsprozesses vornimmt. Diese werden als einzelne Deduktionsschritte aufgefasst und entsprechend ihrer Einbettungstiefe gewichtet.
Damit berücksichtigt der Deductive Score sowohl die Anzahl der durchgeführten Deduktionsschritte als auch deren strukturelle Komplexität. Ein höherer Wert weist auf einen größeren deduktiven Aufwand und damit auf eine höhere Schwierigkeit der Instanz hin.

2.5.1. Zählung der Aktualisierungen

Für die Berechnung des Deductive Score werden nur solche Zustandsaktualisierungen berücksichtigt, die nach einem Deduktionsschritt tatsächlich in den globalen Puzzlezustand übernommen werden. Zwischenschritte, die ausschließlich innerhalb hypothetischer Prüfungen bei Shaving oder Agreement auftreten, gehen nicht direkt in die Zählung ein.
Die erfassten Aktualisierungen werden nach der Art des Deduktionsschritts klassifiziert, durch den sie in den globalen Zustand übernommen werden. Dabei werden Vereinfachungen durch deterministische Deduktion der Kategorie `Simplify` zugeordnet, Eliminierungen aufgrund widersprüchlicher Hypothesen der Kategorie `Shaving` und gemeinsame Folgerungen aus mehreren konsistenten Hypothesen der Kategorie `Agreement`.
Entsprechend werden folgende Größen erfasst:

- S_0: Anzahl der Updates durch `Simplify`-Schritte,
- A_1: Anzahl der Updates durch `Agreement`-Schritte auf LoE1,
- A_2: Anzahl der Updates durch `Agreement`-Schritte auf LoE2,
- V_1: Anzahl der Updates durch `Shaving`-Schritte auf LoE1,
- V_2: Anzahl der Updates durch `Shaving`-Schritte auf LoE2.

Ein Update entspricht dabei entweder der Festlegung einer Zelle als Stern (*forced*) oder dem Ausschluss einer Zelle als mögliche Sternposition (*forbidden*). Die Anzahl der von einem Deduktionsschritt erzeugten Updates ergibt sich somit als Summe der in diesem Schritt neu festgelegten und neu ausgeschlossenen Zellen.
Führt ein Shaving- oder Agreement-Schritt zu einer neuen Festlegung, so wird anschließend erneut die deterministische Simplify-Phase bis zum Fixpunkt ausgeführt. Die dabei entstehenden zusätzlichen Updates werden als `Simplify`-Schritte gezählt.

2.5.2. Gewichtung nach Einbettungstiefe

Die Gewichtung der Updates richtet sich nach der LoE-Stufe desjenigen Deduktionsschritts, durch den ein Update in den aktuellen Puzzlezustand übernommen wird. Maßgeblich ist also nicht eine abstrakte logische Verschachtelung hypothetischer Zwischeninferenzen, sondern die in der Implementierung protokollierte LoE-Stufe des jeweiligen Schritts.
Ein durch `Shaving` oder `Agreement` auf LoE1 beziehungsweise LoE2 übernommenes Update erhält daher das Gewicht der entsprechenden LoE-Stufe. Deterministische Folgeschritte, die nach einer solchen Übernahme in der erneut gestarteten `Simplify`-Phase auftreten, werden entsprechend ihrer protokollierten LoE-Stufe gezählt und gewichtet.
Für die LoE-Stufen 0, 1 und 2 ergeben sich damit aus der Gewichtungsfunktion

$$(n+1)^2$$

die Gewichte

$$1, \quad 4, \quad 9,$$

wobei n die jeweilige LoE-Stufe bezeichnet.
Diese Gewichtung geht in die Berechnung des Deductive Score ein, indem die protokollierten Updates mit dem jeweiligen Faktor gewichtet werden.

2.5.3. Sondergewichtung bestimmter Simplify-Regeln

Im Gegensatz zum ursprünglichen Ansatz von Browne werden in der vorliegenden Implementierung nicht alle Simplify-Schritte gleich behandelt.
Insbesondere wird die im Solver als `SIMPLIFY/R3` bezeichnete Regel stärker gewichtet. Dabei handelt es sich um die in Abschnitt 2.4.3 beschriebene globale Segmentbeschränkung.
Die entsprechenden Updates entstehen in der Funktion `step_global_segment_bounds()`, die innerhalb von `simplify_0LoE()` ausgeführt wird. Für nummerierte schwarze Zellen werden dabei die zugehörigen Sichtsegmente betrachtet und untere sowie obere Schranken für die Anzahl möglicher Sterne in einem Segment bestimmt.
Auf Grundlage dieser Schranken werden konkrete Aktualisierungen durchgeführt. Felder werden gesetzt, wenn ein Segment vollständig belegt werden muss, und ausgeschlossen, wenn in einem Segment keine weiteren Sterne möglich sind. Alle Updates, die aus dieser Regel resultieren, werden der Kategorie `SIMPLIFY/R3` zugeordnet und bei der Schwierigkeitsbewertung gesondert erfasst.
Sei $S_{0,R3}$ die Anzahl der durch diese Regel verursachten Updates. Dann ergibt sich die gewichtete Anzahl der Simplify-Updates als

$$S_0^{(w)} = S_0 + (w_{R3} - 1) \cdot S_{0,R3}$$

mit

$$w_{R3} = 2.$$

Die Wahl von $w_{R3} = 2$ ist eine bewusste heuristische Modellierungsentscheidung. Sie beruht auf der Annahme, dass die globale Segmentbeschränkung kognitiv anspruchsvoller ist als rein lokale deterministische Setz- oder Ausschlussregeln. Da sie Informationen mehrerer Hinweiszellen über gemeinsame Segmente zusammenführt, wird ihr Beitrag zur Schwierigkeit in der vorliegenden Arbeit stärker gewichtet.

2.5.4. Definition des Deductive Score

Der endgültige Schwierigkeitswert wird als gewichtete Summe der während des Lösungsprozesses erzeugten Updates berechnet. Dabei werden die durch `Simplify`, `Shaving` und `Agreement` erzeugten Aktualisierungen entsprechend ihrer Einbettungstiefe berücksichtigt.
In der vorliegenden Implementierung ergibt sich der Deductive Score einer Instanz S als

$$\mathrm{DS}(S) = \frac{S_0^{(w)} + 4 \cdot (V_1 + A_1) + 9 \cdot (V_2 + A_2)}{\sum_i D_i}.$$

Hierbei bezeichnet $S_0^{(w)}$ die gewichtete Anzahl der durch `Simplify` erzeugten Updates. Die Größen V_n und A_n bezeichnen die Anzahl der durch `Shaving` beziehungsweise `Agreement` erzeugten Updates auf der jeweiligen LoE-Stufe.
Der Nenner $\sum_i D_i$ dient zur Normierung auf die Instanzgröße. In der vorliegenden Implementierung wird dieser Wert als

$$\sum_i D_i = 2 \cdot W \cdot H$$

berechnet, wobei W und H die Breite und Höhe des Gitters bezeichnen.

Der Faktor 2 ergibt sich daraus, dass für jedes Feld zwei mögliche binäre Zustandszuweisungen betrachtet werden: Ein Feld kann entweder als Stern festgelegt oder als mögliche Sternposition ausgeschlossen werden. Damit dient $2 \cdot W \cdot H$ als größenabhängige Normierung der möglichen Zustandsaktualisierungen im Gitter.
Der deduktive Solver ist in der vorliegenden Arbeit als bewusst regelbasierter Auswertungsmechanismus zu verstehen. Er dient nicht als vollständiges Entscheidungsverfahren, sondern als Modell eines begrenzten deduktiven Lösungsverhaltens. Entsprechend ist nicht garantiert, dass jede gültige Buraitoraito-Instanz allein durch die implementierten Deduktionsregeln vollständig gelöst werden kann. Bleibt eine Instanz unter den verwendeten Regeln unvollständig, so ist dies als Grenze des gewählten deduktiven Modells zu interpretieren und nicht als Hinweis auf die Ungültigkeit der Instanz.

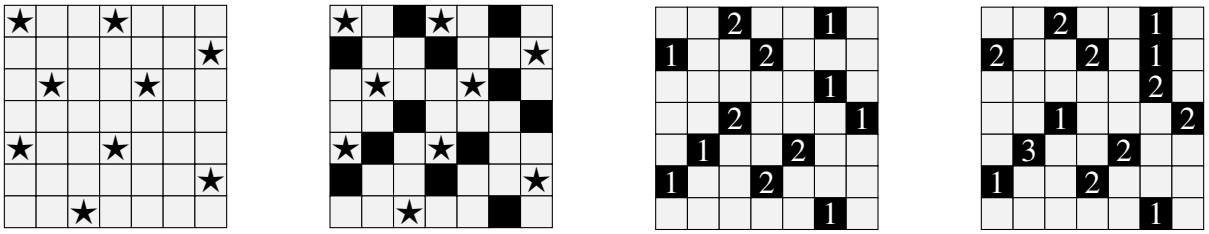
2.5.5. CP-Solver zur Validierung und Eindeutigkeitsprüfung

Constraint Programming (CP) bezeichnet einen Ansatz zur Lösung kombinatorischer Probleme, bei dem Variablen, deren Wertebereiche und Nebenbedingungen modelliert werden [6, vgl. Kap. 1–2].
Neben dem deduktiven Solver wird in dieser Arbeit ein Constraint-Programming-Solver (CP-Solver) als Referenzverfahren verwendet. Seine Aufgabe besteht nicht in der Bestimmung des Deductive Score, sondern in der formalen Validierung erzeugter Instanzen. Insbesondere wird der CP-Solver dazu eingesetzt, die Lösbarkeit und Eindeutigkeit von Puzzlekandidaten im Generierungsprozess zu prüfen.
Der CP-Solver kommt an mehreren Stellen zum Einsatz. Nach dem Erzeugen einer Kandidateninstanz wird geprüft, ob diese eindeutig lösbar ist. Derselbe Test wird bei der iterativen Ergänzung zusätzlicher Hinweisfelder sowie bei der optionalen Minimierung von Hinweisfeldern erneut durchgeführt. Auf diese Weise dient der CP-Solver als Referenzverfahren, um sicherzustellen, dass nur gültige und eindeutig lösbare Instanzen in den finalen Datensatz aufgenommen werden.
Methodisch erfüllt der CP-Solver damit eine andere Rolle als der deduktive Solver: Während der deduktive Solver ein begrenztes regelbasiertes Lösungsverhalten modelliert und als Grundlage des Schwierigkeitsmaßes dient, übernimmt der CP-Solver die Aufgabe eines vollständigeren Prüfverfahrens für Korrektheit und Eindeutigkeit. Die Implementierung basiert auf dem CP-SAT-Solver aus OR-Tools [7].

2.6. Instanzgenerierung

Die Generierung neuer Buraitoraito-Instanzen erfolgt mit einem iterativen Generate-and-Test-Verfahren. Dieser Ansatz wird häufig zur automatischen Generierung logischer Puzzle verwendet: Dabei werden zunächst Kandidateninstanzen erzeugt und anschließend mit Hilfe eines Solvers auf Lösbarkeit und weitere Eigenschaften überprüft. Instanzen, die die Anforderungen nicht erfüllen, werden verworfen und durch neue Kandidaten ersetzt [8].
In der vorliegenden Arbeit wird dieses Verfahren in einer Top-Down-Variante umgesetzt. Dabei wird nicht unmittelbar ein vollständiges Puzzle konstruiert, sondern zunächst eine gültige Sternbelegung erzeugt. Aus dieser Lösung wird anschließend ein Puzzlegerüst abgeleitet, indem schwarze Felder so platziert werden, dass die Sichtlinienbeziehungen der Sterne mit der gewählten Puzzlestruktur vereinbar sind. Danach werden den schwarzen Feldern Hinweiszahlen zugewiesen. Die so entstandene Instanz wird anschließend hinsichtlich struktureller Gültigkeit, eindeutiger Lösbarkeit sowie ihrer de-

duktiven Schwierigkeit überprüft. Ist ein Kandidat noch nicht eindeutig lösbar, so können zusätzliche Hinweisfelder ergänzt werden. Kandidaten, die die Anforderungen nicht erfüllen, werden verworfen und durch neue Kandidaten ersetzt.
Algorithmus 1 beschreibt den detaillierten Ablauf des Generate-and-Test-Verfahrens, während Abbildung 2.5 den Generierungsprozess in Form eines Ablaufdiagramms darstellt. Ergänzend zeigt Abbildung 2.4 die einzelnen Stadien der Generierung anhand eines konkreten Beispielpuzzles.
Die folgende Beschreibung erläutert die einzelnen Schritte der Implementierung im Detail.



(a) Erzeugte Sternbelegung (b) Abgeleitetes Puzzlegerüst (c) Initiale Hinweisfelder (d) Finale Instanz

Abbildung 2.4.: Stadien der Instanzgenerierung eines Beispielpuzzles.

Teilabbildung (a) zeigt die zunächst erzeugte gültige Sternbelegung, die als Ausgangslösung des Generierungsverfahrens dient. In Teilabbildung (b) wird daraus durch das Einfügen schwarzer Felder ein Puzzlegerüst abgeleitet, das die Sichtlinienbeziehungen der Sterne mit der gewählten Puzzlestruktur kompatibel macht. Teilabbildung (c) zeigt die daraus entstehende erste Puzzleinstanz nach dem Setzen initialer Hinweisfelder. Teilabbildung (d) stellt die finale Instanz dar, nachdem zusätzliche Hinweisfelder ergänzt wurden, um eindeutige Lösbarkeit sicherzustellen.

Algorithm 1 Generate-and-Test-Verfahren zur Instanzgenerierung

Require: Zielschwierigkeit, Generatorparameter und Ergänzungslimit `max_additions`
Ensure: Eine gültige, eindeutig lösbare Instanz innerhalb des Zielbereichs der Schwierigkeit

repeat
    Erzeuge eine gültige Sternbelegung
    Platziere schwarze Felder so, dass die Sichtlinienkonflikte zwischen den Sternen unterbrochen werden, und erhalte dadurch ein Puzzlegerüst
    Weise den schwarzen Feldern Hinweiszahlen zu
    Prüfe die strukturelle Gültigkeit der Instanz
    if Instanz strukturell gültig **then**
        Prüfe, ob die Instanz eindeutig lösbar ist
        $a \leftarrow 0$
        while Instanz nicht eindeutig lösbar **and** $a <$ `max_additions` **do**
            Ergänze ein zusätzliches Hinweisfeld
            Aktualisiere die Hinweiszahlen
            Prüfe erneut, ob die Instanz eindeutig lösbar ist
            $a \leftarrow a + 1$
        end while
        if Instanz eindeutig lösbar **then**
            Bestimme die deduktive Schwierigkeit der Instanz
        end if
    end if
until Instanz gültig, eindeutig lösbar und innerhalb des Zielbereichs der Schwierigkeit
return Instanz

Die Ergänzung von Hinweisfeldern erfolgt iterativ. Nach jeder Ergänzung wird erneut geprüft, ob die Instanz eindeutig lösbar ist. Dieser Prozess endet entweder mit einer eindeutig lösbaren Instanz oder mit dem Verwerfen des Kandidaten, falls das vorgegebene Ergänzungslimit erreicht wird.

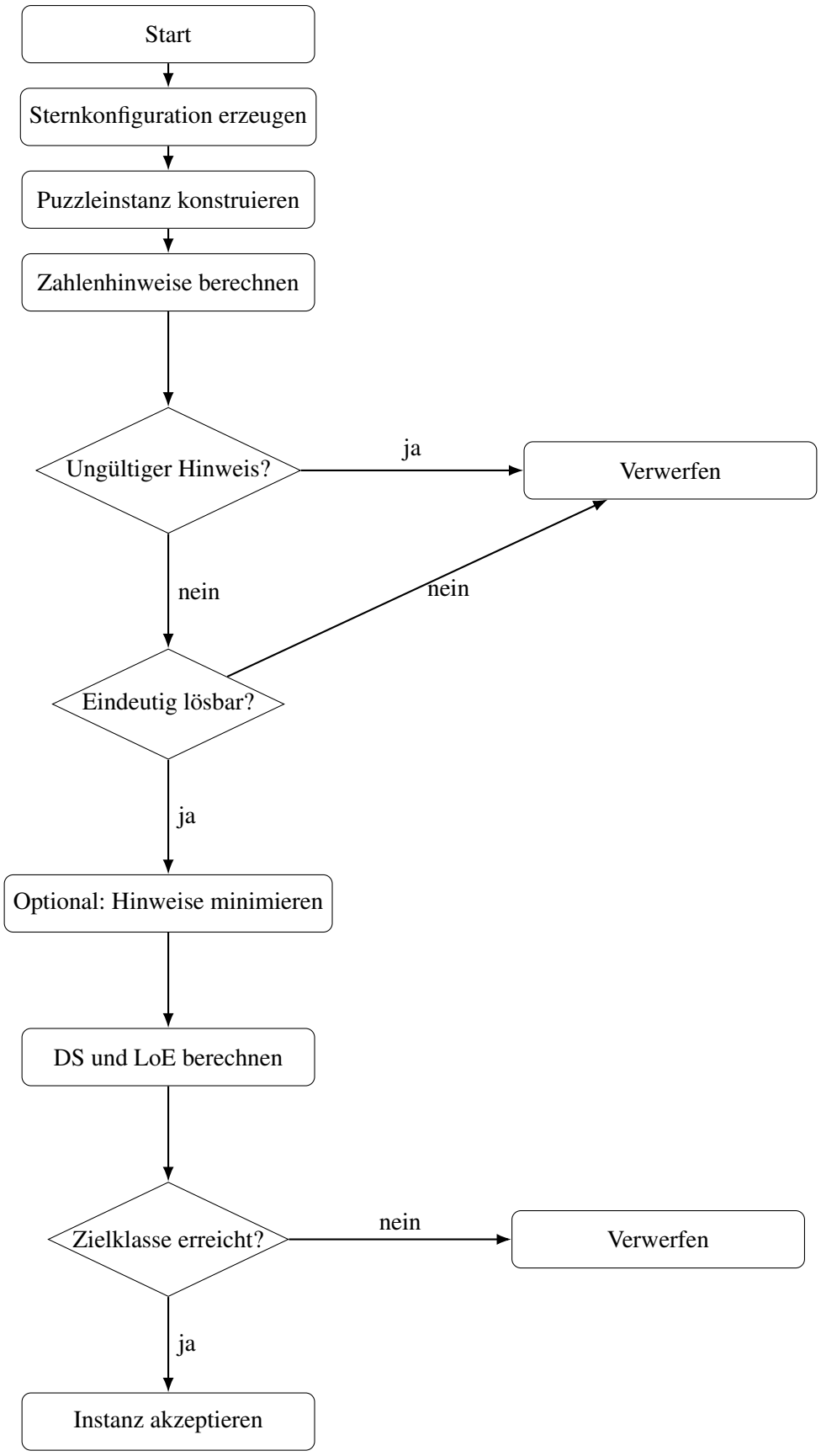


Abbildung 2.5.: Ablauf der Top-Down-Instanzgenerierung im Generate-and-Test-Verfahren.

2.6.1. Implementierungsdetails des Generierungsprozesses

Die in Abbildung 2.5 dargestellten Schritte entsprechen direkt der Implementierung des Generators. Zu Beginn wird eine vollständige gültige Sternkonfiguration erzeugt, die als Lösung dient. Aus dieser wird durch Einfügen von Blockern zwischen sichtbaren Sternen sowie durch Berechnung der Zahlenhinweise eine Puzzleinstanz konstruiert.

Anschließend wird die Instanz auf formale Gültigkeit und eindeutige Lösbarkeit geprüft. Ungültige Instanzen werden verworfen. Ist die Lösung nicht eindeutig, werden iterativ zusätzliche Hinweisfelder ergänzt. Falls keine eindeutige Lösung erreicht wird, wird die Instanz verworfen.

Optional werden Hinweisfelder minimiert, wobei nach jeder Änderung erneut Validität und Eindeutigkeit geprüft werden. Im Anschluss erfolgt die Bewertung mittels deduktivem Solver, der Deductive Score (DS), LoE-Stufen sowie die Schrittverteilung bestimmt.

Die Klassifikation erfolgt anhand der verwendeten Deduktionsstufen (vgl. Tabelle 2.1). Leichte Instanzen benötigen ausschließlich LoE0, während höhere Schwierigkeitsgrade den Einsatz von LoE1 bzw. LoE2 erfordern. Sehr schwere Instanzen zeichnen sich dadurch aus, dass LoE2 benötigt wird, jedoch keine LoE1-Schritte auftreten.

Der Prozess wird iterativ wiederholt, bis die gewünschte Anzahl an Instanzen der jeweiligen Zielklasse erzeugt wurde.

Tabelle 2.1.: Klassifikation der Instanzen nach Level-of-Embedding (LoE)

Schwierigkeit	**LoE0**	**LoE1**	**LoE2**
Leicht	✓	×	×
Normal	✓	✓	×
Schwer	✓	✓	✓
Sehr schwer	✓	×	✓

2.6.2. Parameter der Instanzgenerierung

Obwohl der Deductive Score in Abschnitt 2.5 als differenziertes Maß der deduktiven Schwierigkeit eingeführt wurde, wird er in der Instanzgenerierung nicht direkt zur Einteilung in Zielschwierigkeitsstufen verwendet. Stattdessen erfolgt die Generierungssteuerung über diskrete Zielklassen, die sich an den vom deduktiven Solver tatsächlich benötigten LoE-Stufen orientieren. Der Deductive Score wird im weiteren Verlauf vor allem zur feineren Analyse und zum Vergleich von Instanzen innerhalb und zwischen diesen Schwierigkeitsklassen herangezogen.
Die Instanzgenerierung wird durch mehrere Parameter gesteuert, die den Aufbau der Puzzleinstanzen sowie den Suchprozess beeinflussen.
Die Parameter bestimmen insbesondere die Größe der generierten Instanzen, die Anzahl der Generierungsversuche sowie die Art und Intensität der Nachbearbeitungsschritte wie das Hinzufügen zusätzlicher Hinweise und deren Minimierung.

Zur Reproduzierbarkeit der Generierung werden die verwendeten Parameter im Folgenden explizit angegeben.

Parameter	**Wert**	**Beschreibung**
W, H	7×7	Größe des Gitters
`max_solutions`	20000	Maximale Anzahl von Generierungsversuchen
`max_additions`	60	Maximale Anzahl zusätzlicher Ergänzungsschritte zur Herstellung eindeutiger Lösbarkeit
`minimize`	aktiv	Aktiviert die Minimierung der Hinweise
`minimize_trials`	500	Anzahl der Versuche zur Hinweisreduktion
`relax_schwer`	deaktiviert	Strikte Klassifikation der Schwierigkeit

Tabelle 2.2.: Verwendete Parameter der Instanzgenerierung

Der Parameter `max_additions` gibt die maximale Anzahl zusätzlicher Ergänzungsschritte bei der Herstellung eindeutiger Lösbarkeit an. Der Parameter `minimize_trials` legt fest, wie viele Versuche zur Reduktion der Hinweisfelder während der optionalen Minimierung durchgeführt werden. Die übrigen Parameter steuern die Gittergröße, die maximale Anzahl von Generierungsversuchen sowie die Verwendung einer strikten Schwierigkeitsklassifikation.
Die Parameter wurden für alle Generierungsexperimente konstant gehalten, um eine vergleichbare Datengrundlage zwischen den Schwierigkeitsklassen zu gewährleisten.

3. Ergebnisse

3.1. Überblick über die ausgewerteten Daten

Die Ergebnisse dieser Arbeit basieren auf mehreren unterschiedlichen Datenquellen von Buraitoraito-Instanzen (vgl. Anhang A).
Zum einen wurde ein automatisch generierter Instanzkorpus verwendet, der im Rahmen dieser Arbeit mithilfe des in Abschnitt 2.6 vorgestellten GaT-Verfahrens erzeugt wurde. Diese Instanzen werden mittels deduktiver Solveranalyse bewertet und anhand der auftretenden LoE-Signaturen in die Schwierigkeitsklassen *leicht*, *normal*, *schwer* und *sehr_schwer* eingeteilt.
Zum anderen wurden externe Puzzlequellen analysiert. Dazu gehören insbesondere Instanzen der Plattform GrandGames[1], für die Schwierigkeitsstufen (1–4) vorliegen, sowie Instanzen der Plattform Janko[2], für die teilweise explizite Schwierigkeitsangaben gegeben sind.
Zusätzlich wurden die Datensätze *kidlogic100*[3] sowie *360 Nights of Buraitoraito*[4] berücksichtigt, die als Buchsammlungen dienen. Eine Übersicht über die verwendeten Datenquellen ist in Tabelle 3.1 dargestellt.
Für alle Instanzen wurden der Deductive Score (DS) sowie das Level of Embedding (LoE) mithilfe des implementierten deduktionsgeleiteten Solvers berechnet.

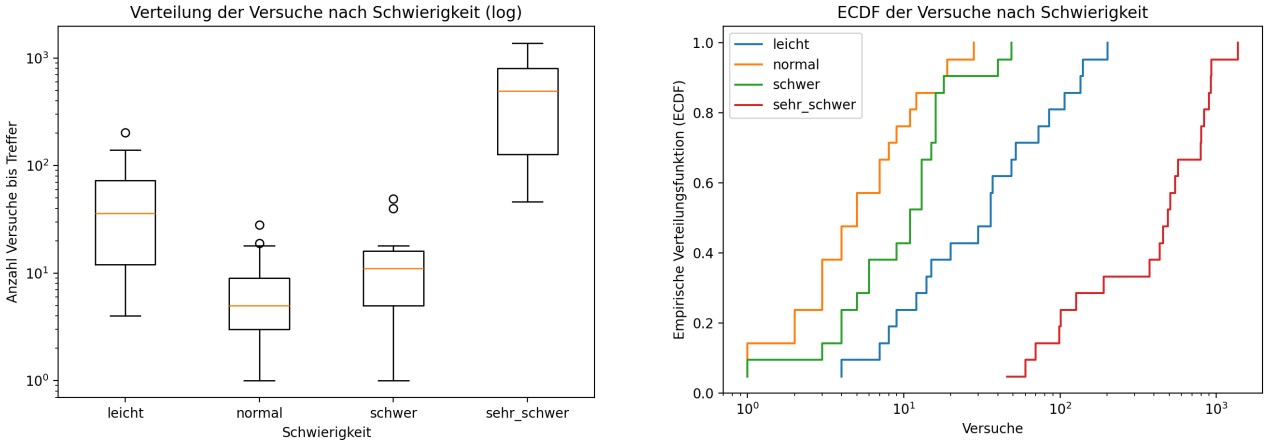
Quelle	Anzahl	Schwierigkeit gegeben	Quellentyp
Generator (eigene Arbeit)	80	ja	generiert
GrandGames	287	ja	Online-Plattform
Janko	20	ja	Online-Plattform
Janko5	5	nein	Online-Plattform
kidlogic100	100	nein	Buchdatensatz
360 Nights of Buraitoraito	360	nein	Buchdatensatz

Tabelle 3.1.: Übersicht der verwendeten Instanzquellen (vgl. Anhang A).

[1] https://de.grandgames.net/starlight, Zugriff am 03.03.2026.
[2] https://www.janko.at/Raetsel/Buraitoraito/index.htm, Zugriff am 27.02.2026.
[3] *Kid Logic Puzzles: Buraitoraito Teaser*, Amazon-Produktseite, https://www.amazon.de/Kid-Logic-Puzzles-Buraitoraito-Teaser/dp/1792971702, Zugriff am 18.03.2026.
[4] *360 Nights of Buraitoraito*, Amazon-Produktseite, https://www.amazon.de/dp/B0CWDXKWND, Zugriff am 18.03.2026.

3.2. Generierbarkeit nach LoE-Schwierigkeitsklasse

Im Rahmen der Instanzgenerierung wird für jede Ziel-Schwierigkeitsklasse wiederholt versucht, gültige Puzzleinstanzen zu erzeugen. Die Anzahl der notwendigen Versuche bis zum ersten erfolgreichen Treffer variiert dabei deutlich zwischen den Zielklassen.



(a) Boxplots der Versuchszahlen pro Zielklasse auf logarithmischer y-Achse. Kreise kennzeichnen Ausreißer.

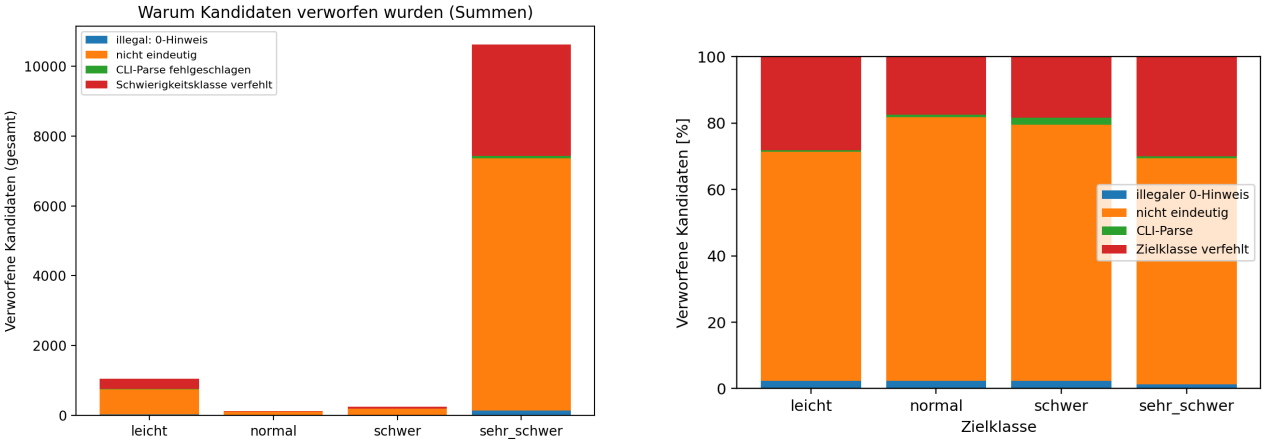
(b) Empirische Verteilungsfunktionen (ECDF) der Versuchszahlen auf logarithmischer x-Achse.

Abbildung 3.1.: Anzahl der Generierungsversuche bis zur erfolgreichen Instanz in Abhängigkeit von der Ziel-Schwierigkeitsklasse.

Teilabbildung (a) zeigt die Verteilung der Generierungsversuche pro Zielklasse in Form von Boxplots, die zusätzlich eingezeichneten Kreise markieren Ausreißer. Dabei handelt es sich um einzelne Instanzen, deren Anzahl an Generierungsversuchen deutlich außerhalb des für die jeweilige Zielklasse typischen Wertebereichs liegt.
Teilabbildung (b) zeigt die entsprechenden empirischen Verteilungsfunktionen (ECDF). Insgesamt zeigt sich eine Verteilung mit Maxima in den Randbereichen des Schwierigkeitsspektrums: Sehr leichte und sehr schwere Instanzen erfordern im Vergleich zu den mittleren Zielklassen mehr Generierungsversuche.
Die große Streuung und das Auftreten von Ausreißern verdeutlichen, dass geeignete Kandidaten je nach Zielklasse mit unterschiedlicher Wahrscheinlichkeit entstehen. Insbesondere an den beiden Rändern des Schwierigkeitsspektrums werden die geforderten Eigenschaften seltener erreicht.

3.3. Verwerfungsgründe im Generierungsprozess

Im Verlauf der Instanzgenerierung werden zahlreiche erzeugte Kandidaten verworfen, da sie die geforderten Eigenschaften nicht erfüllen. Die Verwerfungsgründe liefern zusätzliche Hinweise darauf, an welchen Stellen der Generate-and-Test-Prozess bei den verschiedenen angestrebten Schwierigkeitsklassen besonders häufig scheitert.



(a) Absolute Häufigkeiten.

(b) Normierte Anteile in %.

Abbildung 3.2.: Verwerfungsgründe im Generierungsprozess nach Ziel-Schwierigkeitsklasse.

Teilabbildung (a) zeigt die absoluten Häufigkeiten der einzelnen Verwerfungsgründe, Teilabbildung (b) deren normierte Anteile relativ zur Gesamtzahl der verworfenen Kandidaten innerhalb der jeweiligen Zielklasse. Die normierte Darstellung erleichtert den Vergleich zwischen den Zielklassen, da Unterschiede in der absoluten Anzahl verworfener Kandidaten ausgeblendet werden.
Der Verwerfungsgrund *CLI-Parse* bezeichnet Kandidaten, die beim Einlesen oder bei der Weiterverarbeitung des erzeugten Puzzles im verwendeten Austauschformat nicht erfolgreich verarbeitet werden konnten. Der Grund *illegaler 0-Hinweis* bezeichnet Kandidaten, bei denen beim Setzen der Hinweiszahlen ein schwarzes Feld den Wert 0 erhalten würde. Solche Fälle könnten zwar bereits unmittelbar bei der Hinweisvergabe ausgeschlossen werden. In der vorliegenden Implementierung erfolgt die Prüfung jedoch nachgelagert, da dies den Generierungsprozess vereinfacht: Zunächst wird ein vollständiger Kandidat erzeugt, anschließend werden formale Ungültigkeiten in einem separaten Validierungsschritt erkannt.
Die Gegenüberstellung beider Darstellungen zeigt, dass sich nicht nur die absolute Zahl verworfener Kandidaten zwischen den Zielklassen unterscheidet, sondern auch die relative Bedeutung einzelner Verwerfungsgründe. Auf diese Weise wird deutlicher, welche Arten von Fehlschlägen für bestimmte Ziel-Schwierigkeitsklassen besonders charakteristisch sind.

3.4. Verteilung der Schwierigkeitsklassen bei unfiltrierter Generierung

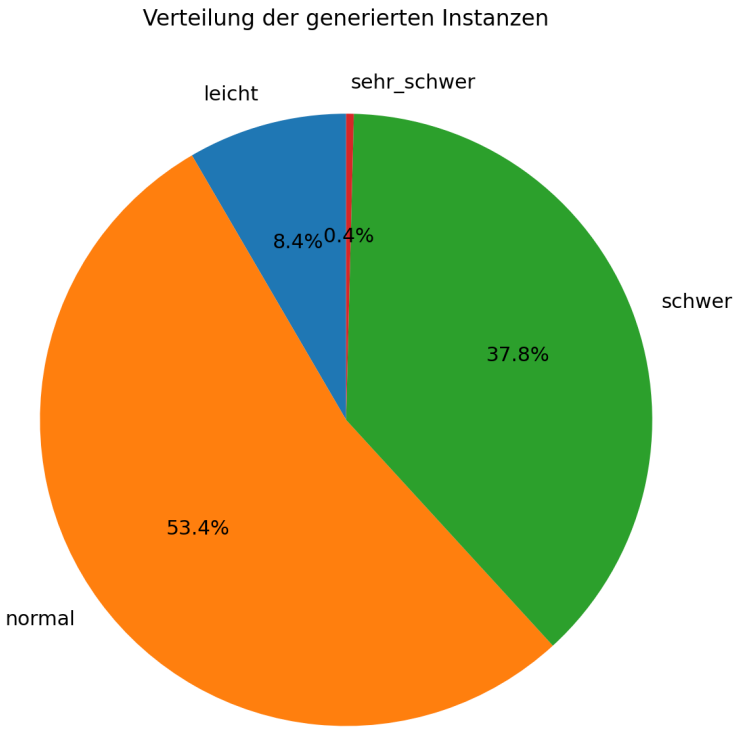


Abbildung 3.3.: Verteilung der Schwierigkeitsklassen bei unfiltrierter Generierung.

Abbildung 3.3 basiert auf einem Datensatz von 500 generierten Buraitoraito-Instanzen der Größe 7×7. Die Instanzen wurden mit dem in Abschnitt 2.6 beschriebenen Generate-and-Test-Verfahren erzeugt. Die verwendeten Generierungsparameter entsprechen den in Tabelle 2.2 angegebenen Werten. Der Datensatz ist dabei *unfiltriert*, das heißt, während der Generierung wurde keine Auswahl nach einer vorgegebenen Schwierigkeitsklasse vorgenommen. Die Zuordnung zu den Klassen *leicht*, *normal*, *schwer* und *sehr schwer* erfolgt erst nachträglich auf Grundlage der beim Lösen beobachteten LoE-Signatur.

Tabelle 3.2.: Zuordnung der beobachteten LoE-Signaturen zu den Schwierigkeitsklassen.

Schwierigkeitsklasse	**LoE1**	**LoE2**
leicht	keine Schritte	keine Schritte
normal	mindestens ein Schritt	keine Schritte
schwer	mindestens ein Schritt	mindestens ein Schritt
sehr schwer	keine Schritte	mindestens ein Schritt

3.5. Deduktiver Schwierigkeitswert und externe Schwierigkeitsstufen

In diesem Abschnitt wird untersucht, inwieweit die vom deduktiven Solver bestimmten Schwierigkeitsmaße mit externen Schwierigkeitsangaben übereinstimmen. Damit beginnt ein neuer Auswertungsaspekt. Während die vorangehenden Abschnitte die Generierung und interne Charakterisierung der Instanzen behandeln, steht hier der Vergleich mit extern vergebenen Schwierigkeitsstufen im Vordergrund.
Dabei ist zu beachten, dass der deduktive Solver bewusst auf die in Abschnitt 2.5 beschriebenen regelbasierten Deduktionsmechanismen beschränkt ist und daher nicht notwendigerweise jede Instanz vollständig löst. Für den überwiegenden Teil der betrachteten Instanzen konnte der Solver dennoch eine vollständige Lösung herleiten. Nicht vollständig rein deduktiv lösbar waren lediglich 7 von 287 GrandGames-Instanzen, 4 von 100 Instanzen aus *Kidlogic100* sowie 4 von 360 Instanzen aus *360 Nights of Buraitoraito*. Alle übrigen betrachteten Instanzen wurden vollständig gelöst.

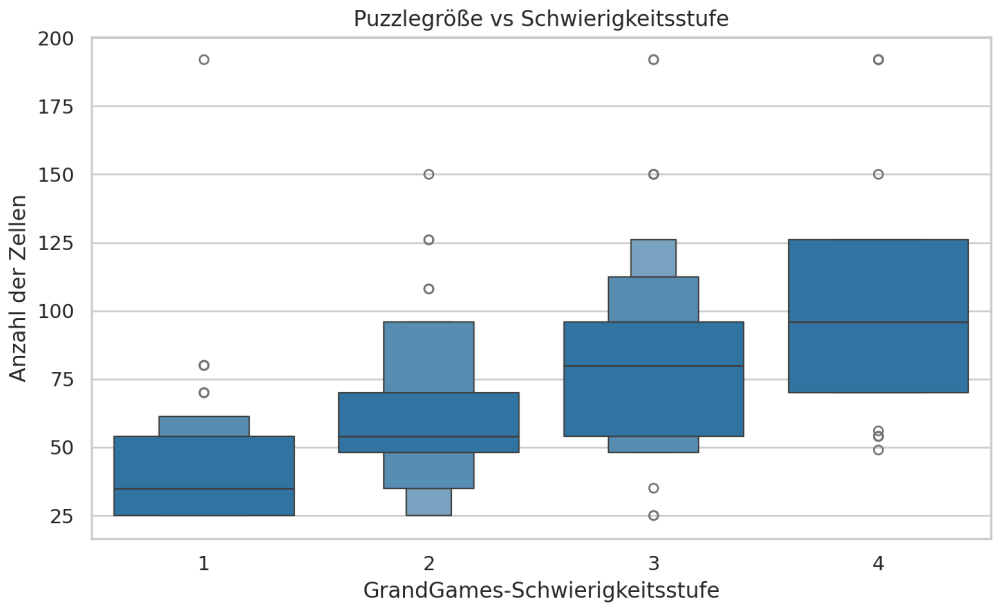
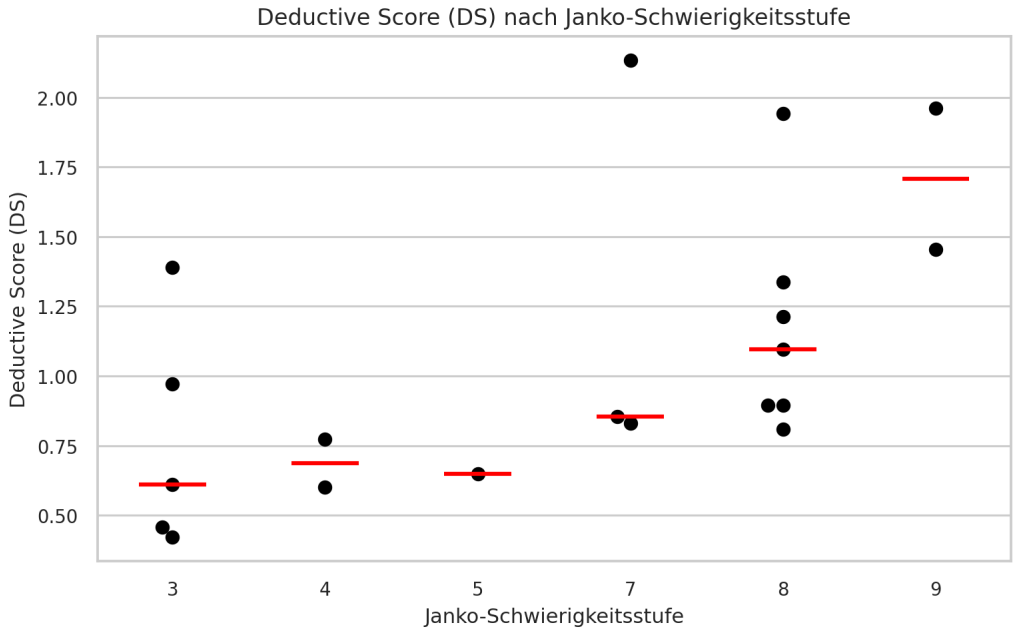
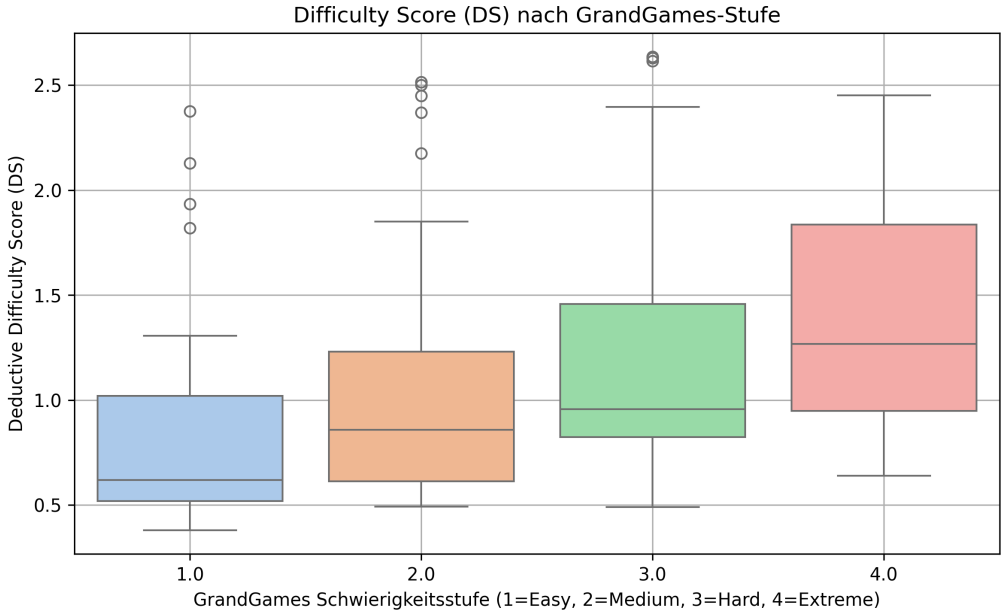


Abbildung 3.4.: Puzzlegröße, gemessen als Anzahl der Zellen, in Abhängigkeit von der externen Schwierigkeitsstufe. Dargestellt sind die Verteilungen der Puzzlegrößen innerhalb der jeweiligen Schwierigkeitsklassen.



(a) Janko.



(b) GrandGames.

Abbildung 3.5.: Deductive Score in Abhängigkeit von externen Schwierigkeitsstufen für die Datensätze Janko und GrandGames. Für Janko sind die einzelnen Beobachtungen dargestellt. Die roten Querstriche markieren den Median der jeweiligen Kategorie.

Abbildung 3.4 zeigt die Verteilungen der Puzzlegrößen in den jeweiligen externen Schwierigkeitsstufen. Zwischen den Klassen bestehen Überlappungen.
Teilabbildung 3.5a zeigt die einzelnen DS-Werte für die auf Janko vergebenen Schwierigkeitsstufen.

Die roten Querstriche markieren den Median der jeweiligen Kategorie. Teilabbildung 3.5b zeigt die Verteilung der DS-Werte für die GrandGames-Schwierigkeitsstufen in Form von Boxplots.
In beiden Teilabbildungen sind Unterschiede zwischen den Schwierigkeitsstufen sowie Streuungen innerhalb der einzelnen Kategorien sichtbar.

3.6. Puzzlegröße, DS und GrandGames-Stufen

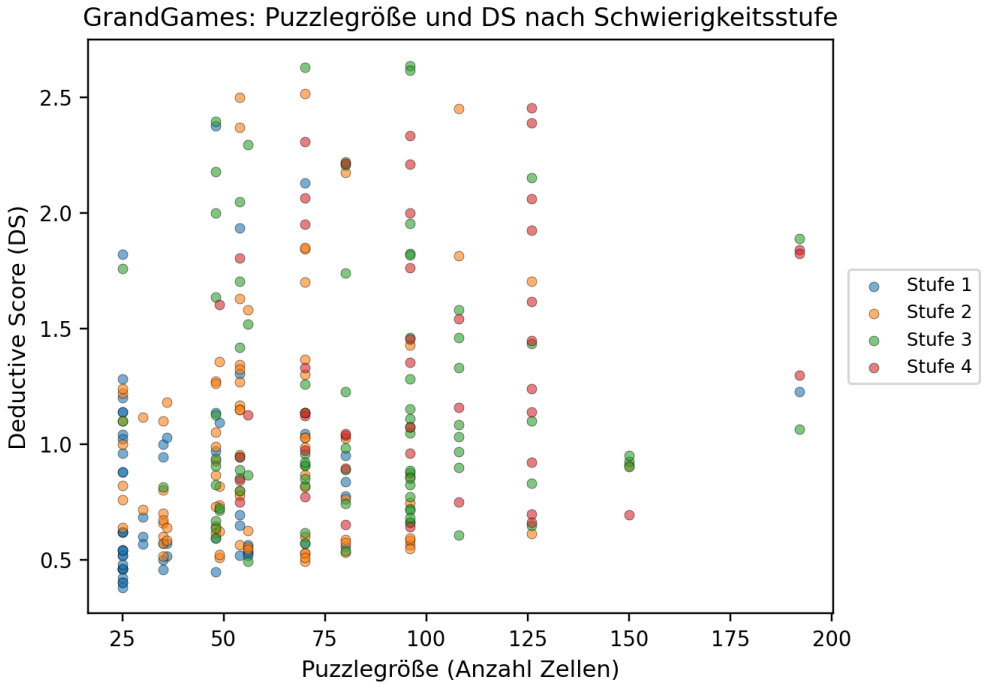


Abbildung 3.6.: Zusammenhang zwischen Puzzlegröße und Deductive Score für die GrandGames-Instanzen. Die Punkte sind nach der externen Schwierigkeitsstufe eingefärbt.

Abbildung 3.6 zeigt den Zusammenhang zwischen Puzzlegröße und Deductive Score für die GrandGames-Instanzen. Für viele Puzzlegrößen sind mehrere Schwierigkeitsstufen vertreten. Zudem liegen die beobachteten DS-Werte bei größeren Puzzleinstanzen über einen breiteren Wertebereich verteilt.

3.7. LoE-Kategorien und GrandGames-Stufen

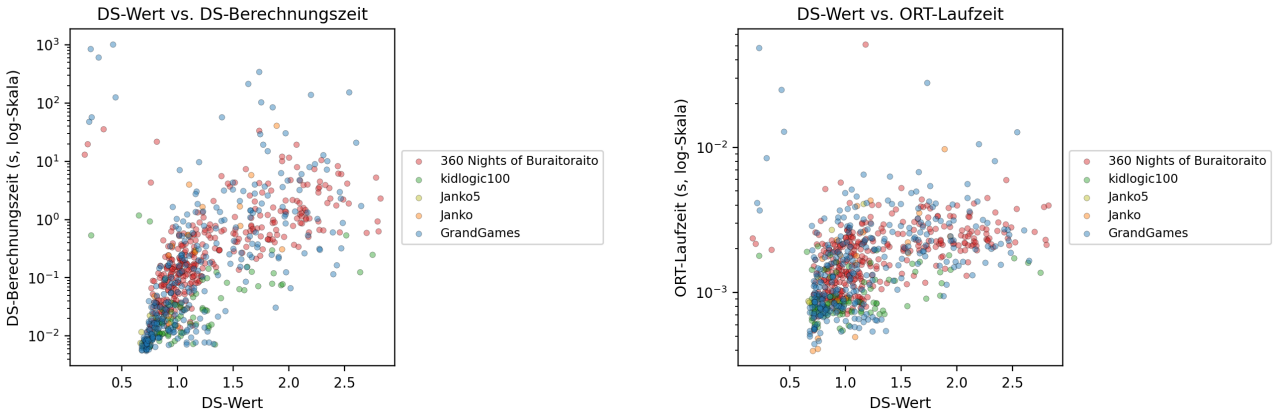
Tabelle 3.3 zeigt für jede auf GrandGames vergebene Schwierigkeitsstufe, welche maximale LoE-Stufe zur Lösung der jeweiligen Instanzen durch den deduktiven Solver erforderlich war.
Mit steigender GrandGames-Stufe verschiebt sich die Verteilung der Instanzen zu höheren LoE-Kategorien. Während in den niedrigeren Stufen noch ein nennenswerter Anteil von Instanzen mit maximal LoE 0 auftritt, nimmt dieser Anteil in den höheren Stufen deutlich ab und verschwindet in Stufe 4 vollständig. Gleichzeitig steigt der Anteil von Instanzen, deren Lösung maximal LoE 2 erfordert, kontinuierlich an. LoE 1 bleibt in allen Stufen vertreten, ist jedoch vor allem in den mittleren Stufen die häufigste Kategorie.

Tabelle 3.3.: Verteilung der maximal zur Lösung benötigten LoE-Kategorien innerhalb der GrandGames-Schwierigkeitsstufen.

Stufe	LoE-0	LoE-1	LoE-2
1	21 (34.4%)	31 (50.8%)	9 (14.8%)
2	7 (8.0%)	54 (62.1%)	26 (29.9%)
3	2 (2.4%)	45 (52.9%)	38 (44.7%)
4	0 (0.0%)	14 (30.4%)	32 (69.6%)

3.8. Zusammenhang zwischen DS und Laufzeiten

Die im Folgenden dargestellten Laufzeitmessungen wurden auf einem Rechner mit einem *AMD Ryzen 5 7500F 6-Core Processor* durchgeführt. Die gemessenen Zeiten sind daher von der verwendeten Hardware abhängig und vor allem als Vergleich innerhalb der vorliegenden Auswertung zu verstehen.



(a) DS-Berechnungszeit.

(b) OR-Tools-Laufzeit.

Abbildung 3.7.: Zusammenhang zwischen Deductive Score (DS) und Laufzeiten für die in Tabelle 3.1 beschriebenen externen Datensätze. Die Datensätze werden in beiden Teilabbildungen durch konsistente Farben unterschieden.

Teilabbildung 3.7a zeigt den Zusammenhang zwischen Deductive Score und der Berechnungszeit des deduktiven Solvers, Teilabbildung 3.7b den Zusammenhang zwischen Deductive Score und der Laufzeit des OR-Tools-Solvers. In beide Darstellungen fließen mehrere Datensätze ein, die zur besseren Unterscheidbarkeit farblich kodiert sind.
In Teilabbildung 3.7a liegen höhere DS-Werte überwiegend bei höheren Laufzeiten des deduktiven Solvers. Zudem ist bei größeren DS-Werten eine breitere Streuung der Laufzeiten sichtbar. Teilabbildung 3.7b zeigt ebenfalls höhere Laufzeiten bei größeren DS-Werten, jedoch in einem insgesamt engeren Wertebereich als Teilabbildung 3.7a.

3.9. Verteilung der DS-Werte

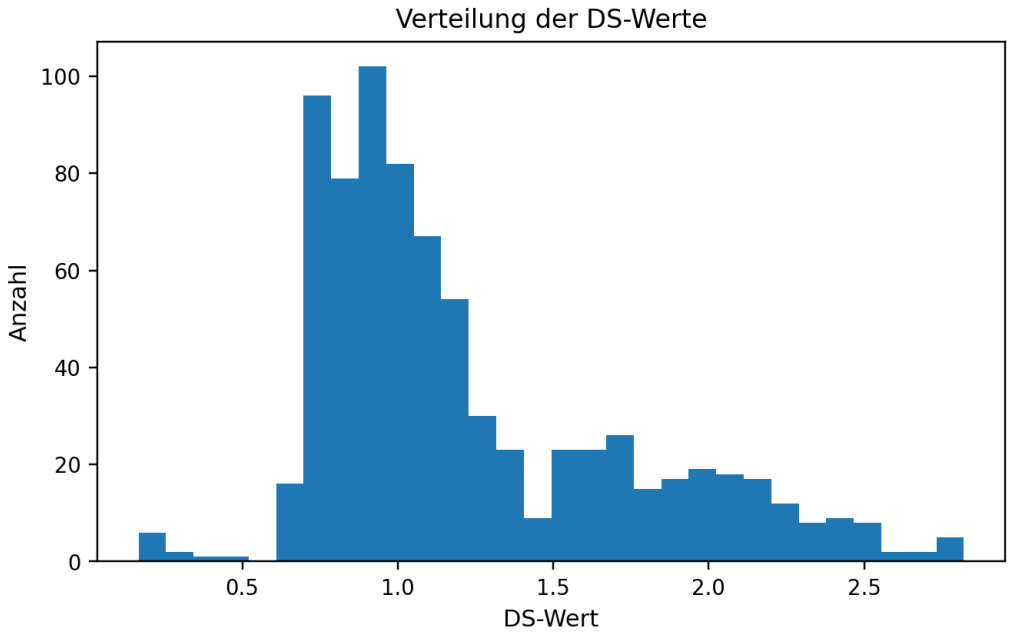


Abbildung 3.8.: Verteilung der Deductive-Score-Werte über die in Tabelle 3.1 beschriebenen externen Datensätze.

Abbildung 3.8 zeigt eine rechtsschiefe Verteilung der DS-Werte. Ein großer Teil der Instanzen liegt im Bereich ungefähr zwischen 0,7 und 1,3. Höhere DS-Werte treten seltener auf und bilden einen längeren rechten Verteilungsschwanz.

4. Diskussion

4.1. Interpretation der Ergebnisse

Die Ergebnisse zeigen, dass der Deductive Score (DS) eine geeignete Kennzahl zur quantitativen Beschreibung der Schwierigkeit von Buraitoraito-Instanzen darstellt.
Sowohl für die GrandGames-Daten (Abbildung 3.5b) als auch für die Janko-Instanzen (Abbildung 3.5a) zeigt sich ein positiver Zusammenhang zwischen externer Schwierigkeit und DS-Wert. Mit steigender Schwierigkeit nimmt der Median des DS-Werts zu.
Bei der Einordnung dieser Ergebnisse ist zu beachten, dass für Buraitoraito kein Datensatz mit klar und einheitlich vergebenen Schwierigkeitsstufen vorliegt. Die Angaben von GrandGames und Janko sind deshalb kein fester Maßstab, sondern eher praktische Vergleichswerte. Umso bemerkenswerter ist es, dass insbesondere die GrandGames-Daten auf eine starke Korrelation zwischen den externen Schwierigkeitsstufen und dem Deductive Score hinweisen, wie in Abbildung 3.5b zu sehen ist. Für Janko zeigt sich ebenfalls eine positive Tendenz, die aufgrund der kleineren Datengrundlage jedoch vorsichtiger zu bewerten ist.
Die Überlappung der Verteilungen deutet darauf hin, dass externe Bewertungen nicht ausschließlich auf deduktiver Komplexität beruhen. Neben dem vom Solver erfassten Deduktionsaufwand dürften auch weitere Merkmale wie Puzzlegröße oder strukturelle Eigenschaften der Instanzen eine Rolle spielen.
Die Analyse der LoE-Kategorien bestätigt diese Beobachtung. Höhere Schwierigkeitsstufen sind mit komplexeren Deduktionsprozessen verbunden, insbesondere durch das vermehrte Auftreten von LoE-2-Schritten. Damit spricht die Auswertung dafür, dass der DS nicht nur die Anzahl von Lösungsschritten, sondern auch die Tiefe der erforderlichen Deduktion in sinnvoller Weise abbildet.

4.2. Einfluss der Puzzlegröße

Die Ergebnisse aus Abbildung 3.4 und Abbildung 3.6 zeigen, dass die Puzzlegröße mit der Schwierigkeit zusammenhängt, diese jedoch nicht allein erklärt. Größere Puzzle treten häufiger in höheren GrandGames-Stufen auf und weisen tendenziell auch höhere DS-Werte auf. Gleichzeitig zeigt die Darstellung, dass für viele Puzzlegrößen mehrere Schwierigkeitsstufen auftreten.
Die Puzzlegröße ist damit ein relevanter, aber kein hinreichender Erklärungsfaktor. Die Schwierigkeit wird nicht allein durch die Anzahl der Zellen bestimmt, sondern wesentlich auch durch die Struktur der Sichtbeziehungen, die Platzierung der Hinweiszellen und die daraus resultierenden Deduktionsmöglichkeiten.

4.3. Generierbarkeit der Schwierigkeitsklassen

Die Generierungsergebnisse zeigen, dass sich die Schwierigkeitsklassen nicht mit gleicher Leichtigkeit erzeugen lassen. Insbesondere die Auswertungen zu den benötigten Versuchen und den Verwerfungsgründen machen deutlich, dass die gezielte Erzeugung einer bestimmten deduktiven Schwierigkeit eine zentrale Herausforderung des Generierungsprozesses ist.
Die hohe Zahl an verworfenen Kandidaten aufgrund verfehlter Zielschwierigkeit zeigt, dass gültige und eindeutig lösbare Instanzen vergleichsweise häufig erzeugt werden können, die gewünschte deduktive Struktur jedoch deutlich seltener entsteht. Dies spricht dafür, dass Schwierigkeitseigenschaften nicht direkt aus der bloßen Erzeugung einer gültigen Lösung folgen, sondern das Ergebnis spezifischer struktureller Konstellationen sind.
Die ergänzende Auswertung der unfiltrierten Generierung zeigt zudem, dass die erzeugten Instanzen nicht gleichmäßig über die Schwierigkeitsklassen verteilt sind. Der Generator besitzt also eine erkennbare Tendenz, bestimmte deduktive Profile häufiger hervorzubringen als andere. Dies ist für die zukünftige Weiterentwicklung des Verfahrens relevant, wenn bestimmte Zielklassen gezielter oder effizienter erreicht werden sollen.

4.4. Laufzeitverhalten der Solver

Die Analyse der Laufzeiten zeigt deutliche Unterschiede zwischen dem deduktiven Solver und dem OR-Tools-Solver. Während die Laufzeit des deduktiven Solvers mit steigendem DS-Wert sichtbar zunimmt, bleibt die Laufzeit des OR-Tools-Solvers vergleichsweise stabil (Abbildung 3.7).
Dies ist inhaltlich plausibel, da der deduktive Solver gerade jene regelbasierten Schlussprozesse ausführt, die dem Schwierigkeitsmaß zugrunde liegen. Der OR-Tools-Solver dient dagegen primär als allgemeines Referenzverfahren zur Prüfung von Lösbarkeit und Eindeutigkeit und ist nicht darauf ausgelegt, menschlich nachvollziehbare Deduktionsschritte abzubilden. Die Laufzeitanalyse stützt damit die Interpretation, dass der DS vor allem die Schwierigkeit der deduktiven Auflösung misst und nicht bloß allgemeine algorithmische Lösungszeit.
Bei der Einordnung dieses Unterschieds ist außerdem zu beachten, dass OR-Tools ein stark optimierter Solver ist. Der deduktive Solver dieser Arbeit verfolgt dagegen bewusst einen regelbasierten Ansatz und wurde nicht primär auf maximale Rechengeschwindigkeit hin optimiert.
Dennoch lassen sich einzelne Ideen aus dem Constraint Programming übernehmen, um auch den deduktiven Solver effizienter zu gestalten. Ein möglicher Ansatz betrifft die Reihenfolge, in der der Solver Zellen bei Shaving- und Agreement-Schritten auswählt. Anstatt alle Zellen in einer festen Reihenfolge zu durchlaufen, könnte etwa eine Heuristik wie *Minimum Remaining Values* (MRV) eingesetzt werden, bei der zuerst Zellen mit nur wenigen verbleibenden Möglichkeiten betrachtet werden. Auf diese Weise ließen sich Widersprüche möglicherweise früher erkennen und der Rechenaufwand des deduktiven Solvers verringern.

4.5. Vergleich mit bestehender Literatur

Der in dieser Arbeit verwendete Ansatz knüpft an das Deductive-Search-Modell von Browne an, bei dem Schwierigkeit über regelbasierte Deduktionsschritte unterschiedlicher Einbettungstiefe erfasst wird. Die Übertragung dieses Grundgedankens auf Buraitoraito erweist sich als tragfähig, da die

beobachteten DS-Werte sowohl mit den LoE-Kategorien als auch mit externen Schwierigkeitsangaben in nachvollziehbarer Weise zusammenhängen.
Gegenüber dem allgemeinen Ansatz besteht der Beitrag dieser Arbeit in der konkreten Modellierung eines deduktiven Solvers für Buraitoraito, in der Definition einer darauf abgestimmten Schwierigkeitsmetrik sowie in der Verbindung dieser Metrik mit einem Verfahren zur Instanzgenerierung. Damit wird das Deductive-Search-Prinzip nicht nur auf ein weiteres Puzzle übertragen, sondern auch um eine praktische Anwendung im Kontext automatischer Generierung erweitert.

4.6. Limitationen

Die Arbeit weist mehrere Einschränkungen auf. Erstens berücksichtigt der deduktive Solver nur eine begrenzte Menge an Regeln. Menschliche Spieler können zusätzliche Strategien verwenden, die im Modell nicht enthalten sind. Dadurch kann die tatsächliche subjektive Schwierigkeit einzelner Instanzen von der durch den DS beschriebenen Schwierigkeit abweichen.
Zweitens beruhen die externen Schwierigkeitsstufen von GrandGames und Janko nicht auf einem transparent dokumentierten und einheitlich kuratierten Verfahren. Sie sind daher als nützliche Vergleichswerte, nicht jedoch als objektiver Standard zu verstehen.
Drittens basiert ein Teil der Analysen auf vergleichsweise kleinen oder unausgewogenen Datensätzen. Dies betrifft insbesondere die Janko-Daten, für die nur wenige bewertete Instanzen verfügbar waren. Aussagen über deren Zusammenhang mit dem DS sollten daher vorsichtig interpretiert werden.
Viertens ist auch der Generierungsansatz in seiner jetzigen Form stark suchbasiert. Die hohe Zahl verworfener Kandidaten zeigt, dass die zielgerichtete Erzeugung bestimmter Schwierigkeitsklassen noch nicht effizient erfolgt.

4.7. Beitrag der Arbeit

Diese Arbeit zeigt, dass deduktionsbasierte Schwierigkeitsmetriken eine fundierte quantitative Analyse von Buraitoraito ermöglichen.
Dazu wurden ein deduktiver Solver, ein CP-basiertes Verfahren zur Validierung und Eindeutigkeitsprüfung sowie ein Generate-and-Test-Verfahren zur automatischen Instanzgenerierung entwickelt und miteinander verbunden. Auf dieser Grundlage konnte gezeigt werden, dass der Deductive Score in sinnvoller Beziehung zu externen Schwierigkeitsstufen, LoE-Kategorien, Puzzlegröße und Generierungsaufwand steht.
Die Kombination aus Solver, Metriken und Generierung liefert damit neue Einblicke in die Struktur und Schwierigkeit von Buraitoraito-Puzzles und schafft eine Grundlage für weiterführende Arbeiten zur Bewertung und automatischen Erzeugung solcher Instanzen.

5. Fazit

Ziel dieser Arbeit war die Entwicklung eines Verfahrens zur automatischen Generierung von Buraitoraito-Instanzen sowie zur objektiven Bewertung ihrer Schwierigkeit auf Basis eines deduktiven Lösungsprozesses.
Hierzu wurde ein Generate-and-Test-Verfahren implementiert, das aus vollständigen Lösungen gültige Puzzleinstanzen erzeugt und diese auf eindeutige Lösbarkeit sowie auf ihre Schwierigkeit überprüft. Zur Bewertung der Schwierigkeit wurde ein deduktiver Solver mit einer darauf aufbauenden Metrik eingesetzt, die sowohl die Anzahl als auch die Art der im Lösungsprozess auftretenden Deduktionsschritte berücksichtigt.
Die Ergebnisse zeigen, dass sich Buraitoraito-Instanzen unterschiedlicher Schwierigkeit automatisch erzeugen und mit dem entwickelten Ansatz systematisch bewerten lassen. Damit liefert die Arbeit eine Grundlage für die automatisierte Generierung und Schwierigkeitsanalyse dieses Puzzletyps.
Für zukünftige Arbeiten erscheint insbesondere die Untersuchung alternativer Generierungsverfahren interessant. Ein möglicher Ansatz ist der Einsatz von Monte-Carlo Tree Search (MCTS), das sich in vielen Spiel- und Planungskontexten bewährt hat [9]. Auch für die automatische Inhalts- und Puzzle-Generierung wurden suchbasierte Verfahren bereits systematisch untersucht [10]. Für Puzzle wurde MCTS unter anderem zur Generierung von Sokoban-Instanzen eingesetzt [11]. Im Vergleich zu dem in dieser Arbeit verwendeten Generate-and-Test-Verfahren könnte MCTS eine gezieltere Suche nach Instanzen mit bestimmten Schwierigkeitseigenschaften ermöglichen.

A. Instanzquellen

In dieser Arbeit wurden Buraitoraito-Instanzen aus mehreren unterschiedlichen Quellen verwendet. Die Datengrundlage umfasst sowohl automatisch generierte Instanzen als auch externe Datensätze und Puzzleplattformen.

A.1. Automatisch generierte Instanzen

Ein Teil der untersuchten Instanzen wurde im Rahmen dieser Arbeit mithilfe des in Abschnitt 2.6 vorgestellten Generate-and-Test-Verfahrens erzeugt. Die generierten Instanzen sind im zugehörigen Repository dieser Arbeit gespeichert und stehen für eine Reproduktion der Ergebnisse zur Verfügung.

A.2. Externe Puzzleplattformen und Datensätze

Zusätzlich wurden Instanzen aus öffentlich zugänglichen Puzzleplattformen sowie weiteren Datensätzen verwendet:

- `instances/grandgames`: Instanzen der Plattform GrandGames.[1]
- `instances/janko`: Instanzen der Plattform Janko.[2]
- `instances/janko5`: Instanzen aus der auf der Janko-Plattform bereitgestellten Beispielsammlung („Einige Beispiele“).[3]
- `instances/360-Nights-of-buraitoraito`: Instanzen aus dem Puzzleband *360 Nights of Buraitoraito*.[4]
- `instances/kidlogic100`: Instanzen aus dem Puzzleband Kid Logic Puzzles.[5]

[1] https://de.grandgames.net/starlight, Zugriff am 03.03.2026.
[2] https://www.janko.at/Raetsel/Buraitoraito/index.htm, Zugriff am 27.02.2026.
[3] https://www.janko.at/Raetsel/Naoki/Buraitoraito.htm, Zugriff am 27.02.2026.
[4] *360 Nights of Buraitoraito*, Amazon-Produktseite, https://www.amazon.de/dp/B0CWDXKWND, Zugriff am 18.03.2026.
[5] *Kid Logic Puzzles: Buraitoraito Teaser*, Amazon-Produktseite, https://www.amazon.de/Kid-Logic-Puzzles-Buraitoraito-Teaser/dp/1792971702, Zugriff am 18.03.2026.

Erklärung zum Einsatz von KI-Tools

Im Rahmen dieser Arbeit wurde folgendes KI-Tool unterstützend eingesetzt:

- ChatGPT (GPT-5 Thinking), OpenAI; Zugriff über https://chat.openai.com

Umfang und Art der Nutzung

Die Unterstützung umfasste sprachliche Überarbeitungen (Kürzungen, Umformulierungen, Titel- und Bildunterschriften sowie die einheitliche Verwendung von Fachterminologie), Hinweise zur Gliederung und zur konsistenten Darstellung von Tabellen und Abbildungen sowie programmiertechnische Unterstützung (z. B. C++/Python-Boilerplate, Refactoring-Hinweise und LaTeX-Hilfen).
Übernommene Codefragmente wurden eigenständig angepasst, getestet und verifiziert.

Verantwortlichkeit

Sämtliche fachlichen Inhalte (Modellentwurf, Implementierung, Experimente, Datenauswertung und Interpretation) stammen von mir. KI-generierte Vorschläge wurden geprüft, gegebenenfalls angepasst und ausschließlich in eigener Verantwortung übernommen. Wörtliche Übernahmen aus externen Quellen sind als solche kenntlich gemacht.

Literatur

[1] Stuart Russell und Peter Norvig. *Artificial Intelligence: A Modern Approach*. 3. Aufl. Prentice Hall, 2010, Link.

[2] Cameron Browne. "Deductive Search for Logic Puzzles". In: *Proceedings of the IEEE Conference on Computational Intelligence in Games (CIG)*. 2013, S. 1–8. DOI, Link.

[3] Brandon McPhail. *Light Up is NP-Complete*. Unpublished manuscript. 2005, Link.

[4] Erik D. Demaine u. a. "The Fewest Clues Problem". In: *8th International Conference on Fun with Algorithms (FUN 2016)*. Bd. 49. Leibniz International Proceedings in Informatics (LIPIcs). Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2016, 12:1–12:12. DOI.

[5] Cameron Browne. "Bitboard Methods for Games". In: *ICGA Journal* 37.2 (2014), S. 67–84, Link.

[6] Krzysztof R. Apt. *Principles of Constraint Programming*. Cambridge University Press, 2003. DOI.

[7] Laurent Perron, Frédéric Didier und Steven Gay. "The CP-SAT-LP Solver". In: *29th International Conference on Principles and Practice of Constraint Programming (CP 2023)*. Bd. 280. Leibniz International Proceedings in Informatics (LIPIcs). Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2023, 3:1–3:2. DOI, Link.

[8] Barbara De Kegel und Mads Haahr. "Procedural Puzzle Generation: A Survey". In: *IEEE Transactions on Games* 12.1 (2020), S. 21–40. DOI, Link.

[9] Cameron B. Browne u. a. "A Survey of Monte Carlo Tree Search Methods". In: *IEEE Transactions on Computational Intelligence and AI in Games* 4.1 (2012), S. 1–43. DOI, Link.

[10] Julian Togelius u. a. "Search-Based Procedural Content Generation: A Taxonomy and Survey". In: *IEEE Transactions on Computational Intelligence and AI in Games* 3.3 (2011), S. 172–186. DOI.

[11] Bilal Kartal, Nick Sohre und Stephen J. Guy. "Data-Driven Sokoban Puzzle Generation with Monte Carlo Tree Search". In: *Proceedings of the Twelfth AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment (AIIDE)*. 2016, S. 58–64. DOI, Link.