Systematischer Vergleich von Ant Colony Optimization und Constraint Programming zur Lösung von Akari-Puzzles

Bachelorarbeit

Maxim Mjakinin
0480592

3. November 2025

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 das japanische Logikrätsel *Akari* (auch bekannt als *Light Up*), das sich durch einfache Regeln, aber eine hohe kombinatorische Komplexität auszeichnet und nachweislich NP-vollständig ist. Ziel ist ein systematischer Vergleich verschiedener Algorithmusansätze zur Lösung von Akari-Puzzles.
Dazu wurden drei unterschiedliche Solver implementiert: ein deduktiver Ansatz (Deductive Search), der durch regelbasierte Schlüsse auch eine Einschätzung der Schwierigkeit einzelner Instanzen ermöglicht, ein auf *Ant Colony Optimization* (ACO) basierender heuristischer Ansatz, der die Lösungsfindung stochastisch unterstützt, sowie ein Constraint-Programming-Ansatz mittels OR-Tools, der das Puzzle als SAT/CP-Modell formuliert. Alle Solver wurden auf Benchmark-Puzzles aus externen Quellen angewandt und ihre Ergebnisse systematisch miteinander verglichen.
Die Resultate zeigen, dass der Deductive-Search-Ansatz nicht nur Lösungen deterministisch findet, sondern auch eine plausible Schwierigkeitseinschätzung im Vergleich zu menschlichen Referenzwerten liefert. Der ACO-Ansatz erreicht bei geeigneter Parametrierung ebenfalls hohe Lösungsraten, weist jedoch stochastische Schwankungen auf. Der CP-/SAT-Solver überzeugt durch seine Zuverlässigkeit, zeigt aber höhere Laufzeiten bei kleinen Instanzen.
Damit leistet die Arbeit einen Beitrag zum Verständnis der Stärken und Schwächen verschiedener algorithmischer Paradigmen für Akari und diskutiert deren jeweilige Eignung für praktische Anwendungen sowie die Implikationen für die Bewertung von Puzzleschwierigkeit.

Inhaltsverzeichnis

Tabellenverzeichnis

1. Einleitung

1.1. Motivation und Relevanz

Logikrätsel sind kleine Systeme mit klaren Regeln und überraschend reicher Dynamik. Sie dienen nicht nur der Unterhaltung, sondern auch als Testfelder für Suchstrategien, Heuristiken und Constraint-Modelle. In dieser Arbeit steht das Nikoli-Rätsel *Akari* im Mittelpunkt. Bereits wenige, intuitive Regeln erzeugen starke Wechselwirkungen: Lampen beleuchten ganze Zeilen und Spalten, Zahlenfelder erzwingen lokale Muster, und daraus ergibt sich ein globales Konsistenzproblem.

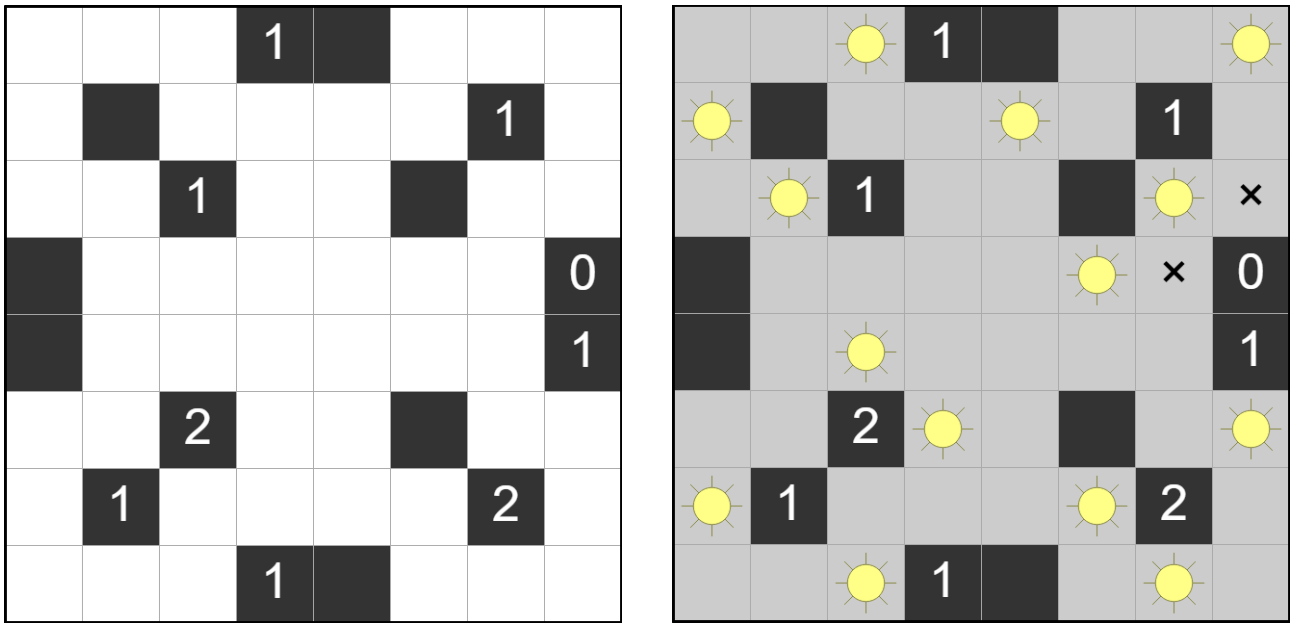
Aus algorithmischer Sicht ist Akari besonders interessant, weil wenige lokale Regeln weitreichende Abhängigkeiten erzeugen und den Suchraum stark verzweigen. Damit eignet sich das Rätsel, um unterschiedliche Paradigmen unter vergleichbaren Bedingungen zu bewerten: deduktive (regelbasierte) Verfahren, stochastische Metaheuristiken und deklarative SAT/CP-Modelle. Neben der reinen Lösbarkeit spielt in der Praxis die *Schwierigkeit* eine zentrale Rolle, verstanden als logische Tiefe und der Aufwand, den sowohl menschliche als auch algorithmische Löser benötigen. Ein reproduzierbares, datenbasiertes Schwierigkeitsmaß zu entwickeln und gegen externe Labels zu kalibrieren ist daher ebenso relevant wie das Lösen selbst.

Vor diesem Hintergrund untersucht diese Arbeit *Akari* als kompaktes, aber anspruchsvolles Benchmark-Problem: Wir implementieren und vereinheitlichen drei Solver (Deductive Search, CP–SAT, ACO), vermessen sie auf einer gemeinsamen *Benchmark-Sammlung* und gleichen die aus Deduktion abgeleitete Schwierigkeit mit Herausgeberangaben ab. Ziel ist eine objektive, systematische Gegenüberstellung von Lösungsfähigkeit, Laufzeit und Stabilität als Grundlage für belastbare Aussagen über Stärken, Schwächen und sinnvolle Einsatzszenarien der Ansätze.

1.2. Spielbeschreibung

Das Rätsel *Akari* (auch *Light Up*) wird auf einem rechteckigen Gitter aus weißen und schwarzen Feldern gespielt. Ziel ist es, Lampen auf *weißen* Feldern so zu platzieren, dass alle Regeln erfüllt sind:

1. **Abdeckung:** *Alle weißen Felder müssen beleuchtet sein* (mindestens von einer Lampe).
2. **Beleuchtungswirkung:** Jede Lampe beleuchtet alle weißen Felder in ihrer Zeile und Spalte, bis ein schwarzes Feld oder der Gitterrand erreicht wird.
3. **Keine Konflikte:** Zwei Lampen dürfen sich nicht gegenseitig beleuchten, das heißt sie dürfen nicht in derselben Zeile oder Spalte ohne dazwischenliegendes schwarzes Feld stehen.
4. **Zahlenhinweise:** Eine Zahl in einem schwarzen Feld gibt *genau* an, wie viele Lampen in den vier orthogonal angrenzenden Feldern platziert werden müssen. Schwarze Felder ohne Zahl stellen keine weiteren Einschränkungen dar.



(a) Puzzle (b) Lösung

Abbildung 1.1.: Akari-Beispiel: (a) Puzzle und (b) eindeutige Lösung. Weiße Felder müssen beleuchtet sein, Zahlen in schwarzen Feldern geben die Anzahl von Lampen in den vier orthogonal angrenzenden Feldern an. Beispiel nach Janko, Akari Nr. 522[1].

Akari verbindet lokale Bedingungen durch Zahlen an einzelnen Feldern mit globalen Abhängigkeiten durch die Beleuchtung über ganze Zeilen und Spalten. Diese Kopplung führt bereits auf kleinen Gittern zu hoher kombinatorischer Komplexität. Veröffentlichte Instanzen sind in der Regel so konstruiert, dass die Lösung eindeutig ist. Das macht Akari sowohl für menschliche Lösende als auch für algorithmische Untersuchungen attraktiv.

[1] https://www.janko.at/Raetsel/Akari/522.a.htm

1.3. Relevanz und Literaturüberblick

Komplexität und Einordnung

Akari gehört zu den Nikoli-Logikrätseln mit einfachen Regeln und hoher kombinatorischer Vielfalt. Die Entscheidungsvariante ist NP-vollständig. McPhail zeigt dies durch Reduktion von CIRCUIT-SAT [2]. Pulles [3] untersucht Varianten mit reduzierten Zahlenhinweisen und vergleicht Backtracking- und SAT-Solver als algorithmische Referenz. Übergreifende Darstellungen zur Komplexität von Spielen und Puzzles bieten Costa [4] sowie Hearn [5].

Deduktive und regelbasierte Ansätze

Browne [6] beschreibt *Deductive Search* als beschränkte Constraint-Propagation mit fester Suchtiefe (Level of Embedding, LoE), wobei *Deducibility* die Lösbarkeit *auf einer gegebenen LoE* bezeichnet. Der Ansatz begrenzt gedankliche Einbettungen, um menschliche Lösungsprozesse nachzubilden und Schwierigkeitsgrade abzuleiten. Dekker [7] entwickelt darauf aufbauend ein algorithmisches Schwierigkeitsmaß, das sich an menschlichen Deduktionsmustern orientiert und Puzzles nach deduktiven bzw. Rateschritten klassifiziert.

Metaheuristische Verfahren

Rosberg et al. [1] präsentieren einen zweiphasigen *Ant Colony Optimization*-Algorithmus mit logischem Preprocessing, der Instanzen bis 40×30 löst. Verwandte evolutionäre und hybride Verfahren stammen von Salcedo-Sanz et al. [8] und Ortiz-García et al. [9], die genetische und neuronale Komponenten kombinieren. Beispiele stochastischer Solver finden sich in einer Python-Implementierung mit Hill Climb, Simulated Annealing und DNN-gestützter Heuristik[2]. Grundlagen liefert Dorigo & Stützle [10].

Deklarative Modelle (SAT / ASP / CP)

Çelik et al. [11] vergleichen *Answer Set Programming* und *Constraint Programming* an vier Nikoli-Puzzles (u. a. Akari) und zeigen die gute Regelabbildung. Pulles [3] kombiniert eine SAT-Modellierung mit Puzzle-Generierung und Validierung eindeutiger Lösungen. Praktische Beispiele: Z3[3] und OR-Tools[4].

Varianten und Schwierigkeitsbewertung

Web-Sammlungen ordnen *Akari* nach empirischen Kriterien. Bei *Janko* erfolgt die Sortierung nach mittlerer Lösungszeit[5], während in der *Portable Puzzle Collection* »Easy« Instanzen ohne und »Hard« solche mit Backtracking kennzeichnet[6]. Diese Angaben dienen als Referenz für den Vergleich algorithmisch bestimmter Schwierigkeitsmaße.

Zwischenfazit

Die Forschung zu *Akari* reicht von logikbasierten Deduktionsverfahren über metaheuristische bis zu deklarativen Ansätzen. Diese Arbeit vergleicht Deductive Search, CP–SAT und ACO unter gleichen Bedingungen und leitet ein datenbasiertes Schwierigkeitsmaß ab.

[2] github.com/rperera12/AKARI-LightUp-GameSolver
[3] https://github.com/Borroot/akari
[4] https://github.com/SmilingWayne/PuzzleSolver/blob/main/Puzzles/Akari.ipynb
[5] https://www.janko.at/Raetsel/index.htm#schwierigkeit
[6] https://www.chiark.greenend.org.uk/~sgtatham/puzzles/doc/lightup.html

1.4. Zielsetzung

Ziel dieser Arbeit ist der systematische und reproduzierbare Vergleich dreier algorithmischer Ansätze zur Lösung des Logikrätsels *Akari (Light Up)*: (i) ein deduktiver, regelbasierter Solver (*Deductive Search*, DS), (ii) ein metaheuristischer Ansatz auf Basis der *Ant Colony Optimization* (ACO), und (iii) ein deklaratives Modell mit *CP–SAT* aus Google OR-Tools. Untersucht werden Lösungsfähigkeit, Laufzeit und Skalierungsverhalten. Für den deduktiven Ansatz wird zusätzlich ein internes Schwierigkeitsmaß entwickelt und evaluiert. Beim ACO-Verfahren steht der Einfluss des Pheromonparameters α ($\alpha = 0$ vs. $\alpha = 1$) im Fokus.

Forschungsfragen

- **Effizienz und Skalierung:** Wie unterscheiden sich DS, ACO und CP–SAT hinsichtlich Erfolgsrate und Laufzeit über eine gemeinsame Instanzmenge und verschiedene Gittergrößen?
- **Einfluss von α im ACO:** Welche Effekte zeigen sich zwischen ACO ohne Pheromon ($\alpha = 0$) und mit Pheromonsteuerung ($\alpha = 1$) in Bezug auf Zeit bis zur Lösung und Abdeckung?
- **Schwierigkeit:** Lässt sich aus der DS-Propagation ein nachvollziehbares, größenrobustes Schwierigkeitsmaß ableiten, und wie verhält es sich zu externen Einstufungen (z. B. Janko-Labels)?
- **Erklärbarkeit und Performance:** Welche Trade-offs ergeben sich zwischen nachvollziehbaren Regelableitungen (DS), stochastischer Suche (ACO) und deklarativer Optimierung (CP–SAT)?

Metriken

- *Korrektheit/Erfolg:* Anteil gelöster Instanzen, bei denen alle Regeln erfüllt sind.
- *Laufzeit:* Wandzeit bis zur ersten gültigen Lösung (Median und Quartile).
- *Suchaufwand:* Anzahl der Regelapplikationen (DS) bzw. Iterationen (ACO).
- *Robustheit:* Streuung der Ergebnisse über Mehrfachläufe, kein expliziter Seed-Sweep.
- *Schwierigkeit:* aggregierte logische Ableitungsschritte (DS) als internes Maß. Zusätzlich betrachten wir die Korrelation mit externen Schwierigkeitsangaben.

Versuchsaufbau

Alle Verfahren werden auf identischen Instanzen unter gleichen Rahmenbedingungen ausgeführt (gemeinsame Hardware, einheitliche Zeit- und Speicherkontingente). Für ACO werden pro Instanz mehrere unabhängige Läufe durchgeführt. Berichtet werden Mittelwerte und Streuungen. Die Hyperparameter folgen den in der Literatur üblichen Standardwerten. Systematisch variiert wird ausschließlich α. Deductive Search und CP–SAT verwenden feste, dokumentierte Standardeinstellungen. Ein gemeinsames Preprocessing (Fixpunkt-Rule-Engine) wird, wo relevant, konsistent zwischen den Ansätzen eingesetzt und separat ausgewiesen.

Abgrenzung

Nicht Gegenstand der Arbeit sind Puzzle-Generierung, neue Komplexitätsbeweise oder die Entwicklung neuer allgemeiner Solver. Der Fokus liegt auf einem fairen, transparenten Vergleich bestehender Paradigmen sowie auf der Analyse ihrer Stärken und Schwächen anhand realer Instanzen.

Ergebnisse/Artefakte.

Code und Instanzlisten werden bereitgestellt. Die Reproduktion erfolgt durch Ausführen der Programme mit den im Repository hinterlegten festen Zufallsseeds für ACO. Weitere Hyperparameter oder Voreinstellungen für DS und CP–SAT werden nicht vorgegeben.

1.5. Struktur der Arbeit

Kapitel 1: **Einleitung** führt in das Thema ein, stellt *Akari* vor, ordnet literaturseitig ein und formuliert die Zielsetzung (1).

Kapitel 2: **Methoden** beschreibt die drei Ansätze (Deductive Search, CP–SAT/OR-Tools und ACO) einschließlich Problemformulierung, Modellierungsentscheidungen, Pseudocode und Parametrisierung. Außerdem werden Benchmarks, Ausführungsprotokoll sowie Metriken und Auswertung definiert (2).

Kapitel 3: **Ergebnisse** präsentiert die Experimente: Kurzüberblick zum Versuchsrahmen (Hardware, Zeit- und Speicherbudgets), Resultate je Ansatz (Lösungsabdeckung, Laufzeiten), methodenübergreifender Vergleich und Zusammenfassung der Befunde (3).

Kapitel 4: **Diskussion** interpretiert die Ergebnisse und diskutiert Stärken, Schwächen, Limitationen und Validität (4).

Kapitel 5: **Fazit** fasst die wichtigsten Erkenntnisse zusammen und skizziert mögliche Anschlussarbeiten (5).

2. Methoden

2.1. Problemformulierung und gemeinsame Infrastruktur

Dieses Kapitel beschreibt zunächst die formale Problemstruktur von *Akari* und eine gemeinsame technische Infrastruktur, auf der alle drei Solver-Implementierungen (Deductive Search, CP–SAT und ACO) aufbauen. Anschließend werden die spezifischen Methoden der einzelnen Ansätze beschrieben.

Formale Formulierung (solver-agnostisch)

Gegeben ist ein rechteckiges Gitter G mit Höhe H und Breite W. Weiße Zellen tragen binäre Variablen $x_{r,c} \in \{0, 1\}$ (Lampe ja/nein). Ein *Run* ist ein maximal zusammenhängendes Weißsegment in einer Zeile oder Spalte, getrennt durch eine schwarze Zelle oder den Rand. Für eine weiße Zelle (r, c) sei $\mathrm{vis}(r, c)$ die Menge aller weißen Zellen in derselben Zeile oder Spalte bis zum nächsten Block einschließlich (r, c). Für eine nummerierte Schwarzzelle b mit Zahl $k \in \{0, \ldots, 4\}$ sei $N(b)$ die orthogonale Nachbarschaft.
Die Regeln des Puzzles lauten:

$$\textbf{(Kein Lampenkonflikt)} \quad \sum_{(r,c)\in \mathrm{Run}} x_{r,c} \le 1 \qquad \text{für jeden Run,} \tag{2.1}$$

$$\textbf{(Beleuchtung)} \quad \sum_{(r',c')\in \mathrm{vis}(r,c)} x_{r',c'} \ge 1 \qquad \text{für jede weiße Zelle } (r, c), \tag{2.2}$$

$$\textbf{(Zahlenhinweise)} \quad \sum_{(r',c')\in N(b)} x_{r',c'} = k \qquad \text{für jede Hinweiszelle } b \text{ mit Zahl } k. \tag{2.3}$$

Diese Formulierung bildet die Grundlage für das CP–SAT-Modell, während DS und ACO dieselbe Logik über Propagation umsetzen.

Gemeinsame Infrastruktur (für DS/ACO/CP)

Gitterrepräsentation

Das Spielfeld wird als zweidimensionales Array von Zellstrukturen gespeichert. Für jede Position (r, c) wird der Typ (weiß, schwarz, Zahl k) hinterlegt. Zur effizienten Zustandsverwaltung werden drei Bitmasken verwendet: (i) `lampMask` markiert gesetzte Lampen, (ii) `litMask` markiert aktuell beleuchtete Felder und (iii) `allowedMask` kennzeichnet zulässige Lampenpositionen. Nach jeder Zustandsänderung aktualisiert eine Routine `propagateLight()` die Beleuchtung, indem sie in vier orthogonalen Richtungen bis zum nächsten Block oder Gitterrand fortgeführt wird. Diese Infrastruktur wird in allen Solvern verwendet, um Konsistenzprüfungen und Beleuchtungsregeln effizient umzusetzen.

Einsatz von Bitmasken

Bitmasken reduzieren den Speicher auf 1 Bit pro Zelle und erlauben wortbreite AND/OR-Operationen sowie Bitzählung (`popcount` = Anzahl gesetzter Bits). Sichtlinienprüfungen, Beleuchtungs-Updates und Kandidatenstreichungen erfolgen damit in wenigen, cachefreundlichen Maschinenoperationen, typischerweise schneller als mit reinen 2D-Arrays oder `std::set`.

Begründung der Programmiersprachenwahl

Die Implementierungen von Deductive Search, CP–SAT und ACO wurden in C++ realisiert. C++ bietet eine gute Balance aus Effizienz, Kontrolle über Speicherstrukturen und Verfügbarkeit numerischer Bibliotheken. Für dieses Projekt war insbesondere die Möglichkeit relevant, Bitmasken und Gitteroperationen speichereffizient zu implementieren und direkt zu profilieren. Zudem lassen sich OR-Tools (für den CP–SAT-Solver) und eigene C++-Module einheitlich in einer gemeinsamen Build-Umgebung (CMake) integrieren. Die Wahl fiel daher pragmatisch auf C++ als performante und etablierte Sprache mit hoher Portabilität, auch aufgrund vorhandener Erfahrung aus früheren Projekten.

2.1.1. Fixpunkt-RuleEngine (Schritt-für-Schritt-Beispiel)

Vor der eigentlichen Suche werden in Deductive Search (*simplify*) und ACO (*preprocessing*) deterministische Regeln in fester Reihenfolge bis zum Fixpunkt ausgeführt. Für *CP–SAT* erfolgt kein Preprocessing. Die logischen Implikationen werden durch die Modellklauseln und die Inferenz des Solvers abgedeckt. Jede Regelanwendung zählt als *Deduktionsschritt*. Koordinaten sind als (r, c) notiert (0-indiziert, r=Zeile, c=Spalte).

Ablauf (nur DS/ACO): Zunächst werden R1 und R2 einmal pro Gitter ausgeführt, anschließend R3 bis R5 iterativ, bis keine Änderungen mehr auftreten.

One-shot-Regeln (einmal pro Gitter):

R1 Clue-Zero (`prune0`): Bei Hinweis $k = 0$ werden alle vier orthogonalen Nachbarn als Lampenkandidaten ausgeschlossen.

R2 Clue-Three (`prune3`): Bei $k = 3$ sind alle vier Diagonalfelder durch mindestens eine der drei noch zu setzenden Nachbarlampen indirekt beleuchtet; eine Lampe auf einem Diagonalfeld würde somit immer einen Konflikt erzeugen $\Rightarrow$ Diagonalfelder streichen.

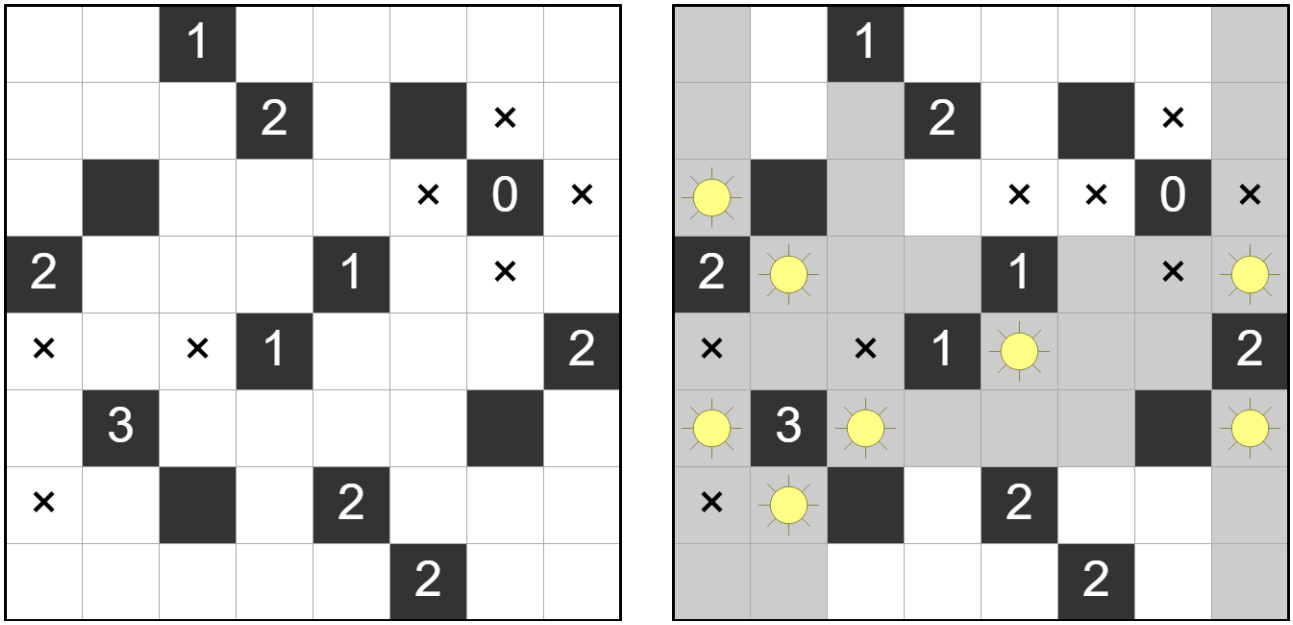
Fixpunkt-Regeln (iterativ bis zum Fixpunkt):

R3 Clue-Saturation (`forceNeighbors`): Für Hinweiszelle b mit Zahl k:
Falls |gesetzte Nachbarn| + |noch erlaubte Nachbarn| $= k$, werden alle erlaubten Nachbarn zu Lampen gesetzt.

R4 Clue-Closure (`pruneIfClueSatisfied`): Hat ein Hinweis bereits genau k Lampen, werden alle übrigen Nachbarn als Kandidaten ausgeschlossen.

R5 Singleton (Coverage/Run, `singleton`): Besitzt ein (noch) unbeleuchtetes Feld genau *eine* mögliche Beleuchter-Position in seiner Sichtlinie, muss dort eine Lampe gesetzt werden.

Erklärung und Demonstration der Fixpunkt-Regeln an einem Beispiel



(a) Anfangszustand nach R1 und R2: × markieren gestrichene Kandidaten (Clue-Zero bei $(2, 6)$, Clue-Three bei $(5, 1)$).

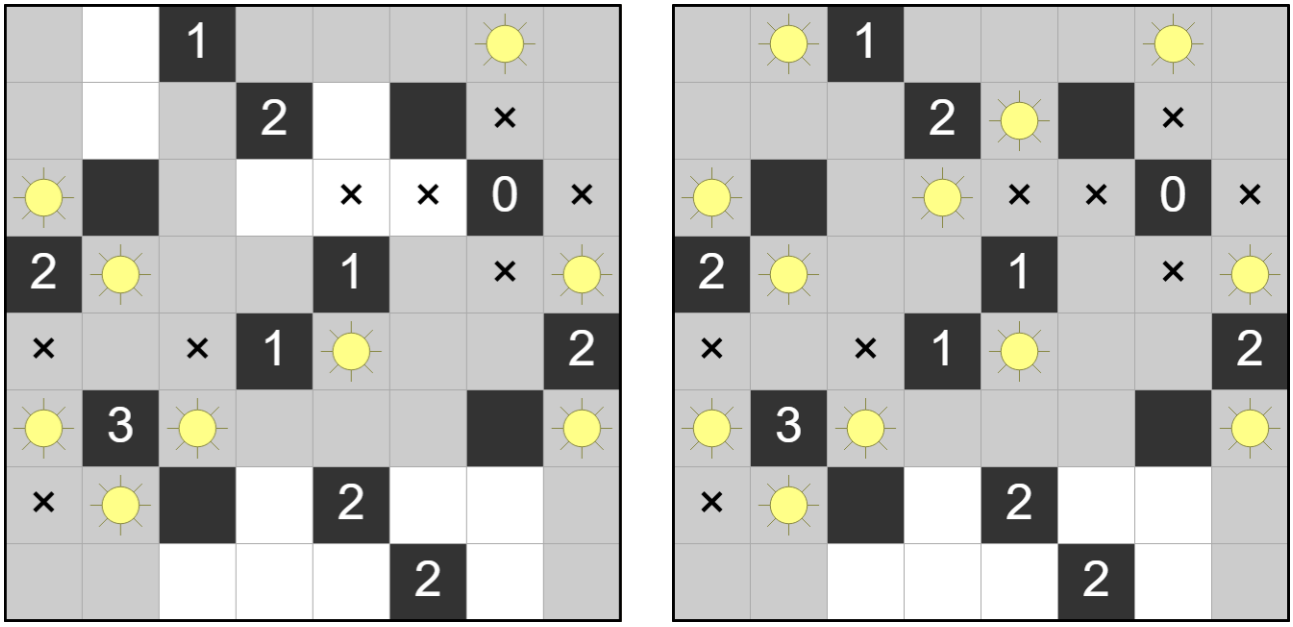
(b) Folgen von R3 und R4: notwendige Nachbarlampen werden gesetzt (R3), anschließend überzählige Nachbarn verboten (R4).

Abbildung 2.1.: Erster Schrittblock: Kandidatenstreichungen (R1 und R2) und daraus resultierende erzwungene Nachbarlampen (R3) mit Abschluss (R4).

Koordinatenkonvention: Koordinaten werden als (r, c) angegeben (0-indiziert) mit Ursprung oben links: r zählt die *Zeilen* von oben nach unten, c die *Spalten* von links nach rechts. Beispiele: $(3, 0)$ bedeutet 4. Zeile/1. Spalte; $(0, 6)$ bedeutet oberste Zeile/7. Spalte.

Zunächst greifen R1 und R2: Der Nullhinweis bei $(2, 6)$ verbietet alle vier orthogonalen Nachbarn als Lampen. Die Felder werden mit × markiert (Abbildung 2.1a). Beim 3er-Hinweis bei $(5, 1)$ werden gemäß R2 zusätzlich alle Diagonalen gestrichen, da sie unabhängig von der späteren Anordnung der drei Nachbarlampen stets in einer Sichtlinie liegen und dadurch im Konflikt stünden, auch diese erscheinen mit × (ebenfalls Abbildung 2.1a). Nach jeder Streichung wird die Beleuchtung entlang von Zeilen und Spalten propagiert. Beleuchtete Felder werden nicht weiter als Lampenkandidaten geführt.

Durch diese Bereinigung verbleiben am 2er-Hinweis $(3, 0)$ genau zwei zulässige Nachbarpositionen. Da $k = 2$ gilt, müssen beide Nachbarn Lampen werden (R3). Die so ausgelöste Kettenreaktion erzwingt weitere eindeutige Nachbarlampen, u. a. für den 3er bei $(5, 1)$, den 1er bei $(4, 3)$ und den 2er bei $(4, 7)$ (sichtbar als gesetzte Lampen in Abbildung 2.1b). Sobald ein Hinweis seine Sollzahl erreicht hat, verbietet R4 alle übrigen orthogonalen Nachbarn, beim 1er bei $(3, 4)$ werden daher die verbleibenden Nachbarn gestrichen (rechts in Abbildung 2.1b).



(a) R5 (Singleton/Coverage): Lampe auf $(0, 6)$ wird erzwungen, da $(1, 6)$ sonst unbeleuchtet bliebe.

(b) Fixpunkt-Teillösung nach R1 bis R5; von hier startet die eigentliche Suche (DS/ACO).

Abbildung 2.2.: Zweiter Schrittblock: Coverage-Singleton (R5) und resultierender Fixpunktzustand.

Im Anschluss greift R5 (Singleton/Coverage). Das Feld (1,6) ist noch unbeleuchtet und besitzt in seinen Sichtlinien nur eine zulässige Lampe auf Position (0,6). Daher wird dort eine Lampe gesetzt. (siehe Abbildung 2.2a).

Nach jeder Setzung werden Beleuchtung und Kandidatenmengen erneut aktualisiert, und die Regeln R3 bis R5 werden in fester Reihenfolge wieder geprüft. Dieser Zyklus endet im dargestellten Fixpunkt (Abbildung 2.2b), in dem keine weitere deterministische Regel mehr anwendbar ist, ohne Raten zu erfordern.

Einordnung

Deductive Search (*simplify*) und *ACO* (*preprocessing*) verwenden diesen Fixpunktzustand als Ausgangsbasis. Für *CP–SAT* erfolgt kein Preprocessing. Die Inferenz wird direkt durch das Modell und den Solver ausgeführt. Die quantitative Wirkung des deterministischen Preprocessings wird in Kapitel 3 empirisch belegt.

2.2. Deductive Search (DS)

2.2.1. Leitidee und Bezug zur Literatur

Deductive Search ist eine *breitensuchartige, tiefenbegrenzte* Propagationsstrategie für Deduktionspuzzles. Sie kombiniert drei Bausteine [6]:

- **Simplify:** reine Constraint-Propagation durch die Rule-Engine bis zum Fixpunkt (kann bereits Lösung oder Widerspruch erkennen, ganz ohne Hypothesen).
- **Shaving:** teste eine Hypothese (z. B. „Lampe setzen“ oder „verbieten“), führt ein Ast zum Widerspruch, wird der Gegenast erzwungen bzw. der widersprechende Wert ausgeschlossen.
- **Agreement:** sind *beide* Hypothesen konsistent, werden Konsequenzen übernommen, die in *beiden* Ästen gleich ausfallen (Schnittmenge der Implikationen).

In dieser Arbeit wird LoE $\in \{0, 1, 2\}$ verwendet (vgl. Experimente). Die *Rule-Engine* aus Kap. 2.1 liefert die Domänen-Propagation für *Akari*.

2.2.2. Notation und Parameter

- **LoE** (Level of Embedding): Einbettungstiefe der Hypothesensuche:
  - 0: nur *Simplify*
  - 1: Hypothesen mit *Shave/Agreement* gegen den Fixpunkt
  - 2: rekursives Shaving/Agreement auf einer weiteren Ebene.
- **max_LoE**: Äußeres Limit.
- **useDirty** $\in \{$`true`, `false`$\}$: Kandidatenwahl via *Dirty-Queue* statt Vollscan (siehe Abschnitt 2.2.4).
- **Cand**: Momentan erlaubte Lampenkandidaten, formal $\mathrm{Cand}(g)$ für die Kandidatenmenge einer Gruppe g.
- **Simplify**$(\cdot)$: Fixpunkt-Propagation über die Regeln aus Kap. 2.1.

2.2.3. Originalansatz nach Browne (Baseline)

Operationen (formal)

Variablenmodell: Sei $\mathcal{X} = \{X_1, \ldots, X_n\}$ die Menge der Entscheidungsvariablen. Indizes i, j bezeichnen beliebige Variablen aus $\mathcal{X}$. Die Domain sei D (bei *Akari* binär, $D = \{\text{Lampe}, \text{ keine Lampe}\}$). Ein Zustand S ist die Gesamtheit der aktuellen Domains aller Variablen, also $S = (X_1^*, \ldots, X_n^*)$. Die aktuelle Domain einer Variable schreiben wir $X_k^* \subseteq D$. Eine Variable heißt *unaufgelöst*, wenn X_k^* kein Einzelwert ist. Werte bezeichnen wir mit $v, w \in D$.
Notation: S^* ist die Menge der unaufgelösten Variablen. Eine Hypothese schreiben wir $X_k{\cdot}v$ (temporär $X_k \leftarrow v$). Eine endgültige Festlegung schreiben wir $X_k \leftarrow X_k \cdot v$. Eine Eliminierung schreiben wir $X_k \leftarrow X_k \setminus v$. Das Symbol $\perp$ bezeichnet einen Widerspruch.
Für eine unaufgelöste Variable X_i gelten:
Shaving (eliminiert alle Werte, die zu einem Widerspruch führen):

$$\forall v \in X_i^* : \ (X_i \cdot v \Rightarrow \perp) \Rightarrow X_i \leftarrow X_i \setminus v. \tag{2.4}$$

Agreement (übernimmt, was in *allen* Hypothesen folgt):

$$(\forall v \in X_i^* : \ X_i \cdot v \Rightarrow X_j \cdot w) \Rightarrow X_j \leftarrow X_j \cdot w. \tag{2.5}$$

$$(\nexists v \in X_i^* : \ X_i \cdot v \Rightarrow w \in X_j^*) \Rightarrow X_j \leftarrow X_j \setminus w. \tag{2.6}$$

Hinweis: In den Prämissen verwenden wir Brownes Kurzform $X_j \cdot w$. Für (2.6) referenzieren wir explizit die Domänenmitgliedschaft $w \in X_j^*$. Elimination ist nur gerechtfertigt, wenn w in *allen* Hypothesen aus der Domain fällt. In binären Domains fällt dies mit einer Festlegung zusammen. In nicht-binären bleibt die Mitgliedschaftsform notwendig.

Ablauf

(1) **0-LoE: *Simplify* bis zum Fixpunkt.** Ausgehend vom aktuellen Zustand werden alle durch die Constraints erzwungenen Updates propagiert, bis keine Änderung mehr entsteht. Tritt eine Verletzung auf, ist der Status CONTRADICTION. Sind alle Variablen festgelegt, lautet der Status SOLVED.

(2) **1-LoE: *Shave/Agreement* über alle noch ungeklärten Variablen.** Für jede solche Variable X_i werden beide Hypothesen $X_i \cdot v$ geprüft, jeweils mit anschließendem lokalen *Simplify*:

- *Shave:* Führt genau eine Hypothese zum Widerspruch, wird der *Gegenfall* im Arbeitszustand sofort vollzogen (Wert setzen bzw. eliminieren).
- *Agreement:* Sind *beide* Hypothesen konsistent, werden nur jene Konsequenzen übernommen, die in *allen* Hypothesen identisch folgen (Schnittmenge neuer Fakten).
- Sind *beide* Hypothesen widersprüchlich, ist der Gesamtzustand CONTRADICTION.

Nach jeder Übernahme (Shave/Agreement) wird erneut *Simplify* bis zum Fixpunkt ausgeführt. Schritt (2) wird wiederholt, bis keine Änderungen mehr auftreten oder SOLVED/CONTRADICTION erreicht ist.

(3) **Optional: 2-LoE.** Wenn nach wiederholten 1-LoE-Pässen keine weiteren Updates entstehen, kann ein tieferer Pass auf einer zusätzlichen Einbettungsebene ausgeführt werden, anschließend erfolgt eine Rückkehr zu Schritt (2).

Schwierigkeitsmaß

Browne klassifiziert die Schwierigkeit über die in DS anfallenden Updates pro Einbettungstiefe (LoE): Fixpunkt-Updates durch **Simplify** (S_n), durch **Shaving** erzwungene Verbote/Festlegungen (V_n) und durch **Agreement** übernommene Festlegungen (A_n). Das Maß lautet [6]:

$$\mathrm{diff}(S) = \frac{S_0 + 4\,(S_1 + V_1 + A_1) + 9\,(S_2 + V_2 + A_2)}{\sum_{i=1}^{n} D_i}, \tag{2.7}$$

wobei D_i die Domaingröße von X_i ist. Der Nenner entspricht der Gesamtzahl potenzieller Werte.
Zählweise: S_n zählt ausschließlich *Simplify*-Fixpunktupdates, die *im Arbeitszustand S* auf Ebene n ausgeführt werden. *Nicht* mitgezählt werden testweise Simplify-Aufrufe innerhalb von Hypothesen, da deren interne Regelanwendungen verworfen werden. V_n zählt die durch *Shaving* in S übernommenen Verbote/Festlegungen, A_n die durch *Agreement* übernommenen Festlegungen.
Interpretation: Die Gewichte $1, 4, 9 = (n{+}1)^2$ modellieren steigenden kognitiven Aufwand auf höheren LoE-Ebenen. Der Nenner normalisiert auf die Instanzgröße. Viele Updates auf LoE $1/2$ erhöhen $\mathrm{diff}(S)$, während überwiegende LoE= 0-Lösungen niedrige Werte ergeben.

2.2.4. Erweiterungen & Abgleich

Inkrementelle Verarbeitung via Dirty-Set: Statt Vollscans über alle Zellen wenden wir *Simplify* sowie *Shaving/Agreement* nur auf Bereiche an, die seit dem letzten Update betroffen sind
(ΔS; vgl. Browne, Abschnitt *E. Interface*). Wir realisieren ΔS als bitset-basiertes Dirty-Set: Nach Setzen oder Verbieten markieren wir Zelle, orthogonale Nachbarn und Sichtstrahlen und iterieren bevorzugt über dieses Set. Der Aufwand pro Pass sinkt so von $O(|G|)$ auf $O(\Delta)$ mit Δ = Anzahl tatsächlich berührter Zellen.
Neue Schwierigkeitsbewertung: Gezählt werden pro Einbettungstiefe LoE die Updates am Arbeitszustand S: S_n sind *Simplify*-Fixpunktupdates auf Ebene n *ohne* testweise Simplify in Hypothesen, V_n sind übernommene Verbote/Festlegungen durch *Shaving*, A_n die übernommenen Festlegungen durch *Agreement*. Normalisierung über $Z = \sum_i D_i$ (bei *Akari* $Z = 2 \cdot \#$ anfängliche leere Felder). Die Bewertung lautet wie folgt:

$$S_n = \text{Updates auf LoE} = n, \qquad n \in \{0, 1, 2\}, \tag{2.8}$$

$$\text{base} = \frac{S_0 + 4\,S_1 + 9\,S_2}{Z}, \tag{2.9}$$

$$\text{diff}_{\text{base}} = \text{base} \cdot \big(1 + \lambda \cdot \text{max\_LoE}\big), \tag{2.10}$$

$$\text{effort} = \frac{\log\big(1 + \text{hypoFail}\big)}{\log\big(1 + \#\text{anfängliche leere Felder}\big)}, \tag{2.11}$$

$$\text{diff} = \text{diff}_{\text{base}} + \mu \cdot \text{effort}. \tag{2.12}$$

Dabei ist (2.10) ein *multiplikativer* LoE-Bonus auf der Basisbewertung, (2.12) ergänzt *additiv* einen leichten Aufwandsterm für fehlgeschlagene Hypothesen. Aus diff (Gl. (2.12)) wird abschließend eine Klasse $c \in \{0, \ldots, 9\}$ anhand fester Schwellwerte (Janko-orientiert) abgeleitet.

2.2.5. Pseudocode (kompakt, LoE ≤ 2)

Algorithm 1 DeductiveSearch(G, max_LoE): Kompakter Ablauf

```
1: // status ∈ {OK, Solved, Contradiction}; progressLocal: Fortschritt dieses DEDUCE-Aufrufs;
   anyChange: Fortschritt in dieser while-Runde
2: status ← SIMPLIFY(G, 0)                                   ▷ [0-LoE] Fixpunkt
3: if status ∈ {SOLVED, CONTRADICTION} then
4:     return status
5: while TRUE do                                  ▷ endet via return oder NONDEDUCIBLE
6:     anyChange ← false
7:     if max_LoE ≥ 1 then                                    ▷ [1-LoE]-Runden
8:         repeat
9:             (progressLocal, status) ← DEDUCE(G, 1)        ▷ Shave/Agreement auf Tiefe 1
10:            anyChange ← anyChange or progressLocal
11:            if status ∈ {SOLVED, CONTRADICTION} then
12:                return status
13:            if progressLocal then
14:                status ← SIMPLIFY(G, 1)          ▷ [1-LoE] Fixpunkt nach Übernahmen
15:                if status ∈ {SOLVED, CONTRADICTION} then
16:                    return status
17:        until not progressLocal                  ▷ keine Änderungen mehr auf [1-LoE]
18:    if not anyChange and max_LoE ≥ 2 then                ▷ einmalige [2-LoE]-Runde
19:        (progressLocal, status) ← DEDUCE(G, 2)
20:        if status ∈ {SOLVED, CONTRADICTION} then
21:            return status
22:        if progressLocal then
23:            status ← SIMPLIFY(G, 2)                      ▷ [2-LoE] Fixpunkt
24:            if status ∈ {SOLVED, CONTRADICTION} then
25:                return status
26:            continue                  ▷ zurück zum Schleifenkopf  wieder [1-LoE]
27:    if not anyChange then
28:        return NONDEDUCIBLE
```

Semantik von DEDUCE(G, *depth*)

Für jede ungeklärte Variable (bei *Akari*: Zelle (r, c) mit Kandidat) prüfe *beide* Hypothesen $X_{r,c}$=1 (Lampe) und $X_{r,c}$=0 (verboten), jeweils mit lokalem SIMPLIFY auf derselben Tiefe:

- **Shave:** genau eine Hypothese inkonsistent $\Rightarrow$ Gegenfall *erzwingen*.
- **Agreement:** beide konsistent $\Rightarrow$ nur identische Konsequenzen *übernehmen*.
- sind beide inkonsistent, dann CONTRADICTION.
- falls *depth* > 1, rekursiv auf der Hypothese fortsetzen, sonst nur lokaler Fixpunkt (Tiefe = 1).

2.3. CP–SAT-Modell (OR-Tools)

Modellidee

Das *Akari*-Rätsel wird als boolesches Machbarkeitsproblem in OR-Tools modelliert. Die in Abschnitt 2.1 definierten Constraints (2.1) bis (2.3) werden direkt in das CP–SAT-Modell überführt. Eine Zielfunktion ist nicht erforderlich, da jede konsistente Belegung eine gültige Lösung darstellt.

Variablen

Für jede weiße Zelle (r, c) wird eine Boolesche Variable $x_{r,c} \in \{0, 1\}$ erzeugt (1 = Lampe, 0 = leer). Schwarze und nummerierte Felder werden mit `AddEquality` auf 0 fixiert.

Constraint-Erzeugung

Das Modell entspricht formal den Gleichungen (2.1) bis (2.3) aus der Problemformulierung:

- **Run-Constraints** (2.1): Konfliktfreiheit innerhalb jeder Sichtlinie wird mit `AddLessOrEqual(LinearExpr::Sum(vis), 1)` umgesetzt. Dies verhindert mehrere Lampen in einer Zeile oder Spalte ohne Blocker.
- **Beleuchtungsbedingungen** (2.2): Für jedes weiße Feld (r, c) wird eine Summe über alle sichtbaren Felder $\mathrm{vis}(r, c)$ gebildet und per `AddGreaterOrEqual(LinearExpr::Sum(vis), 1)` als Mindestbedingung modelliert.
- **Hinweis-Constraints** (2.3): Orthogonale Nachbarn einer Hinweiszelle b werden zu einer Liste zusammengefasst und mit `AddEquality(LinearExpr::Sum(neigh), clue)` auf die vorgegebene Zahl k beschränkt.

Solver-Konfiguration

Der Solver verwendet die Standardparameter des OR-Tools-CP–SAT-Moduls. Zur Laufzeitbegrenzung ist `max_time_in_seconds = 10` gesetzt.
Logging ist deaktiviert (`set_log_search_progress(false)`), und der Rückgabestatus wird als `FEASIBLE` bzw. `OPTIMAL` (gelöst) oder `UNKNOWN` (Abbruch) interpretiert.

Auswertung

Nach erfolgreichem Lösen wird das Ergebnis aus der `CpSolverResponse` übernommen, in das Gitter eingetragen und die Beleuchtung automatisch aktualisiert. Für die Analyse werden Laufzeit und Suchstatistiken protokolliert (`WallTime`, `num_conflicts`, `num_branches`, `deterministic_time`).

Abgrenzung

Das CP–SAT-Modell arbeitet rein deklarativ und nutzt kein deduktives Preprocessing (vgl. DS und ACO in Kap. 2.1). Die Inferenz erfolgt ausschließlich über SAT-Propagation, also durch Unit Propagation und Constraint Filtering. Damit dient CP–SAT als neutrale Baseline für den Vergleich von Lösungszeit, Abdeckung und Robustheit.

2.4. Ant Colony Optimization (ACO)

2.4.1. Leitidee und Bezug zur Literatur

Das verwendete ACO-Framework folgt dem zweiphasigen Ansatz von Rosberg et al. für *Akari* [1] und orientiert sich konzeptionell am Standardmodell von Dorigo & Stützle [10]. Eine Kolonie von algorithmischen „Ameisen“ konstruiert Lösungen stochastisch. Die Auswahl wird durch Pheromone τ (erfahrungsbasiert) und eine statische Heuristik η (problembezogen) gesteuert. Die Bezeichnung ist biologisch inspiriert und lehnt sich an das Verhalten realer Ameisen an, die über Pheromonspuren erfolgreiche Wege markieren. Im Algorithmus stehen die „Ameisen“ jedoch für softwarebasierte Agenten, die Suchräume explorieren und gute Entscheidungen verstärken. Die Iteration endet, sobald eine fehlerfreie Belegung gefunden ist, d. h. es gibt keine unbeleuchteten weißen Felder und keine verletzten Hinweiszahlen mehr.
Als Gütemaß verwenden wir, wie in [1],

$$E(S) \;=\; \#\texttt{unlit}(S) \;+\; \#\texttt{violated\_clues}(S). \tag{2.13}$$

Eine Lösung ist genau dann gültig, wenn $E(S) = 0$.

Zweiphasiger Rahmen

1. *Preprocessing (Fixpunkt):* Die deterministische Rule-Engine (Kap. 2.1) reduziert zu Beginn die Kandidatenmenge (`allowedMask`).
2. *Koloniekonstruktion:* In jeder Iteration bauen mehrere Ameisen unabhängig voneinander Kandidatenlösungen auf. Am Ende verdunstet das Pheromon global und die beste Ameise der Iteration deponiert Pheromon auf ihren Lampenzellen.

2.4.2. Notation und Parameter

Wir verwenden:

- n_{ants}: Zahl der Ameisen pro Iteration.
- α, β: Exponenten für Pheromon bzw. Heuristik in (2.16).
- $\rho \in (0, 1)$: Verdunstungsrate (vgl. (2.17)).
- $Q > 0$: Einzahlungsfaktor beim Deposit (vgl. (2.18)).
- max_iters: Iterationsbudget (äquivalent zu einem Zeitbudget mit Stopkriterium).
- `groupMode`$\in$ {`row`, `col`, `both`}: Segmentierung in Gruppen.
- `fixpointMode`$\in$ {`none`, `during`}: Lokaler Fixpunkt nach Setzung.
- $\tau_{r,c}$: Pheromonwert der Zelle (r, c), kodiert Attraktivität aus guten Lösungen.
- $\eta_{r,c}$: Statische Heuristik nach (2.15), Gütewert der Zelle, unabhängig vom Pheromon.
- $\mathrm{Cand}(g)$: Erlaubte Zellen einer Gruppe g, $N_L(r, c)$: Benachbarte Hinweisfelder, $\mathrm{Lamps}(S_k)$: Lampenzellen in der von Ameise k konstruierten Lösung.
- **Seed** s: Initialisiert den Pseudozufallszahlengenerator (PRNG) für reproduzierbare Läufe. Pro Iteration it und Ameise k wird deterministisch ein Ant-Seed aus s abgeleitet. (Bei serieller Ausführung sind alle Ziehungen reproduzierbar.)

2.4.3. Referenzansatz nach Rosberg et al. (Baseline)

Intuition: ACO kombiniert schnelle lokale Entscheidungen mit einfachem Lernen über Iterationen. Regeln aus der Puzzlelogik reduzieren zuerst die Suche, danach setzen Ameisen in ausgewählten Gruppen jeweils eine Lampe, geführt von Pheromon (τ) und Heuristik (η). Gelingt ein fehlerfreier Durchlauf, stoppt der Prozess.

1. **Preprocessing (Fixpunkt):** Deterministische Regeln reduzieren zu Beginn die Kandidatenmenge auf erlaubte Zellen.

2. **Gruppenbildung:** Zerlege jede Zeile in zusammenhängende Weißsegmente (*Runs*).
   Eine Gruppe g der Länge n_g wird überhaupt aktiviert mit Wahrscheinlichkeit

$$P_g \;=\; 1 \;-\; \frac{1}{n_g + 1} \tag{2.14}$$

   Dadurch werden sehr kurze Segmente seltener ausgewählt, frühe Streuung statt Ballung.

3. **Heuristik:** Für eine erlaubte Zelle (r, c) gilt

$$\eta_{r,c} \;=\; 1 \;+\; \sum_{b \in N_L(r,c)} \mathrm{value}(b) \tag{2.15}$$

   Nähe zu „hungrigen“ Hinweisen macht eine Zelle attraktiver. Verbotene Felder und Lampen haben $\eta = 0$.

4. **Auswahl in aktiven Gruppen:** In einer aktivierten Gruppe g wählt die Ameise genau *eine* erlaubte Zelle proportional zu

$$p_{r,c} \;=\; \frac{\tau_{r,c}^{\alpha}\, \eta_{r,c}^{\beta}}{\displaystyle\sum_{z \in \mathrm{Cand}(g)} \tau_z^{\alpha}\, \eta_z^{\beta}} \tag{2.16}$$

   und setzt dort eine Lampe, sofern konfliktfrei. So verbinden sich Erfahrung (τ) und Strukturhinweis (η).

5. **Bewertung und Abbruch:** Nach der Gruppenrunde wird $E(S)$ gemäß (2.13) berechnet.
   Gilt $E = 0$, wird sofort terminiert, gültige Belegung gefunden.

6. **Lernen über Iterationen:** Andernfalls folgt die globale Verdunstung

$$\tau_{r,c} \;\leftarrow\; (1 - \rho)\, \tau_{r,c} \qquad \text{für alle erlaubten } (r, c) \tag{2.17}$$

   und die iterationsbeste(n) Ameise(n) deponieren Pheromon mit $E^\star = \min_k E(S_k)$ und $\mathcal{A}^\star = \{k \mid E(S_k) = E^\star\}$ auf ihren Lampenzellen:

$$\tau_{r,c} \;\leftarrow\; \tau_{r,c} \;+\; \sum_{\substack{k \in \mathcal{A}^\star \\ (r,c) \in \mathrm{Lamps}(S_k)}} \frac{Q}{E^\star} \tag{2.18}$$

   Die Bedingung unter dem Summenzeichen bedeutet: addiert wird nur, wenn (r, c) von Ameise k als Lampe gesetzt wurde. Dadurch werden erfolgversprechende Entscheidungen wahrscheinlicher.

2.4.4. Erweiterungen gegenüber dem Paper

Gruppenmodus `row/col/both`:

Rosberg et al. verwenden nur Zeilengruppen. Wir ergänzen Spaltengruppen sowie einen kombinierten Modus `both`. Seien $\mathcal{G}_{\text{row}}$ die zusammenhängenden Weißsegmente je Zeile und $\mathcal{G}_{\text{col}}$ die analogen Segmente je Spalte. Die in einer Iteration verarbeitete Gruppenfolge ist

$$\Pi = \begin{cases} \text{perm}(\mathcal{G}_{\text{row}}) & \texttt{row}, \\ \text{perm}(\mathcal{G}_{\text{col}}) & \texttt{col}, \\ \text{perm}(\mathcal{G}_{\text{row}} \cup \mathcal{G}_{\text{col}}) & \texttt{both}, \end{cases} \tag{2.19}$$

wobei $\text{perm}(\cdot)$ eine zufällige Permutation bezeichnet. Die Aktivierung einer Gruppe erfolgt weiterhin gemäß Gl. 2.14. Alles andere bleibt unverändert (Kandidaten, Heuristik, Auswahlregel, Verdunstung, Deposit; vgl. Gl. 2.15, 2.16, 2.17, 2.18).

Hybrid-ACO mit lokalem Fixpunkt:

Hybrid-ACO kombiniert den stochastischen ACO-Aufbau mit deterministischer Logikpropagation. Nach jeder Lampensetzung wird im betroffenen Umfeld ein lokaler Fixpunkt ausgeführt (`fixpointMode=during`). Dieser Schritt wendet die Regeln (vgl. 2.1.1) sofort an, leitet erzwungene Setzungen und Verbote früh ab und entfernt inkonsistente Teilzustände. Dadurch wird die Suche fokussiert, Widersprüche werden unmittelbar sichtbar und fehlgeschlagene Konstruktionen werden früh abgebrochen. Beispiel: Setzt eine Ameise eine Lampe so, dass ein benachbartes Hinweisfeld dadurch nicht mehr erfüllbar wäre, wird dies sofort erkannt und die aktuelle Konstruktion verworfen. Ohne Interleaving (`fixpointMode=none`) entspricht der Ablauf der Rosberg-Baseline.

2.4.5. Pseudocode (vereinfacht)

Algorithm 2 ACO für *Akari* (vereinfacht, Iterationsbudget)

Require: Grid G; Parameter $(n_{\text{ants}}, \alpha, \beta, \rho, Q, \text{max\_iters})$; `groupMode`$\in\{$`row`, `col`, `both`$\}$; `fixpointMode`$\in\{$`none`, `during`$\}$

preprocess: RUNFIXPOINT(G) ▷ globale Kandidatenreduktion
Initialisiere $\tau_{r,c} \leftarrow 1$ auf erlaubten Feldern; berechne η via (2.15)
for $it = 1$ **to** max_iters **do**
  $E^\star \leftarrow +\infty$; $\mathcal{A}^\star \leftarrow \emptyset$; $G^\star \leftarrow$ `null`
  for $k = 1$ **to** n_{ants} **do**
    $G' \leftarrow$ Kopie von G; $\Pi \leftarrow$PERMUTEGROUPS(G', `groupMode`)
    for jede Gruppe $g \in \Pi$ **do**
      $C \leftarrow \mathrm{Cand}(g)$ ▷ aktuelle erlaubte Zellen der Gruppe
      if $C = \emptyset$ **then**
        continue
      if ACTIVATED($g; P_g$) ▷ P_g gemäß (2.14) **then**
        wähle $z \in C$ mit p_z gemäß (2.16); PLACELAMP(G', z)
        if `fixpointMode=during` **then**
          RUNFIXPOINTLOCAL(G', z)
    $E \leftarrow \#\texttt{unlit}(G') + \#\texttt{violated\_clues}(G')$ ▷ vgl. (2.13)
    if $E = 0$ **then**
      return G'
    $S_k \leftarrow G'$ ▷ Lösung der k-ten Ameise merken
    if $E < E^\star$ **then**
      $E^\star \leftarrow E$; $G^\star \leftarrow G'$; $\mathcal{A}^\star \leftarrow \{k\}$
    else if $E = E^\star$ **then**
      $\mathcal{A}^\star \leftarrow \mathcal{A}^\star \cup \{k\}$
  EVAPORATE(τ, ρ) ▷ vgl. (2.17)
  for jede Ameise $k \in \mathcal{A}^\star$ **do**
    DEPOSIT($\tau, \mathrm{Lamps}(S_k), Q/E^\star$) ▷ vgl. (2.18)
return bestes beobachtetes Gitter $G^\star$

- Fixpunkt-Preprocessing reduziert Kandidaten. Setze $\tau = 1$ und berechne η.
- Pro Iteration arbeitet jede Ameise auf einer Kopie G'. Die Gruppen werden gemäß `groupMode` permutiert.
- Pro Gruppe: Wenn $\mathrm{Cand}(g) = \emptyset$, wird sie übersprungen. Andernfalls wird sie mit Wahrscheinlichkeit P_g aktiviert. Bei Aktivierung wird eine Zelle nach (2.16) gewählt und als Lampe gesetzt. Bei `fixpointMode=during` folgt ein lokaler Fixpunkt.
- Nach allen Gruppen werden die Kosten E berechnet. Bei $E{=}0$ wird die Lösung sofort zurückgegeben. Andernfalls werden $E^\star$, $G^\star$ und $\mathcal{A}^\star$ aktualisiert.
- Am Iterationsende erfolgt die Verdunstung (2.17). Die Ameisen aus $\mathcal{A}^\star$ zahlen auf ihren Lampenzellen jeweils $Q/E^\star$ ein (2.18).

3. Ergebnisse

3.1. Versuchsrahmen (Kurzüberblick)

Dieses Kapitel berichtet die Resultate der drei Ansätze auf einer gemeinsamen Benchmark-Sammlung und wertet sie mit einheitlichen Metriken aus. Alle Laufzeiten sind Wandzeiten und, sofern nicht anders angegeben, in Sekunden (**s**) angegeben. Sämtliche Messungen erfolgten auf identischer Hardware (Intel Core i5-12400F, 32 GB RAM). Stochastische Verfahren werden über mehrere unabhängige Läufe aggregiert; angegeben werden robuste Lage- und Streuemaße (Median, IQR). Läufe, die das globale Zeitlimit erreichen, gelten als *zensiert* und gehen in Zeit-Erfolgs-Kurven entsprechend ein (die Kurven steigen dann nicht bis 1).

Korpus und Größenklassen

Der Datensatz umfasst **1 345** Akari-Instanzen aus sieben Quellen. Für eine einheitliche Darstellung ordnen wir alle Instanzen nach der längeren Kante $L = \max(H, W)$ in Größen-Buckets ein: ≤ 10, 11–15, 16–20, 21–25, > 25.[1]

Tabelle 3.1.: Instanzüberblick nach Quelle und Größen-Buckets. Vollständige Quellenangaben siehe Anhang A.

Quelle	≤10	11–15	16–20	21–25	>25	N gesamt
janko_akari [A]	230	80	361	181	98	950
puzzle-light-up.com [B]	5	5	0	4	0	14
minijuegos [C]	10	0	0	0	0	10
chiark.greenend.org.uk [D]	0	0	5	0	0	5
Marugoto [E]	21	0	33	37	21	112
Penpa [F]	84	0	49	0	3	136
Puzzlebox [G]	83	1	34	0	0	118
Summe	**433**	**86**	**482**	**224**	**120**	**1345**

Vergleichbarkeit

Alle Verfahren laufen auf derselben Maschine unter identischen Zeit- und Speicherbudgets. Preprocessing erfolgt gemäß Abschnitt 2.1.1. Deductive Search und ACO starten vom Fixpunkt der Rule-Engine. CP–SAT arbeitet ohne domänenspezifische Preprocessing direkt auf dem Modell. Die Quellenverteilung ist in Tab. 3.1 ersichtlich und Querschnittsvergleiche werden über Größen-Buckets kontrolliert.

[1] Eine Einteilung nach Fläche oder Zahl weißer Zellen wäre ebenfalls möglich; hier werden L-Buckets konsequent für alle Abbildungen/Tabellen verwendet.

3.2. Deductive Search (DS)

Dieser Abschnitt zeigt die Resultate des regelbasierten Solvers (*Deductive Search*). Standardmäßig verwenden wir den produktiven *Dirty-Queue*-Modus, zum Vergleich ist der vollständige Scan (*full*) ausgewiesen. Die Auswertung folgt den zuvor definierten Größen-Buckets nach der längeren Gitterkante $L = \max(H, W)$.

3.2.1. Lösungsabdeckung und Laufzeit

Der DS-Solver löst den gesamten Instanzkorpus (N=1345) vollständig. Tabelle 3.2 zeigt die Lösungsabdeckung sowie Median und Interquartilsabstand (IQR) der Wandzeiten pro Größen-Bucket für beide Betriebsarten. Sondergrößen einzelner großer Instanzen sind in Tabelle 3.3 separat aufgeführt.

Tabelle 3.2.: Deductive Search: Lösungsrate und Wandzeit je Größen-Bucket (nach $\max(W, H)$). Zeiten in s.

Bucket	% gelöst	Median (Dirty) [s]	IQR (Dirty) [s]	Median (Full) [s]	N
< 10	100.00	0.001	0.000	0.003	36
10	100.00	0.001	0.001	0.002	397
11–15	100.00	0.002	0.005	0.003	86
16–20	100.00	0.005	0.010	0.007	482
21–25	100.00	0.005	0.011	0.007	222
26–36	100.00	0.025	0.087	0.041	107
37–55	100.00	0.177	0.851	0.347	12
Gesamt*					**1342**

* ohne Sondergrößen, vgl. Tab. 3.3.

Tabelle 3.3.: Deductive Search: Einzelwerte für drei größte Instanzen. Zeiten in s.

Größe	% gelöst	Zeit (Dirty)	Zeit (Full)	N
50×50	100.00	0.878	4.151	1
50×80	100.00	25.014	90.956	1
100×100	100.00	45.473	160.654	1
Gesamt				**3**

Hinweis: Spalten zeigen den Anteil gelöster Instanzen (%), Median- und Streuwerte der Wandzeiten (IQR = Q_{75} - Q_{25}). Alle Instanzen werden erfolgreich gelöst und das mit einer maximalen LoE-Stufe von 1. Mit wachsender Gittergröße steigt die Laufzeit mit wachsender Gittergröße an.

3.2.2. Skalierung mit der Instanzgröße

Für *Deductive Search* bestimmt in erster Linie die Anzahl der *weißen Zellen* die effektive Problemgröße. Jede weiße Zelle entspricht einer Booleschen Variablen mit zugehörigen Sichtlinien, entlang derer Propagation und Hypothesenbildung erfolgen. Die reine Gittergröße $W \times H$ ist daher weniger aussagekräftig, da unterschiedliche Blockdichten sehr verschieden viele Weißfelder erzeugen. Aus diesem Grund wird die Wandzeit in Abhängigkeit von der Anzahl der weißen Zellen dargestellt, beide Achsen in logarithmischer Skalierung.

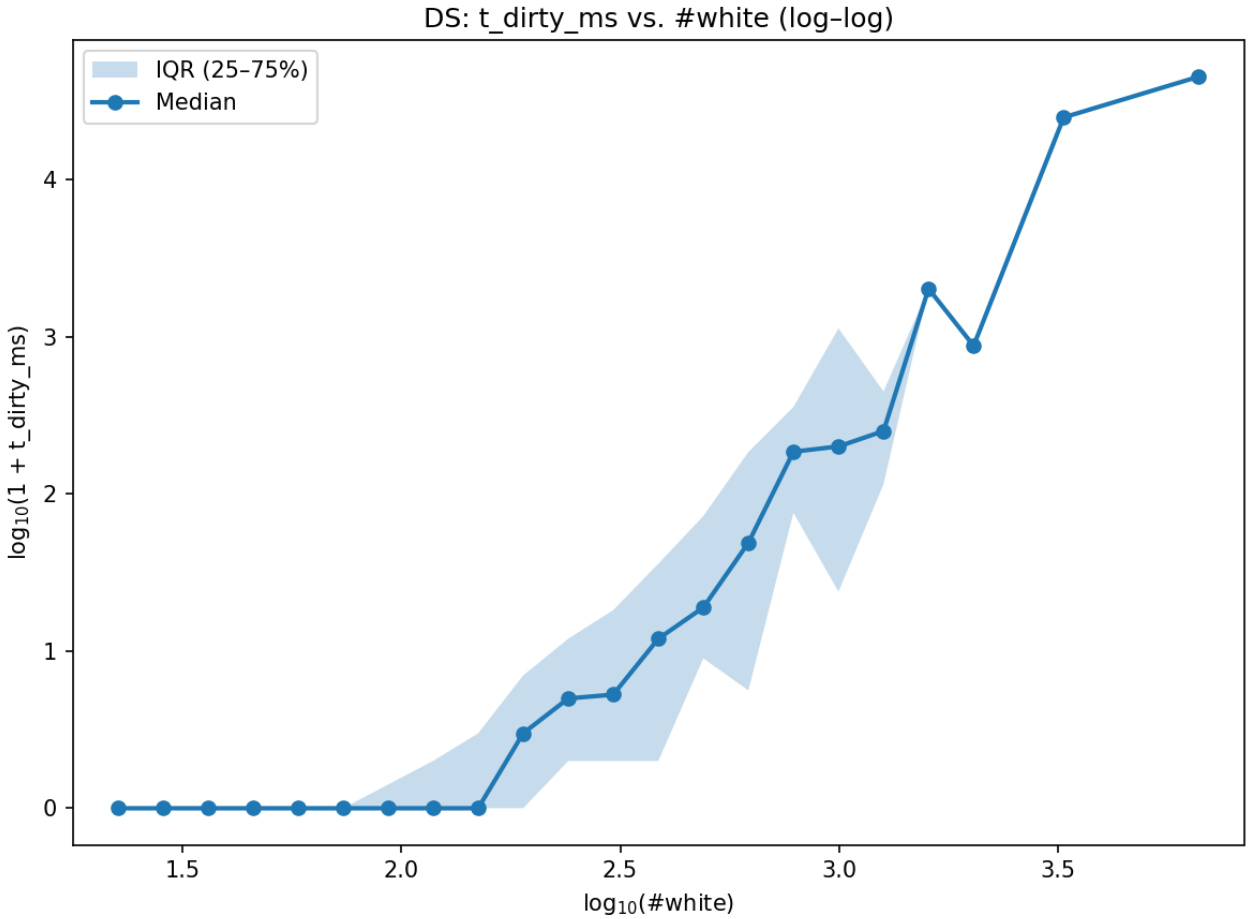


Abbildung 3.1.: Deductive Search (Dirty-Modus): Median (Linie) und Interquartilsabstand (IQR, schattierte Fläche) der Wandzeiten in Abhängigkeit von $\log_{10}(\#\text{white})$.

Abbildung 3.1 zeigt, dass die Medianwerte mit wachsender Anzahl weißer Zellen deutlich ansteigen und dass die Streuung (IQR) ebenfalls zunimmt. Der Zeitboden nahe null Sekunden entspricht Instanzen, die bereits im *0-LoE*-Fixpunkt gelöst werden und keine Hypothesenbildung erfordern. Mit wachsender Instanzgröße treten sowohl sehr schnelle Fälle (vollständige Lösung im Fixpunkt) als auch deutlich langsamere auf, die zusätzliche *1-LoE*-Shave- oder Agreement-Pässe benötigen. Für die größten Instanzen entfällt das IQR-Band, da dort nur wenige Datenpunkte vorliegen.

3.2.3. Schwierigkeit: Abgleich mit externen Labels

Die interne *DS-Klasse* entsteht aus der in Abschnitt 2.2.4 beschriebenen Zählung der Regel-Updates und deren Aggregation nach Gleichung (2.9) bis (2.12). Ziel ist eine größenunabhängige, regelbasierte Einordnung nach dem *logischen Aufwand* der Lösung.

Gewichtungen und Faktoren

Für die Fixpunktregeln wurden in allen Experimenten folgende Gewichte verwendet:

$$W_{\text{forceNeighbors}} = 2.0, \quad W_{\text{singleton}} = 5.0, \quad W_{\text{pruneClue}} = 1.0, \quad W_{\text{prune0}} = 1.0, \quad W_{\text{prune3}} = 3.0.$$

Der resultierende Schwierigkeitswert berücksichtigt zusätzlich einen *LoE-Bonus* mit $\lambda = 0.25$ nach Gleichung (2.10) sowie einen *Effort-Term* mit $\mu = 0.25$ nach (2.12).

Vergleich mit Janko-Labels ($n = 950$)

Als externe Referenz dienen die Herausgeberstufen von *janko.at* (*leicht*, Stufen 1–8, *schwer*). Für die Analyse werden diese auf eine numerische Skala von 0–9 abgebildet. Die DS-Klassen sind bewusst größenunabhängig kalibriert und leiten sich deterministisch aus dem logischen Ableitungsaufwand ab (Fixpunkt-Updates, Shave- und Agreement-Pässe). Die Janko-Labels spiegeln dagegen eine erwartete Lösedauer bzw. subjektive Einschätzung wider und sind damit indirekt von der Instanzgröße beeinflusst: Größere Gitter erfordern meist mehr Zeit, ohne notwendigerweise komplexere logische Schritte zu beinhalten. Das DS-Maß zielt daher auf eine algorithmische, regelbasierte Annäherung an den menschlichen *Denkaufwand* und nicht an die reine Laufzeit. Eine exakte Übereinstimmung beider Skalen ist somit nicht zu erwarten. Im Folgenden werden die Verteilungen beider Skalen gegenübergestellt.

Tabelle 3.4.: Verteilung der Schwierigkeitswerte (0–9): Janko-Label (JD) versus DS-Klasse.

	Janko (JD)		**DS-Klasse**	
Wert	**Anzahl**	**Anteil [%]**	**Anzahl**	**Anteil [%]**
0	158	16.63	19	2.00
1	166	17.47	47	4.95
2	171	18.00	60	6.32
3	221	23.26	20	2.11
4	143	15.05	49	5.16
5	61	6.42	139	14.63
6	11	1.16	193	20.32
7	7	0.74	194	20.42
8	10	1.05	168	17.68
9	2	0.21	61	6.42
Gesamt	**950**	**100.00**	**950**	**100.00**

Verteilung und Kalibrierung

Die Gegenüberstellung in Tabelle 3.4 zeigt eine Verschiebung der Masse in die mittleren bis höheren DS-Klassen, insbesondere 6–8, während die unteren Klassen unterrepräsentiert sind. Dies entspricht der Zielsetzung einer größenunabhängigen, regelbasierten Skala des *logischen Aufwands*. Im gesamten Datensatz erreicht keine Instanz mit $\mathrm{LoE} = 0$ eine DS-Klasse oberhalb von 3. Höhere Klassen treten in der Regel erst bei Instanzen mit $\mathrm{LoE} \geq 1$ auf.

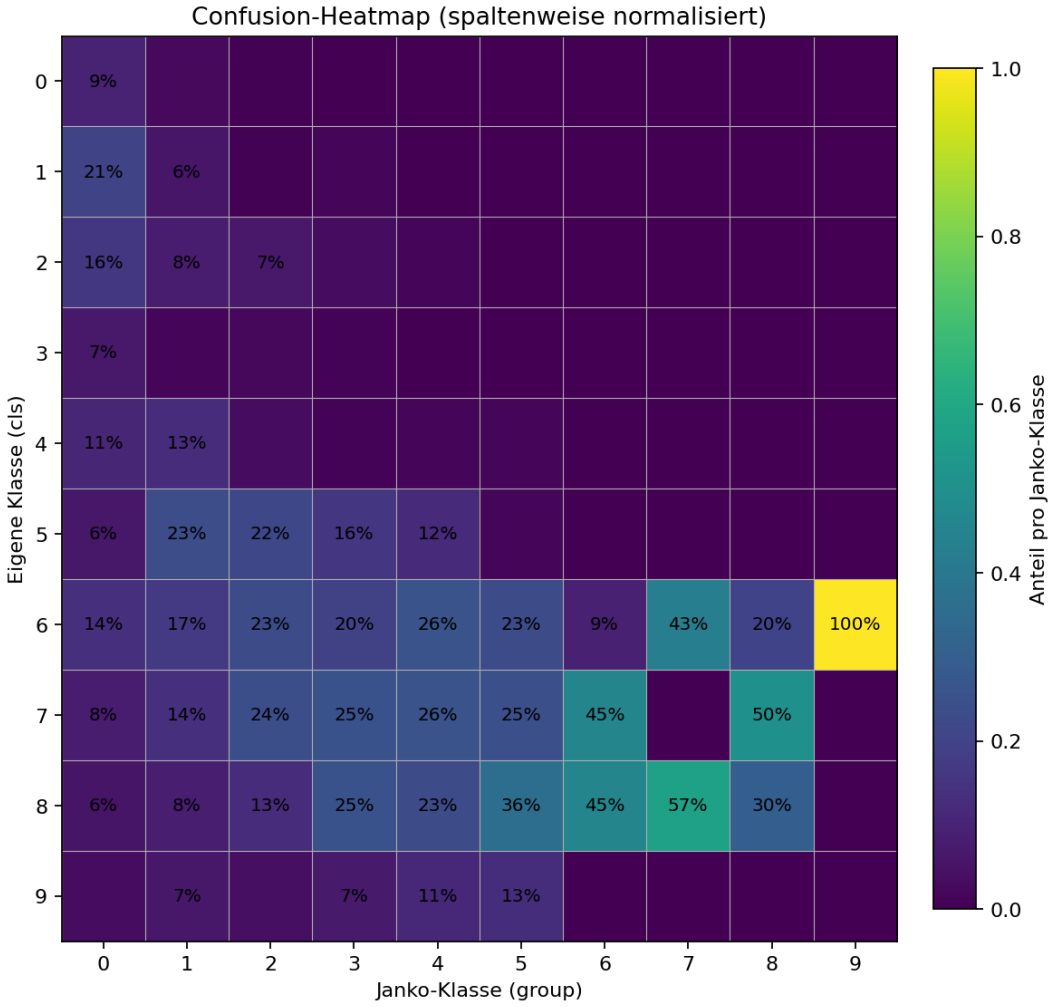


Abbildung 3.2.: Konfusions-Heatmap: Anteil der DS-Klassen je Janko-Klasse spaltenweise normalisiert.

Abbildung 3.2 zeigt die spaltenweise normalisierte Zuordnung zwischen den Herausgeberstufen von *Janko* auf der x-Achse und den algorithmisch bestimmten DS-Klassen auf der y-Achse. Jede Spalte summiert zu eins und beschreibt die Verteilung einer Janko-Stufe über die DS-Skala. Für niedrige Janko-Stufen von 0 bis 2 ergibt sich eine breite Streuung über mehrere DS-Klassen. Ab etwa Janko 3 konzentriert sich die Masse zunehmend auf die Klassen 5 bis 8. Janko 7 und 8 werden überwiegend als DS 6 bis 8 eingeordnet. Eine Besonderheit sind die beiden größten Puzzles im Korpus mit Janko 9, die durch DS der Klasse 6 zugeordnet werden. Das zeigt, dass DS nicht die Gittergröße oder Laufzeit bewertet, sondern den erforderlichen *logischen Ableitungsaufwand*.

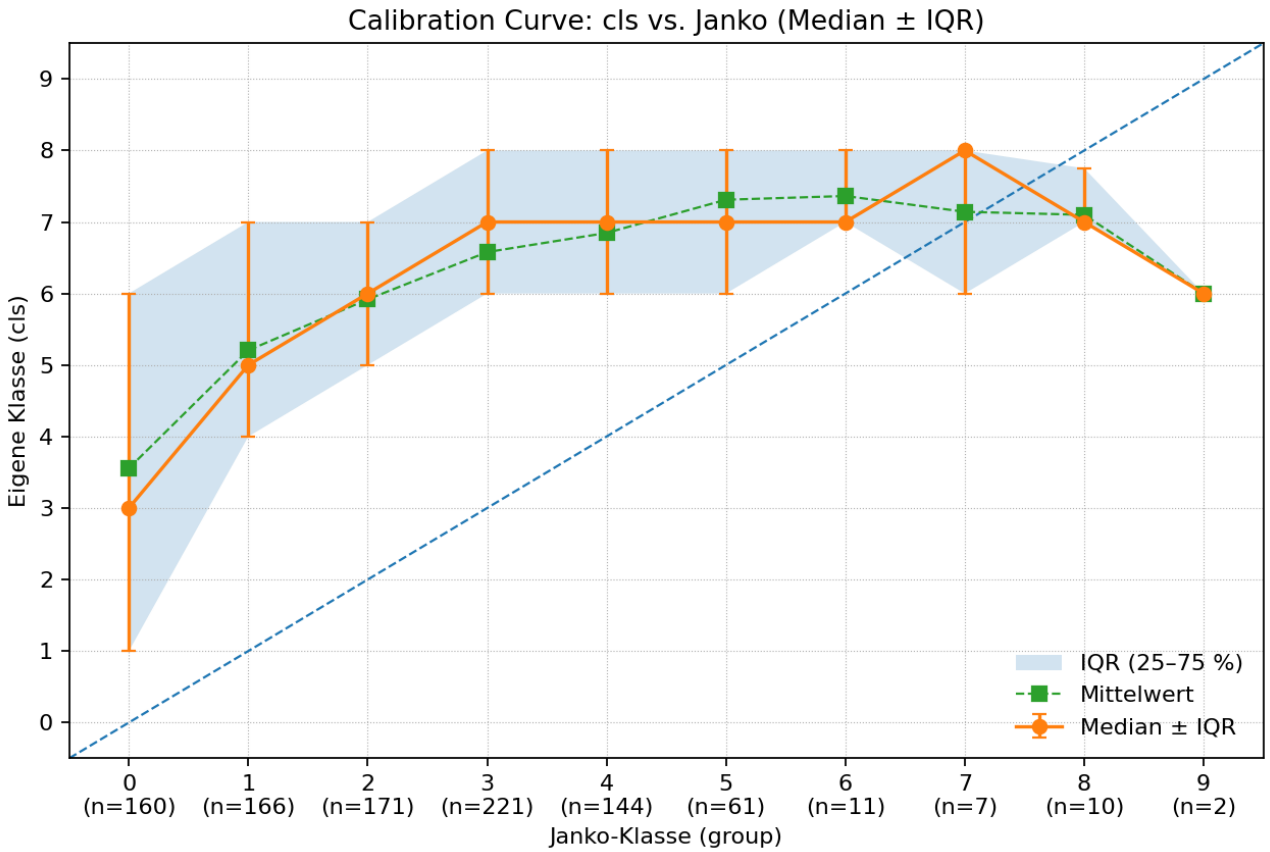


Abbildung 3.3.: Kalibrationskurve: DS-Klasse vs. Janko-Klasse (Median ± IQR, gruppiert). Die gestrichelte Diagonale kennzeichnet ideale Übereinstimmung.

Abbildung 3.3 zeigt den Zusammenhang zwischen den redaktionellen Janko-Stufen (x-Achse) und den algorithmisch bestimmten DS-Klassen (y-Achse). Die Medianwerte steigen bis etwa Janko 7 nahezu monoton an und liegen überwiegend oberhalb der Diagonalen $y = x$. Damit bewertet DS den logischen Aufwand in diesen Bereichen tendenziell höher, als es die Herausgeberstufe nahelegt. Ab Janko 8 flacht die Kurve ab und verläuft leicht unterhalb der Diagonalen (Janko 8 $\rightarrow$ DS $\approx$ 7, Janko 9 $\rightarrow$ DS $\approx$ 6). Dies weist darauf hin, dass bei sehr großen Instanzen die Janko-Stufen weiter ansteigen, während der deduktive Aufwand im DS-Sinn nicht mehr zunimmt.
Die hellblauen IQR-Bänder (25–75 %) zeigen die Streuung innerhalb der Janko-Gruppen. In den niedrigen Stufen ist sie am größten, was auf eine Mischung aus $\mathrm{LoE}{=}0$- und $\mathrm{LoE}{\geq}1$-Fällen hinweist. In den mittleren Stufen verengt sich die Verteilung deutlich, während sie in den oberen Stufen aufgrund kleiner Fallzahlen n nur eingeschränkt aussagekräftig ist.

Hinweis zur Kalibrierung: Die Lage und Steigung der Kurve hängen von den gewählten Parametern der DS-Metrik ab, insbesondere von den Gewichten der Fixpunktregeln $W_\cdot$ sowie den Faktoren für LoE-Bonus λ und Effort-Term μ (vgl. die oben genannten Standardwerte).

Vergleich mit weiteren Labels

Neben den Janko-Stufen werden exemplarisch zwei weitere Quellen mit redaktionellen Schwierigkeitsangaben betrachtet. Tabelle 3.5 zeigt einen Detailauszug, nach Gittergröße sortiert und innerhalb jeder Größe von *easy* über *normal* bis *hard* geordnet.

Tabelle 3.5.: Externe Schwierigkeitslabels (easy / normal / hard) im Vergleich zur DS-Klasse, sortiert nach Größe und Label. Quellen: (B) puzzle-light-up.com, (D) Portable Puzzle Collection (Light Up). Siehe Anhang A.

ID	Größe	Externes Label	DS-Klasse	Quelle
677565	10×10	easy	1	B
4147326	10×10	easy	2	B
9116291	10×10	normal	5	B
1738340	10×10	normal	6	B
7872871	10×10	hard	6	B
1602558	14×14	easy	1	B
9366021	14×14	easy	2	B
898033	14×14	normal	5	B
5183645	14×14	normal	7	B
912406	14×14	hard	5	B
20x20d	20×20	easy	5	D
20x20e	20×20	normal	6	D
20x20b	20×20	hard	6	D
20x20c	20×20	hard	6	D
9855822	25×25	easy	3	B
6595195	25×25	normal	6	B
814847	25×25	hard	7	B
1664040	25×25	hard	7	B

Die Quelle (B) nennt keine expliziten Kriterien für ihre Schwierigkeitslabels. Für Quelle (D) heißt es in der offiziellen Dokumentation[2]: „*'Easy' means that the puzzles should be soluble without backtracking or guessing, 'Hard' means that some guesses will probably be necessary.*“ Ob die Einstufungen aus Heuristiken, einem regelbasierten Solver oder reinem Backtracking stammen, bleibt offen. Über die Größen hinweg zeigt sich dennoch ein konsistentes Muster. Innerhalb einer Gittergröße steigen die DS-Klassen typischerweise von *easy* über *normal* zu *hard*, etwa bei 10×10 (1–2 → 5–6 → 6). Einzelne Abweichungen treten auf, zum Beispiel 14×14 mit *hard* = 5 unterhalb von *normal* = 7 oder 20×20 mit *easy* = 5. Die Stichproben pro Größe sind klein, daher sind die Befunde deskriptiv zu verstehen.

[2] https://www.chiark.greenend.org.uk/~sgtatham/puzzles/doc/lightup.html

Größenabhängigkeit der Schwierigkeit

Abbildung 3.4 zeigt die Verteilung der *weißen Felder* (Gitterzellen ohne Blöcke) je DS-Klassen-Bucket. Die x-Achse entspricht der DS-Klasse, die y-Achse ist logarithmisch und gibt die Anzahl weißer Zellen pro Instanz an.

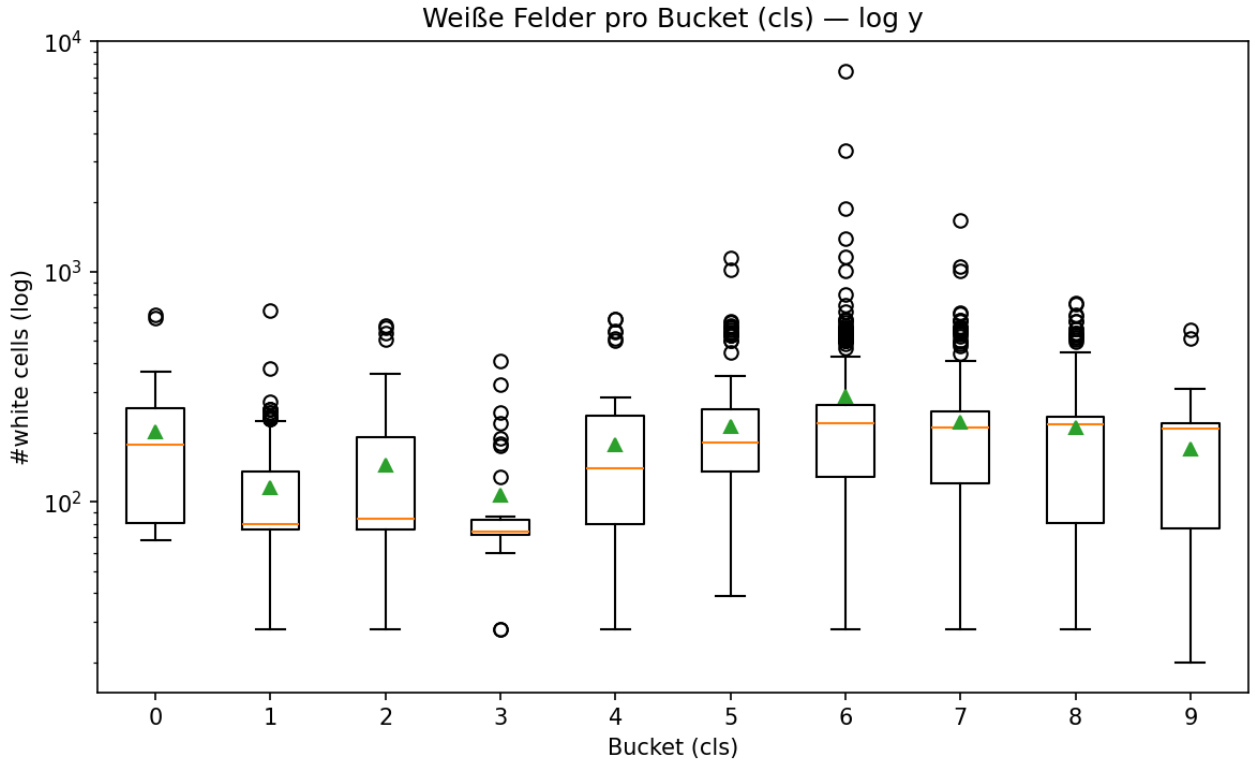


Abbildung 3.4.: Weiße Felder pro DS-Klassen-Bucket (logarithmische y-Achse). Box = Interquartilsbereich (IQR, 25–75 %), orange Linie = Median, grünes Dreieck = Mittelwert, Whisker = Tukey-Bereich bis $1.5 \times$ IQR, darüber hinausliegende Punkte markieren Ausreißer.

Lesart der Grafik

Jede Box fasst die Verteilung innerhalb einer Klasse zusammen. Die Boxen zeigen den IQR, die orange Linie den Median und das grüne Dreieck den arithmetischen Mittelwert. Die Whisker reichen bis zum $1.5 \times$ IQR-Bereich, darüber hinausliegende Kreise kennzeichnen einzelne Ausreißer. Durch die logarithmische Skalierung werden sehr kleine und sehr große Gitter vergleichbar dargestellt.

Deskriptive Beobachtungen

Über die DS-Klassen hinweg liegen die Medianwerte überwiegend auf ähnlichem Niveau, und die IQR-Bereiche überlappen stark. In den Klassen 6–8 treten vermehrt große Ausreißer auf, die einzelnen Instanzen mit vielen weißen Feldern entsprechen. Dies betrifft vor allem die größten Gitter im Datensatz (zum Beispiel 50×80 und 100×100), die als Ausreißer sichtbar werden. Der globale Median über alle Instanzen beträgt $\tilde{w} = 162$ weiße Felder, der Mittelwert $\bar{w} = 206$.

3.3. Constraint Programming (CP–SAT)

Dieser Abschnitt beschreibt die Resultate des OR-Tools CP–SAT-Solvers auf dem vollständigen Datensatz (N=1345; Quellen siehe Anhang A). Alle Zeiten sind Wandzeiten in Millisekunden. Die Auswertung verwendet dieselben Größen-Buckets wie im Abschnitt zu *Deductive Search*. Die drei größten Gitter 50×50, 50×80 und 100×100 werden als Sondergrößen separat betrachtet.

3.3.1. Lösungsabdeckung und Laufzeit

Tabelle 3.6.: CP–SAT: Laufzeit pro Größen-Bucket sowie Gesamtwerte.

Bucket	N	Gelöst [%]	Median t [ms]	P_{90} [ms]
< 10	36	100.00	29	37
10	397	100.00	29	33
11–15	86	100.00	31	37
16–20	482	100.00	32	39
21–25	222	100.00	33	40
26–36	107	100.00	40	48
37–55	12	100.00	49	63
Gesamt(inkl. Sondergrößen)	**1345**	**100.00**	**32,0**	**40,0**

In Tabelle 3.6 sind die Laufzeiten pro Größen-Bucket sowie die Gesamtwerte zusammengefasst. Alle Instanzen konnten gelöst werden. Mit wachsender Gittergröße steigen die Laufzeiten moderat an. Die Mediane reichen von etwa 29 ms bei kleinen Instanzen bis etwa 49 ms im größten Bucket.

3.3.2. Skalierung nach Größe

Tabelle 3.7.: CP–SAT: Größte Einzelinstanzen.

Größe	N	Zeit t [ms]
50×50	1	64
50×80	1	108
100×100	1	221

Tabelle 3.7 listet die größten Einzelinstanzen. Für diese steigt die Laufzeit überproportional und erreicht bis zu rund 220 ms beim 100×100-Gitter, bleibt aber klar unter einer Sekunde.

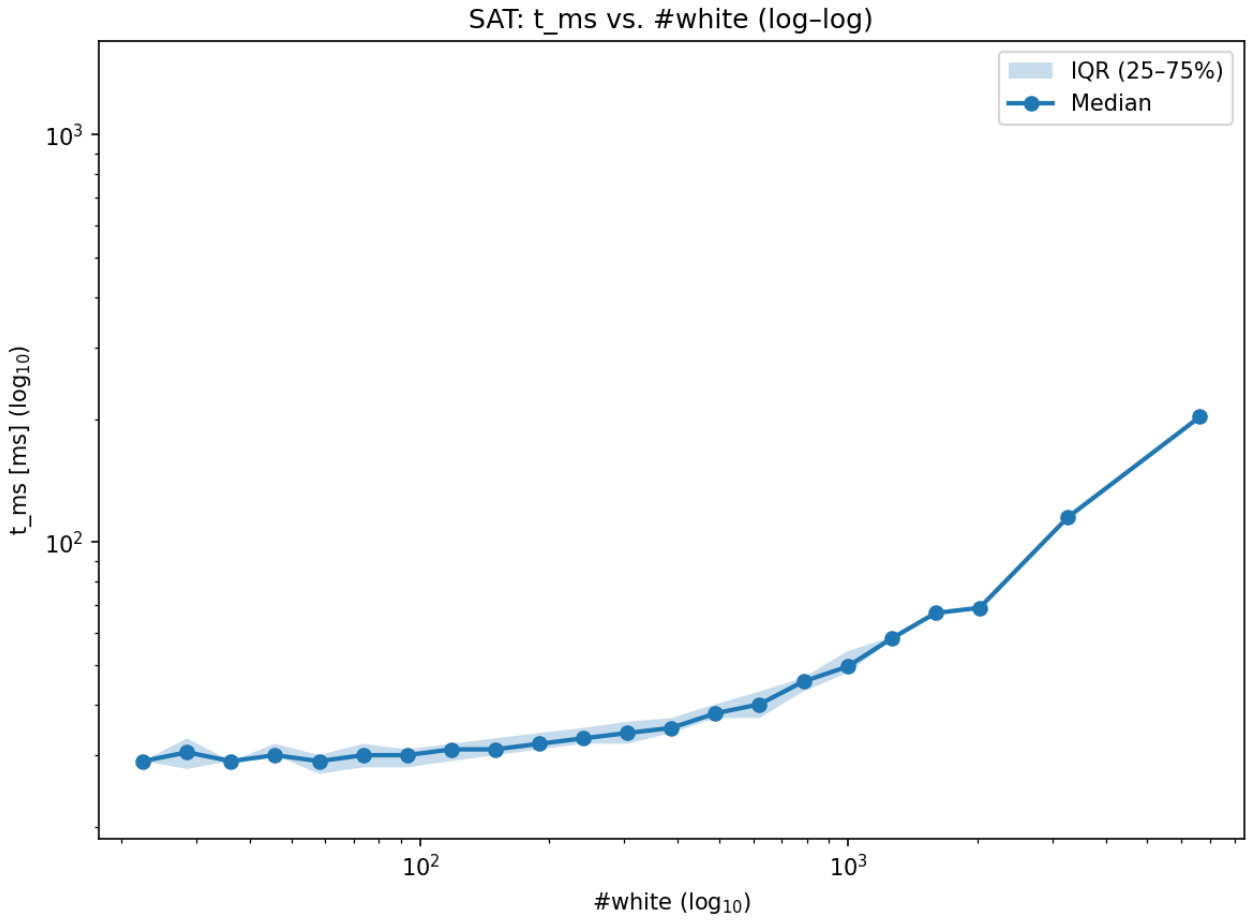


Abbildung 3.5.: CP–SAT: Median (Linie) und Interquartilsabstand (IQR, schattierte Fläche) der Wandzeiten t_{ms} in Abhängigkeit von $\log_{10}(\#\text{white})$ bei logarithmischer Skalierung beider Achsen.

Abbildung 3.5 zeigt eine weitgehend gleichmäßige und nahezu lineare Zunahme der Wandzeiten mit wachsender Anzahl weißer Zellen. Über den größten Teil der Größenklassen bleibt die Streuung gering und nimmt erst bei sehr großen $\#\text{white}$ leicht zu. In den äußersten Größenklassen fehlt das IQR-Band, da dort nur wenige Datenpunkte vorliegen.

3.4. Ant Colony Optimization (ACO)

Dieser Abschnitt beschreibt die Resultate des ACO-Solvers auf der vollständigen Benchmark-Sammlung. Die Implementierung folgt eng der Variante von Rosberg et al. für *Light Up* (zeilenorientierte Konstruktion mit kombinierter Pheromon- und Heuristikkomponente), integriert jedoch zusätzlich eigene Erweiterungen, wie in Abschnitt 2.4.4 beschrieben sind. Dadurch ist ein direkter, aber zugleich erweiterter Vergleich mit der Literatur möglich.[3]

3.4.1. Verwendete Parameter

Die Parameter wurden entsprechend der Publikation von Rosberg et al. übernommen. Die Autoren geben an: *"The parameters were set in preliminary experiments."* Damit orientiert sich die vorliegende Implementierung vollständig an den empfohlenen Standardwerten der Originalarbeit (Tab. 3.8). Eine eigenständige Parameteranalyse wäre methodisch aufwendig und würde den Rahmen dieser Arbeit überschreiten.

Tabelle 3.8.: Parameter des ACO-Algorithmus nach [1], Tab. II.

Größe	**#Ameisen**	**#max_Iter**	Q	ρ
7×7	10	200	1	0.4
10×10	1000	1000	5	0.4
14×14	1000	1000	8	0.1
20×20	1000	10 000	20	0.1
25×25	1000	100 000	20	0.1
≥40×30	1000	100 000	50	0.1

[3] Parameterwahl und Grundaufbau nach [1].

3.4.2. Ohne Preprocessing: Lösungsabdeckung und Konvergenz

Tabelle 3.9.: ACO ohne Preprocessing: Laufzeiten, Iterationen und Konvergenzraten pro Instanz für *Rows*, *Cols* und *Both*. Quellen: (B) Puzzle-Light-Up, (C) Minijuegos.

			Rows			**Cols**			**Both**			**Quelle**
ID	**Größe**	**Typ**	T [s]	#it	rate	T [s]	#it	rate	T [s]	#it	rate	
1	7×7	–	0.0	3.0	90	0.0	12.4	100	0.0	2.2	100	(C)
2	7×7	–	0.0	8.1	100	0.0	13.2	50	0.0	4.7	100	(C)
3	7×7	–	0.0	15.3	100	0.0	9.4	70	0.0	6.2	100	(C)
4	7×7	–	0.0	133.0	40	0.0	9.0	10	0.0	8.7	100	(C)
5	7×7	–	0.0	30.6	100	0.0	14.1	100	0.0	13.2	100	(C)
6	7×7	–	0.0	56.1	100	0.0	32.4	100	0.0	5.5	100	(C)
7	7×7	–	0.0	9.9	100	0.0	3.6	100	0.0	3.0	100	(C)
8	7×7	–	0.0	31.6	100	0.0	7.6	80	0.0	5.5	100	(C)
9	7×7	–	0.0	31.6	80	0.0	11.6	50	0.0	20.4	100	(C)
10	7×7	–	0.0	18.6	70	0.0	12.0	70	0.0	5.1	80	(C)
677565	10×10	easy	2.3	146.1	80	0.1	8.0	10	0.2	9.4	100	(B)
4147326	10×10	easy	0.6	38.0	30	3.3	206.3	30	1.9	81.5	60	(B)
1738340	10×10	normal	5.1	351.0	20	0.7	41.0	30	2.0	93.8	40	(B)
9116291	10×10	normal	0.3	20.0	10	0.0	0.0	0	0.0	0.0	0	(B)
7872871	10×10	hard	0.1	9.3	30	0.2	14.0	10	6.2	272.6	100	(B)
1602558	14×14	easy	0.0	0.0	0	0.0	0.0	0	2.5	54.2	50	(B)
9366021	14×14	easy	0.0	0.0	0	0.0	0.0	0	0.0	0.0	0	(B)
898033	14×14	normal	0.0	0.0	0	0.0	0.0	0	0.0	0.0	0	(B)
5183645	14×14	normal	0.0	0.0	0	0.0	0.0	0	6.2	148.4	50	(B)
912406	14×14	hard	0.0	0.0	0	0.0	0.0	0	9.1	167.3	30	(B)

Tabelle 3.9 reproduziert das *ACO-only*-Setting der Originalarbeit mit denselben Instanzen aus den Quellen (B) und (C) (vgl. [1, Tab. III]). Pro Instanz wurden $R = 10$ unabhängige Läufe durchgeführt. Ausgewiesen sind die Mittelwerte der Wandzeit T, der Iterationszahl $\#it$ und die Erfolgsrate *rate* in Prozent. Neben der im Original verwendeten *Row*-Konstruktion werden hier zusätzlich *Col* (spaltenorientiert) und *Both* (kombiniert) berichtet.

Wie Tabelle 3.9 zeigt, erreichen auf 7×7 alle Varianten hohe Raten, *Both* liegt meist vorn und benötigt typischerweise weniger Iterationen als *Row* und *Col*. Mit wachsender Größe von 10×10 bis 14×14 sinken die Konvergenzraten im Mittel und die Iterationszahlen steigen, jedoch nicht streng monoton. In 10×10 übertrifft *Both* die Einzelrichtungen häufig deutlich
(z. B. 677565: *Both* 100 % vs. *Row* 80 %, *Col* 10 %; 7872871: *Both* 100 % vs. 30 % und 10 %), während einzelne Fälle auftreten, in denen *Row* vorn liegt (9116291: *Row* 10 %, *Col* 0 %, *Both* 0 %). Für 14×14 erzielen *Row* und *Col* keine Treffer, *Both* löst dagegen einen Teil der Instanzen (30–50 %). Insgesamt spricht dies für eine höhere Robustheit der kombinierten Konstruktion bei größeren Instanzen.

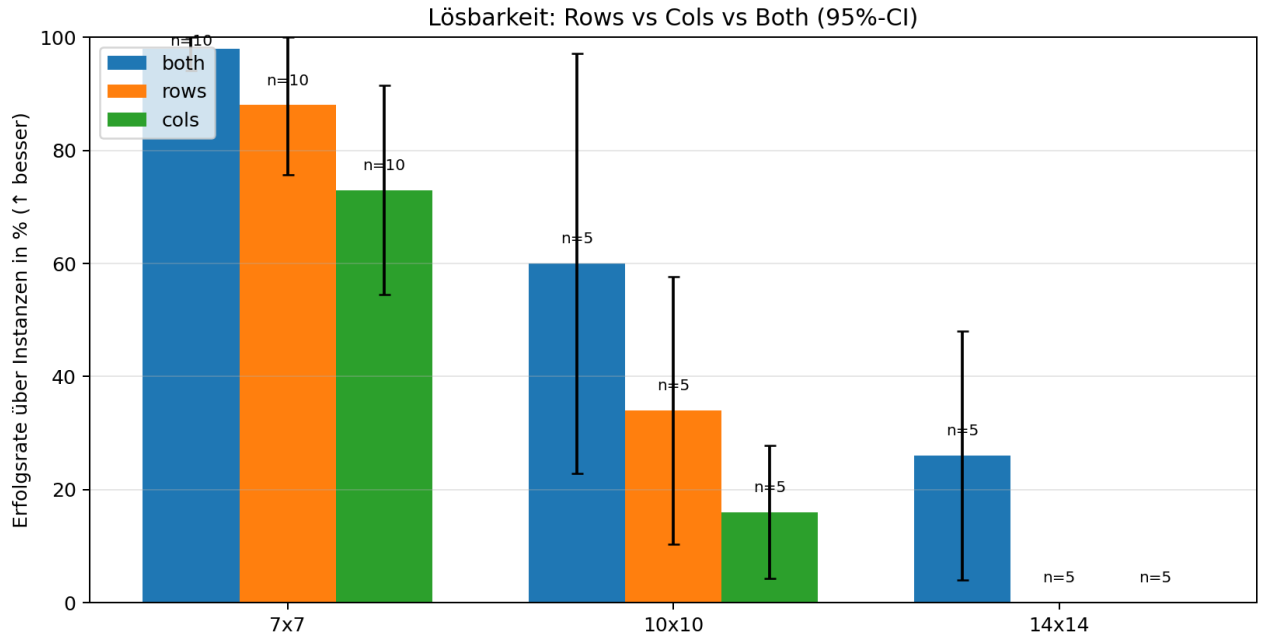


Abbildung 3.6.: ACO ohne Preprocessing: Mittlere Erfolgsraten (%) für *Rows*, *Cols* und *Both* pro Größenklasse. Balkenhöhen zeigen die mittlere Rate, schwarze Fehlerbalken die 95 %-Konfidenzintervalle. Die Annotationen $n = \cdot$ geben die Fallzahl je Klasse an.

Abbildung 3.6 zeigt die mittleren Erfolgsraten pro Größenklasse. Auf 7×7 ergibt sich ein kleiner Vorteil für *Both* gegenüber *Rows* und *Cols*. Bei 10×10 liegt *Both* deutlich vorn, *Rows* folgt mit Abstand und *Cols* bildet das Schlusslicht. Für 14×14 erzielt in dieser Stichprobe nur *Both* Treffer, *Rows* und *Cols* erreichen $0\,\%$. Die Konfidenzintervalle werden mit wachsender Größe breiter, was auf kleinere Fallzahlen und höhere Varianz hinweist. In den anschließenden Hybrid-ACO-Experimenten wird daher die *Both*-Konstruktion verwendet.

3.4.3. Mit Preprocessing / Hybrid (α-Experimente)

Tabelle 3.10.: Hybrid-ACO mit Preprocessing: Ergebnisse pro Instanz für $\alpha = 0$ und $\alpha = 1$. T: Mittlere Wandzeit (s), #it: Mittlere Iterationszahl, rate: Erfolgsquote (%). Quellen: (C) Minijuegos, (B) Puzzle-Light-Up, (D) Chiark Portable Puzzle Collection.

ID	**Größe**	**$\alpha = 0$**			**$\alpha = 1$**			**Quelle**
		T [s]	#it	rate	T [s]	#it	rate	
1	7×7	0.0	1.0	100	0.0	1.0	100	(C)
2	7×7	0.0	1.0	100	0.0	1.0	100	(C)
3	7×7	0.0	1.0	100	0.0	1.0	100	(C)
4	7×7	0.0	1.0	100	0.0	1.0	100	(C)
5	7×7	0.0	1.0	100	0.0	1.0	100	(C)
6	7×7	0.0	1.0	100	0.0	1.0	100	(C)
7	7×7	0.0	1.0	100	0.0	1.0	100	(C)
8	7×7	0.0	1.0	100	0.0	1.0	100	(C)
9	7×7	0.0	1.0	100	0.0	1.0	100	(C)
10	7×7	0.0	1.0	100	0.0	1.0	100	(C)
1738340	10×10	0.0	1.0	100	0.0	1.0	100	(B)
4147326	10×10	0.0	0.0	100	0.0	0.0	100	(B)
677565	10×10	0.0	0.0	100	0.0	0.0	100	(B)
7872871	10×10	0.0	1.0	100	0.0	1.0	100	(B)
9116291	10×10	0.0	1.0	100	0.0	1.0	100	(B)
1602558	14×14	0.0	0.0	100	0.0	0.0	100	(B)
5183645	14×14	0.0	1.0	100	0.0	1.0	100	(B)
898033	14×14	0.0	1.0	100	0.0	1.0	100	(B)
912406	14×14	0.0	1.0	100	0.0	1.0	100	(B)
9366021	14×14	0.0	0.0	100	0.0	0.0	100	(B)
20x20a	20×20	0.0	0.0	100	0.0	0.0	100	(D)
20x20b	20×20	0.0	1.0	100	0.0	1.0	100	(D)
20x20c	20×20	0.1	1.2	100	0.1	1.2	100	(D)
20x20d	20×20	0.0	1.0	100	0.0	1.0	100	(D)
20x20e	20×20	0.0	1.0	100	0.0	1.0	100	(D)
1664040	25×25	90.4	193.3	100	40.6	83.5	100	(B)
6595195	25×25	0.1	1.0	100	0.1	1.0	100	(B)
814847	25×25	2.5	10.0	100	2.6	10.0	100	(B)
9855822	25×25	0.0	0.0	100	0.0	0.0	100	(B)

Tabelle 3.10 zeigt die Resultate des Hybrid-ACO auf allen getesteten Instanzen, getrennt nach den Einstellungen $\alpha = 0$ und $\alpha = 1$. Beide Varianten wurden jeweils auf denselben Instanzen mit $R = 10$ unabhängigen Läufen ausgeführt. $\alpha = 0$ entspricht der Konfiguration ohne Pheromonkomponente, in der nur die statische Heuristik verwendet wird, während $\alpha = 1$ die vollständige ACO-Variante mit Pheromonsteuerung darstellt. Für jede Instanz sind die mittlere Wandzeit T, die mittlere Iterationszahl $\#it$ bis zur ersten gültigen Lösung sowie die Erfolgsrate (*rate*) angegeben. Die Struktur folgt der Darstellung in [1, Tab. IV] und ermöglicht einen direkten Vergleich mit der Quelle. Beide Varianten wurden auf allen Instanzen mit $R = 10$ Läufen ausgeführt.

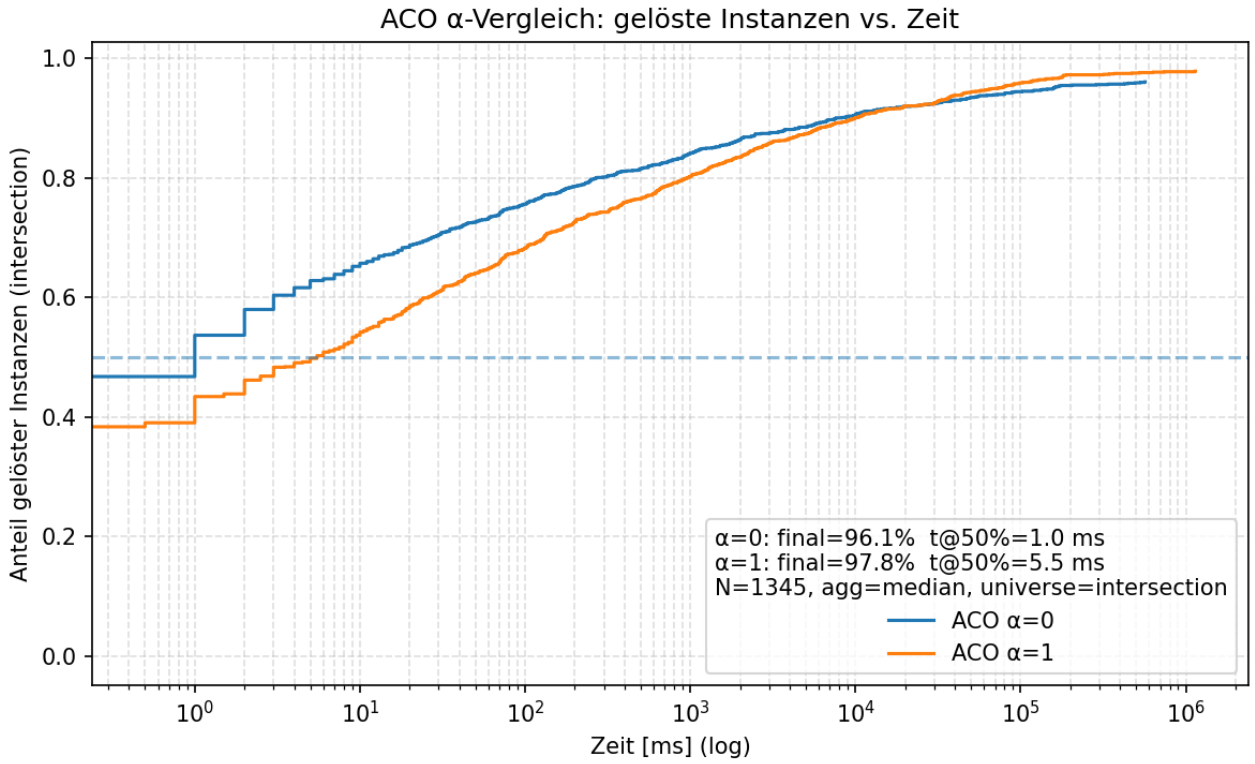


Abbildung 3.7.: Hybrid-ACO: Kumulativer Anteil gelöster Instanzen in Abhängigkeit der Zeit für $\alpha = 0$ (ohne Pheromon, nur Heuristik) und $\alpha = 1$ (mit Pheromon). Beide Achsen logarithmisch skaliert.

Abbildung 3.7 vergleicht die Lösungsabdeckung beider Hybrid-ACO-Varianten über der Zeit. Bei kurzen Laufzeiten liegt $\alpha = 0$ oberhalb von $\alpha = 1$ und erreicht früher den Medianpunkt. Im mittleren Bereich nähern sich die Kurven an. Bei langen Laufzeiten erzielt $\alpha = 1$ eine etwas höhere Endabdeckung, $\alpha = 0$ bleibt knapp darunter. Die Kurven steigen nicht bis 1, da nicht alle Instanzen gelöst wurden.

3.5. Vergleich der Solver

Dieser Abschnitt stellt die drei Solver-Ansätze im direkten Vergleich vor: Deductive Search (DS), CP–SAT (OR-Tools) und den Hybrid-ACO (Preprocessing und *Both*, α=1). Alle wurden auf denselben Instanzen ausgeführt und in identischer Umgebung gemessen. Der Vergleich zeigt den zeitlichen Verlauf der Lösungsabdeckung sowie das Verhalten in Abhängigkeit von der Problemgröße.

3.5.1. Abdeckung über die Zeit

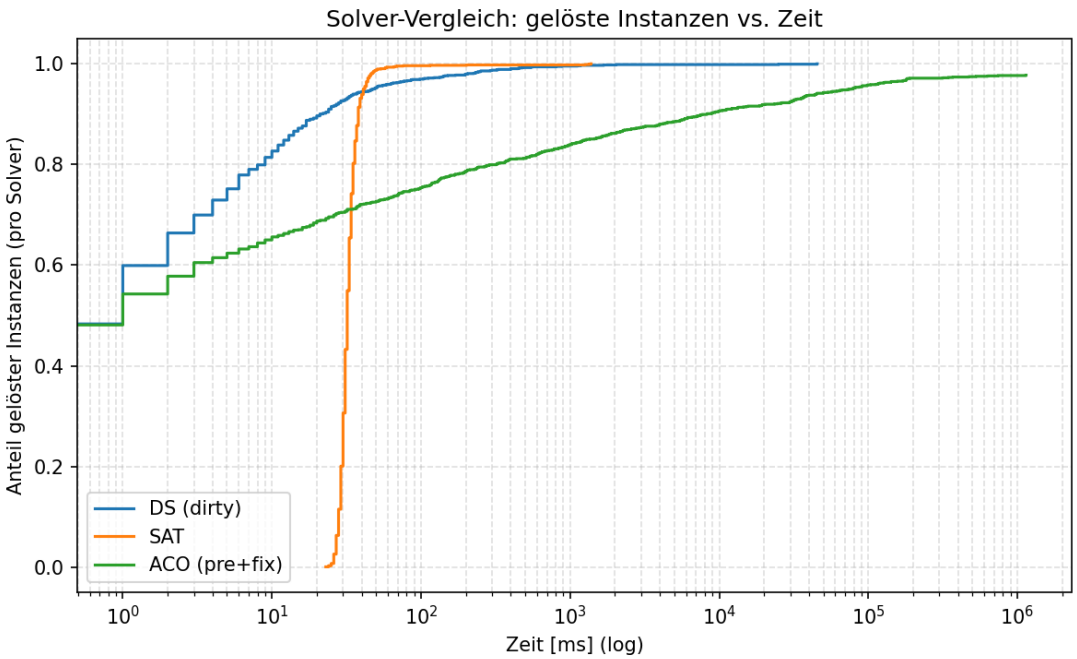


Abbildung 3.8.: Gelöste Instanzen in Abhängigkeit der Zeit (logarithmische Zeitachse). Dargestellt sind Deductive Search (DS), CP–SAT und Hybrid-ACO (*Both*, α=1).

Abbildung 3.8 zeigt, wie sich der Anteil gelöster Instanzen im Zeitverlauf entwickelt. Die Kurve von DS steigt bereits zu Beginn an und erreicht früh einen stabilen Bereich. CP–SAT folgt mit einem steileren Anstieg und erreicht die vollständige Abdeckung in kürzester Zeit. Hybrid-ACO wächst dagegen gleichmäßiger und benötigt deutlich längere Laufzeiten, bis ein vergleichbarer Anteil erreicht ist. Die unterschiedlichen Verläufe veranschaulichen, dass die deterministischen Verfahren (DS und CP–SAT) schneller sättigen, während Hybrid-ACO eine flachere, über den gesamten Zeitbereich ansteigende Kurve zeigt. Die ACO-Kurve erreicht nicht 1, da mit der gewählten Parametrisierung einzelne Instanzen ungelöst bleiben.

3.5.2. Skalierung mit der Problemgröße

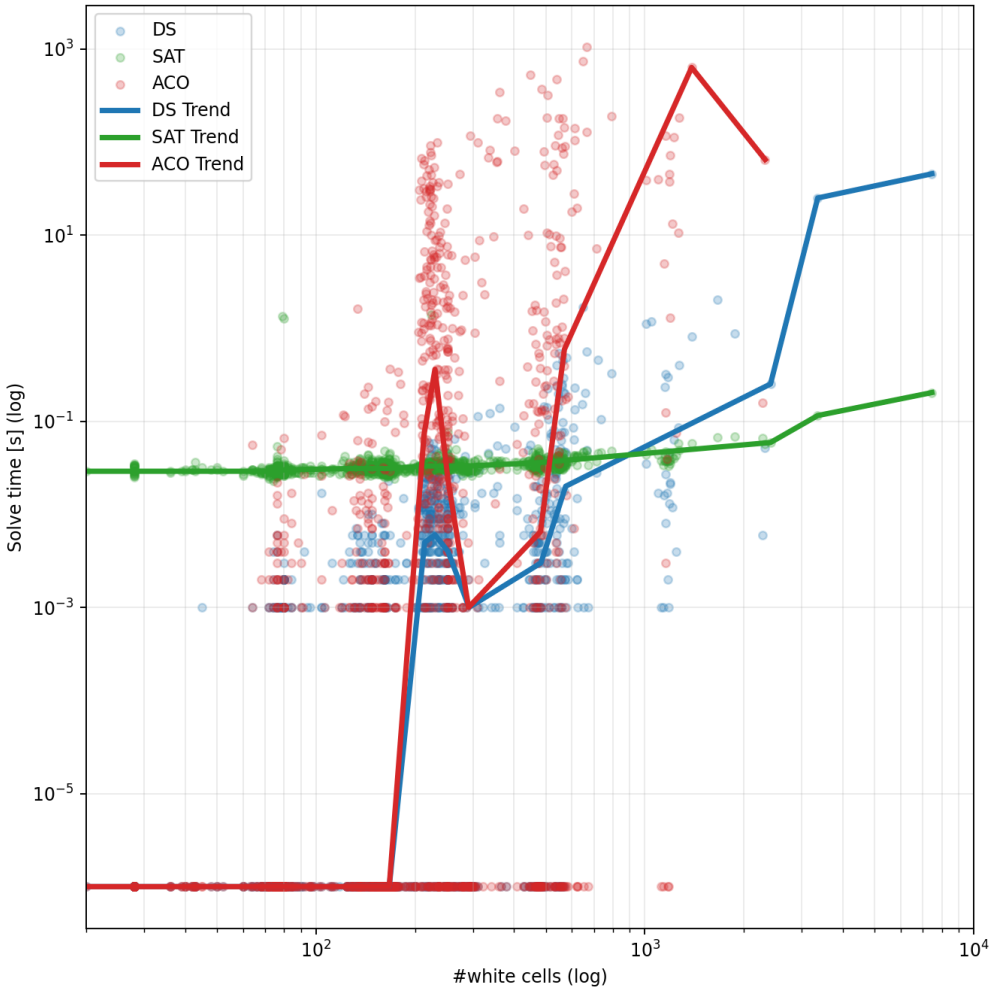


Abbildung 3.9.: Solververgleich: Laufzeiten in Abhängigkeit der Anzahl weißer Zellen (#white). Beide Achsen sind logarithmisch skaliert. Punkte zeigen Einzelinstanzen (nur gelöste Instanzen der jeweiligen Solver), Linien die pro Größenklasse berechneten Medianwerte.

Abbildung 3.9 zeigt die Laufzeiten der drei Solver in Abhängigkeit der effektiven Problemgröße. CP-SAT (grün) bleibt über weite Bereiche nahezu konstant und steigt erst bei sehr großen Instanzen leicht an. Deductive Search (blau) zeigt ein ähnliches Verhalten mit insgesamt geringen Laufzeiten, weist jedoch bei den größten Instanzen einen deutlicheren Anstieg auf. Hybrid-ACO (rot) besitzt zwei charakteristische Bereiche, viele Instanzen werden durch das Preprocessing sehr schnell gelöst, die verbleibenden weisen mit zunehmender Größe deutlich längere Laufzeiten auf. Die Streuung der ACO-Ergebnisse ist im Vergleich zu den deterministischen Verfahren am größten, insbesondere im oberen Größenbereich. Die Darstellung umfasst ausschließlich gelöste Instanzen, nicht gelöste Instanzen sind nicht enthalten. CP-SAT und Deductive Search lösen alle Instanzen. Hybrid-ACO löst mit den gewählten Parametern einige Instanzen nicht, insbesondere die größten, daher fehlen rechts Punkte und die ACO-Trendlinie endet vor der letzten Größenklasse.

4. Diskussion

Dieses Kapitel deutet und kontextualisiert die Resultate aus den Abschnitten 3.2–3.5. Es ordnet die Beobachtungen den Forschungsfragen zu, stellt Bezüge zur Literatur her und reflektiert Grenzen sowie Anschlussmöglichkeiten.

4.1. Haupterkenntnisse in Kürze

- **CP–SAT** löst den gesamten Datensatz schnell und stabil. Die Laufzeiten steigen mit der Instanzgröße nur moderat an und liegen für fast alle Fälle im zweistelligen Millisekundenbereich (vgl. Tab. 3.6, Abbildung 3.8, 3.9).
- **Deductive Search (DS)** liegt zeitlich dicht hinter CP–SAT, löst sehr viele Instanzen bereits im frühen Zeitbereich und liefert zusätzlich ein erklärbares, größenrobustes Schwierigkeitsmaß (vgl. Abschnitt 3.2.3).
- **ACO** profitiert stark vom Fixpunkt-Preprocessing. Ohne Preprocessing ist die kombinierte *Both*–Konstruktion robuster als *Rows*/*Cols*. Mit Preprocessing zeigt α=1 gegenüber α=0 Vorteile bei größeren Instanzen, während kleine Fälle häufig schon im Fixpunkt fallen (vgl. Abschnitt 3.4.3).
- **Skalierung:** In der Größenperspektive bleibt CP–SAT über weite Bereiche nahezu flach, DS zeigt einen ähnlichen Verlauf mit Anstieg in den größten Bins, ACO weist den stärksten Zuwachs und die größte Streuung auf (Abbildung 3.9).

4.2. Deductive Search: Verhalten, Designentscheidungen und Difficulty

Dirty–Queue statt Full–Scan

DS nutzt eine Dirty–Queue und betrachtet nach lokalen Änderungen nur potenziell betroffene Regionen statt jedes Mal das gesamte Gitter (Tab. 3.2, 3.3). Dadurch landen Hypothesen häufiger an *wirksamen* Positionen mit starken Folgeeffekten. *Agreement*-Übernahmen sind seltener als beim Full-Scan, weil Prüfstellen früher divergieren. Das korreliert mit der beobachteten Reduktion notwendiger Hypothesenprüfungen. Eine Ablationsstudie fehlt und bleibt zukünftiger Arbeit vorbehalten.

Abbruchkriterium (Early Stop) gegenüber Browne

Im Unterschied zu Browne [6] brechen wir nach der ersten konsistenten Gesamtbelegung ab. In Kombination mit der Dirty–Queue bringt das klare Laufzeitgewinne, zusätzliche LoE= 1-Prüfungen erhöhen primär die Laufzeit ohne erkennbaren Gewinn für den Difficulty-Score. Wir priorisieren damit effiziente, menschennähere Progression gegenüber vollständiger Deduktionsabdeckung.

LoE–Ebenen und Zeitstreuung

Über alle Instanzen genügen LoE≤ 1. LoE$= 2$ trat nicht auf. Das passt zu Akari, wo Sichtlinien oft große Bereiche auflösen. Die Zeitstreuung zeigt zwei Modi: LoE$= 0$-Fixpunktfälle bilden den Zeitboden, LoE$= 1$-Fälle steigen durch zusätzliche Shave/Agreement-Pässe an. Bei kleinen Gittern genügen mitunter wenige Hypothesen, bei großen überwiegt der Effekt zusätzlicher Pässe.

Schwierigkeitsmaß: Gewichtung und Kalibrierung

Die Difficulty aggregiert Fixpunkt-Updates, Shaves und Agreements pro LoE und normalisiert auf die Domänengröße ($\sum_i D_i$). Zwei modellierte Feinheiten sind hervorzuheben: (i) ein LoE-Bonus, der die klare Trennung LoE $= 0$ vs. LoE ≥ 1 stärkt, da Hypothesen kognitiv anspruchsvoller sind, und (ii) ein kleiner *effort*-Zusatz für fehlgeschlagene Hypothesen (`hypoFail`). Letzterer erfasst Prüfungen an Positionen mit hohem propagativen Einfluss, also an Regionen, deren Setzen/Verbieten typischerweise starke Folgeeffekte im Fixpunkt auslöst, scheitern solche Prüfungen, wird dies als leicht erhöhter Aufwand gewertet. Die Gewichte der Fixpunktregeln wurden so gewählt, dass sie einer laiennahen Wahrnehmung entsprechen. Beispielsweise gilt ein Coverage-Singleton als anspruchsvoller als das reine „Zuschließen“ bereits erfüllter Hinweise. Im Abgleich mit externen Labels zeigt DS eine größenrobuste Skala. Abweichungen zu Janko sind zu erwarten, da dort die Einordnung primär zeitbasiert erfolgt. Große, logisch einfache Puzzles werden dort oft höher eingestuft, während DS die logische Tiefe misst. Die Kalibrationskurve (Median ± IQR) liegt bis etwa Janko 7 überwiegend oberhalb der Diagonalen und flacht bei den höchsten Stufen ab, was zum Befund passt, dass die DS-Bewertung nicht die Fläche, sondern den Deduktionsaufwand abbildet. Einzelne Ausreißer bleiben, die mit einem Feintuning der Gewichte oder quellenweisen Schwellwerten adressiert werden können.

Mehrdeutige Instanzen

Bei mehreren Lösungen liefert reine Deduktion keine erzwungene Belegung. DS endet dann als *NON_DEDUCIBLE* (vgl. [6]). Unsere Implementierung prüft danach Kandidaten sequenziell und setzt im *Completion–Fallback* verbleibende Lampen bis zur Vollbelegung. Dieser Modus dient nur der Komplettierung und geht nicht in die DS–Difficulty ein, da er `hypoFail` kumuliert und die Skala verzerren würde. Das markiert eine Grenze von DS, weil die Schwierigkeitsschätzung bei Ambiguität weniger menschennah und nur eingeschränkt aussagekräftig ist.

4.3. CP–SAT: Stabilität, Skalierung und Potenzial für Hybridisierung

CP–SAT löst alle Instanzen bei niedrigen und eng streuenden Laufzeiten. Die nahezu lineare Zunahme in der log–log–Darstellung (vgl. Abbildung 3.5) weist auf ein gutmütiges Skalierungsverhalten im hier betrachteten Bereich hin. Die IQR-Bänder bleiben über weite Größenbereiche schmal (ebenfalls Abbildung 3.5). Das ist konsistent mit der deterministischen Arbeitsweise und der starken internen Propagation: Es gibt keinen Seed-Einfluss, und das Suchverhalten ist über Instanzen ähnlicher Größe sehr ähnlich. In den größten Einzelfällen steigen die Zeiten sichtbar an, bleiben jedoch klar unter einer Sekunde. Ein naheliegender nächster Schritt wäre eine *Hybridisierung*: Preprocessing durch die Rule-Engine und Start von CP–SAT vom resultierenden Teilzustand. Die Ergebnisse legen nahe, dass sich damit insbesondere bei kleinen Instanzen der Frühbereich weiter beschleunigen ließe (siehe Abbildung 3.9).

4.4. ACO: Reproduktion, Varianten und Rolle des Preprocessings

Ohne Preprocessing zeigt die kombinierte *Both*-Variante höhere Erfolgsraten als rein zeilen- oder spaltenorientierte Konstruktionen (vgl. Abbildung 3.6). Das deckt sich mit der Intuition, dass eine größere Gruppenvielfalt vorteilhaft ist, auch wenn einzelne Iterationen dadurch länger dauern. Der exakte ACO-Only-Abgleich mit [1] fällt nicht in allen Zahlen übereinstimmend aus. Wahrscheinliche Ursachen liegen in Implementierungsunterschieden, insbesondere bei den Gruppierungsheuristiken und der konkreten Auswertung der Fitnessfunktion. Da das Original keine vollständigen Details zur Initialisierung und internen Zufallsseeds angibt, ist eine exakte Reproduktion schwierig. Insgesamt deuten die Abweichungen jedoch darauf hin, dass die Variation *Both* eine erhöhte Diversität im Suchverlauf bietet.
Mit Fixpunkt-Preprocessing verschwinden die Unterschiede bei kleinen und mittleren Größen weitgehend, da viele Instanzen bereits deterministisch gelöst werden. Unterschiede zwischen α=0 und α=1 treten vor allem bei größeren Instanzen auf: α=1 reduziert die mittlere Iterationszahl und Laufzeit, die Erfolgsraten bleiben gleich. Der verwendete Hybrid-Modus mit lokalem Fixpunkt während der Konstruktionsphase beschleunigt Propagationseffekte und erlaubt eine frühzeitige Erkennung von Widersprüchen, erhöht jedoch gleichzeitig die Rechenzeit pro Schritt.
Eine systematische Ablations- und Parameterstudie (mit und ohne Interleaving, Variation von ρ, Q, β und Koloniegröße) wurde in dieser Arbeit nicht durchgeführt. In Abbildung 3.9 zeigt sich, dass bei ACO einzelne, insbesondere große, Instanzen nicht gelöst wurden und die Medianlinie deshalb früher endet. Mit einer angepassten Parametrisierung könnte sich die Lösungsabdeckung erhöhen, dies bleibt jedoch künftiger Arbeit vorbehalten.

4.5. Solververgleich und Einsatzszenarien

Die Zeit-Abdeckungskurven (Abbildung 3.8) zeigen drei charakteristische Verläufe: DS steigt sehr früh an, CP–SAT steigt danach am steilsten an und erreicht das Plateau am schnellsten, ACO steigt kontinuierlich mit langem Tail. In der Größenperspektive (Abbildung 3.9) bleibt CP–SAT am flachsten, DS folgt mit Anstieg im oberen Bereich, ACO zeigt den stärksten Zuwachs und die größte Varianz.
Daraus ergeben sich praxisnahe Empfehlungen:

- **Strikte Latenzbudgets und Servicebetrieb:** CP–SAT ist die verlässlichste Wahl. Preprocessing-gestützte Varianten (DS, Hybrid-ACO) lösen zahlreiche kleine Instanzen bereits im Preprocessing.
- **Erklärbarkeit und Schwierigkeitsbewertung:** DS bietet regelbasierte Schrittprotokolle (*Traces*) und eine konsistente, größenrobuste Schwierigkeitsmetrik, stößt jedoch bei mehrdeutigen Instanzen an konzeptionelle Grenzen.
- **Diversität und Kandidatenfindung:** ACO ist als Hybrid-Baustein sinnvoll (*Both*, α=1), insbesondere in Pipelines, die Kandidaten an DS oder CP–SAT weiterreichen.

4.6. Vergleich mit der Literatur

Die DS-Ergebnisse stehen im Einklang mit [6], berücksichtigen jedoch ein Early-Stop anstelle vollständiger Exhaustion. Look-ahead (LoE) erhöht den Ableitungsaufwand und dient als brauchbarer Proxy für wahrgenommene Schwierigkeit. Die ACO-Experimente bestätigen qualitativ [1]: Preprocessing schrumpft den Suchraum, Pheromone beschleunigen die Konvergenz gegenüber einer rein heuristischen Variante. Abweichungen sind aufgrund unterschiedlicher Implementierungen, Gruppierungen und Datensätze erwartbar. Für CP-SAT gibt es keine direkt vergleichbaren Akari-Resultate. Das beobachtete Stabilitätsprofil entspricht Erfahrungen mit OR-Tools bei ähnlichen Gitter-CSPs.

4.7. Limitierungen und Validität

- **Parametrik und Ablationsstudien:** Die ACO-Parameter wurden an die Literatur angelehnt und nicht systematisch variiert. Eine systematische Ablation des Hybrid-Interleavings fehlt.
- **Datensatz:** Sehr große Instanzen sind selten, daher sind Aussagen am oberen Ende des Größenspektrums deskriptiv zu verstehen.
- **Vergleichsfairness:** CP–SAT lief bewusst ohne domänenspezifisches Preprocessing, DS und ACO hingegen mit Fixpunkt-Preprocessing. Das begünstigt CP–SAT nicht, hält aber die Paradigmen klar getrennt.
- **Interpretation:** Einzelne mechanistische Erklärungen (Dirty-Queue $\rightarrow$ weniger Agreements, Interleaving $\rightarrow$ frühere Widerspruchserkennung) sind konsistent mit den Daten, wurden jedoch nicht kausal nachgewiesen.
- **Mehrdeutige Instanzen:** DS kann Puzzles mit mehreren Lösungen nur partiell deduzieren. Der Completion-Fallback liefert zwar Vollbelegungen, aber keine verlässliche Schwierigkeitsbewertung.

4.8. Ausblick

- **ACO-Studie:** Systematischer Sweep über ρ, Q, Koloniegröße und β. Gezielte Ablation von *Both* und Interleaving. ECDF-basierte Auswertung einschließlich Varianzbetrachtung.
- **Hybride Pipelines:** DS-/Rule-Preprocessing vor CP–SAT. ACO als Kandidatengenerator mit Übergabe an DS oder CP–SAT. Adaptives Umschalten nach einem kurzen Initiallauf zur Bewertung der aktuellen Suchdynamik.
- **Difficulty-Kalibration:** Feintuning der Regelgewichte und quellenweise Schwellwerte. Kopplung an Zeitmetriken. Ausweitung des Abgleichs mit der *Portable Puzzle Collection* und weiteren Kuratierungen.
- **Generatoren:** DS-Score als Zielfunktion für difficultygesteuerte Puzzle-Generierung und als Kontrollgröße für Größe-vs.-Tiefe-Trade-offs.

5. Fazit

Diese Arbeit wurden drei Ansätze zur Lösung des Logikrätsels *Akari* verglichen: einen deduktiven, regelbasierten Solver (DS), einen metaheuristischen Ansatz auf Basis der *Ant Colony Optimization* (ACO) sowie ein deklaratives Modell mit *CP–SAT*. Ziel war ein systematischer, reproduzierbarer Vergleich der Ansätze hinsichtlich Lösungsfähigkeit, Laufzeit, Skalierung und Robustheit. Zusätzlich wurde für DS ein internes Schwierigkeitsmaß entwickelt und bei ACO der Einfluss des Pheromonparameters α untersucht.

Zentrale Ergebnisse: CP–SAT war am effizientesten und stabil. Vollständige Abdeckung, Medianzeiten im zweistelligen Millisekundenbereich und geringe Streuung über die Größe. DS lag zeitlich knapp dahinter und lieferte erklärbare Schrittfolgen sowie ein größenrobustes Schwierigkeitsmaß. ACO ohne Preprocessing zeigte geringere Erfolgsraten und höhere Varianz. Mit Fixpunkt-Preprocessing und *Both* stieg die Abdeckung deutlich. Der Vergleich α=0 gegenüber α=1 bestätigte den Nutzen der Pheromonkomponente, insbesondere bei größeren Instanzen. In der Skalierung blieben CP–SAT und DS weitgehend flach, ACO stieg mit der Größe am stärksten.

Implikationen: Für strikte Zeitbudgets empfiehlt sich CP–SAT. DS ist attraktiv, wenn Erklärbarkeit oder Schwierigkeitsbewertung benötigt wird. ACO eignet sich in Hybridform: Preprocessing+*Both* liefert diversere Kandidaten, die an DS oder CP–SAT übergeben werden können.

Einschränkungen: Sehr große Instanzen sind selten. Aussagen am oberen Größenende sind daher deskriptiv. ACO-Parameter (ρ, Q, β und Koloniegröße) wurden an Literaturwerte angelehnt und nicht breit optimiert. Für CP-SAT wurde nur eine Modellierungsvariante betrachtet. Bei mehrdeutigen Puzzles stößt DS an seine Grenzen, da reine Deduktion keine eindeutige Lösung erzwingt und der Completion–Fallback die Difficulty nur bedingt abbildet.

Ausblick: Empfehlenswert sind eine systematische ACO-Parameterstudie einschließlich ECDF- und Varianzanalysen, Ablationen des Hybrid-Interleavings sowie adaptive Pipelines, bei denen ACO als Kandidatengenerator und DS oder CP–SAT als Verifikationsmodule dienen. Für DS bieten sich eine Feinkalibration der Regelgewichte und eine breitere Validierung des Schwierigkeitsmaßes an. Langfristig kann der DS-Score difficultygesteuerte Generatoren steuern.

Schlussfolgerung: *CP–SAT dominiert in Effizienz und Stabilität, DS verbindet Lösungsweg und Bewertung, zeigt jedoch bei Mehrdeutigkeit methodische Grenzen, ACO trägt als Hybrid durch Suchdiversität und Kandidatengenerierung bei.*

A. Instanzquellen

A Janko Akari: https://www.janko.at/Raetsel/Akari/index.htm

B Puzzle-Light-Up: https://www.puzzle-light-up.com/

C Minijuegos: https://www.minijuegos.com/juego/light-up

D Chiark Portable Puzzle Collection (Light Up):
https://www.chiark.greenend.org.uk/~sgtatham/puzzles/js/lightup.html

```
Instances identification:
20x20a:
20x20:e1aBBb3b1BcBaB2bBaBd2e00Bh2jBBa1BeB0bBb0bBaBe0aBhBbB
BiBe1b2c1bB3d0k0c2fBbBbBaBcBBc1c02b2eBc2b2cBBcBB11a2cBdBb1c1
cBa1B0bBBBeBBgBgBaBBBBBbBaBbB0b1aBa0aBaBd2d1c2d1Bb0cBe4b
Ba1i0aBBBdBa1a1Ba0bBB2bBhBBjBd2aBcBaBaBc
20x20b:
20x20:cBa2a2hB0aBaBdBbBc1b1d2d2cBBcB0aBBdBdBd1BbBb3a0cBaBB
aBd3aBBdBcBhBgBBB2b1e1bBa0d1aBd1bBgBb2d0b1bBaBBa0c1bBaBdB
1aB2aBcBbBa0dBf2bBdBbBc1a1dBa3d0b3gB1BBb1eBa0dBcBhBbBa0c1a
B0a1dBaBdBdBd1BbBbBdBc2BcBBa12a0dBbBc0b0fBa3aBhBBaB
20x20c:
20x20:BBBBBBbBd0fB1a2d1b1g1aBcBf1BBc1BaBcBaBaBi1bB0a1aBaB00
b2a1gBb11eBaBf2aBe11BaBfBaBkBa10Ba2dBbBeB0bBbBcB1lBcBd0B1a1
bBa2a0dBdBfBe2c1g3aBc4bBcBa0BB0dBf0cBBcB2BBa0aBd1eBBjBoBcB
bBBBBcBB0aBb3b2bBBa2b02aBBdBc2aBj
20x20d:
20x20:2dB1fBBa1eBBaBgBBaBB0aBBBaB1dBbBBBi1BBb1a1a0d0B2dBe
Bf1cBd1bBbBb2k0aBa21b2c03eBBaBdBbBcBa0b2a2a3dBcBfBbBbBb0Ba1
b1dB1qBe2cBB2h2BdBbBb1b1cBdBaB1cBBbBpBa0rBbB2aBa0a2bBeBaBe
Bb0B1bBdBaBBc1cBBBBb1aBaBBaBbB1c0bBb0
20x20e:
20x20:c2bBg2d1eBbBa1BdBBf1a0aBb3bBb0eB0bBaBb2aBdBa0bBc2e1dBc
BBa2cB0cB0a1e1fBj1hB0c0eBa0c1e2aBa0a1cBB0a1d2jBBfBaBBBa0BcBc
Ba1j1BaBaBbBeBbB2a1c2bBB0dBlBd0a2a2aBB0Ba0dBb1BbBaBa0BBaBc
BBcBgBc2cBa0eBc0f2BBBB2bBa2fBBaBa2eB2c
```

E Marugoto-Sammlung: https://nikoli.stores.jp/items/5e8e72a7e20b040815d77ae9

F Penpa-Sammlung (Jahrgänge 2011, 12, 15, 16, 20, 23):
https://nikoli.stores.jp/?category_id=62fed8b2110dda01c85312
https://nikoli.stores.jp/?category_id=65556a72917665003550488e

G Puzzlebox-Sammlung (Hefte 4-13):
https://nikoli.stores.jp/?category_id=6679c897401c31154b80e814

Erklärung zum Einsatz von KI-Tools

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

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

Umfang und Art der Nutzung

Die Unterstützung umfasste sprachliche Überarbeitung (Kürzungen, Umformulierungen, Titel- und Bildunterschriften, einheitliche Verwendung der Fachterminologie), Hinweise zur Gliederung und zur einheitlichen Darstellung von Tabellen und Abbildungen sowie programmiertechnische Unterstützung (C++/Python-Boilerplate, Refactoring-Hinweise, LaTeX-Hilfen). Gelegentlich übernommene Codefragmente wurden von mir angepasst, getestet und verifiziert.

Verantwortlichkeit

Sämtliche fachlichen Inhalte (Modelldesign, Experimente, Datenauswertung und Interpretation) stammen von mir. KI-generierte Vorschläge wurden geprüft, gegebenenfalls verändert und nur in eigener Verantwortung übernommen. Wörtliche Übernahmen aus externen Quellen sind als Zitat gekennzeichnet.

Literaturverzeichnis

[1] I. Rosberg, E. F. G. A. Xavier, and M. C. Goldbarg, "Solving the light up with ant colony optimization," in *2011 IEEE Congress on Evolutionary Computation (CEC)*, IEEE, 2011. https://ieeexplore.ieee.org/document/5949669.

[2] B. McPhail, "Light up is np-complete." http://www.mountainvistasoft.com/docs/lightup-is-np-complete.pdf, 2005.

[3] B. Pulles, "Analysis of akari," Bachelor's thesis, Radboud University, Nijmegen, The Netherlands, 2021. Supervisors: H. Zantema, H. Geuvers. https://www.cs.ru.nl/bachelors-theses/2021/Bram_Pulles___1015194___Analysis_of_Akari.pdf.

[4] D. M. dos Santos Costa, "Computational complexity of games and puzzles," Master's thesis, Instituto Superior Técnico, University of Lisbon, Lisbon, Portugal, 2018. Master's thesis. Supervisors: L. M. S. Russo, A. P. L. Francisco. https://scholar.tecnico.ulisboa.pt/api/records/wYoodVm9Xz7UlnKYq7HZHSlHth3C-K7ud2HQ/file/3e88e99ae96230fb99bff50e0bcff8a217cb19f5a03e9e3e18ccbe9d01bcc0f1.pdf.

[5] R. A. Hearn, *Games, Puzzles, and Computation*. PhD thesis, Massachusetts Institute of Technology, Cambridge, MA, 2006. https://erikdemaine.org/theses/bhearn.pdf.

[6] C. Browne, "Deductive search for logic puzzles," in *2013 IEEE Conference on Computational Intelligence in Games (CIG)*, pp. 359–366, IEEE, 2013. https://ieeexplore.ieee.org/document/6633649.

[7] V. Dekker, "A difficulty measure for light up puzzles," Bachelor's thesis, Leiden University, Leiden Institute of Advanced Computer Science (LIACS), Leiden, The Netherlands, 2015. Supervisors: W. A. Kosters, H. J. Hoogeboom. https://theses.liacs.nl/pdf/2014-2015Victor-Dekker.pdf.

[8] S. Salcedo-Sanz, L. Carro-Calvo, E. G. Ortiz-García, Á. M. Pérez-Bellido, and J. A. Portilla-Figueras, "A nested two-steps evolutionary algorithm for the light-up puzzle," *ICGA Journal*, vol. 32, Sept. 2009. https://www.researchgate.net/publication/220174496_A_Nested_Two-steps_Evolutionary_Algorithm_for_the_Light-up_Puzzle.

[9] E. G. Ortiz-García, S. Salcedo-Sanz, Á. M. Pérez-Bellido, and A. Portilla-Figueras, "A hybrid hopfield network–genetic algorithm approach for the lights-up puzzle," in *2007 IEEE Congress on Evolutionary Computation (CEC)*, pp. 1403–1407, IEEE, 2007. https://ieeexplore.ieee.org/document/4424635.

[10] M. Dorigo and T. Stützle, *Ant Colony Optimization*. Cambridge, MA: MIT Press, 2004. https://www.researchgate.net/publication/308953674_Ant_Colony_Optimization.

[11] M. Çelik, H. Erdoğan, F. Tahaoğlu, T. Uras, and E. Erdem, "Comparing ASP and CP on four grid puzzles: Kakuro, nurikabe, heyawake, and akari," in *Proceedings of the 16th International RCRA Workshop on Experimental Evaluation of Algorithms for Solving Problems with Combinatorial Explosion (RCRA 2009)*, vol. 589 of *CEUR Workshop Proceedings*, 2009. https://ceur-ws.org/Vol-589/paper01.pdf.