Beluga Challenge mit Reinforcement Learning

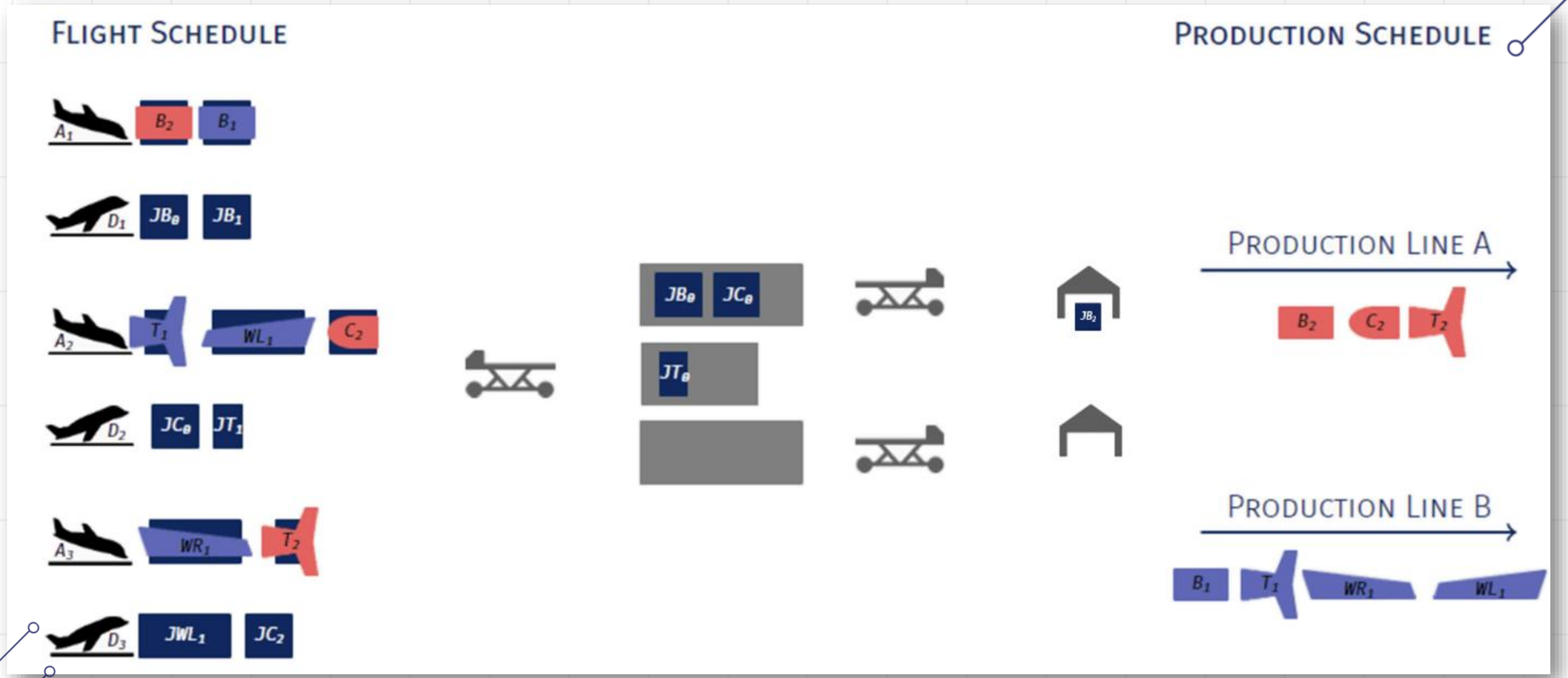
Fortgeschrittenen Praktikum Dr.Gnad

Youssef Daoudi El Boukhrissi
Jan Kirschbaum
Nils-Frederik Schulze

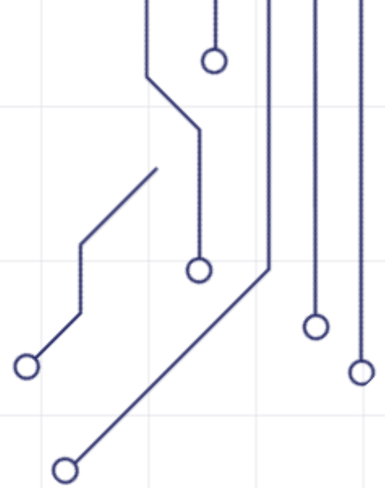
GLIEDERUNG

1. **Fragestellung & Anforderungen**
2. **Ansatz & Implementierung**
3. **Ergebnisse**
4. **Lessons Learned**
5. **Fazit**

BELUGA-PROBLEM:



FRAGESTELLUNG & ANFORDERUNGEN



1. **Ziel: Eigenständige Lösung des Beluga-Problems**

2. **Herausforderung: NP-schwer – keine „einfache“ Lösung möglich**

3. **Erste Ansätze im Team:**
   - Mathematischer Ansatz
   - Theoretische Informatik (Reduktion)
   - Reinforcement Learning

4. **Entscheidung: Machine Learning (RL) – wegen Flexibilität & Realisierbarkeit**

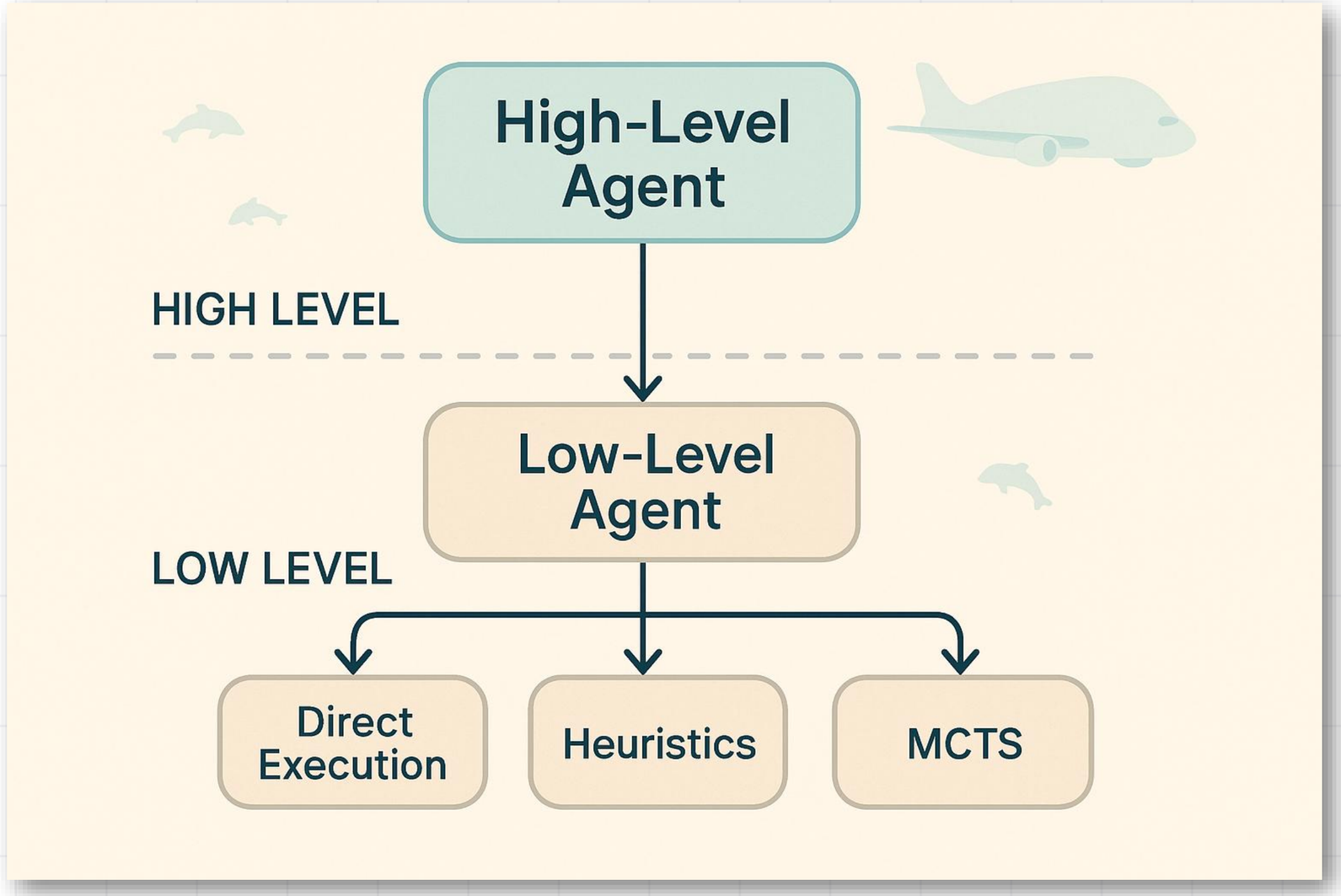
UNSER ANSATZ & IMPLEMENTIERUNG

1. **Einstieg: Reinforcement Learning als finaler Ansatz**
2. **Analyse des GitHub-Toolkits und JSON-Struktur**
3. **State-Definition**
4. **Python-Umsetzung des Problemzustands (Listenstruktur)**
5. **Zielzustand: Production line/Beluga-Listen sind leer**

ZUSTAND:

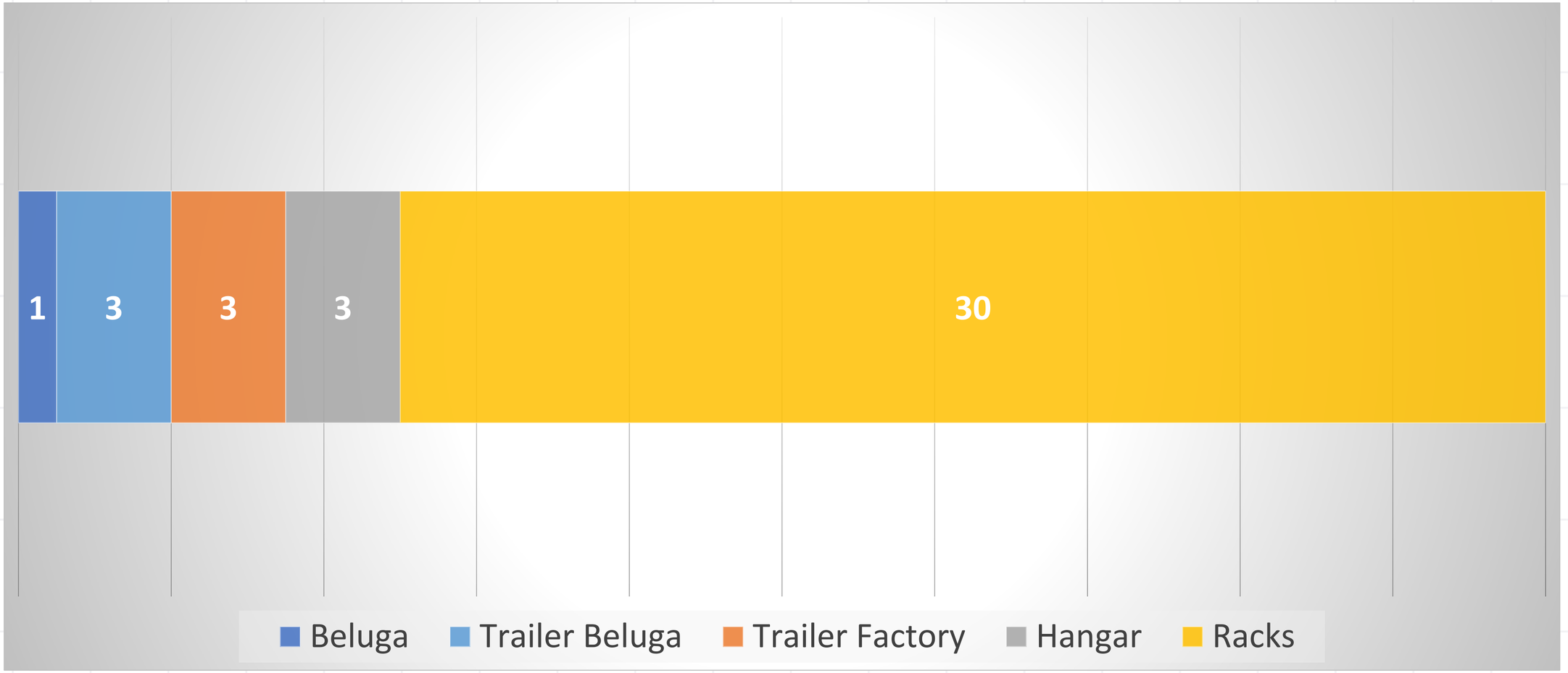
```
jigs:
    0: typeB | False
    1: typeB | False
    2: typeB | False
    3: typeB | False
belugas:
    0: current_jigs = [0, 1, 2] | outgoing = []
    1: current_jigs = [3] | outgoing = []
    2: current_jigs = [] | outgoing = [typeB, typeB, typeB]
trailers_beluga: [None, None]
trailers_factory: [None, None, None]
racks:
    0: size = 26 | current_jigs = []
    1: size = 28 | current_jigs = []
production_lines:
    0: scheduled_jigs = [1, 3]
    1: scheduled_jigs = [2]
    2: scheduled_jigs = [0]
hangars: [None]
```

HYBRIDER RL-ANSATZ – PPO + MCTS



HIGH-LEVEL-AGENT: ENTSCHEIDUNG MIT PPO

1 Eingabe: 40-dimensionaler State-Vektor



HIGH-LEVEL-AGENT: ENTSCHEIDUNG MIT PPO

1. **Ausgabe: Wahrscheinlichkeitsverteilung über Aktionen (Stack, Unstack, Load Bealug, …)**

2. **Der PPO-Agent wird auf Basis folgender Reward-Struktur trainiert:**

- Ziel erreicht → + 10.000
- Keine ausführbare Aktion mehr vorhanden (Deadend) → – 10.000
- Schleifenverhalten erkannt → – 200
- Aktion nicht ausführbar → – 1000
- Beluga abgeschlossen → + 2000
- Teilweises Be- oder Entladen des Belugas → + 100
- (Un)Stacking von Racks (links oder rechts) → - 10
- Normale Lieferung oder Abholung am Hangar → + 200 / + 50 …

DIRECT EXECUTION & HEURISTICS

1 Direct Execution: unload_beluga

- Einzige Aktion ohne Parameter
- Immer: nimm ersten verfügbaren Jig aus dem ersten Beluga
- Keine Suche nötig

2 Heuristikbasierte Entscheidungen

- Wenn der Zustand eine klare, sinnvolle Aktion nahelegt
- Entscheidung basiert direkt auf Observation

Beispiel: „Ein Jig, der rechts im Rack liegt und gebraucht wird“

→ Trailer rechts ist frei

→ right_unstack_rack + deliver_to_hangar

MCTS – Monte Carlo Tree Search

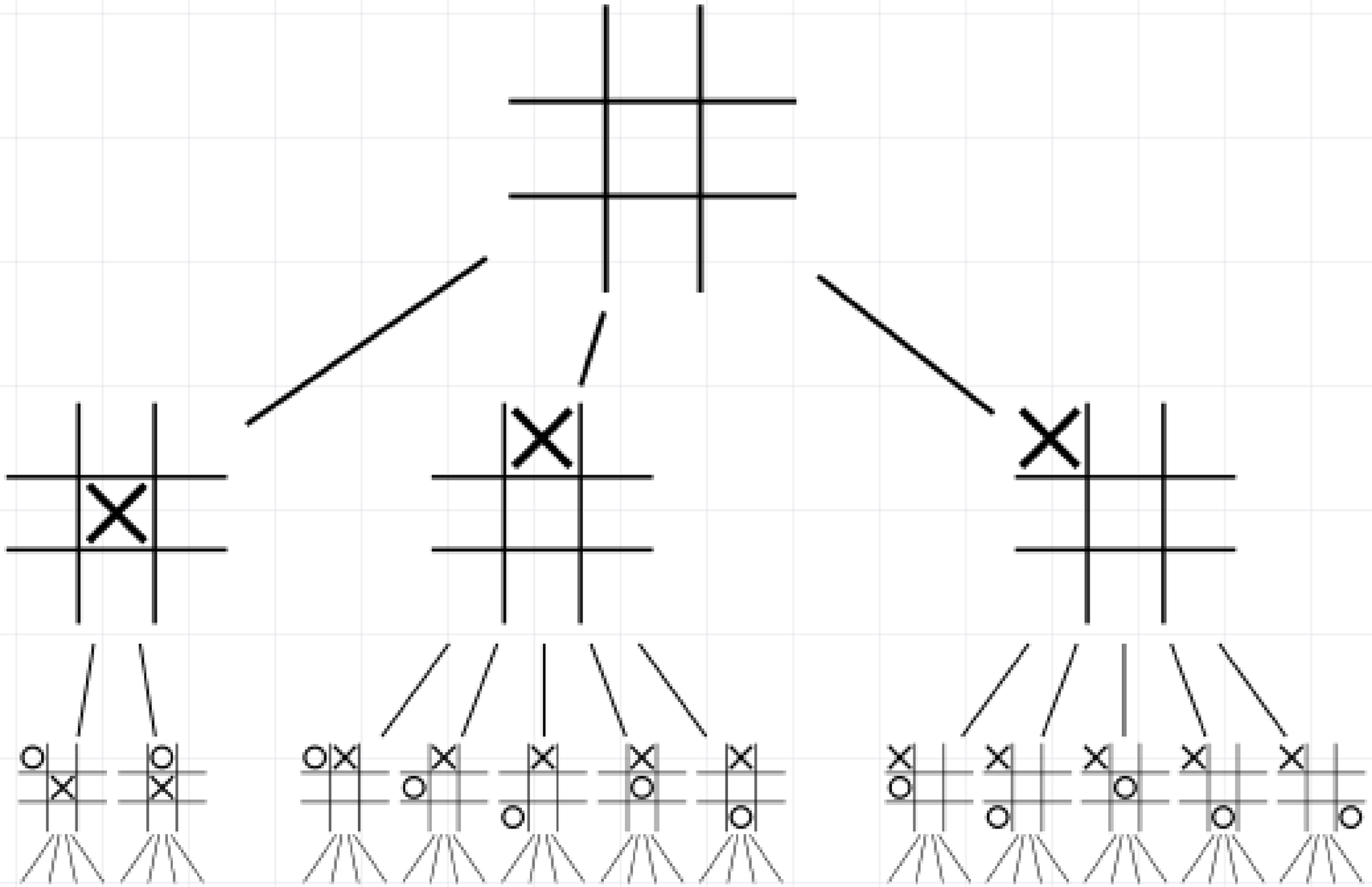
1 Kernproblem: Hohe Komplexität

- Reihenfolge der Aktionen sehr relevant
- Welche Aktion mit welchen Parametern ist am besten im aktuellen State
- Welche Auswirkung hat Aktion-x in State-1 für den State-n

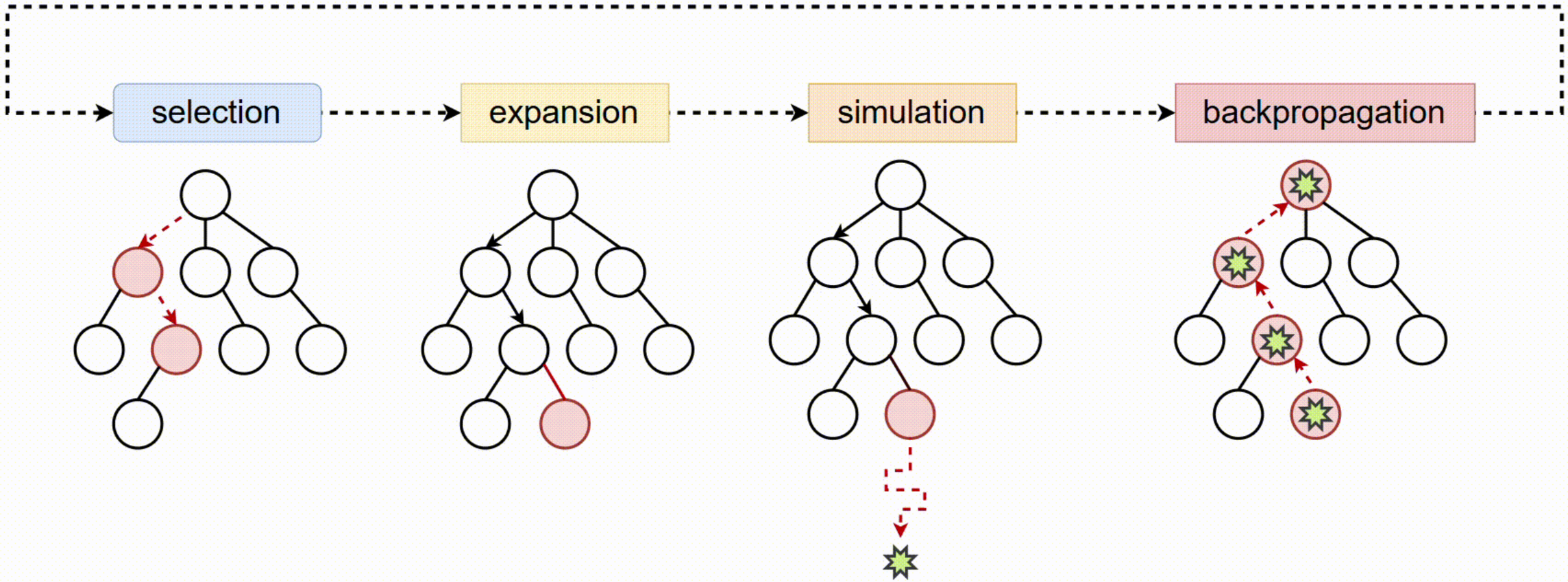
2 Lösung: Monte Carlo Tree Search

- Ideal für sequentielle Entscheidungsprobleme **(Such-Algorithmus)**
- Lösungssuche ohne Betrachtung des gesamten Aktionsraumes
- Rechenzeit wird intelligent auf die vielversprechendsten Sequenzen konzentriert

MCTS – Beispiel TicTacToe



MCTS – Algorithmus in vier Phasen



MCTS – Wie erkennt man gute Knoten?

1 UCT: Upper confidence bounds applied to Trees

$$\frac{w_i}{n_i} + c\sqrt{\frac{\ln N_i}{n_i}}$$

- n: Anzahl "Besuche"
- w: Wert des States der mit dem Knoten assoziiert ist
- c: Parameter zum Gewichten der Exploration

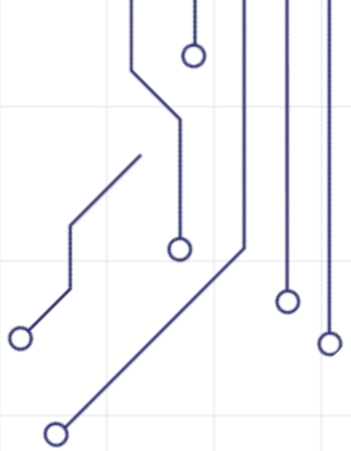
2 State-Bewertung: „w“

- Keine exakte Bestimmung möglich, Wert wird geschätzt
- Binäres Endergebnis erschwert Bewertung von Zwischenzuständen
- **Lösung:** Einführung von Zwischenzielen (Subgoals)
    - Beluga entladen (+ 50)
    - Beluga beladen (+ 100)
    - Produktionslinie vollständig (+ 150)

PROBLEMGENERATOR & TRAININGSSTRATEGIE

1. **Eigener Generator für neue Problem-Instanzen**
   - Für eine Trainingspipeline
2. **Schwierigkeit: Erkennung lösbarer vs. unlösbarer Fälle**
3. **Curriculum Learning: Start mit einfachen Problemen → Steigerung**
4. **Ziel: Robustes Modell**

POSTPROCESSING & OPTIMIERUNG



1. **Problem: Zu viele unnötige Aktionen in Lösung**

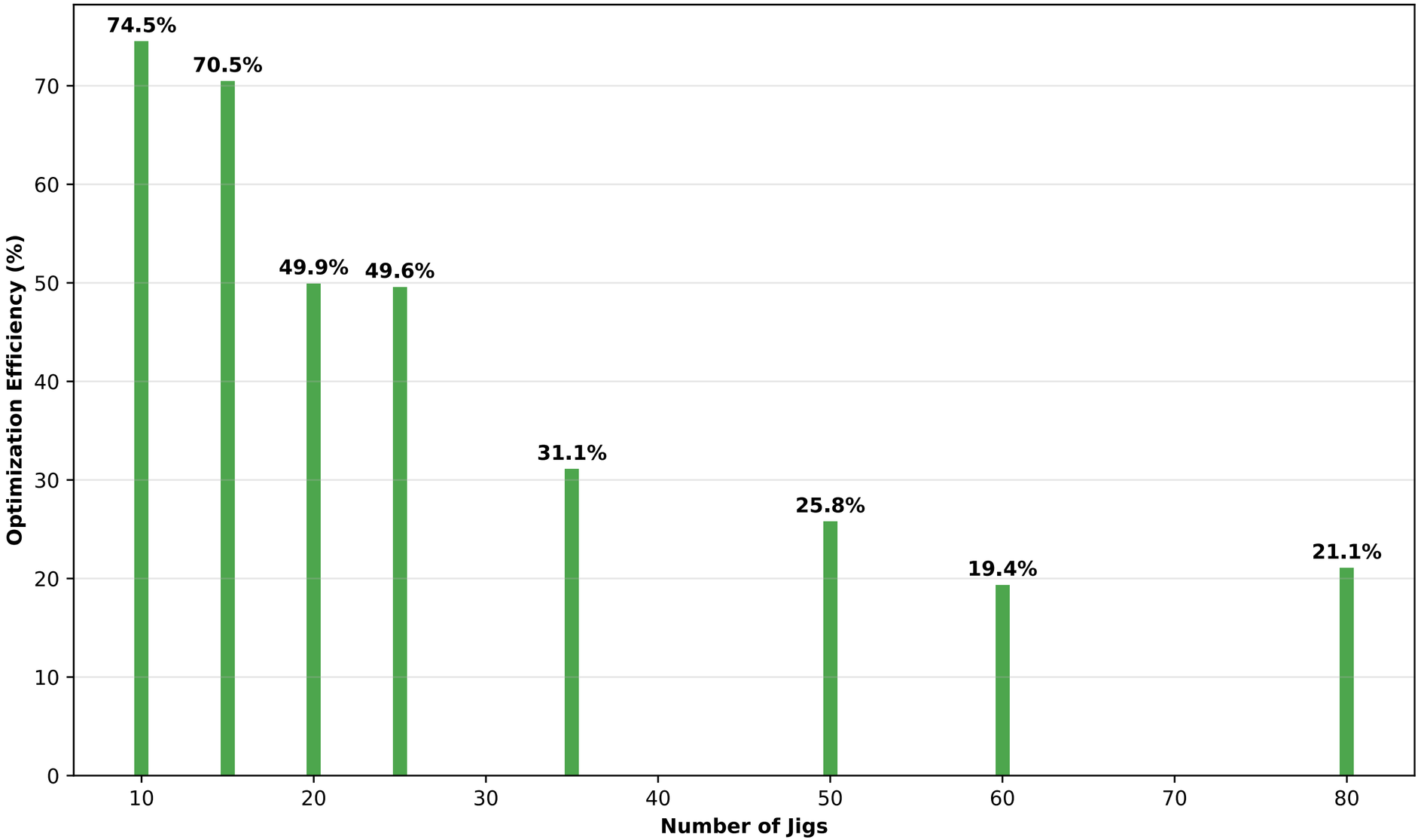
2. **Lösung: Zustandsvergleich via Hashing**
   - Entfernen redundanter Aktionspfade

3. **Skript via Terminal steuerbar (Train / Evaluation / Problem lösen)**

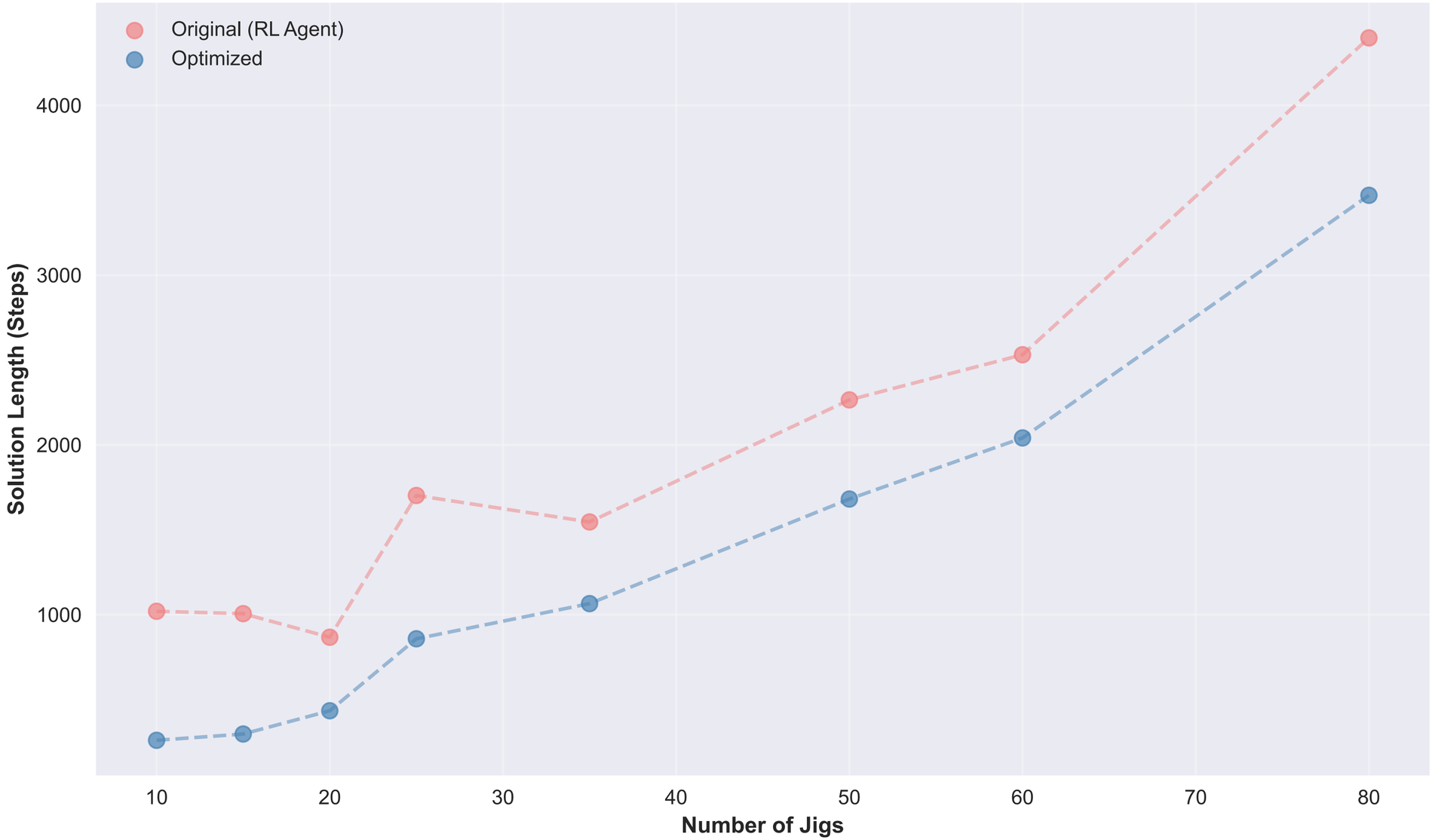
4. **MCTS in C++ umzusetzen und CPU-Parallelisierung zu nutzen**

   → **hohe Integrationskomplexität**

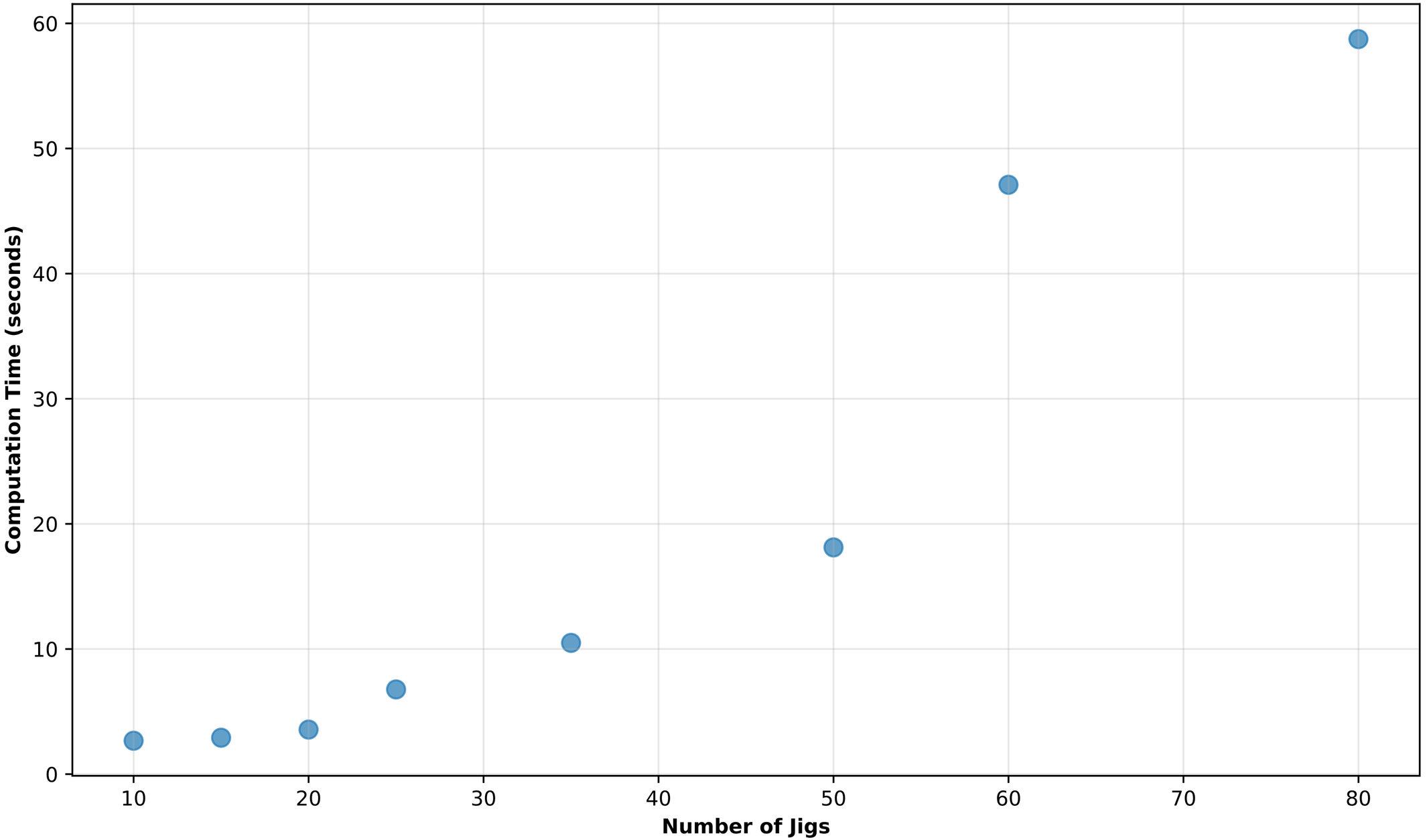
POSTPROCESSING & OPTIMIERUNG

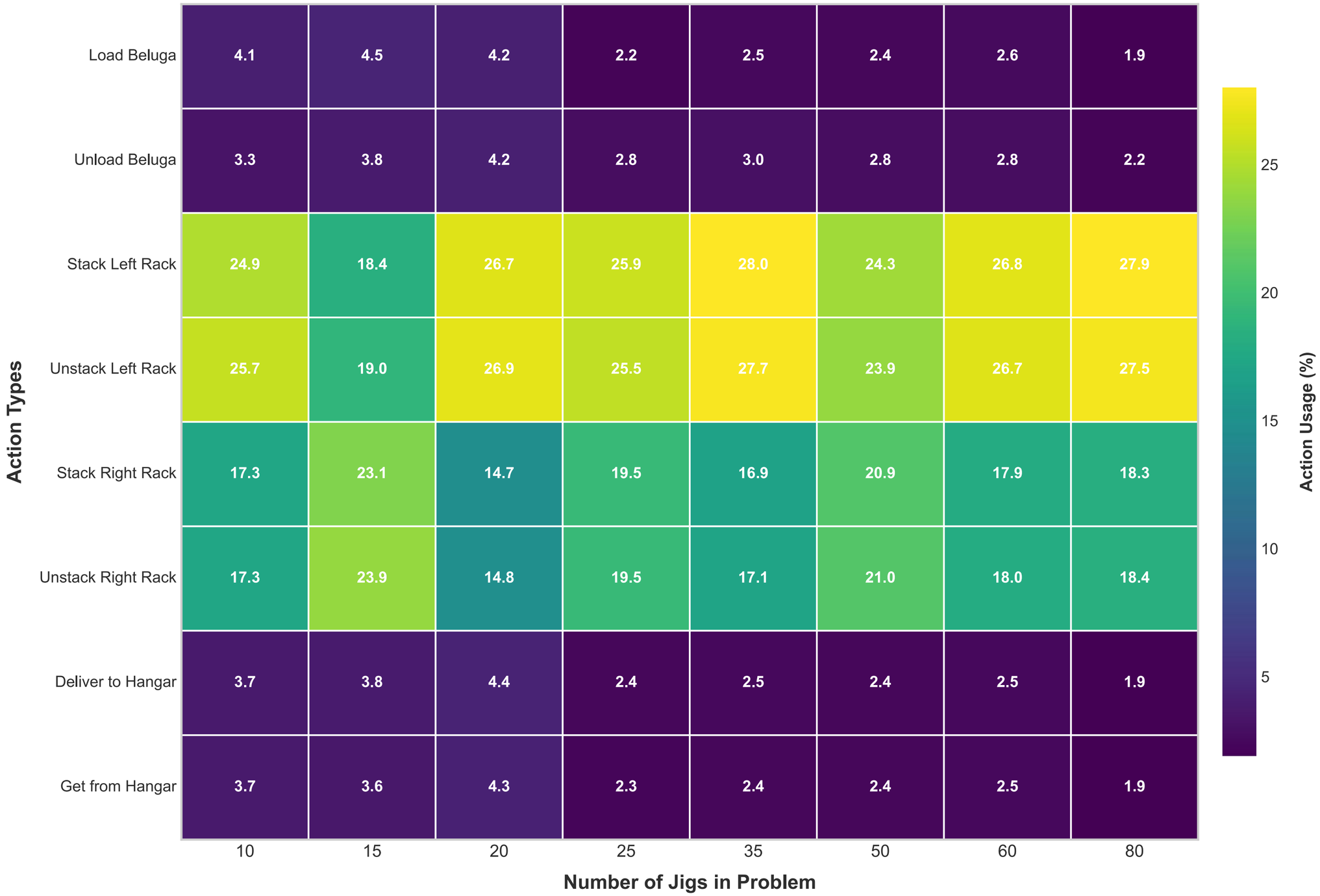


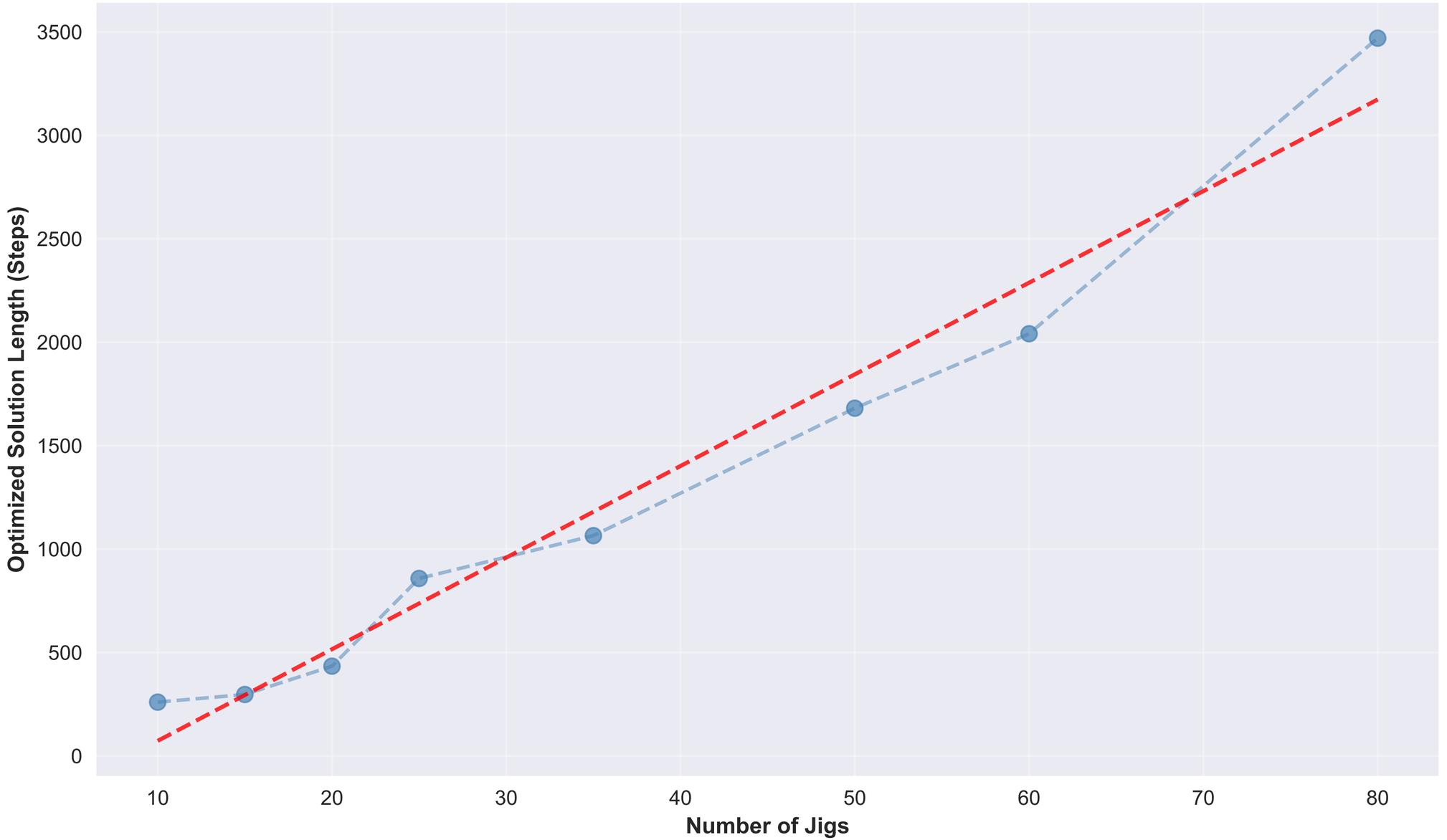
POSTPROCESSING & OPTIMIERUNG



Ergebnisse/Analyse


Computation Time (seconds)
0
10
20
30
40
50
60
10
20
30
40
50
60
70
80
Number of Jigs


Load Beluga
4.1 4.5 4.2 2.2 2.5 2.4 2.6 1.9
Unload Beluga
3.3 3.8 4.2 2.8 3.0 2.8 2.8 2.2
Stack Left Rack
24.9 18.4 26.7 25.9 28.0 24.3 26.8 27.9
Unstack Left Rack
25.7 19.0 26.9 25.5 27.7 23.9 26.7 27.5
Stack Right Rack
17.3 23.1 14.7 19.5 16.9 20.9 17.9 18.3
Unstack Right Rack
17.3 23.9 14.8 19.5 17.1 21.0 18.0 18.4
Deliver to Hangar
3.7 3.8 4.4 2.4 2.5 2.4 2.5 1.9
Get from Hangar
3.7 3.6 4.3 2.3 2.4 2.4 2.5 1.9
Action Types
10 15 20 25 35 50 60 80
Number of Jigs in Problem
25
20
15
10
5
Action Usage (%)


Optimized Solution Length (Steps)
3500
3000
2500
2000
1500
1000
500
0
10
20
30
40
50
60
70
80
Number of Jigs

LESSONS LEARNED

1. **Teamarbeit: sehr effektiv, gute Kommunikation**
   - Regelmäßige Team-Meetings waren entscheidend

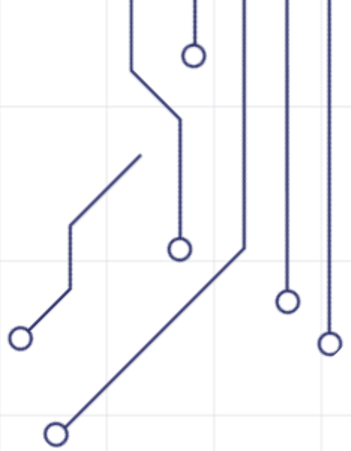
2. **Nutzung eines Kanban-Boards zur Aufgabenzuteilung**

3. **Analyse statt nur Optimierung!**

4. **Oft war es schwer, die Gesamtstruktur zu durchblicken**

   → **Einführung von „Pair-Programming“ am gemeinsamen Rechner**

FAZIT



1. **Erfolgreiche Entwicklung eines skalierbaren RL-Systems**

2. **Gutes Zusammenspiel von PPO + MCTS**

3. **Viel gelernt: Reinforcement Learning, Problemmodellierung & Teamarbeit**

4. **Offene Punkte:**
   - Lösungslänge weiter minimieren
   - Lösungsgarantie für generierte Probleme (Für die Trainings-Pipeline)

DANKE FÜRS ZUHÖREN.

Haben Sie noch Fragen?