

# **HTN-Planung unter Ressourcen- und Zeiteinschränkungen in hybriden Bergwerken**

---

## **Bachelor-Thesis**

vorgelegt von

Kilian Kramer

aus

Bergisch Gladbach

am

11. Januar 2021

Gutachter:

Prof. Dr. Alexander Ferrein

Dr. Stefan Schiffer

# **Abstrakt**

Die optimierte Einsatzplanung autonomer Fahrzeuge in hybriden Bergwerken ist eine komplexe und schwierige Aufgabe und nicht ausschließlich von den verfügbaren Ressourcen und der Zeit abhängig. Intelligente Koordination kann dazu beitragen, die Abläufe der Fahrzeugflotten nachhaltig zu verbessern. In der Praxis gibt es eine Vielzahl an Modellen, die das Thema Flottenmanagement auf unterschiedliche Weise angehen. In dieser Arbeit wird ein Ansatz aus der Künstlichen Intelligenz vorgestellt, der die hierarchische Task-Netzwerk-Planung für die Verbesserung der Prozessabläufe einsetzt.



## **Erklärung**

Hiermit versichere ich, dass ich die vorliegende Arbeit selbstständig verfaßt und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe, dass alle Stellen der Arbeit, die wörtlich oder sinngemäß aus anderen Quellen übernommen wurden, als solche kenntlich gemacht sind und dass die Arbeit in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegt wurde.

Aachen, den 11. Januar 2021

*K.Kramer*

---

Kilian Kramer



# Inhaltsverzeichnis

<b>1 Einleitung</b>	<b>1</b>
1.1 Einführung . . . . .	1
1.2 Ziel der Arbeit . . . . .	2
1.3 Übersicht der Kapitel . . . . .	3
<b>2 Stand der Technik und verwandte Arbeiten</b>	<b>5</b>
2.1 Flottenmanagement Strategien in der Industrie . . . . .	5
2.2 Verwandte Methoden in der KI-Forschung . . . . .	9
2.2.1 CSP . . . . .	9
2.2.2 Wissensbasierte Agenten . . . . .	10
2.2.3 Genetische Algorithmen . . . . .	11
2.3 HTN-Forschung . . . . .	11
<b>3 Mathematischer / technischer Hintergrund</b>	<b>17</b>
3.1 Klassische Planung . . . . .	17
3.2 STRIPS . . . . .	19
3.3 HTN-Planung . . . . .	20
3.4 SHOP3 . . . . .	23
3.4.1 Domäne und Problembeschreibung . . . . .	23
3.4.2 Plansuche . . . . .	24
3.4.3 Atome . . . . .	25
3.4.4 Axiome . . . . .	25
3.4.5 Operatoren . . . . .	26
3.4.6 Methoden . . . . .	27
<b>4 Eigener Ansatz</b>	<b>29</b>
4.1 Transportplanung in Bergwerken . . . . .	29
4.2 Das Modell . . . . .	31
4.3 Temporäre Planung . . . . .	35
4.4 Einordnung in das Gesamtsystem . . . . .	37
<b>5 Experimentelle Ergebnisse</b>	<b>39</b>
5.1 Anwendungsszenario 1 . . . . .	40
5.1.1 Planausgabe . . . . .	41
5.1.2 Evaluation . . . . .	42
5.2 Anwendungsszenario 2 . . . . .	45
5.2.1 Evaluation . . . . .	47

<b>6 Zusammenfassung</b>	<b>53</b>
6.1 Fazit . . . . .	53
6.2 Ausblick . . . . .	54
<b>A Implementierung</b>	<b>57</b>
A.1 Problembeschreibung . . . . .	57
A.2 Domäne . . . . .	60
<b>Literatur</b>	<b>69</b>





# Abbildungsverzeichnis

2.1	Die 1-Truck-n-Schaufeln Strategie . . . . .	6
2.2	Die m-Trucks-für-1-Schaufel Strategie . . . . .	8
2.3	Die m-Trucks-für-n-Schaufeln Strategie . . . . .	9
3.1	Repräsentation des Zustands durch Atome . . . . .	18
3.2	Veränderung des Zustands durch Aktionen . . . . .	20
3.3	Keine Transition . . . . .	20
3.4	Hierarchisches-Task-Netzwerk . . . . .	22
4.1	Einteilung der Mine . . . . .	30
4.2	Vermischen des Erzes . . . . .	30
4.3	Warteschlangensystem . . . . .	32
4.4	HTN-Planungssystem . . . . .	33
4.5	Zuweisung der Transportladungen . . . . .	34
4.6	Temporäre Planung mittels Zeitfenstern . . . . .	36
4.7	Blockierungen der Ressourcen . . . . .	37
4.8	Architektur des Projekts ARTUS . . . . .	38
5.1	Anwendungsszenario 1 . . . . .	41
5.2	Planausgabe von Shop . . . . .	42
5.3	Anwendungsszenario 1 - Ergebnisse der Gesamttonnage . . . . .	43
5.4	Beispiel - lange Wartezeiten . . . . .	43
5.5	Anwendungsszenario 2 . . . . .	45
5.6	Anwendungsszenario 2 - Ergebnisse der Gesamttonnage . . . . .	47
5.7	Gesamtwartezeit aller Fahrzeuge . . . . .	48
5.8	Zurückgelegte Gesamtstrecke aller Fahrzeuge . . . . .	49
5.9	Kumulierte Leerlaufzeiten aller Schaufeln . . . . .	50
6.1	Grafik zur Modellerweiterung . . . . .	55



# Kapitel 1

## Einleitung

### 1.1 Einführung

Das Projekt ARTUS (Autonomes robustes Transportsystem für hybride umweltschonende Rohstoffgewinnung auf Basis knickgelenkter Sonderfahrzeuge) ist ein Forschungsprojekt bei dem das Maskor Institut (Mobile autonome Systeme und kognitive Robotik) (Webseite: <https://www.maskor.fh-aachen.de/projects/ARTUS/>) der Fachhochschule Aachen (FH Aachen), die Rheinisch-Westfälische Technische Hochschule Aachen (RWTH) sowie weitere Unternehmen aus der Bergbauindustrie beteiligt sind. Ziel des Projekts ist das Entwickeln einer autonomen Flotte aus knickgelenkten Sonderfahrzeugen, die eigenständig die Rohstoffe in hybriden Bergwerken (Untertage und Übertage) zu den Prozessanlagen transportieren sollen. Hierfür werden von dem Maskor Institut zwei Minidumper ([https://www.wackerneuson.de/de/produkte/dumper/raddumper/-Modell 1501](https://www.wackerneuson.de/de/produkte/dumper/raddumper/-Modell-1501)) für den autonomen Betrieb umgebaut und mit verschiedenen Sensoren (UWB, IMU, GPS, Lidar, Kameras) ausgestattet. Die Fahrzeuge dienen als Testfahrzeuge auf dem Campusgelände der FH Aachen oder auf anderen leerstehenden Testgeländen. Die dafür zu entwickelnden Lernalgorithmen können dadurch trainiert und verbessert werden. Ziel des Projekts ist das System so zu konzipieren, dass es später auf größere Fahrzeuge (z.B. knickgelenkte Muldenkipper) angewendet werden kann. Neben der Entwicklung einer autonomen Fahrzeugarchitektur ist ein weiteres Projektziel ein intelligentes Flottenmanagementsystem zu implementieren. Das System soll auf der einen Seite die Fahrzeugflotte koordinieren und auf der anderen Seite die verfügbaren Ressourcen und die Prozesse innerhalb der Mine überwachen. Die Ergebnisse dieser Arbeit dienen als Grundlage für das Flottenmanagementsystem.

## 1.2 Ziel der Arbeit

Im Rahmen des Projekts ARTUS untersucht die vorliegende Arbeit den allgemeinen Transportprozess der Rohstoffe von Fahrzeugflotten (knickgelenkte Muldenkipper, Radlader usw.) in hybriden Bergwerken mit dem Fokus auf die Herausforderungen, die für die Transportplanung entstehen und entwickelt mit Hilfe methodischer Ansätze aus der Künstlichen Intelligenz ein Planungssystem, das optimierte Ablaufpläne auf höherer Planungsebene für eine autonome Fahrzeugflotte erzeugen kann.

Der automatisierte Transportprozess in einer Mine ist ein komplexes Planungsproblem, denn die Tourenplanung der Fahrzeuge ist von einigen Faktoren, wie der dynamischen Infrastruktur des Bergwerks, den täglichen Leitungsvorgaben, der Größe der Fahrzeugflotten und den Aktionsplanungen der Fahrzeuge untereinander abhängig. Dadurch steigt der Suchraum zum Finden eines reibungslosen und effizienten Ablaufplans. Temporäre Anpassungen sind durch neue Anweisungen von außen oder durch unerwartete Ereignisse erforderlich, sodass aufgrund der Komplexität des Problems ein leistungsstarkes und schnelles Planungssystem mit guten Heuristiken benötigt wird. Aus diesem Anlass modelliert die vorliegende Arbeit das Planungsproblem mit einem HTN-Planer (Hierarchical Task Network). HTN-Planer sind populäre Werkzeuge zur Modellierung von komplexen Planungsproblemen und gelten als besonders effizient, da sie die Möglichkeit bieten, Wissen über die Domäne zu implementieren. Abstrahiert besteht die Einsatzplanung der Fahrzeuge aus den einzelnen Fahrten zwischen den Ladestellen und den Entladestellen. Da es sich bei einer Fahrzeugflotte um ein Multiagentensystem handelt, entstehen zeitliche Restriktionen für die Fahrzeuge und Kollisionen an den Ressourcen können auftreten. Die dynamische Umgebung des Bergwerks und die Vorgaben an den Transportprozess müssen ebenfalls in die Planung einbezogen werden. Neben der temporären Planung und den Vorgaben an den Transportprozess steht die Optimierung der Flottenkoordination im Vordergrund. Ein guter Ablaufplan sollte versuchen die Produktion zu verbessern, wie zum Beispiel die Gesamttonnage zu maximieren oder Kosten, wie die Wartezeiten und die Leerlaufzeiten minimieren.

Für die Implementierung verwendet diese Arbeit ein Open Source HTN-Planungssystem namens SHOP3 (Simple Hierarchical Order Planner) [21, vgl. Goldman und Kutur,

2019]. SHOP3 (Download: <https://github.com/shop-planner/shop3>) ist in Common Lisp programmiert und ein Domänen unabhängiges Planungssystem mit integriertem Solver, der ermöglicht, Aktionspläne für ein beliebiges Planungsproblem automatisiert zu generieren. Als Eingabe erhält SHOP3 eine Domäne, die das Planungsproblem definiert. Da in SHOP3 temporäre Aktionen nicht standardmäßig implementiert sind, zeigt die Arbeit, wie SHOP3 modifiziert werden kann und verwendet dafür ein Konzept namens „Durative Actions“ [18, vgl. Fox und Long, 2003].

### 1.3 Übersicht der Kapitel

Im zweiten Kapitel werden der Stand der Technik und verwandte Arbeiten betrachtet. Das zweite Kapitel gliedert sich in drei Abschnitte. Zunächst werden verschiedene Strategien und Systeme zum Flottenmanagement diskutiert. Danach werden verwandte Methoden aus der Künstlichen Intelligenz vorgestellt, mit denen das Planungssystem auf eine andere Art und Weise modelliert werden kann. Abschließend betrachtet Kapitel 2 die Entwicklung und den Stand von HTN-Planungssystemen. In Kapitel 3 werden die Grundlagen über die Klassische Planung und die HTN-Planung besprochen sowie eine Einführung über SHOP3 und dessen zentralen Funktionen gegeben. Im vierten Kapitel wird der eigene Ansatz vorgestellt. Dafür wirft die Arbeit einen allgemeinen Blick auf das Thema hybride Bergwerke und charakterisiert die Merkmale des Transportprozesses, die im Modell berücksichtigt werden. Aufbauend darauf wird im zweiten Teil der Lösungsweg und das Konzept des Planungssystems präsentiert. Im fünften Kapitel wird die Arbeit evaluiert. Für die Evaluation wird das Planungssystem an Hand von zwei Anwendungsszenarien demonstriert, dessen Ergebnisse und Messungen bewertet werden. Zum Abschluss fasst das sechste Kapitel die gesamte Arbeit und die gewonnenen Erkenntnisse aus dem fünften Kapitel zusammen und gibt einen Ausblick darüber, wie das Planungssystem erweitert werden könnte.



## Kapitel 2

# Stand der Technik und verwandte Arbeiten

### 2.1 Flottenmanagement Strategien in der Industrie

Allgemein heißen die Systeme zur Überwachung der Fahrzeugflotten in den Bergwerken Dispatching Systeme (Versandsysteme). In der Industrie gibt es eine Vielzahl an unterschiedlichen Implementierungen der Systeme, denn in der realen Welt ist das Flottenmanagement eine komplexe Aufgabe. Entweder ist das System überlastet und die Fahrzeuge müssen warten bis eine Schaufel verfügbar ist oder das System nicht ausgelastet und die Schaufeln befinden sich im Leerlauf. Der Hauptnutzen eines Flottenmanagementsystems ist die Optimierung der Produktivität und die Kostenreduktion. Im Folgenden werden die verschiedenen Ansätze näher betrachtet, die in der einschlägigen Literatur diskutiert werden.

Eine allgemeine Klassifizierung der verschiedenen Flottenmanagementmodelle für den Betrieb im Tagebau zeigt die Veröffentlichung von Alarie und Gamache [1, vgl. Alarie und Gamache, 2002]. In ihrer Arbeit vergleichen die Autoren die verschiedenen Strategien zur Optimierung des Transportprozesses miteinander. Alarie und Gamache unterteilen die Systeme in zwei Kategorien: einstufige und mehrstufige Systeme. Die einstufigen Systeme sind die meist verbreitetsten Systeme in der Bergbauindustrie. Bei dem einstufigen Ansatz werden die Trucks ohne Berücksichtigung spezifischer Produktionsziele oder -nebenbedingungen zu den Schaufeln geschickt. Der Fokus liegt

hauptsächlich auf der Maximierung der aktuellen Produktivität und einem Nutzenvorteil der Schaufeln und Trucks. Es handelt sich häufig um heuristische Methoden, die auf bestimmten „Faustregeln“ basieren. Abbildung 2.1 zeigt das Modell eines einstufigen Systems. Die einstufigen Systemen bezeichnen Alarie und Gamache auch als „1-Truck-für-n-Schaufeln“-Strategien. Bei dieser Strategie fragt ein leerer Truck nach der nächsten Zuordnung zu einer Schaufel. Unter Berücksichtigung definierter Heuristiken sendet der Dispatcher das Fahrzeug zu der Schaufel, die das größte Potenzial bietet. Dieser Vorgang wird jedes Mal wiederholt, wenn ein leerer Truck nach einem neuen Auftrag fragt. Chaowasakoo et al. [11, vgl. Chaowasakoo et al., 2017], Munirathinam et al. [33, vgl. Munirathinam und Yingling, 1994] und Kolonja et al. [28, vgl. Kolonja et al., 1993] fassen die Heuristiken wie folgt zusammen:

- Minimierung der Wartezeiten der Trucks (1)
- Minimierung der Zykluszeit (2)
- Minimierung der Wartezeiten der Schaufeln (3)
- Minimierung der Schaufelsättigung (4)

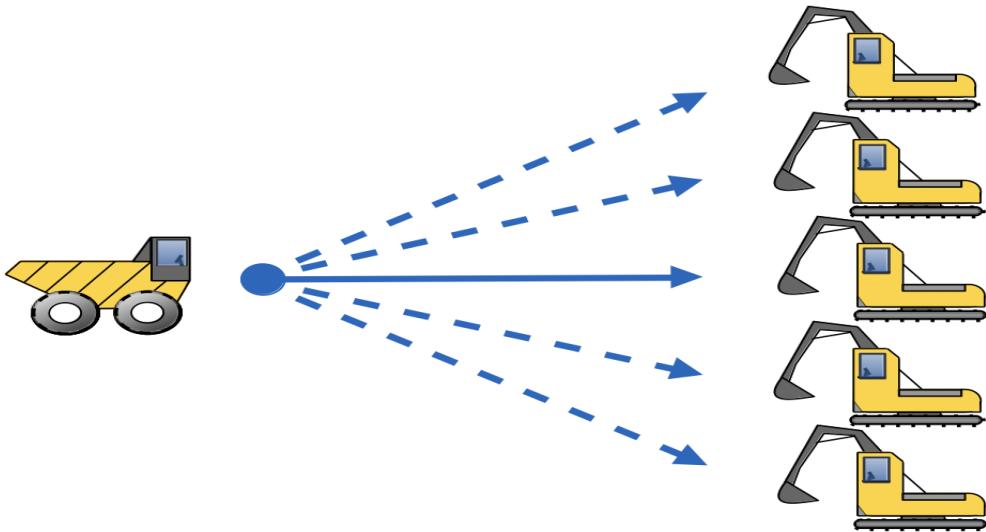


Abbildung 2.1: Die 1-Truck-n-Schaufeln Strategie

- (1) Bei der Minimierung der Wartezeiten der Trucks wird ein leerer Truck der Schaufel zugeordnet, bei der die Ladeoperation am frühstmöglichen Zeitpunkt beginnen kann. Diese Methode vermeidet die Bildung von Warteschlangen an den Schaufeln. Dieser

Ansatz eignet sich allerdings wenig für einen nicht ausgelasteten Betrieb, da die Wahrscheinlichkeit einer Warteschlange gering ist und die Zuordnung dann willkürlich erfolgt. Ferner kann diese Regel zu einer ungleichmäßigen Produktion führen. Denn es kann vorkommen, dass die Beladung an einer Schaufel in der Nähe trotz Wartezeiten früher beginnen kann und weiter entfernte Schaufeln ausgelassen werden [33, vgl. Munirathinam und Yingling, 1994].

- (2)** Das Ziel der Minimierung der Zykluszeit ist die Dauer einer Tour, die ein Truck zum Beladen an einer Schaufel und Entladen beim Brecher insgesamt benötigt, zu reduzieren. Der Fokus liegt auf der Maximierung der Gesamttonnage.
- (3)** Bei der Minimierung der Wartezeiten der Schaufeln wird der nächste leere Truck zu der Schaufel mit der längsten Leerlaufzeit oder zu der, die als Erstes verfügbar ist, geschickt. Diese Regel verteilt die Trucks gleichmäßig zu den Schaufeln. Der Nachteil ist, dass mit dieser Strategie weitere Kosten verbunden sind. Denn obwohl eine leere Schaufel in der Nähe verfügbar ist, kann ein Truck eine weiter entfernte Schaufel aufgrund einer längeren Leerlaufzeit anfahren [33, vgl. Munirathinam und Yingling, 1994].
- (4)** Die Schaufelsättigung wird als Verhältnis zwischen der gewünschten Anzahl und der tatsächlichen Anzahl an Trucks, die einer Schaufel zugewiesen werden, definiert [33, vgl. Munirathinam und Yingling, 1994]. Hierbei sendet der Dispatcher die Trucks in gleichen Zeitintervallen zu den Schaufeln mit dem geringsten Sättigungsgrad. Dadurch wird ein nicht im Leerlauf befindlicher Schaufelbetrieb, wenn genügend Fahrzeugkapazitäten vorhanden sind, aufrecht gehalten.

Die größten Nachteile der einstufigen Systeme sind, dass die Versandentscheidungen nicht unter Rücksichtnahme von zukünftigen Entscheidungen getroffen werden und dadurch auf lange Sicht negative Auswirkungen für das Gesamtziel entstehen. Ein mehrstufiges System hingegen unterteilt das Transportproblem in Teilziele und beobachtet dafür den Gesamtprozess. Diese Systeme bestehen in der Regel aus zwei Komponenten, einer höheren Planungskomponente, die ein Produktionsziel für jede Schaufel vorgibt und einer tieferen Planungskomponente, dem Dispatcher, der die Trucks den Schaufeln zuordnet. Die höhere Planungskomponente basiert oftmals auf linearen Opti-

mierungsprogrammen. Alarie und Gamache unterscheiden bei den mehrstufigen Systemen zwei Strategien. Bei der ersten Strategie, die „m-Trucks-für-1-Schaufel“-Strategie, werden die Schaufeln zunächst absteigend nach einem Prioritätsschema sortiert. Das Prioritätsschema basiert darauf, wie weit eine Schaufel in der Produktion hinter dem Zeitplan liegt. Abbildung 2.2 zeigt das Modell der „m-Trucks-für-1-Schaufel“-Strategie. Zunächst schlägt der Dispatcher der Schaufel mit der höchsten Priorität die nächsten m ankommenden Trucks vor. Anschließend ordnet das System den Truck mit den geringsten Kosten der Schaufel zu. Der Nachteil der „m-Trucks-für-1-Schaufel“-Strategie ist, dass diese Methode ebenfalls nur eine Schaufel bei den Versandentscheidungen berücksichtigt und dadurch die Auswirkungen auf die weiteren Produktionsabläufe nicht berücksichtigt werden.

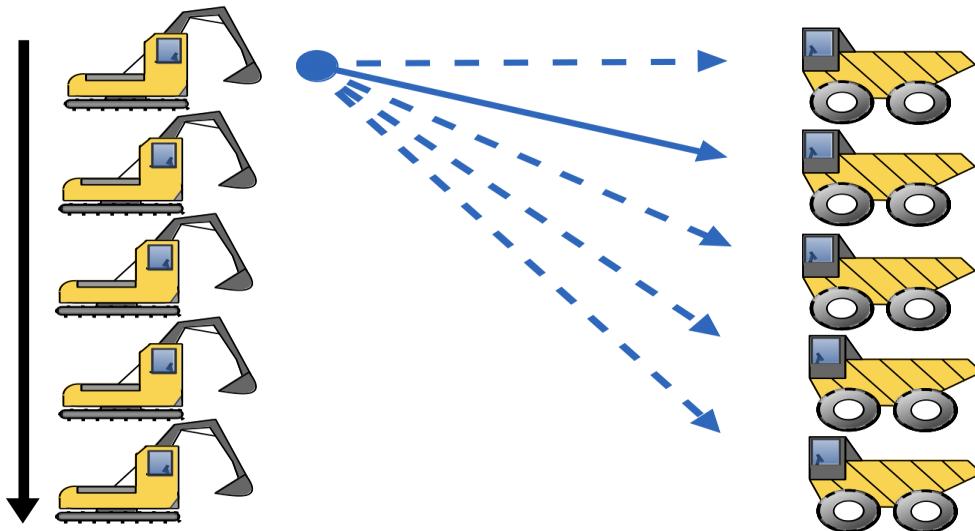


Abbildung 2.2: Die m-Trucks-für-1-Schaufel Strategie

Die zweite Strategie des mehrstufigen Ansatzes ist die „m-Trucks-für-n-Schaufeln“-Strategie. Diese Methode bezieht die nächsten m Trucks und n verfügbaren Schaufeln in die Planung ein. Der nächste leere Truck fragt für sich eine freie Schaufel an. Mittels einer kombinatorischen Optimierungsmethode wird die Schaufel zugewiesen, bei der die prognostizierte Produktivität aller Trucks und Schaufeln auf längere Sicht am Größten ist. Anschließend wird nur der Truck zugewiesen, der die Anfrage gesendet hat. Die Schwierigkeit dieser kombinatorischen Optimierungsstrategie ist, die Komplexität bei großen m und n zu reduzieren. Abbildung 2.3 zeigt die Strategie. Temeng et al. [50, vgl. Temeng et al., 1997] präsentieren ein Modell, das mehrere Schaufeln und Trucks bei einer Versandentscheidung berücksichtigt. Dafür wird eine Methode namens

Zielprogrammierung (Goal Programming) verwendet. Die Zielprogrammierung ist eine Optimierungsmethode zur Lösung von Problemen mit mehreren Zielen.

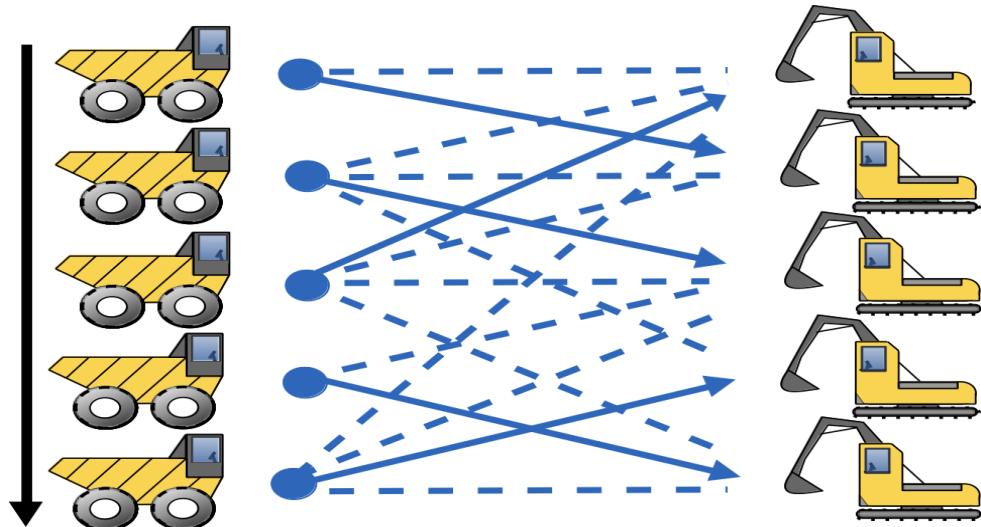


Abbildung 2.3: Die m-Trucks-für-n-Schaufeln Strategie

## 2.2 Verwandte Methoden in der KI-Forschung

Planung und Ablaufplanung (planning and scheduling) sind zentrale Teilgebiete der Künstlichen Intelligenz. Das Forschungsziel besteht darin, geeignete Aktionssequenzen für autonome Systeme (Agenten) zu entwickeln, die aus einem initialen Startzustand ein gewünschtes Ziel erreichen sollen. Neben dem hier vorgestellten Planungsanatz und den Beispielen aus dem Gebiet der HTN-Planung, gibt es zahlreiche andere weit verbreitete Konzepte, die das Thema Planung auf unterschiedliche Weise angehen. Im Folgenden werden einige dieser Techniken vorgestellt.

### 2.2.1 CSP

Ein CSP (Constraint Satisfaction Problem) drückt allgemein ein Planungsproblem unter Rand- und Nebenbedingungen in einer „faktorisierten Darstellung“ [39, Russel und Norvig, 2012, Seite 252] aus. Anstelle das Planungsproblem mit einzelnen Atomen auszudrücken, wie es in der klassischen- und HTN-Planung der Fall ist, wird jeder Zustand durch eine partielle oder vollständige Zuordnung sämtlicher Variablen, die das Planungsproblem beschreiben, abgebildet. Die zulässigen Variablenwerte werden

vorher in spezifischen Domänen, für jede Variable eine, festgelegt. Nebenbedingungen (Constraints) regeln die erlaubten Wertkombinationen zwischen den Variablen und ein CSP wird gelöst, indem jede Variable eine konfliktfreie Zuordnung erhält. Typischerweise werden Backtracking-Algorithmen oder der sogenannte AC-3 Algorithmus [30, vgl. Mackworth, 1977] auf solche in CSP-Form ausgedrückten Planungsprobleme angewendet. In der Praxis finden CSPs häufig Anwendung in der Arbeitsplanung oder bei logistischen Problemen. Beispiele hierzu geben TACT [45, vgl. Simonis et al., 1996] oder die Veröffentlichung von Neague et al. [36, vgl. Neagu et al. 2006], die das Konzept auf die Ablaufplanung bzw. Echtzeitsteuerung von Fahrzeuglotten anwenden. Es gibt verschiedene Open Source Constraint-Programming-Bibliotheken mit integrierten CSP-Solvern, wie zum Beispiel Gecode (C++) oder OptaPlanner (Java). Mit diesen Programmen können CSP-Probleme formuliert und automatisiert gelöst werden.

### 2.2.2 Wissensbasierte Agenten

Neben der Constraintprogrammierung (CP) gibt es einen anderen Ansatz, bei denen die Planungsprobleme in einer Aussagenlogik (auch als Boolesche Erfüllbarkeitsprobleme oder SAT bezeichnet) ausgedrückt werden. Derartige Modelle lassen sich auf weitaus komplexere Probleme anwenden, für die ein CSP allein nicht ausreichen würde und benötigen weniger Expertenwissen über die Domäne als die HTN-Planer. Der Nachteil ist, dass diese Systeme oftmals weniger effizient arbeiten, denn sie verwenden umfangreiche Inferenzmechanismen zum Ableiten der Informationen. Probleme in der Aussagenlogik sind dadurch gekennzeichnet, dass die Zustände durch Konjunktionen und Disjunktionen von booleschen Variablen (Literalen) repräsentiert werden. Im Gegensatz zu Atomen können Literale auch negiert sein, sodass sich das Planungssystem neben dem Planungsprozess zusätzlich eine Wissensbasis in Form von Klauseln (Klauseln sind Mengen von Literalen und jede Klausel repräsentiert einen anderen möglichen Zustand) über die aktuellen Beobachtungen des Planungsproblems aufbaut. Die Klauseln dienen als Wissensgrundlage für die weiteren Entscheidungen im Planungsprozess. Es gibt etliche SAT-Solver, wie z.B. CHAFF [32, vgl. Moskewicz et al., 2001], ein Solver der Millionen Variablen verarbeiten kann, oder MiniSat [14, Eén und Sörensson, 2003], eine Open Source Implementierung. Davis et. al zeigt, wie temporäre Aktionen (Durative Actions) in ein SAT-Modell überführt werden können [44, Davis et al., 2015].

### 2.2.3 Genetische Algorithmen

Aufgrund der Größe der Probleme, zum Beispiel gäbe es in einer Mine mit 30 Muldenkipfern und 100 zu absolvierenden Transportfahrten  $30^{100}$  verschiedene Kombinationen Fahrzeuge einzusetzen, werden in der Praxis andere Verfahren für die Optimierung von komplexen Problemen verwendet, da die Problematik es unmöglich macht, alle Möglichkeiten zu betrachten und eine optimale Lösung zu finden. Abhilfe schaffen genetische Algorithmen, die aus der Biologie inspiriert sind. Genetische Algorithmen (oder auch Evolutionäre Algorithmen genannt) generieren zunächst, wie bei einer Monte-Carlo-Simulation, eine vorgegebene Anzahl an stochastischen Lösungen. Die Anzahl aller gezogen Zufallsergebnisse wird als Population und ein einzelnes Ergebnis als Genom bezeichnet. Die Algorithmen entfernen dann den besseren Teil, auch Chromosom genannt, aus einem Genom und setzen dieses mit einem anderen Chromosom wieder zusammen. Dieser Schritt heißt Mutation und eine Fitnessfunktion überprüft, ob die Chromosomenpaare zueinander passen. Der gesamte Vorgang wird beliebig oft wiederholt und am Ende wählen die Algorithmen das beste Ergebnis aus. Beispiele für die Anwendung von Genetischen Algorithmen zur Tourenplanung in Bergwerken zeigen die Publikationen von Reynolds et al. [12, vgl. Reynolds et al., 2017] und Liu et al. [29, vgl. Liu et al., 2019]. Zäpfel et al. [53, vgl. Zäpfel et al., 2016] verwenden das Konzept für die Maschinenbelegungsplanung (Job Shop Scheduling).

## 2.3 HTN-Forschung

Dieser Abschnitt betrachtet den Stand heutiger HTN-Planungssysteme. Da die HTN-Panung aus der Klassischen Planung entstanden ist, wirft die Arbeit zunächst einen Blick auf die Geschichte der Klassischen Planung.

Die Komplexität von Planungsproblemen in der realen Welt ist eines der Hauptgründe, die die Forschung in der HTN-Planung sowie der Klassischen Planung und dem Entwickeln mathematischer Planungsmodelle und -sprachen vorantreibt. Ein Agent würde keinen vernünftigen Plan in annehmbarer Zeit finden, wenn dieser den Suchraum alleine mithilfe der Breiten- und Tiefensuche und ohne geeigneter Heuristiken traversiert. Denn ohne eine vernünftige Heuristik berücksichtigt der Agent auch alle unnützen Aktionen. Das bekannteste Beispiel für einen informierten Suchalgorithmus, der mit einer

Zielheuristik arbeitet, ist der A\*-Algorithmus [22, vgl. Nilsson et al., 1968]. Wenige Jahre später nach der Entwicklung von A\* entwarf der Erfinder Nils Nilsson im Rahmen der gleichen Arbeit für einen der ersten autonomen Roboter namens „Shakey“ die Planungssprache STRIPS [17, vgl. Fikes und Nilsson, 1971], die heute als eines der fundamentalen Arbeiten in der Klassischen Planung gilt. STRIPS (STanford Research Institute Problem Solver) ist eine Sprache zur Repräsentation von Planungsproblemen und ist aus der mathematischen Mengenlehre abgeleitet. In STRIPS verändern Aktionen die Zustandsräume der Agenten. Dabei haben die Aktionen, bzw. in STRIPS und SHOP3 heißen die Aktionen „Operatoren“, eine besondere Syntax. Die Operatoren bestehen immer aus zwei Teilen, den Vorbedingungen und den Effekten. Diese deterministische Darstellung der Operatoren ermöglicht den Suchraum des Planungsproblems einzuzgrenzen. Es gibt verschiedene Weiterentwicklungen von STRIPS, wie zum Beispiel ABSTRACTIPS (Abstraction-Based STRIPS) [40, vgl. Sacerdoti, 1974], eine der ersten Arbeiten zur Abstraktion in der Planung oder ADL (Action Description Language) [37, vgl. Pednault, 1989][vgl. Pednault, 1989]. Die bekannteste und meist verwendetste Sprache zur Darstellung von Planungsproblemen der heutigen Zeit ist die Planungssprache PDDL (Planning Domain Definition Language), die auf den beiden Sprachen STRIPS und ADL basiert. PDDL wurde 1998 von Drew McDermott et al. für die Internationale Konferenz für Künstlich Intelligente Planungssysteme (International Conference on Artificial Intelligence Planning Systems - kurz: AIPS) entwickelt [20, vgl. Ghallab et al., 1998]. Die AIPS ist eine seit 1992 stattfindende Konferenz über den aktuellen Stand der Forschung und Technik im Bereich der künstlich intelligenten Planungssysteme und ist z.B. von zentraler Bedeutung für das Produktionsmanagement, Raumfahrtsysteme, Robotik, Internet oder militärische Anwendungen. Die AIPS wird seit 2003 im Rahmen der ICAP (International Conference on Automated Planning and Scheduling - <https://www.icaps-conference.org>) organisiert, in dessen Zusammenhang auch die Europäische Konferenz für Planung (ECP) stattfindet. Auf der ICAP wird heute alle zwei Jahre die IPC (International Planning Competition) ausgetragen, ein Wettbewerb mit unterschiedlichen Disziplinen in denen die Planungssysteme miteinander verglichen und bewertet werden. Seit der AIPS-98 (und der ersten IPC) wird PDDL als Standardsprache auf den internationalen Planungswettbewerben eingesetzt und sorgt damit für einen enormen Fortschritt an verfügbaren Benchmarks im Bereich der autonomen Planung. Es gibt einige Weiterentwicklungen von PDDL, eine

Vorgängerversion zur aktuellsten Version von PDDL (aktuell HDDL), ist PDDL2.1. Mit PDDL2.1 wurden die temporären Aktionen (Durative Actions) eingeführt [18, vgl. Fox und Long, 2003]. Neben PDDL gibt es zahlreiche Abwandlungen, wie zum Beispiel MAPL (Multi-Agent-Planning-Language) [9, vgl. Brenner, 2003], eine Erweiterung zu PDDL2.1 für Multi-Agenten-Systeme, PPDDL (Probabilistic PDDL) [52, vgl. Younes und Littman, 2004], mit der sich Unsicherheiten und Markov-Entscheidungsprobleme (Markov Decision Process - kurz: MDP) realisieren lassen oder NDDL (New Domain Definition Language) [42, vgl. Bernardini und Smith, 2007], eines von der NASA entwickelten Planungssystem.

Zum ersten Mal wurde der ICP-Wettbewerb auf der ICAP 2020 ausschließlich für HTN-Planer ausgelegt (siehe: <http://gki.informatik.uni-freiburg.de/competition/>) [6, vgl. Behnke et al., 2019], da die Popularität der HTN-Planer bereits seit einigen Jahren größer geworden ist. Im Wesentlichen unterscheidet sich ein HTN-Planungssystem von einem klassischen Planungssystem darin, dass der klassische Planer mit nur einem Typ von Aktionen plant, den primitiven Aktionen. Hingegen arbeiten HTN-Planer mit zwei unterschiedlichen Typen von Aktionen, den abstrakten und den primitiven Aktionen. Durch das Zusammensetzen der primitiven Aktionen zu höheren (abstrakten) Aktionen entsteht ein hierarchisches Aufgabennetzwerk (Hierarchical Task Network), das mehr Wissen über die Domäne besitzt und durch die Vernetzung den Suchraum stärker determiniert. Zu den bekanntesten älteren HTN-Planern gehören:

- NOAH (Nets of Action Hierarchies), der erste HTN-Planer [41, vgl. Sacerdoti, 1975]
- NONLIN (Non-Linear) [49, vgl. Tate, 1976]
- SIPE (System for Interactive Planning and Execution)  
bzw. SIPE2 [51, vgl. Wilkins, 1990]
- O-Plan (Open Plan Architecture) [49, vgl. Currie und Tate, 1990]  
und O-Plan2 [48, vgl. Tate et al., 1994]
- SHOP [34, Nau et al., 1999]  
und SHOP2 [35, vgl. Nau et al., 2003]
- UMCP (Universal Method Composition Planner) [15, vgl. Erol, 2003]

- SIADEX [4, vgl. et al. Castillo, 2005]

Georgievski und Aiello [19, vgl. Georgievski und Aiello, 2015] vergleichen und analysieren in ihrer Arbeit die genannten HTN-Planer hinsichtlich der Domänenstruktur, Ausdrucksstärke, Laufzeit, Kompetenz und Anwendbarkeit. Aufgrund der Popularität wurden in den letzten Jahren zahlreiche neue HTN-Planungssysteme entwickelt und viele für die Klassische Planung entwickelten Methoden für die HTN-Planung übernommen.

Zu den neueren HTN-Planungssystemen gehören [6, vgl. Behnke et al., 2019]:

- FAPE (Flexible Acting and Planning Environment) [13, vgl. Dvorák et al., 2014]
- HTN2STRIPS [2, vgl. Alford et al., 2016]
- GTOHP (Ground Total Order Hierarchical Planner) [38, vgl. Ramoul et al., 2017]
- totSAT (Totally-Ordered Hierarchical Planning through SAT) [7, vgl. Behnke et al., 2018]
- PANDA (Planning and Acting in Network Decomposition Architecture) [8, vgl. Bercher et al., 2017] und PANDApro [26, vgl. Höller et al., 2019]
- Tree-Rex (Tree-like Reduction Exploration) [43, vgl. Schreiber et al., 2019]
- partSAT (Partial Order in SAT-based HTN Planning) [27, vgl. Behnke et al., 2019]

FAPE ist ein auf ANML basierender HTN-Planer, der Planen und Handeln miteinander kombiniert und für Echtzeitsysteme, wie etwa einen Serviceroboter, weiterentwickelt wird. ANML (Action Notation Modeling Language) [vgl. Smith, Frank und Cashing, 2008], eine alternative Sprache zu PDDL, kombiniert die temporäre Planung mittels Zeitleisten (Timelines) mit der HTN-Planung. Während der Ausführung kann FAPE, die in den Plänen gesetzten Zeitstempel mit den Zeitpunkten der Beobachtungen synchronisieren und führt dafür Planreparaturen durch oder ersetzt veraltete Pläne durch neue Pläne. HTN2STRIPS kann HTN-Planungsprobleme in ein klassisches Planungsproblem automatisiert übersetzen. GTOHP wendet eine Methode zur Effizienzsteigerung, das Erdungsverfahren (Grounding), auf die HTN-Planer an. Das Erdungsverfahren ist von zwei Suchverfahren aus der Klassischen Planung namens Fast-forward [24, vgl. Hoffmann, 2001] und Fastdownward [23, vgl. Helmert, 2006] inspiriert.

Hierbei werden zunächst in einem Vorberechnungsschritt alle möglichen Wertkombinationen der Operatoren und Methoden instanziert. So lässt sich die Erreichbarkeit der Ziele schneller erfassen, Heuristiken ableiten und die Pläne können beispielsweise auch in den CSP- und SAT-Formalismus übersetzt werden. totSAT und Tree-Rex übersetzen vollständig geordnete HTN-Planungsprobleme in ein SAT-Problem. totSAT verwendet zum effizienteren Planen eine maximale Zerlegungstiefe. PANDA basiert auf einer domänenunabhängigen Heuristik, der mit Hilfe einer Datenstruktur namens “Task Decomposition Graph” den Suchraum eingrenzen kann, indem nicht erreichbare Ziele ignoriert werden. partSAT überführt partiell geordnete HTN-Planungsprobleme in die Aussagenlogik.

Zum Anlass der HTN-IPC auf der ICAP 2020 wurde eine neue Version von PDDL entwickelt, HDDL. HDDL (Hierarchical Domain Definition Language) [25, vgl. Höller et al., 2019] erweitert PDDL um ein paar Elemente. Mit *:method* und *:task*. können zusammengesetzte Methoden im HDDL-Formalismus definiert werden. Algorithmus 1 zeigt ein Beispiel einer solchen Aktion für eine Transportfahrt eines Fahrzeugs.

```

1 (:task drive-to :parameters (?t - truck ?from ?to - location)... )
2 (:task load-truck :parameters (?i1 ?i2 - inventory ?t - truck ?l - loader)... )
3 (:task unload-truck :parameters (?i1 ?i2 - inventory ?t - truck ?u - unloader).)
4 (:method haulage-tour
    :parameters (?it ?il ?iu - inventory ?t - truck
                ?l - loader ?u - unloader ?pos ?from ?to - location)
    :ordered-subtasks (and
        (drive-to ?t ?pos ?from)
        (load-truck ?it ?il ?t ?l)
        (drive-to ?t ?from ?to)
        (unload-truck ?it ?iu ?t ?u)))

```

**Algorithm 1:** HDDL-Beispiel einer High-Level-Aktion für eine Transportfahrt



## Kapitel 3

# Mathematischer / technischer Hintergrund

In diesem Kapitel wird eine kurze Einführung über das in dieser Arbeit verwendete Programm SHOP3 gegeben. Zum besseren Verständnis werden dafür zunächst die Grundlagen der Klassischen Planung und der HTN-Planung zusammengefasst.

### 3.1 Klassische Planung

Ein zentrales Thema der KI Forschung sind rationale Entscheidungen und die damit verbundenen Aktionen des Agenten. Mit diesem Teilgebiet befasst sich die Klassische Planung, die als Ziel das Entwickeln von Aktionsplänen hat. Ein Aktionsplan ist eine Sequenz von Aktionen, die aus einem initialen Startzustand einen gewünschten Zielzustand erreicht. Die deskriptiven Modelle, die ein Planungsproblem beschreiben, werden üblicherweise als Planungsdomäne bezeichnet. Eine Domäne besteht im Grunde aus vier Teilen [vgl. Ghallab, Nau und Traverso, 2016 - Seite 25]:

- Zustände
- Aktionen
- Zustandsübergangsfunktionen
- Kostenfunktionen

Die Zustände beschreiben den Status, in dem sich das Planungssystem befindet und die Aktionen verändern diese Zustände. Eine Zustandsübergangsfunktion definiert die

Veränderung, die durch die Aktion ausgelöst wird. Die Kostenfunktion bilden die Kosten einer Aktion ab. Ein Zustand wird in der Klassischen Planung als eine Konjunktion von Atomen repräsentiert. Atome sind Variablen, die einen Teil des aktuellen Zustands definieren. Ein komplexeres Planungsproblem hat eine Vielzahl solcher grundlegenden Atome. Dabei kann ein Atom alles Beliebige repräsentieren, die Bedeutung der Atome ist vom Modelleur und der Umsetzung des Planungsproblems abhängig. Ein Atom kann zum Beispiel ein existierendes Objekt, die Position des Agenten, eine Tatsache, das Wetter oder den Füllstand des Fahrzeugs abbilden (Abbildung 3.1). Alle Atome zusammen bilden den aktuellen Weltzustand. Im folgenden Beispiel wäre ein Atom *WEATHER(snow)* oder *AT(robot1, location\_A)*, das mehrere Werte besitzt. Zum Beispiel wäre *WEATHER(snow)* auch als *SNOWING* darstellbar, die Syntax der Atome spielt dabei erst bei dem verwendeten Planungssystem eine Rolle. In der Klassischen Planung wird die Annahme der geschlossenen Welt den Planungsdomänen unterstellt. Die Annahme der geschlossenen Welt besagt, dass negative Atome in der Domäne nicht erlaubt sind und nur das wahr ist, was auch bekannt ist Russel und Norvig, 2004 - Seite 439. Ein Aussage wie etwa *not(CUBE(b))* ist somit wahr, genauso impliziert die obige Domäne ein *not(ATOM\_XYZ)*, da es nicht explizit angegeben werden muss und folglich alles der Welt nicht bekannte automatisch *not(...)* ist, also nicht zutrifft. Damit sich der Zustand der Welt verändern kann, werden Aktionen benötigt. Aktionen haben in den meisten Planungsdomänen eine bestimmte Form. Die hier verwendete Form der Aktionen basiert auf der formalen Sprache STRIPS.

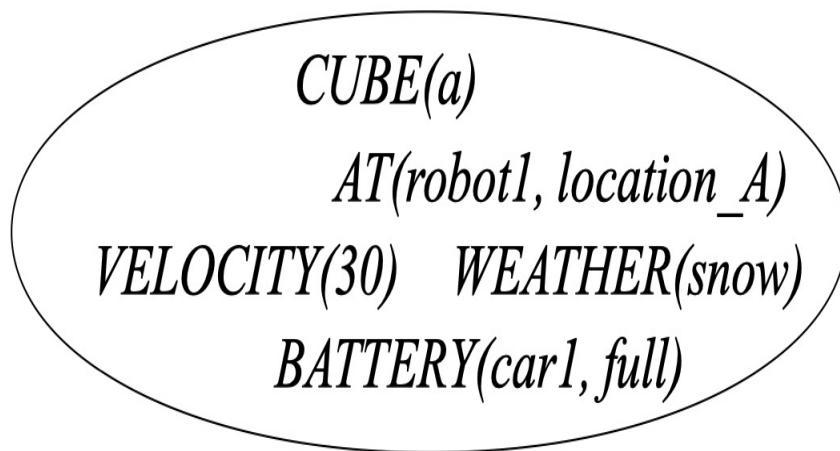


Abbildung 3.1: Repräsentation des Zustands durch Atome

## 3.2 STRIPS

STRIPS (Stanford Research Institute Problem Solver) ist eine von Richard Fikes und Nils Nilsson im Jahre 1971 veröffentlichte formale Sprache, um Planungsprobleme zu lösen. Es gibt eine Vielzahl von entwickelten Methoden und Sprachen aus der Familie der Planungsprobleme, allerdings hat sich STRIPS aufgrund des einfachen Formalismus in der Robotik durchgesetzt. STRIPS ist modellunabhängig. Das heißt, das STRIPS nur beschreibt, wie ein Problem gelöst wird, sodass es auf beliebige Domänen angewendet werden kann. Die Aktionen heißen in STRIPS (sowie in SHOP3) Operatoren und diese können das Weltmodell in ein neues Weltmodell transformieren. Die Operatoren haben dabei immer die gleiche Form [17, vgl. Fikes und Nilsson, 1993]:

```
if preconditions
then ( retract deletions assert additions )
```

**Algorithm 2:** Aufbau der Operatoren in Strips

Der Operator besteht aus zwei Teilen, den Vorbedingungen (preconditions) und den Nachbedingungen (deletions und additions). Ein Operator kann angewendet werden, wenn die Liste in den Vorbedingungen (preconditions) wahr ist. Wenn die Vorbedingungen im aktuellen Weltmodell zutreffen, werden alle Operationen im Rumpf ausgeführt, die den Zustand der Modellwelt verändern. Dies geschieht durch die Listen *deletions* und *additions*. In der Delete-Liste werden all die Atome aufgelistet, die aus dem aktuellen Modellwelt entfernt werden sollen. In der Add-Liste werden all die Atome aufgelistet, die der Modellwelt hinzugefügt werden sollen. Dies können veränderte alte oder neue Atome sein. Das folgende Beispiel zeigt einen Operator, der einen Roboter von Position A zu Position B bewegt.

```
if ( robot, at(robot, A) )
then ( retract ( at(robot, A) ) assert ( at(robot, B) )
```

**Algorithm 3:** Beispiel-Operator in Strips

Wenn *robot* und *at(robot, A)* wahr ist, wird dadurch die in Abbildung 3.2 abgebildete Zustandsveränderung hervorgerufen. Würde zum Beispiel nur *at(robot, A)* zutreffen, könnte der Operator nicht angewendet werden. Genauso könnte der Operator nicht angewendet werden, wenn sich der Roboter zunächst nicht bei A befindet. In beiden Fällen würde sich der Zustand des Planungssystems nicht verändern (Abbildung 3.3).

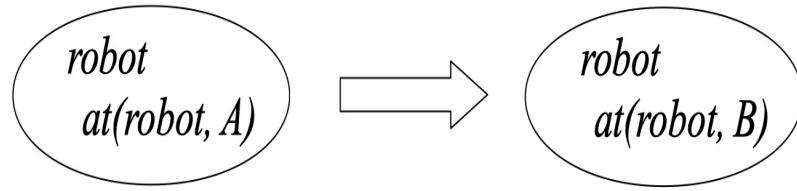


Abbildung 3.2: Veränderung des Zustands durch Aktionen

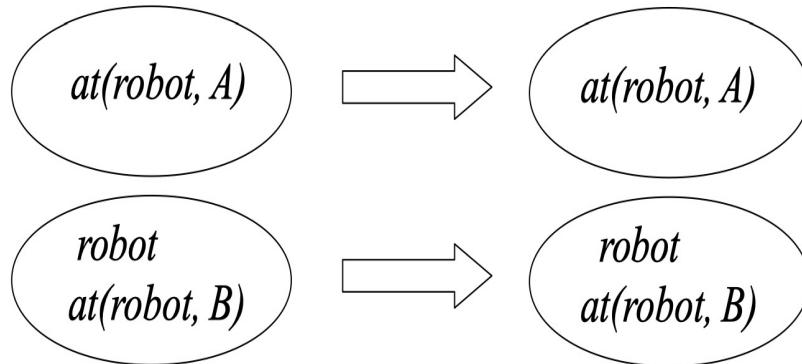


Abbildung 3.3: Keine Transition

### 3.3 HTN-Planung

Eine Problematik, die von der Klassischen Planung nicht berücksichtigt wird, ist die Aufgabe die Komplexität in den Griff zu bekommen. Planungsprobleme in der realen Welt haben oftmals weitaus größere Domänen, insbesondere wenn dem Planungssystem hunderte Aktionen zur Verfügung stehen, mit mehreren Agenten gleichzeitig geplant oder die Zeit berücksichtigt werden muss. Ein umfangreicher Aktionsplan mit  $n = 1000$  Aktionen und  $z = 10$  zulässigen Aktionen in jedem Zustand würde  $O(n) = 10^{1000}$  Aktionskombinationen im schlimmsten Fall durchrechnen müssen. Eine weitere Überlegung zu diesem Thema ist, wie Menschen routiniert Dinge im Leben planen. Ein Plan die Stadt zu besuchen würde, nicht damit beginnen als Erstes die Türklinke des Hauses zu benutzen. Zwar wäre diese Aktion durchaus Teil eines vollständigen Plans, allerdings wird diese, sowie viele andere Details erst zeitnah vom Gehirn geplant, sodass der Plan die Stadt zu besuchen eher verallgemeinernde höhere Aktionen wie „Fahre mit dem Auto zum Parkhaus“, „Besuche das Einkaufszentrum“, „Gehe ins Museum“ usw. beinhaltet. Daher muss dem Planungssystem eine Heuristik implementiert werden, um

ein Problem auf abstrakte Weise lösen zu können. Diese Idee liegt den Hierarchischen-Task-Netzwerken (kurz: HTN) zu Grunde, bei denen ein Problem auf mehreren Ebenen dekomponiert wird. Ein HTN-Planer zerlegt im Allgemeinen zusammengesetzte Aufgaben (in SHOP heißen diese Methoden) rekursiv in immer kleinere Aufgaben. Die nicht weiter verfeinerten Aufgaben, die vom Agenten direkt ausgeführt werden können, heißen primitive Aktionen. Eine höhere Aufgabe, wie beispielsweise „Verlasse das Haus“, könnte durch die primitiven Aktionen „Gehe in den Flur“, „Schuhe anziehen“ und „Öffne die Tür“ realisiert werden. Eine höhere (abstrakte) Aktion kann auch weitere abstrakte Aktionen, oder beide Aktionstypen, primitive und abstrakte Aktionen, kombiniert enthalten. Auf diese Weise wird ein hierarchisches Netzwerk aus Aktionen konstruiert. Eine weitere Besonderheit an HTN-Planungssystemen ist, dass die Aktionen in einer zusammengesetzten Aktion bei Zerlegung nicht zwingend geordnet ausgeführt werden müssen, sodass durch die partielle Anordnung das Planungssystem eine Aktion auf unterschiedliche Weise zerlegen kann. Dadurch können beispielsweise die Aktionen im Plan simultan ausgeführt werden, nachteilig würde das auch den Suchraum vergrößern. Eine Möglichkeit den Suchraum einzuschränken, ist vernünftige Vorbedingungen für die höheren Aktionen zu definieren, denn diese besitzen genauso wie die Operatoren Vorbedingungen. Anstelle der Add- und Delete-Listen im Rumpf werden dort die auszuführenden Aktionen definiert. Der Schlüssel von HTN-Planern ist, also Wissen in Form einer Planbibliothek aus Methoden aufzubauen. Jedoch muss ein intelligenter HTN-Planer nicht zwingend mühsam von Hand konstruiert werden, sondern kann auch neue Methoden selbstständig erlernen. Hierfür könnten zum Beispiel die erzeugten Pläne abgespeichert und auf wiederkehrende Muster bzw. identische Aktionsfolgen untersucht werden, die dann zu einer neuen verallgemeinernden Methode vom Computer zusammengefasst werden. In der Regel sowie in dieser Arbeit wird ein HTN-Planungsproblem mit einer einzigen höheren Aktion namens „Act“ [39, vgl. Russell und Norvig, 2012 - Seite 485] zerlegt. Abbildung 3.4 (auf der nächsten Seite) zeigt das Beispiel eines Hierarchischen-Task-Netzwerks für den Stadtbesuch.

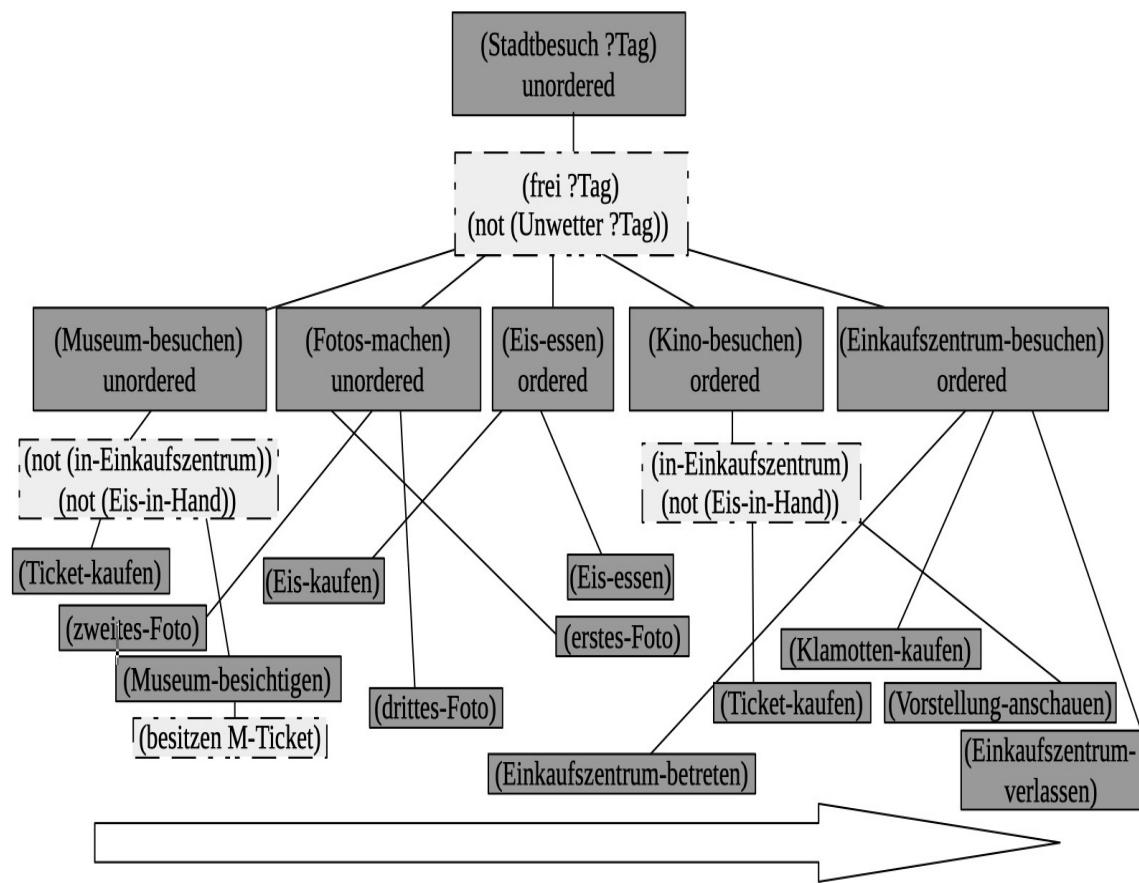


Abbildung 3.4: Hierarchisches-Task-Netzwerk

## 3.4 SHOP3

Aufbauend auf den Grundlagen über die HTN-Planung im vorangegangen Abschnitt werden in diesem Kapitel die zentralen Funktionen von SHOP3 [21, vgl. Goldman und Kutur, 2019] (Download: <https://github.com/shop-planner/shop3>) vorgestellt. Die Implementierung des für diese Arbeit entwickelten FLottenmanagementsystems ist im Anhang für Beispielzwecke zu finden. Für weitere spezifische Programmierungsfragen wird auf das Handbuch von SHOP3 verwiesen (SHOP3 Manual). SHOP3 ist in Common Lisp programmiert. Zum besseren Syntax-Verständnis sei daher erwähnt, dass alle Elemente (Variablen, Funktionen, Ausdrücke usw.) in Common Lisp in Klammern „(...)“ verschachtelt werden. Die Entwickler von SHOP3 empfehlen zur einfacheren Nutzung „Steel Bank Common Lisp“ (SBCL) und „Superior Lisp Interaction Mode for Emacs“ (SLIME) in Kombination mit Emacs (Download: <https://www.gnu.org/software/emacs/>) zu installieren. Außerdem wird vor dem Start der Programmierung empfohlen, die Compiler-Einstellungen anzupassen, da SHOP3 bei komplexeren Domänen Speicherplatzprobleme verursachen kann (SHOP3 Readme). Hierfür muss entweder in der Konfigurationsdatei von SBCL, diese lässt sich in der Benutzeroberfläche von Emacs mit der Tastenkombination *[Steuerung + X + F]* unter „*~/.sbclrc*“ finden, oder in der Projektdatei selbst (vor dem eigentlichen Quellcode) folgender Code ergänzt werden:

```
(DECLAIM (OPTIMIZE (speed 3) (space 3)))
```

**Algorithm 4:** Compiler-Einstellungen für SHOP3

### 3.4.1 Domäne und Problembeschreibung

SHOP3 erhält, wie im Allgemeinen alle KI-Planungssysteme, zwei Dinge als Eingabe. Zum einen die Domäne, eine Beschreibung der Planungswelt mit allen Methoden, Operatoren und Axiomen und deren Relationen und zum anderen die Problembeschreibung, die aus zwei Listen besteht. Die erste Liste enthält alle Atome, die das Planungsproblem im Ausgangszustand definieren und die zweite Liste ist eine Aufgabenliste, bestehend aus Methoden und Operatoren, die dekomponiert und ausgeführt werden sollen. Die Domäne wird mit dem Schlüsselwort *defdomain* definiert, gefolgt von dem Domänenamen („beispieldomäne“). Im Rumpf werden alle Methoden, Operatoren und Axiomen definiert. Externe Funktionen, die von der Domäne ausgehend aufgerufen wer-

den, werden außerhalb implementiert:

```
(defdomain beispieldomäne
  ( Domänenrumpf )
)
```

**Algorithm 5:** Aufbau der Domäne

Eine Problembeschreibung beginnt mit dem Schlüsselwort *defproblem*, gefolgt von dem Problemnamen und dem Domänennamen, auf die sich das Problem bezieht. Im Rumpf stehen die Atomliste und die Aufgabenliste:

```
(defproblem beispielproblem beispieldomäne
  ( Atomliste )
  ( Aufgabenliste )
)
```

**Algorithm 6:** Aufbau der Problembeschreibung

### 3.4.2 Plansuche

In SHOP3 ist bereits standardmäßig ein Solver implementiert, der die Aktionspläne zu den modellierten Domänen und Problembeschreibungen automatisiert erstellt. Der Solver wird mit der Funktion *find-plans*, gefolgt von einem Apostroph und dem Namen der Problemschreibung aufgerufen (siehe Algorithmus 7). Zusätzlich können der Funktion verschiedene Argumente mitgegeben werden. Mit *:which* und einem Wert *:wert* kann die Art der Suche festgelegt werden. Folgende Werte sind möglich:

- *:first* - Tiefensuche. Der erste gefundene Plan wird ausgegeben. (Default-Wert)
- *:all* - Tiefensuche. Alle Pläne werden ausgegeben.
- *:shallowest* - Tiefensuche. Der Erste und kürzeste Plan wird ausgegeben.
- *:all-shallowest* - Tiefensuche. Alle kürzesten Pläne werden ausgegeben.
- *:id-first* - Iterative Tiefensuche. Der erste gefundene Plan wird ausgegeben.
- *:id-all* - Iterative Tiefensuche. Alle kürzesten Pläne werden ausgegeben.
- *:random* - Zufällige Suche. Nicht für den normalen Gebrauch gedacht.

Mit *:verbose* gefolgt von *:stats*, *:plans* oder *:long-plans* können erweiterte Informationen zu den Plänen ausgegeben werden. Außerdem sucht die *find-plans*-Funktion solange

nach einem Plan, bis alle Zerlegungsmöglichkeiten bei der Plansuche durchgeführt worden sind. Alternativ kann die Suche mit `:time-limit n` frühzeitiger gestoppt werden, wobei `n` das Zeitlimit in Sekunden angibt. Folgender Beispiel-Funktionsaufruf gibt alle kürzesten Pläne, die innerhalb von 10 Sekunden gefunden werden mit der zugehörigen Aktionsliste aus:

```
(find-plans 'beispielproblem :which :all-shallowest :verbose :plans
:time-limit 10)
```

**Algorithm 7:** Aufruf der find-plans-Funktion

Allerdings würde dieser Funktionsaufruf ohne die Domäne und eine Problembeschreibung einen Fehler zurückgeben.

### 3.4.3 Atome

Atome werden in SHOP3 wie folgt definiert:

```
(predicate t1 t2 ... tn)
```

**Algorithm 8:** Atome in Shop

Das *predicate* ist die Bezeichnung des Atoms. Jedes  $t_n$  repräsentiert einen Term des Atoms und kann entweder eine Zahl, eine Konstante, eine Variable oder eine Liste sein. Folgende Atome definieren zum Beispiel ein Fahrzeug oder die Distanz zwischen zwei Orten.

```
(truck truck1)
(distance shovel1 crusher1 800)
```

### 3.4.4 Axiome

In Algorithmus 9 ist der Aufbau eines Axioms dargestellt. In SHOP3 schlussfolgern Axiome Tatsachen aus den Informationen des aktuellen Weltmodells.  $a$  und  $E$  sind logische Ausdrücke.  $a$  wird abgeleitet, wenn eines der  $E$  wahr ist. Die Namen der  $E_n$  sind optional.

```
(:- a
    name1 E1
    name2 E2
    ...
)
```

**Algorithm 9:** Axiome in Shop

Beispielsweise würde das folgende Axiom ableiten, dass ein Fahrzeug den Status vollgeladen hat, wenn das Fahrzeug nicht leer ist:

```
(:- (status ?truck loaded) ((payload ?truck ?val) (not (= ?val 0))))
```

### 3.4.5 Operatoren

Wie bereits erwähnt, repräsentieren die Atome den aktuellen Weltzustand und definieren in der Atomliste der Problembeschreibung den Startzustand. Auf die Atome in der Problembeschreibung wendet SHOP3 bei der Planzerlegung die Operatoren und Methoden an. Die Operatoren beschreiben in SHOP3 die primitiven Aktionen. Diese können nicht weiter zerlegt werden und bilden die Aktionen in der Aktionsliste des finalen Plans ab. Operatoren werden wie folgt implementiert:

```
(:op (!beispieloperator ?argument1 ?argument2 ... )
      :add ( Add-Liste )
      :delete ( Delete-Liste )
      :precond ( Vorbedingungen )
      :cost Kostenfunktion
)
```

**Algorithm 10:** Operatoren in Shop

Vor dem Namen des Operators steht ein Ausrufezeichen. In SHOP3 gibt es „Interne Operatoren“, die mit zwei Ausrufezeichen vor dem Namen beginnen, diese werden wie unsichtbare Aktionen im Plan ausgeführt. Das heißt, dass ein Interner Operator zwar angewendet und den Zustand der Modellwelt verändert, jedoch nicht im Plan ausgegeben wird und damit für interne Berechnungen nützlich ist. Optional können dem Operator Argumente übergeben werden. Argumente beginnen mit einem Fragezeichen. Da die Operationen im Plan ausgegeben werden, können die Argumente zusätzliche Informationen über die aufgerufenen Aktionen und den Planverlauf geben, wie im Beispiel auf der nächsten Seite zu sehen ist.

```

Option 1: (:op (!drtive-to ?truck ?destination ))
Option 2: (:op (!drtive-to ?truck ?start ?destination ?starttime ))
...
Plan 1: ... (!drtive-to truck1 b) ...
Plan 2: ... (!drtive-to truck1 a b 600) ...

```

In den Vorbedingungen des Operators stehen logische Ausdrücke. Die einzelnen Funktionen des Operators wurden bereits in Kapitel 3.2 erläutert, daher werden an dieser Stelle ein paar ergänzende Informationen zu den Operatoren in SHOP3 gesagt. Wenn die Vorbedingungen leer sind, wird standardmäßig *true* zurück gegeben. SHOP3 bietet einige Möglichkeiten in den Vorbedingungen, wie zum Beispiel numerische Berechnungen oder externe Funktionsaufrufe. Ein logischer Ausdruck kann zum Beispiel ein Atom, eine Negation (*not (beispielatom wert)*), eine Konjunktion (*and a<sub>1</sub> a<sub>2</sub> ... a<sub>n</sub>*), eine Disjunktion (*or a<sub>1</sub> a<sub>2</sub> ... a<sub>n</sub>*), eine Implikation (*a->b*) - wenn *a* gilt, dann muss *b* erfüllt sein, ein universeller Quantifizierer (*forall ...*) - Bedingungen auf eine Menge von Atomen, eine Variablenzuordnungen (*assign ...*) sein. Außerdem können mit der *:sort-by*-Funktion bestimmte Atome vor den Variablenbindungen in den Vorbedingungen sortiert werden. Mit *:first* kann festgelegt werden, dass nur die erste gefundene Variablenbindung berücksichtigt wird. Befinden sich auf dem Heap beispielsweise die Atome (*atom 1*) und (*atom 2*) und in den Vorbedingungen steht als Ausdruck (*atom ?irgendeinwert*), so kann der Variable *?irgendeinwert* 1 oder 2 zugeordnet werden, und wenn *:first* deklariert wird nur 1. Alle Variablen, die in der Add-Liste und der Delete-Liste verwendet werden, müssen entweder als Argument dem Operator übergeben oder hier definiert werden. Alle Elemente *:add*, *:delete*, *:precond* und *:cost* im Operator sind optional. Standardmäßig verursacht jeder Operator (auch die Internen Operatoren) die Kosten 1. Mit *:cost* können definierte Kosten für den Operator festgelegt werden.

### 3.4.6 Methoden

In SHOP3 repräsentieren die Methoden die höheren Aktionen, die zerlegt werden. Eine Methode wird in der Aufgabenliste in weitere Methoden und Operatoren zerlegt. Ein finaler Plan besteht jedoch immer nur aus den primitiven Aktionen, den Operatoren.

Eine Methode wird wie folgt definiert:

```
(:op (beispielmethode ?argument1 ?argument2 ...)
     zerlegung1
     ( Vorbedingungen )
     ( Aufgabenliste )
     zerlegung2
     ( Vorbedingungen )
     ( Aufgabenliste )
     ...
)
```

**Algorithm 11:** Methoden in Shop

Im Rumpf der Methode ist „zerlegung1“ der Name einer möglichen Zerlegung. Zu jeder Zerlegung muss eine Vorbedingungs- und eine Aufgabenliste definiert werden, diese können auch leer sein. Die *Vorbedingungen* haben dieselben Funktionen wie bei den Operatoren. Methoden dürfen beliebig viele Zerlegungen haben. Bei der Dekomposition behandelt SHOP3 die Zerlegung wie ein *else*. Eine weitere zentrale Eigenschaft von SHOP3 ist die Ordnung der Aufgaben bei der Zerlegung. Die abzuarbeitende Aufgabenliste einer Methoden kann entweder vollständig oder partiell geordnet sein. Partiell geordnete Aufgabenlisten erlauben es, die Methoden Operatoren simultan zu zerlegen. Mit *:ordered* und *:unordered* wird die Ordnung definiert. Standardmäßig ist jede Aufgabenliste geordnet. Dafür wird folgendes Beispiel betrachtet:

```
(:method (beispielmethode1)
        zerlegung-m1 () (:unordered (!operator1) (!operator2)) )
(:method (beispielmethode2)
        zerlegung-m2 () (:unordered (!operator3) (!operator4)) )
(:method (high-level-aktion)
        zerlegung-ha () (:unordered (beispielmethode1) (beispielmethode2)) )
```

Angenommen es soll (*beispielmethode1*) zerlegt werden, so gibt es zwei Lösungen und die Pläne lauten:  $((!operator1) (!operator2))$  und  $((!operator2) (!operator1))$ . Wird allerdings (*high-level-aktion*) zerlegt, so gibt es 24 verschiedene Möglichkeiten diese Methode zu zerlegen:  $((!operator1) (!operator3) (!operator2) (!operator4))$  oder  $((!operator1) (!operator3) (!operator4) (!operator2))$  usw..

# Kapitel 4

## Eigener Ansatz

Die Transportplanung in hybriden Bergwerken ist maßgeblich von der Infrastruktur des Bergwerks und der gewählten Methodik zur Rohstoffgewinnung abhängig. Im ersten Teil dieses Kapitels betrachtet die Arbeit das Thema hybride Bergwerke von einem allgemeinen Standpunkt aus. Dafür wird ein Verfahren vorgestellt, das in der Bergbauindustrie als „Blending“ (Vermischen) bezeichnet wird. Aufbauend darauf stellt Kapitel 4.2 den Lösungsansatz des für diese Arbeit entwickelten Planungssystems vor.

### 4.1 Transportplanung in Bergwerken

Abbildung 4.1 auf der nächsten Seite zeigt links das Foto eines Tagebaus für Hartgestein-Metallvorkommen. Üblicherweise ordnet die Betriebsplanung den Boden der Stätte in Blöcke ein [16, vgl. Espinoza et al., 2012] (Abbildung 4.1 rechts). Die Maschinen bauen die Blöcke in einer von der Leitung vorgegebenen Reihenfolge schichtweise ab. Eine Transportstraße, auch Rampe genannt, schlängelt sich vom Boden der Grube bis zur Oberfläche über die Bänke der einzelnen Ebenen. Vor dem Extrahierungsprozess kann das Bergwerk mithilfe von geostatistischer Analyse [10, Mariz et al., 2019] den Anteil der Mineralien in den Blöcken bestimmen, da die Qualität bzw. der Erzanteil im Boden variiert. Zur Gewinnung wird mittels verschiedener Verfahren, z.B. bohren oder sprengen, das Erz von der Felswand entfernt. Zunächst müssen die Fahrzeuge den unbrauchbaren Abraum, wie beispielsweise Erde, zu einer Müllkippe (waste dump) transportieren, sodass mit dem eigentlichen Beförderungsprozess begonnen werden kann. Die Transportziele der Rohstoffe werden dabei von der Leitung vorgegeben und sind von den verschiedenen Anteilen der Mineralien in den Blöcken abhängig. Da die Prozessanlagen

nach der Verarbeitung von überwiegend schlechtem Material nicht den angestrebten Erzanteil im Produkt erzielen können, kann es vorkommen, dass das Bergwerk zum Erreichen einer gleichmäßigen Qualität das Material vor dem Einspeisen in die Prozessanlagen miteinander vermischen muss (Blending). Statsenko et al. [47, Statsenko et al., 2014] und Montiel et. al [31, Montiel et al., 2015] zeigen und untersuchen Anwendungsfälle dafür. Eine Möglichkeit ist, die Transportladungen aus den unterschiedlichen Sektoren der Mine zu Deponien (Stockpiles), die als Zwischenlager dienen, zu bringen. Beispiele schütten hierbei die Fahrzeuge die verschiedenen Transportladungen abwechselnd einen Abhang hinunter, sodass es schichtweise vermischt wird (siehe Abbildung 4.2).

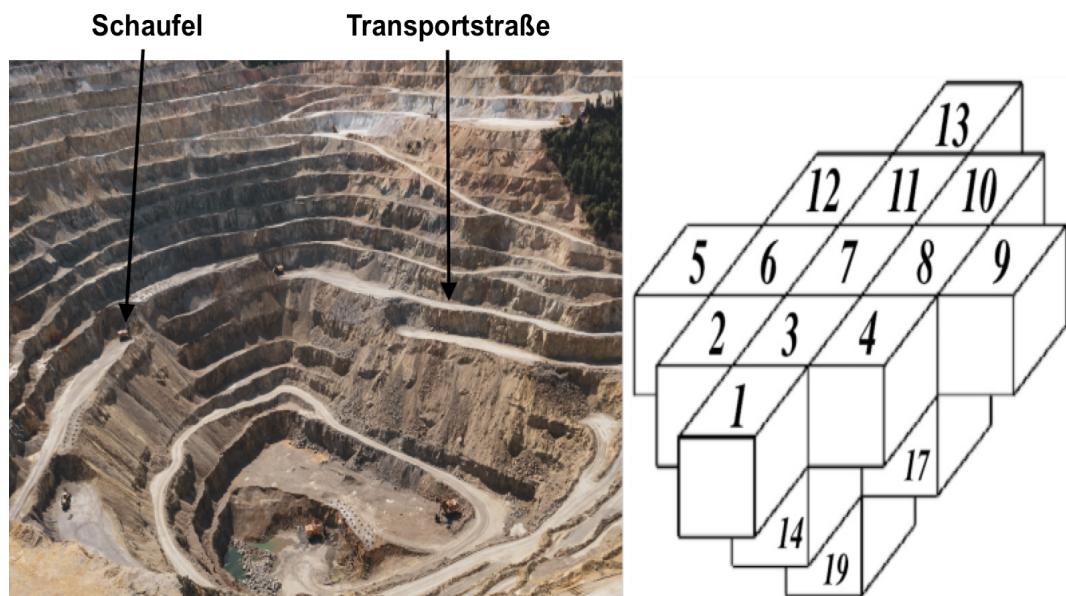


Abbildung 4.1: Einteilung der Mine

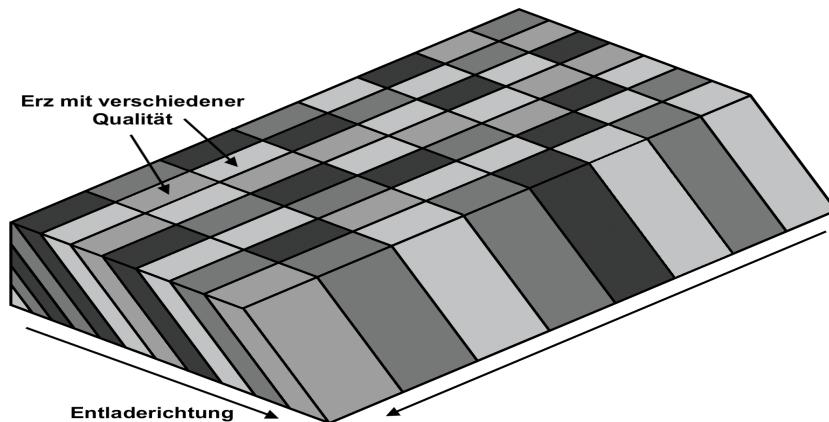


Abbildung 4.2: Vermischen des Erzes

Für die Erzgewinnung im Untertagebau gibt es heutzutage eine Vielzahl an Modellen, wobei die gewählte Methode von der Art des Rohstoffs und der Tiefe der Mine abhängt. Zu den verbreiterten Abbaumethoden im Untertagebau gehören der Kammerbau (Room-and-pillar mining), die Aushöhlung einzelner Ebenen (Sublevel caving), der Blockbruchbau (Block caving) und der Kurzlochabbau (Cut-and-fill mining) [3, vgl. Åstrand, 2018, Seite 24 ff.]. Bei der Room-and-pillar-Methode wird die Mine in Blöcken ausgehöhlt, wobei einzelne Blöcke als Stützpfeiler zurückbleiben. Beim Sublevel caving (flachere Minen) und Block caving (tieferen Minen) wird unter dem Erzvorkommen ein Tunnelsystem gebrochen. Anschließend wird mittels Bohr- und Sprengtechniken und der Schwerkraft das Erz in der Mine aufgefangen. Bei der Cut-and-fill-Methode wird die Mine nach Beendigung des Abbaus mit Erde oder Sand zugeschüttet. Nachdem die Fahrzeuge voll beladen werden, müssen diese entweder (in den flacheren Minen) das Erz an die Oberfläche transportieren oder können es (in den tieferen Minen) Untertage direkt abladen. In diesen Fällen verfügt das Bergwerk über ein separates Transportsystem, dass das Erz beispielsweise durch ein Schienennetz an die Oberfläche befördert. Der Vorteil ist, dass die Fahrzeuge die Mine während der Schicht nicht verlassen müssen. Für die Fahrt nach Untertage und dem Transport an die Oberfläche müssen die Fahrzeuge sich in der Regel die Transportstraße (Rampe) teilen. Das heißt, dass die Straßen zu schmal sind und der Verkehr gleichzeitig nur in eine Richtung möglich ist. Die Straßen verfügen über Einbuchtungen, in denen die Fahrzeuge parken und entgegenkommende Fahrzeuge vorbei lassen können. All diese Methoden und die unterschiedlichen Infrastrukturen im Untertagebau wirken sich anders auf die Prozessdynamiken eines Planungssystems aus.

## 4.2 Das Modell

Das Ziel dieser Arbeit ist ein Flottenmanagementsystem für den allgemeinen Anwendungsfall, also für unterschiedliche Fahrzeugfleotten und beliebige Bergwerke, zu entwickeln. Das „High-Level-Planungssystem“ simuliert die Ablaufplanung der Prozesse, indem unter Berücksichtigung der Nebenbedingungen des Transportsprozesses, den Einschränkungen der Fahrzeuge sowie der Situation im Bergwerk, alle möglichen Kombinationen zur Fahrplanung betrachtet werden. Basierend auf vorgegebenen Heuristiken und den Einschränkungen wählt das Planungssystem dann die nächste Lade- und Entladestelle für ein Fahrzeug aus.

Abbildung 4.3 veranschaulicht das Konzept des Planungssystems. Die Quadrate G1 und G2 repräsentieren die Entladeorte „goals“, zu denen das Erz transportiert wird. S1, S2 und S3 repräsentieren die Ladeorte bzw. Schaufeln „sources“ des Bergwerks. Die Kreise stellen die Fahrzeuge dar. Zwischen jedem Lade- und Entladeort gibt es eine eigene Verbindung (Route(s1,g1) usw.), die zur Übersicht weggelassen sind. Im Grunde kann das Planungssystem als ein „Warteschlangensystem“ beschrieben werden. Die Idee dahinter ist, die Fahrzeuge gleichmäßig auf die Lade- und Entladeorten zu verteilen.

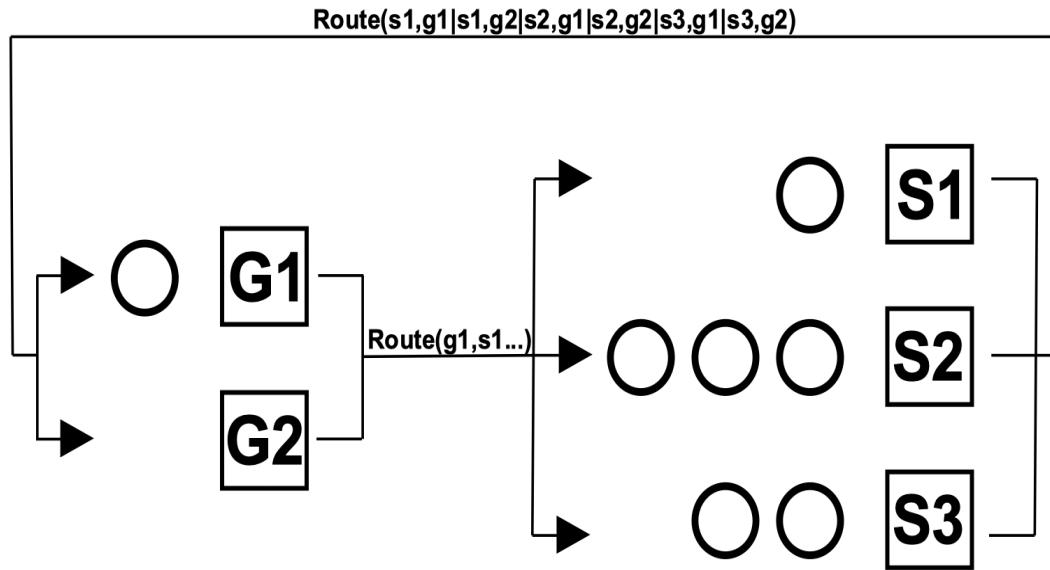


Abbildung 4.3: Warteschlangensystem

Zunächst wird die Planung eines einzigen Fahrzeugs betrachtet. Abbildung 4.4 zeigt den Aufbau des HTN-Planers. Bei jedem Planungsschritt plant das System eine ganze Tour eines Fahrzeugs, dass heißt zu einer Schaufel zum Beladen und zu einer Entladestelle zum Abladen. Dies wird durch die Methode „Dispatch“ repräsentiert. Zur Fahrtplanung verwendet das Planungssystem Heuristiken, die in Kapitel 2.1 bereits vorgestellt worden sind. Die Heuristiken, die der Planer verwendet, sind:

- Kürzeste Wartezeit
- Kürzeste Zykluszeit
- Längste Leerlaufzeit
- Zufällige (kostenbasierte) Suche

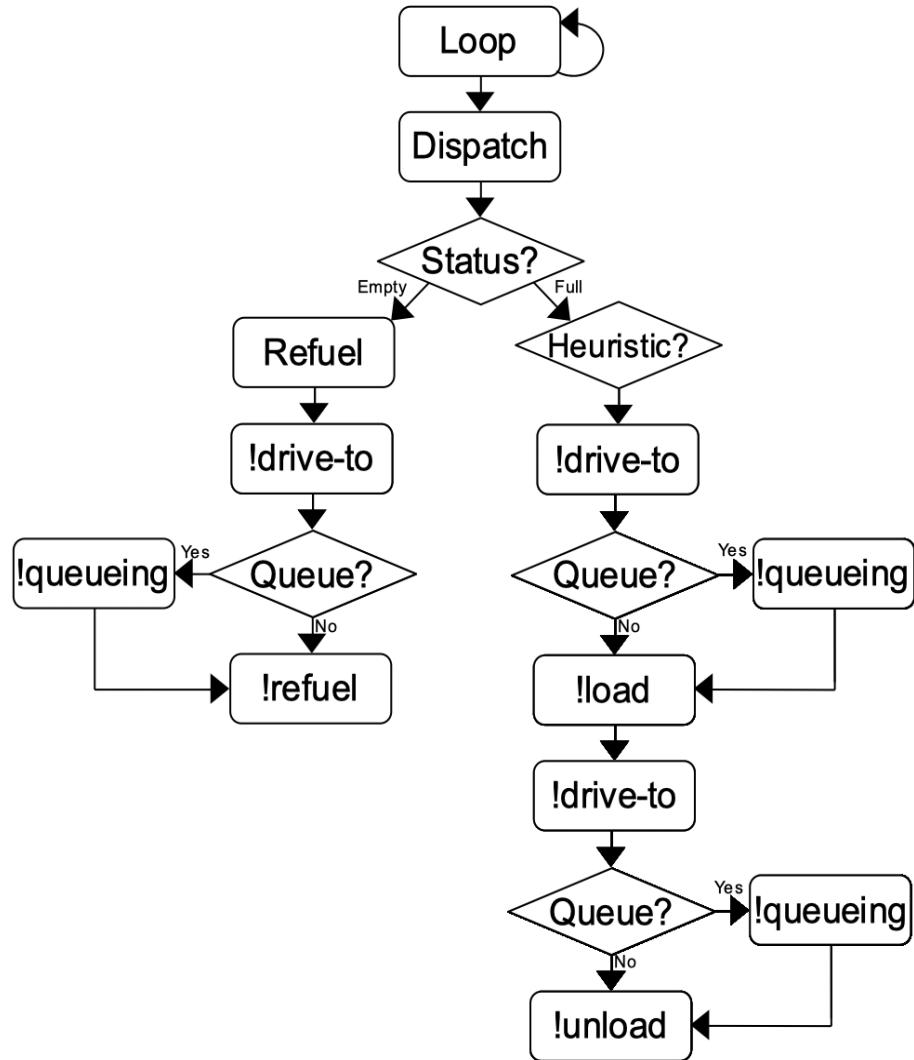


Abbildung 4.4: HTN-Planungssystem

Bei der Heuristik der kürzesten Wartezeit wählt der Planer die Lade- und Entladestelle aus, bei denen jeweils die Wartezeiten am Kleinsten sind. Bei der Heuristik der kürzesten Zykluszeit wählt der Planer die Lade- und Entladestelle aus, bei denen die Gesamtdauer am Kürzesten ist, also der Zeitpunkt nach der Operation „!unload“. Bei dieser Strategie wird auch die Fahrtzeit, sowie die Operationszeiten zum Be- und Entladen einbezogen. Für die längste Leerlaufzeit wählt der Planer bei den Ladestellen die Ladestelle aus, die bereits am Längsten im Leerlauf ist oder die als Erstes verfügbar wird. Bei den Entladestellen verwendet diese Heuristik die Entladestelle mit der kürzesten Wartezeit. Die vierte Strategie setzt jeweils eine zufällige Lade- und Entladestelle ein. In den Vorbedingungen der Methode „Dispatch“ berechnet das Planungssystems mittels der (durchschnittlichen) Fahrzeiten die Ankunftszeit an ei-

nem Ort. Anschließend wird die Wartezeit an dem Ort ermittelt. Dieser Vorgang wird in Abschnitt 4.3 beschrieben. Wenn ein Fahrzeug nicht mehr genügend Kraftstoff besitzt, schickt das Planungssystem das Fahrzeug zu einer Tanksäule, das in Abbildung 4.4 durch den linken Zweig dargestellt ist. Die Fahrzeuge können immer nur am Anfang der nächsten Transportfahrt zum Tanken fahren, wobei der Tankvorgang im Planungssystem eher als nebensächliche Aufgabe modelliert ist. Für die Auswahl einer Lade- und einer Entladestelle berücksichtigt der Planer die Art der Transportladung selbst. Die Rohstoffe sind im Planungssystem als Stapel, für jede Schaufel ein Stapel, implementiert. Für die Abladeziele ist jeweils zu jedem Zeitpunkt im Planungsprozess vorgegeben, welche Transportladung dort abgeladen werden muss. Abbildung 4.5 zeigt ein Beispiel.

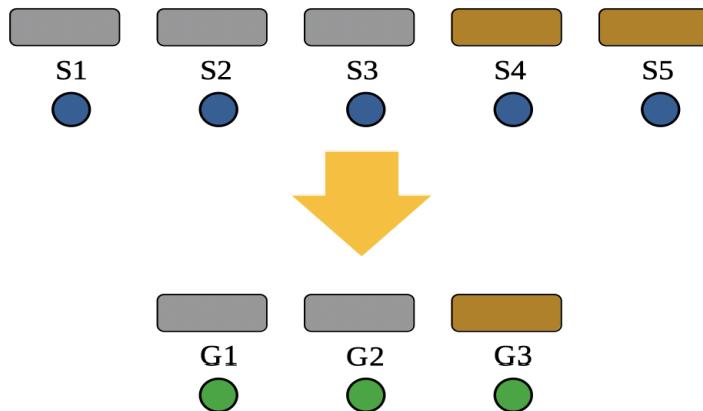


Abbildung 4.5: Zuweisung der Transportladungen

Die grauen Kästchen repräsentieren das Erz und die braunen Kästchen repräsentieren den Abraum, wie etwa Erde. In diesem Beispiel müssen die Transportladungen von S1-S3 zu G1-G2 und von S4-S5 zu G3 transportiert werden. Darüber hinaus kann ein Abladeort mehrere Transportladungen gleichzeitig erwarten oder es kann eine spezifische Abladereihe für eine Entladestelle vorgegeben werden, sodass dadurch der Prozess zum Vermischen der verschiedenen Transportladungen an einem Abladeort berücksichtigt wird. Neben der Fahrtplanung sind die Fahrzeuge Fahrzeugklassen zugeordnet. Für jede Klasse gibt es spezifische Eigenschaften, wie z.B. die maximale Nutzlast, verschiedene Fahrtzeiten der Strecken, unterschiedliche Operationszeiten zum Be- und Entladen, die Tankkapazität und den Kraftstoffverbrauch. Der Kraftstoffverbrauch ist abhängig vom Status des Fahrzeugs und wird unterschieden zwischen Stillstand, Leerfahrt oder Vollfahrt. In den Fahrzeugklassen wird außerdem festgelegt, welche Orte ein

Fahrzeug nicht ansteuern kann, sodass bestimmte Ziele bei der Fahrplanung für ein Fahrzeug nicht in Frage kommen.

Mit den genannten Eigenschaften simuliert das Planungssystem die möglichen Ablaufpläne, indem ein Simulationsziel wie z.B. ein Betrachtungszeitraum vorgegeben wird, oder bis eine bestimmte Menge Erz zu einem Ziel transportiert worden ist. Während der Simulation plant das System die Fahrzeuge simultan, wobei jedes Fahrzeug eine lokale Zeit hat. Das Planungssystem wählt während der Simulation immer das Fahrzeug als nächstes aus, dessen Zeitstempel am Kleinsten ist. Anschließend wird unter Berücksichtigung einer festgelegten Heuristik und den Einschränkungen die nächste Transportfahrt geplant. Sofern ein Fahrzeug aufgrund der Einschränkungen im aktuellen Weltzustand keine Transportfahrt durchführen kann, wählt das Planungssystem automatisch ein anderes Fahrzeug aus. Die gesamte Implementierung des Planungssystems ist im Anhang A2 zu finden.

### 4.3 Temporäre Planung

Temporäre Planung ist in SHOP3 Planer nicht standardmäßig integriert, allerdings kann dies aufgrund der Möglichkeit der numerischen Berechnungen in den Vorbedingungen eigenhändig modelliert werden. Eine Veröffentlichung von SHOP-Entwickler Dana Nau et al. [vgl. Nau et al., 2013], zeigt in einer Anleitung, wie jede beliebige Domäne in SHOP mittels einem Verfahren namens „Multi-Timeline Preprocessing“ in ein temporäres Modell überführt werden kann. Da diese Arbeit eine eigene Darstellung verwendet, wird an dieser Stelle nur kurz das Prinzip hinter dem „Multi-Timeline Preprocessing“ erläutert. Das „Multi-Timeline Preprocessing“ verwendet viele lokale Zeiten für jede dynamische Eigenschaft. Eine dynamische Eigenschaft kann zum Beispiel der Standort eines Fahrzeugs oder der Wert des Tankinhalts sein, die sich während des Planungsprozesses verändert. Jede dynamische Eigenschaft erhält zwei Zeitstempel, eine Lese- und eine Schreibzeit, wie zum Beispiel (*read-time ?vehicle ?location*) und (*write-time ?vehicle ?location*). Die Aktionen regeln in den Vorbedingungen, ob die Aktion eine dynamische Eigenschaft schreiben und/oder lesen kann. Indem eine Aktion nur die Lesezeit verändert, wird verhindert, dass sich zwei Aktionen überlagern.

In dieser Arbeit wird die temporäre Darstellung der Durative Actions aus PDDL2.1 übernommen [18, vgl. Fox und Long, 2003]. Daneben erhält jedes Fahrzeug eine einzige lokale Zeit. In der Problembeschreibung des Planungssystems ist für jede Fahrzeugklasse und Route vorgegeben, wie lange die Aktionen andauern. Die Aktionen werden mit der lokalen Zeit des Fahrzeugs und der jeweiligen Dauer der Aktion ausgeführt, wie zum Beispiel (*!unload truck1 crusher 27 3*). Der vorletzte Wert in dem Operator entspricht der Startzeit und der letzte Wert der Dauer, in diesem Fall startet das Fahrzeug bei Minute 27 und benötigt 3 Minuten zum Entladen. Neben den lokalen Zeiten für die Fahrzeuge gibt es globale Blockierungen für die Ressourcen. Jede Ressource (Ladeort, Entladeort oder Tanksäule) verwaltet dabei eine Liste der eigenen Blockierungen. Die Blockierungen werden als Zeitfenster in der Form (*Startzeit Dauer*) abgespeichert. Sobald ein Fahrzeug eine Ressource benutzen will, wird ein verfügbares Zeitfenster ermittelt. Abbildung 4.7 zeigt das Prinzip (wobei P für Positioning, L für Loading, U für Unloading und R für Refill steht).

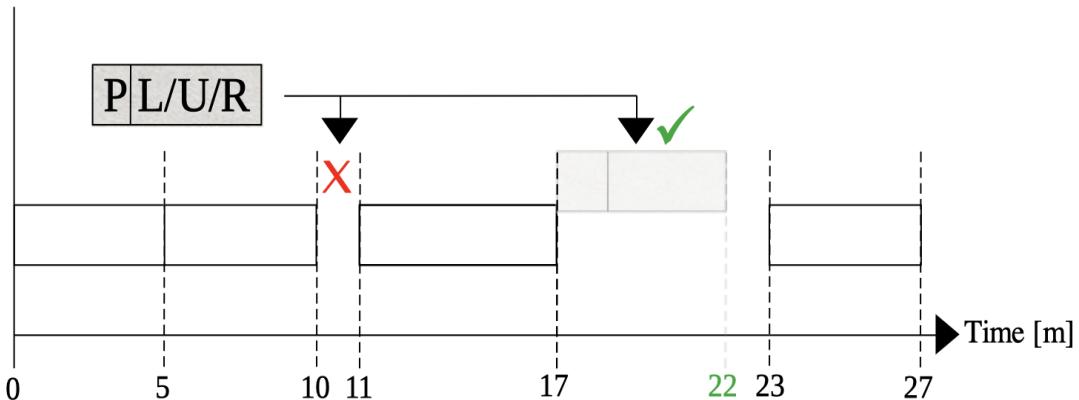


Abbildung 4.6: Temporäre Planung mittels Zeitfenstern

Bevor die Fahrzeuge bestimmte Ressourcen zur Ausführung der Operationen für sich reservieren können, vergleicht der Planer alle Blockierungen ab der Startzeit der Ankunft und berechnet, wann eine Aktion dort ausgeführt werden kann. Sofern das Fahrzeug warten muss, wird die Aktion *!queueing* eingeleitet. Allerdings ist nicht definiert, wo das Fahrzeug wartet und wieviele Fahrzeuge gleichzeitig warten. Abbildung 4.8 zeigt aus einem Testdurchlauf die Zeitfenster der blockierten Ressourcen. Damit die Listen nicht immer größer werden, löscht das Planungssystem in jeder Iteration die Blockierungen, die in der Vergangenheit (aller Fahrzeuge) liegen.

```

(BLOCKED G1
((-1 0) (20 2) (22 2) (25 2) (27 2) (30 2) (42 2) (44 2) (47 2) (49 2) (52 2)
(64 2) (66 2) (69 2) (71 2) (74 2) (86 2) (88 2) (91 2) (93 2) (96 2)
(108 2) (110 2) (113 2) (115 2) (118 2) (130 2) (132 2) (135 2) (137 2)
(140 2) (152 2) (154 2) (157 2) (159 2) (162 2) (174 2) (176 2)))
(BLOCKED S3
((-1 0) (6 5) (11 5) (30 5) (35 5) (52 5) (57 5) (74 5) (79 5) (96 5) (101 5)
(118 5) (123 5) (140 5) (145 5) (162 5)))
(BLOCKED G2
((-1 0) (20 2) (22 2) (25 2) (27 2) (30 2) (42 2) (44 2) (47 2) (49 2) (52 2)
(64 2) (66 2) (69 2) (71 2) (74 2) (86 2) (88 2) (91 2) (93 2) (96 2)
(108 2) (110 2) (113 2) (115 2) (118 2) (130 2) (132 2) (135 2) (137 2)
(140 2) (152 2) (154 2) (157 2) (159 2) (162 2) (174 2)))
(BLOCKED S2
((-1 0) (6 5) (11 5) (16 5) (28 5) (33 5) (38 5) (50 5) (55 5) (60 5) (72 5)
(77 5) (82 5) (94 5) (99 5) (104 5) (116 5) (121 5) (126 5) (138 5) (143 5)
(148 5) (160 5)))
(BLOCKED S1
((-1 0) (6 5) (11 5) (16 5) (28 5) (33 5) (38 5) (50 5) (55 5) (60 5) (72 5)
(77 5) (82 5) (94 5) (99 5) (104 5) (116 5) (121 5) (126 5) (138 5) (143 5)))

```

Abbildung 4.7: Blockierungen der Ressourcen

## 4.4 Einordnung in das Gesamtsystem

Auf der nächsten Seite ist in Abbildung 4.8 die Architektur des Gesamtsystems zu sehen. Das System unterteilt sich in drei Teile: Fahrzeug- und Sensor-spezifische Hardwarekomponenten (rechts unten), Fahrzeug-unabhängige Softwaremodule (links unten) und das Kontrollzentrum (links oben). Der Aufbau des Systems von den Fahrzeugen (Dumper) bis zum Kontrollzentrum (Base Station) und dem High-Level-Planungssystem wird im folgenden beschrieben (Bottom-Up). Die gesammelten Daten der Sensoren der Fahrzeuge zur Kommunikation (UWB), der Fahrzeugkontrolle (IMU), der Lokalisierung (GPS), der räumlichen Orientierung (Lidar) und der Objekterkennung (Camera) werden zur Verarbeitung an die Fahrzeug-unabhängigen Softwaremodule gesendet. Für die Kommunikation und den Datenaustausch wird ROS2 (Robot Operating System) verwendet. Mehrere Reasoner (Lernalgorithmen) verarbeiten die Daten parallel. Dort kommen beispielsweise konvolutionale neuronale Netze zur Bildverarbeitung, Kalman-Filter für verrauschte Daten und Pathfinding-Algorithmen zur Routenplanung zum Einsatz. Ein zentrales Kernstück des Systems ist das Behaviour Planning (Verhaltensplanung). Die Behaviour Engine entscheidet an Hand der von oben vorgegebenen Ziele und den aufbereiteten Sensorinformationen, welche Ziele die Fahrzeuge (Agenten) als nächstes verfolgen bzw. leitet die konkreten Handlungsaktionen ein. Die Stati der Fahrzeuge überwacht die State Machine, ein Zustandsautomat. Das (erlernte) Wissen und die Stati werden in einer Datenbank, dem Weltmodell (World Model), abgelegt. Von

dort erhält das High-Level-Planungssystem und der Dispatcher, einer weiteren Schnittstelle zur Flottensteuerung, die aktuellen Informationen über die Fahrzeuge und das Bergwerk, die zur Planung der Ziele und Verteilung der Aufgaben benötigt werden. Über eine Benutzerschnittstelle (GUI - Graphical User Interface) übergibt der Endanwender (in einem dezentralen Kontrollzentrum) die Befehle an das System und steuert die Fahrzeugflotte.

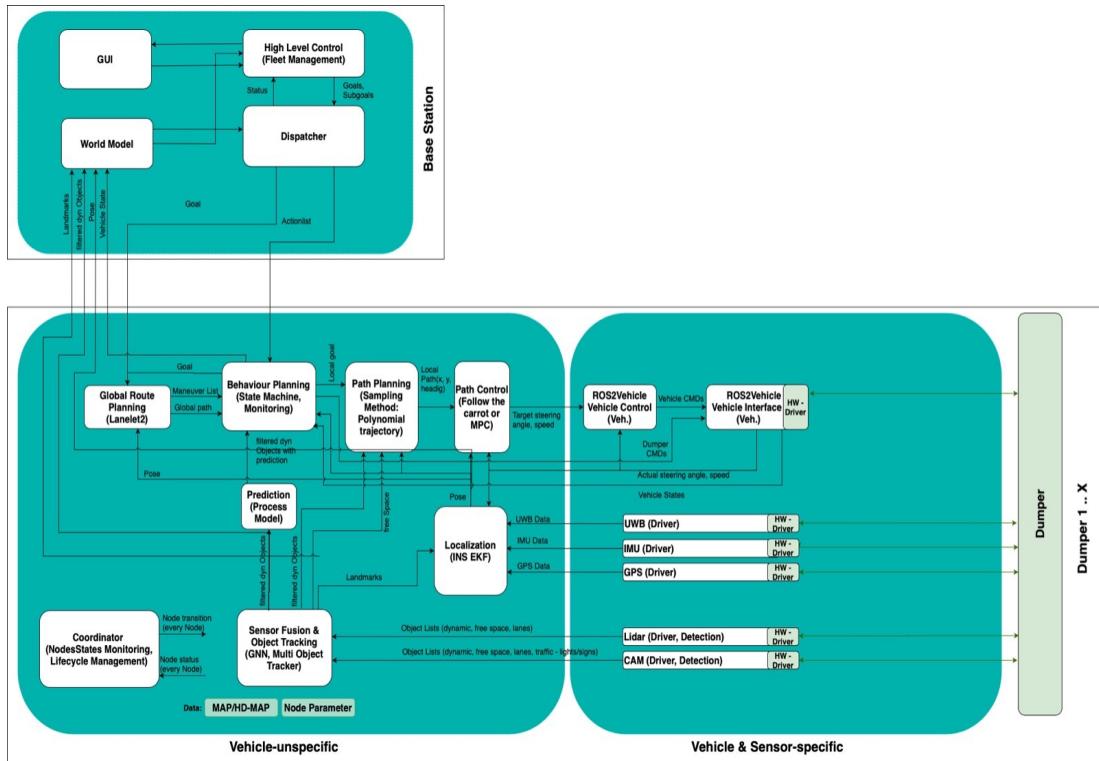


Abbildung 4.8: Architektur des Projekts ARTUS

## Kapitel 5

# Experimentelle Ergebnisse

In diesem Kapitel werden die Ergebnisse des Planungssystems analysiert. Dafür wird das Planungssystem an Hand von zwei Anwendungsszenarien demonstriert. Im Rahmen der Anwendungsszenarien werden die einzelnen Strategien gegenübergestellt und diskutiert. Zur Vergleichbarkeit variieren die Größen der Fahrzeugflotten in den Anwendungsszenarien. Folgende Strategien werden betrachtet:

- Kürzeste Wartezeit
- Kürzeste Zykluszeit
- Längste Leerlaufzeit
- Zufällige kostenbasierte Suche (Tonnage)

Die Strategien kürzeste Wartezeit, kürzeste Zykluszeit und längste Leerlaufzeit erzeugen jeweils einen einzigen Plan. Die kostenbasierte Plansuche generiert 10 zufällige Pläne und wählt darunter den Plan mit der höchsten Gesamttonnage aus. Die Umgebungen und Daten in den Anwendungsszenarien beschreiben fiktive Beispiele und basieren auf den recherchierten Grundlagen über Bergwerke sowie Use-Cases in verwandten Arbeiten [5, Zhang et. al, 2019] und [46, Skawina, 2019]. Für die Bewertung der Strategien werden die Ergebnisse an Leistungskennzahlen (Key Performance Indicators - KPI) gemessen. Folgende KPI's sind definiert:

- Gesamttonnage
- Wartezeit aller Fahreuge
- Leerlaufzeit aller Schaufeln

- Ladeoperationen einer Schaufel
- Gefahrene Distanz aller Fahrzeuge
- Gesamter Kraftstoffverbrauch
- Rechenzeit eines Plans (Technische Kennzahl)

Die Gesamttonnage ist die Summe des durch alle Fahrzeuge geförderten Erzes am Ende einer Schicht. Die Wartezeit ist die kumulierte Wartezeit aller Fahrzeuge, die an blockierten Lade-, Entladestellen und Tanksäulen entstehen. Die Leerlaufzeit ist die Summe der inaktiven Zeit aller Schaufeln/Ladestellen. Die Ladeoperationen einer Schaufel sind die Anzahl der abgefertigten Vollladungen, wobei eine Vollladung einer Transportfahrt entspricht. Da die Fahrzeuge den Schaufeln simultan zugeordnet werden, ist die Anzahl der Fahrten pro Fahrzeug für alle Fahrzeuge in etwa gleich groß und daher irrelevant. Die gefahrene Distanz entspricht der Summe der zurückgelegten Kilometer aller Fahrzeuge. Der Kraftstoffverbrauch bezieht sich auf den Gesamtkraftstoffverbrauch aller Fahrzeuge. Die Rechenzeit eines Plans dient zur Bewertung der Effizienz und des Zeitverhaltens. Die Testdurchläufe werden auf einem MacBook Pro mit einem Intel i5 Kern, 2 Ghz und 10 Gb Arbeitsspeicher durchgeführt. Für die Testfälle in Anwendungsszenario 1 mit 75 Fahrzeugen und Anwendungsszenario 2 wurde der Arbeitsspeicher auf 15 Gb erhöht, da SHOP in diesen Testfällen Speicherplatzprobleme verursacht hat.

## 5.1 Anwendungsszenario 1

Das Ziel des ersten Anwendungsszenario ist, die Ergebnisse der Strategien bei unterschiedlichen Flottengrößen zu bewerten und das Verhalten des Planungssystems bei vielen Fahrzeugen zu demonstrieren. In diesem Anwendungsszenario werden Fahrzeugflotten mit 5, 10, 20, 30, 40 und 75 Fahrzeugen betrachtet. Bei allen Testdurchläufen und Strategien generiert das Planungssystem einen 8-stündigen Ablaufplan für die Fahrzeuge. In dem betrachteten Anwendungsbeispiel gibt es 5 Schaufeln zum Beladen (Sources) und 2 Brecher/Deponien zum Entladen (Goals). Die Distanzen und die durchschnittlichen Fahrtzeiten zwischen den Lade- und Abladestellen sind in Tabelle 5.1 dargestellt. Die Distanzen sind in Kilometer angegeben und die Fahrtzeiten in Minuten. Eine Vollfahrt entspricht der Fahrt von den Schaufeln (Sources) zu den Abladestellen

(Goals) und eine Leerfahrt von den Abladestellen zu den Schaufeln. Außerdem sind alle Transportfahrzeuge vom gleichen Fahrzeugtyp und transportieren pro Fahrt exakt 50 Tonnen Erz. Jedes Fahrzeug benötigt 6 Minuten zum Beladen und 3 Minuten zum Entladen. Zudem wird ausgeschlossen, dass die Fahrzeuge nachtanken müssen. Abraum, der zu einer Müllkippe transportiert werden muss, oder weitere Einschränkungen für den Transportprozess und die Fahrzeuge gibt es nicht. Abbildung 5.1 veranschaulicht das Anwendungsbeispiel. Alle Fahrzeuge starten bei G1.

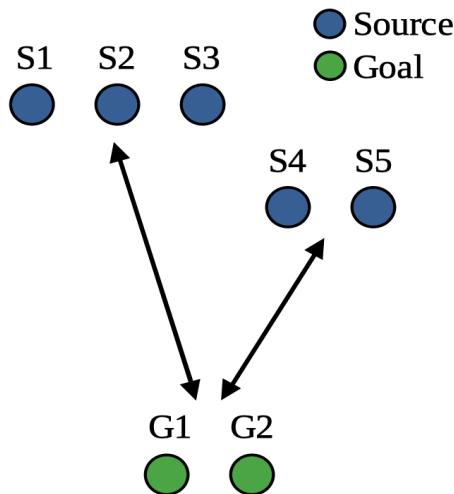


Tabelle 5.1: Distanzen und Fahrtzeiten

von G1-G2 nach	S1-S3	S4-S5
Fahrzeit (Leer)	10	15
Fahrzeit (Voll)	14	19
Distanz	3.5	2.5

Abbildung 5.1: Anwendungsszenario 1

### 5.1.1 Planausgabe

Abbildung 5.2 zeigt einen Ausschnitt der Planausgabe von Anwendungsszenario 1 für den Fall mit fünf Fahrzeugen und unter der Verwendung der heuristischen Methode kürzeste Zykluszeit. Zu sehen sind unter dem Namen der Problembeschreibung die Parameter, die der Plansuche mitgegeben werden. In diesem Fall wird der erste gefundene Plan (*:which = first*) ausgegeben und die dazugehörigen Operationen (*:verbose = plans*) werden angezeigt. Zusätzlich zeigt Shop ein paar statistische Informationen über den Plan selbst an (*Totals...*). Diese beinhalten neben der Anzahl der gefundenen Pläne, die Kosten des besten (*Mincost*) und die Kosten des schlechtesten (*Maxcost*) Plans. Da in diesem Fall die Heuristik kürzeste Zykluszeit verwendet wird, werden bei dieser Strategie keine Kosten berücksichtigt. Schließlich werden noch die Prozessorzeit und die Gesamtzeit, die Shop für die Suche benötigt, angezeigt. Im Grunde bestehen die erzeugten Pläne aus den sich wiederholenden Operationen: *!DRIVE-TO*, *!LOAD*,

*!UNLOAD* und *!QUEUEING*. Intern wählt der Planer nacheinander ein neues Fahrzeug aus und versendet es zu einer Belade- und Entladestelle. Da in diesem Fall die geringste Zykluszeit als Strategie benutzt wird, steuern die ersten beiden Fahrzeuge zunächst S4 und S5 an, da die Fahrtzeit dorthin kürzer ist, als eine Fahrt zu S1-S3. Trotz der dann entstehenden Wartezeit bei S4 und S5 schickt die Strategie die Fahrzeuge 3 und 4 ebenfalls zu den Ladestellen 4 und 5. Dies liegt daran, dass trotz der Wartezeit die Gesamtzeit der Transportfahrt zu diesen Ladestellen immer noch kürzer ist, als eine Fahrt zu S1-S3. Erst für das fünfte Fahrzeug wird die Wartezeit bei S4 oder S5 so groß, dass Fahrzeug 5 S1 ansteuert. Dieser Vorgang wird solange wiederholt, bis das Zeitlimit von 480 Minuten (= 8 Stunden) für alle Fahrzeuge erreicht wird.

```
Defining problem ANWENDUNGSSZENARIO1 ...
```

---

```
Problem #<SHOP3:::PROBLEM ANWENDUNGSSZENARIO1> with :WHICH = :FIRST, :VERBOSE = :PLANS
```

Totals:	Plans	Mincost	Maxcost	Expansions	Inferences	CPU time	Real time
	1	0	0	74	1397	0.008	0.007

```
Plans:
```

```
(((!DRIVE-TO TRUCK1 G1 S4 0 10) (!LOAD TRUCK1 S4 10 6)
  (!DRIVE-TO TRUCK1 S4 G1 16 15) (!UNLOAD TRUCK1 G1 31 3)
  (!DRIVE-TO TRUCK2 G1 S5 0 10) (!LOAD TRUCK2 S5 10 6)
  (!DRIVE-TO TRUCK2 S5 G2 16 15) (!UNLOAD TRUCK2 G2 31 3)
  (!DRIVE-TO TRUCK3 G1 S4 0 10) (!QUEUEING TRUCK3 S4 10 6)
  (!LOAD TRUCK3 S4 16 6) (!DRIVE-TO TRUCK3 S4 G1 22 15)
  (!UNLOAD TRUCK3 G1 37 3) (!DRIVE-TO TRUCK4 G1 S5 0 10)
  (!QUEUEING TRUCK4 S5 10 6) (!LOAD TRUCK4 S5 16 6)
  (!DRIVE-TO TRUCK4 S5 G2 22 15) (!UNLOAD TRUCK4 G2 37 3)
  (!DRIVE-TO TRUCK5 G1 S1 0 14) (!LOAD TRUCK5 S1 14 6)
  (!DRIVE-TO TRUCK5 S1 G1 20 19) (!QUEUEING TRUCK5 G1 39 1)
  (!UNLOAD TRUCK5 G1 40 3) (!DRIVE-TO TRUCK1 G1 S4 34 10)
  (!LOAD TRUCK1 S4 44 6) (!DRIVE-TO TRUCK1 S4 G1 50 15)
  (!UNLOAD TRUCK1 G1 65 3) (!DRIVE-TO TRUCK2 G2 S5 34 10)
  ...
  ...)
```

Abbildung 5.2: Planausgabe von Shop

Im nächsten Abschnitt werden die Ergebnisse von Anwendungsszenario 1 für alle Strategien und Testdurchläufe miteinander verglichen.

### 5.1.2 Evaluation

Zunächst wird die Gesamttonnage betrachtet. Abbildung 5.3 zeigt die Ergebnisse der Gesamttonnage für alle Testdurchläufe in einem Säulendiagramm. Auf der x-Achse ist die Anzahl der Fahrzeuge und auf der y-Achse die Gesamttonnage in Tonnen abgebildet. Die Farben der Säulen repräsentieren die verwendete Strategie. Auf dem ersten

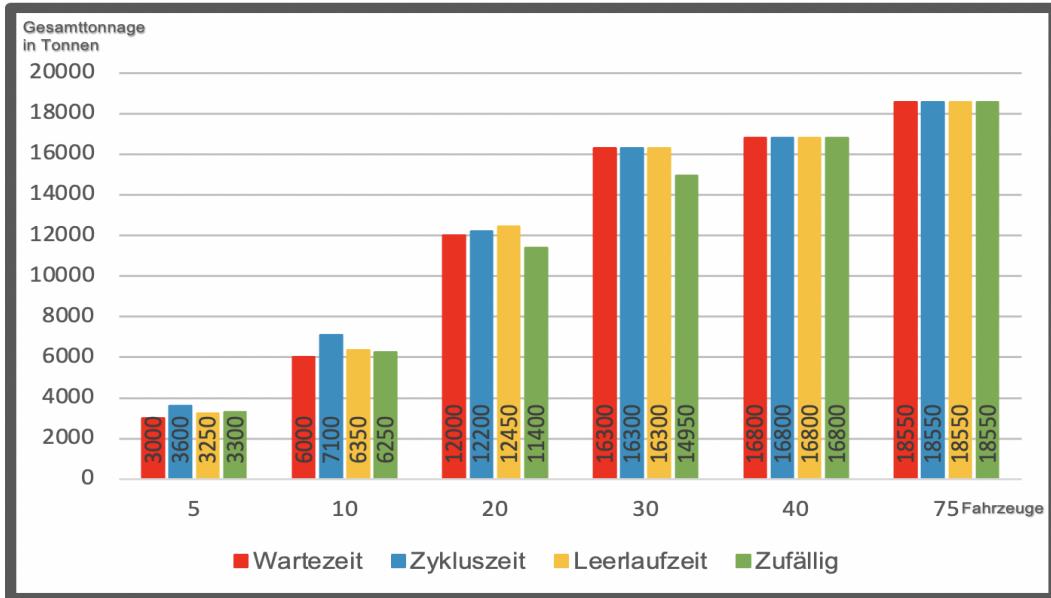


Abbildung 5.3: Anwendungsszenario 1 - Ergebnisse der Gesamttonnage

Blick ist deutlich zu sehen, dass die Anzahl der Fahrzeuge den größten Einfluss auf die Gesamttonnage hat. Außerdem ist zu erkennen, dass bei den Testdurchläufen mit 30, 40 und 75 Fahrzeugen die Gesamttonnage bei (fast) allen Strategien gleich groß ist und weniger zunimmt. Dies liegt an Systemüberlastung und zu langen Warteschlangen an den Ressourcen. Beispielsweise muss in dem letzten Test das 75. Fahrzeug, nachdem es die erste Fahrt beginnt, 114 Minuten warten bevor es von einer Schaufel bedient werden kann (Abbildung 5.4). Tabelle 5.2 zeigt die kumulierten Wartezeiten in Minuten.

```
(!UNLOAD TRUCK74 G2 142 3) (!DRIVE-TO TRUCK75 G1 S4 0 10)
(!QUEUEING TRUCK75 S4 10 114) (!LOAD TRUCK75 S4 124 6)
(!DRIVE-TO TRUCK75 S4 G1 130 15) (!UNLOAD TRUCK75 G1 145 3)
```

Abbildung 5.4: Beispiel - lange Wartezeiten

Tabelle 5.2: Gesamtwartezeit aller Fahrzeuge

Strategie \ Fahrzeuge	5	10	20	30	40	75
Wartezeit	3	54	292	2108	7100	25782
Zykluszeit	16	72	950	2680	7628	26096
Leerlaufzeit	13	99	562	2716	7644	26046
Zufällig	33	250	1150	3488	7388	25938

In allen Testdurchläufen bis auf den Letzten schneidet die zufallsbasierte Plansuche bei der Gesamttonnage am Schlechtesten ab. Interessanterweise erreicht die Strategie der kürzesten Leerlaufzeit für den Fall mit 20 Fahrzeugen eine höhere Gesamttonnage als die kürzeste Zykluszeit (5 Transportfahrten mehr). Insgesamt schneidet die Strategie der kürzesten Zykluszeit bei der Gesamttonnage in allen anderen Fällen mit am Besten ab. Dies liegt daran, dass die Strategie die Fahrtzeit berücksichtigt und es zwei näher gelegene Schaufeln gibt. Dadurch versendet die Strategie die Fahrzeuge öfters zu diesen, wodurch mehrere Fahrten möglich sind. Der Nachteil der Strategie ist, dass dadurch die Schaufeln nicht gleichmäßig beansprucht werden. Tabelle 5.3 zeigt die Anzahl Ladeoperationen der Schaufeln getrennt durch einen Doppelpunkt (S1:S2:S3:S4:S5) für alle Testfälle und Strategien.

Tabelle 5.3: Anzahl der Fahrten zu den Schaufeln (S1:S2:S3:S4:S5)

Strategie \ Fahrzeuge	5	10	20
Wartezeit	23:23:12: <b>1:1</b>	43:35:34: <b>6:2</b>	69:69:65:33:4
Zykluszeit	<b>1:0:0:42:29</b>	<b>3:3:0:68:68</b>	49:39:4:78:74
Leerlaufzeit	12:12:12:15:14	23:23:23:29:29	42:42:40:65:60
Zufällig	12:12:7:20:15	17:19:28:30:31	50:46:48:45:39
	30	40	75
Wartezeit	81:81:79:79: <b>6</b>	81:81:80:80:14	81:81:80:80:49
Zykluszeit	81:81: <b>0:82:82</b>	83:83: <b>0:85:85</b>	92:92: <b>0:94:93</b>
Leerlaufzeit	55:55:55:81:80	58:58:58:81:81	70:70:69:81:81
Zufällig	66:62:59:49:63	70:63:73:64:66	79:73:75:71:73

Die fettgedruckten Zahlen zeigen, dass die Strategien kürzeste Wartezeit und kürzeste Zykluszeit in einigen Testfällen manche Schaufeln weniger bzw. nicht berücksichtigen. Beispielsweise könnte der Anwender (das Bergwerk) mithilfe von diesem Modell eine Schaufel auslassen und dadurch bei gleich bleibender Produktivität Kosten sparen. Tabelle 5.4 zeigt die Dauer der Plansuche in Sekunden für alle Testdurchläufe. Die zufällige Suche benötigt mehr Zeit benötigt, da jeweils 10 Pläne generiert werden.

Tabelle 5.4: Dauer der Plansuche

Strategie \ Fahrzeuge	5	10	20	30	40	75
Wartezeit	0.2	1	5	16	26	52
Zykluszeit	0.4	2	6	17	26	47
Leerlaufzeit	0.3	1	6	16	24	51
Zufällig	3	13	72	87	128	ca. 6 min

## 5.2 Anwendungsszenario 2

Im Rahmen des zweiten Anwendungsszenario werden die funktionalen Fähigkeiten des Planungssystems demonstriert. Dafür werden die Strategien auf ein komplexeres Beispiel angewendet. Es werden erneut Ablaufpläne für eine 8-stündige Schicht erzeugt. Abbildung 5.5 veranschaulicht Anwendungsszenario 2.

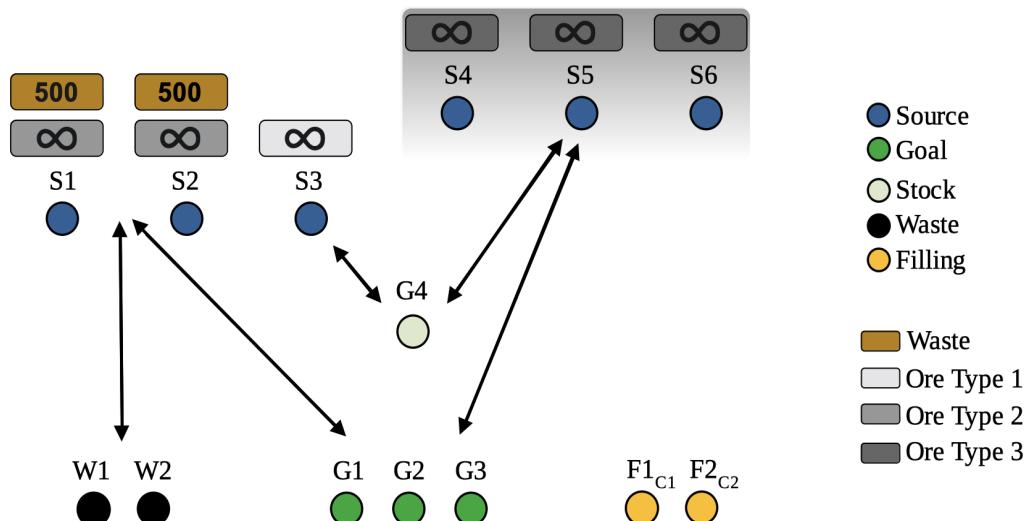


Abbildung 5.5: Anwendungsszenario 2

In dem vorliegenden Anwendungsszenario wird ein hybrides Bergwerk betrachtet. Geplant wird mit drei Schaufeln S1-S3, die sich über Tage und drei weitere Schaufeln S4-S6, die sich unter Tage befinden. Es gibt insgesamt sechs Abladeorte: G1, G2, G3, G4, W1 und W2, die sich alle an der Oberfläche befinden. W1 und W2 sind Mülldeponien. Die kleinen Kästchen über den Schaufeln (*Sources*) repräsentieren die dort abzuholende Erz-/Gesteinart. Bei S1 und S2 befinden sich jeweils 500 Tonnen Abraum (z.B. Erde), die vor dem eigentlichen Gewinnungsprozess zu W1 und W2 gebracht werden müssen. Anschließend soll das dort bereit liegende Erz (Typ 2) zu G1-G3 gebracht werden. G4

ist ein Zwischenlager und akzeptiert ausschließlich Erz von den Schaufeln S3-S6. Da das Erz (Typ 1) bei Schaufel S3 eine zu schlechte Qualität hat, soll es mit dem Erz (Typ 3) von den Schaufeln S4-S6 bei G4 vermischt werden. Dafür müssen beide Erztypen abwechselnd dorthin transportiert werden. Es ist nicht möglich, zweimal hintereinander zu G4 die gleiche Erzladung zu liefern (z.B. 1 1 oder 3 3). Hingegen kann das Erz (Typ 3) von den Schaufeln S4-S6 jederzeit zu G1-G3 transportiert werden. Neben dem Transportprozess wird in diesem Anwendungsszenario das Betanken der Transportfahrzeuge simuliert. Dies ist bei den Tanksäulen F1 und F2 möglich, wobei jeweils eine Tanksäule einer Fahrzeugklasse zugeordnet ist. In dem betrachteten Anwendungsszenario bestehen die Fahrzeugflotten aus zwei verschiedenen Fahrzeugmodellen (Klasse C1 und C2). Folgende Eigenschaften haben die Fahrzeugklassen:

Tabelle 5.5: Technische Daten zu den Fahrzeugklassen

Eigenschaften	Nutzlast	Tankinhalt	K: Leerlauf	K: Leerfahrt	K: Vollfahrt
Klasse C1	25	250	5	10	15
Klasse C2	35	350	6	12	24

Fahrzeugmodell C1 repräsentiert ein kleineres Transportfahrzeug als C2. C1 kann maximal 25 Tonnen Erz transportieren und C2 35. „K“ steht für „Kraftstoffverbrauch“. In Tabelle 5.5 ist der Tankinhalt in Liter und der durchschnittliche Kraftstoffverbrauch in Liter pro Stunde angegeben. Es gibt eine besondere Einschränkung für die Fahrzeuge der Klasse C2. Die Fahrzeuge der Klasse C2 sind zu groß für den Unterbau und können daher ausschließlich an der Oberfläche eingesetzt werden. Das bedeutet, dass die Fahrzeuge nur Erz von den Schaufeln S1-S3 abtransportieren. Die Fahrzeuge der Klasse C2 können alle Schaufeln erreichen. Die Größen der Fahrzeugflotten variieren diesmal in den Testdurchläufen zwischen 15, 20, 25, 30, 35, 40 Fahrzeugen mit jeweils 10 (C1) + 5 (C2) = 15, 10 (C1) + 10 (C2) = 20, 15 (C1) + 10 (C2) = 25, 15 (C1) + 15 (C2) = 30, 20 (C1) + 15 (C2) = 35 und 20 (C1) + 20 (C2) = 40 Fahrzeugen. Die Distanzen und durchschnittlichen Fahrtzeiten zwischen den Orten sind in Tabelle 5.6 aufgelistet. Die Zeiten sind in Minuten und die Distanzen in Kilometer angegeben. Der linke Wert bei den Fahrtzeiten bezieht sich auf die Fahrzeuge der Klasse C1 und der rechte Wert auf die Klasse C2. Die Leerfahrt ist die Hinfahrt von den Abladestellen zu den Schaufeln und die Vollfahrt die Rückfahrt von den Schaufeln zu den Abladestellen. Das Beladen und Tanken dauert für die Fahrzeuge der Klassen C1 4 und für Klasse C2 5 Minuten.

ten. Das Entladen dauert bei beiden Fahrzeugtypen jeweils 3 Minuten. Alle Fahrzeuge starten bei G1. Die gesamte Problembeschreibung ist im Anhang A1 zu finden.

Tabelle 5.6: Distanzen und Fahrtzeiten

von G1-G3 nach	S1-S3	S4-S6	G4	S1-S3	S4-S6	W1-W2	S1-S3	S4-S6
Fahrtzeit (Leer)	13/14	22		1.5/1.5	10		13/14	22
Fahrtzeit (Voll)	17/18	30		2	14		17/18	30
Distanz	3	5.5		0.5	2.5		3	5.5

### 5.2.1 Evaluation

Zunächst wird die Gesamttonnage beurteilt (Abbildung 5.6). Das Planungssystem erzielt auch für den Fall mit 40 Fahrzeugen einen sichtbaren Anstieg der Gesamttonnage. Das lässt darauf schließen, dass aufgrund der größeren Entfernung zwischen den Ablade- /Entladeorten sowie einer zusätzlichen Schaufel weniger Engpässe entstehen.

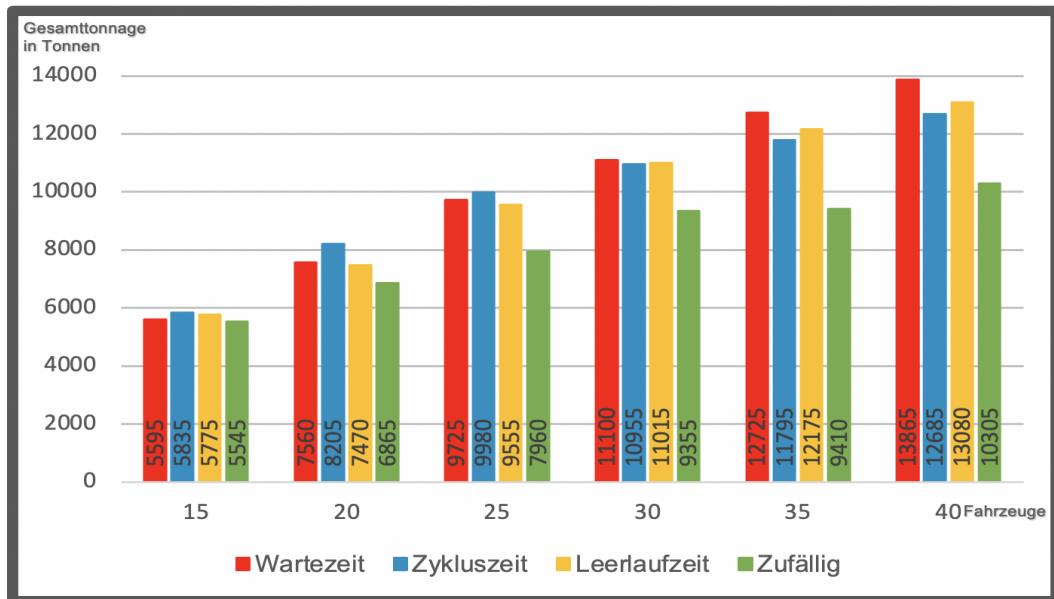


Abbildung 5.6: Anwendungsszenario 2 - Ergebnisse der Gesamttonnage

Interessant ist, dass die Heuristik kürzeste Zykluszeit nur für die Testfälle bis 25 Fahrzeuge am Besten abschneidet. Danach wird diese von beiden anderen Strategien überholt. Die „gierigere“ Strategie kürzeste Wartezeit erzielt bei größeren Fahrzeugflotten die größte Gesamttonnage. Im Test mit 40 Fahrzeugen erzielt die heuristische Methode kürzeste Wartezeit in der selben Zeit 9,3 Prozent ( $13865/12685-1$ ) mehr Produktivität als die kürzeste Zykluszeit.

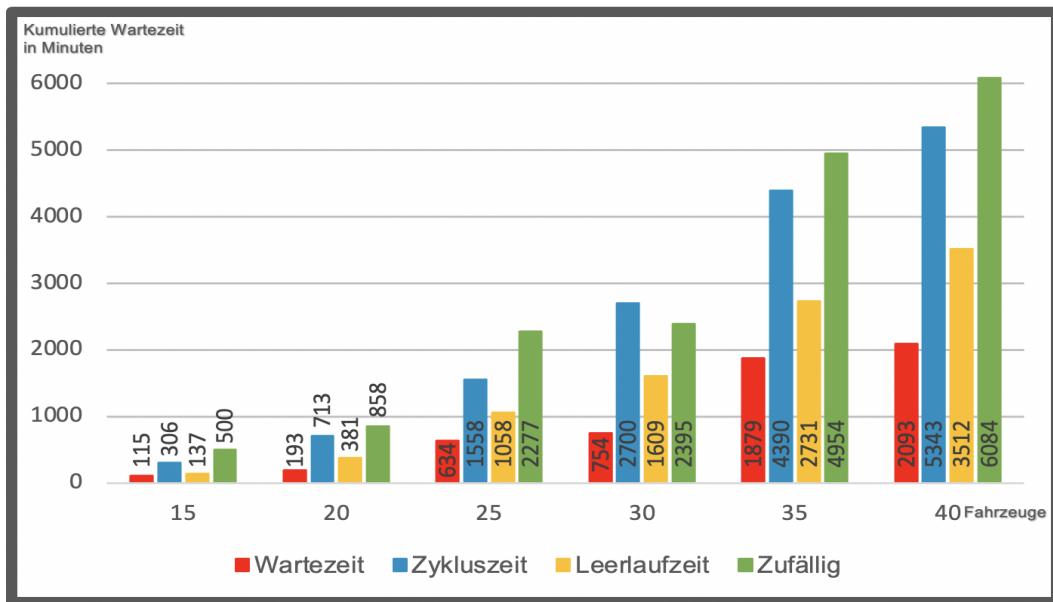


Abbildung 5.7: Gesamtwartezeit aller Fahrzeuge

Es ist in Abbildung 5.7 zu sehen, dass die Strategie der kürzesten Wartezeit in allen Testfällen am wenigsten Wartezeit erzeugt. Obwohl die Strategie kürzeste Zykluszeit die Wartezeiten ebenfalls berücksichtigt, verursacht diese trotzdem insgesamt mehr Wartezeiten. Die Wartezeiten betragen beispielsweise im Test mit 40 Fahrzeugen für die Strategie kürzeste Wartezeit 52,3 (2093/40), für die Strategie kürzeste Zykluszeit 133,6 und für die Strategie längste Leerlaufzeit 87,8 Minuten pro Fahrzeug (bezogen auf 8 Stunden). Abbildung 5.8 zeigt die zurückgelegte Gesamtstrecke für alle Fahrzeuge. Daran lässt sich erkennen, dass die Fahrzeuge aufgrund der langen Wartezeiten bei S1 und S2 öfter einen längeren Fahrtweg in den Untertagebau in Kauf nehmen. Neben den Wartezeiten zeigt Tabelle 5.7, dass durch die Einschränkungen des Transportprozesses und den Fahrzeugen Schaufeln S1 und S2 am Häufigsten besucht werden. Tabelle 5.8 veranschaulicht den gesamten Kraftstoffverbrauch in Liter. Schließlich zeigt Abbildung 5.9 (auf der übernächsten Seite) die kumulierten Leerlaufzeiten aller Schaufeln, also die Zeit in denen keine Fahrzeuge beladen werden. Da in diesem Anwendungsszenario die Zeiten zum Beladen bei allen Schaufeln (abhängig von der Fahrzeugklasse) gleich sind, variieren die Werte weniger und korrelieren/fallen mit höherer Gesamttonnage.

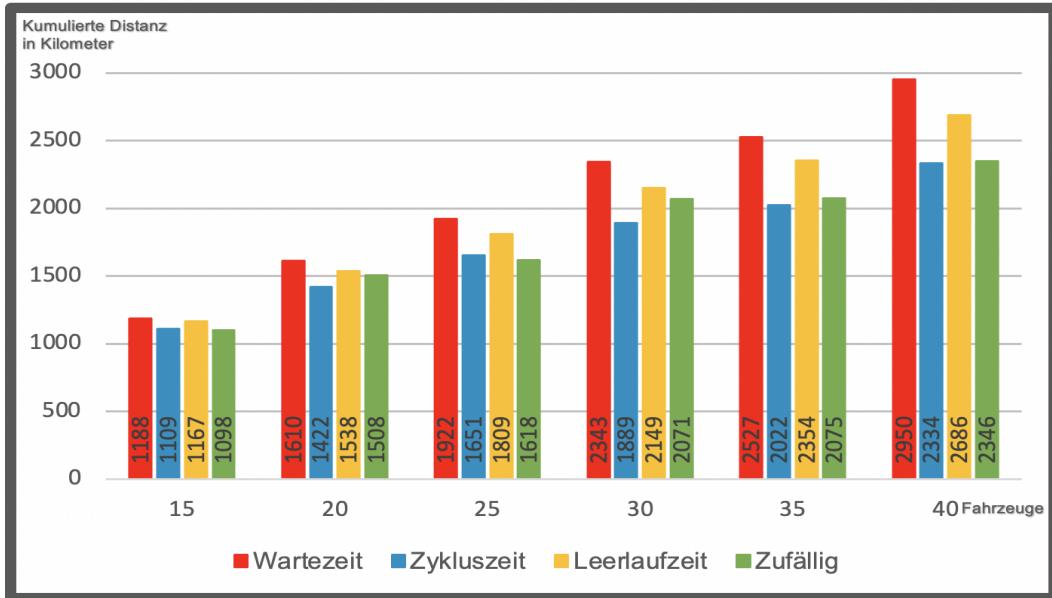


Abbildung 5.8: Zurückgelegte Gesamtstrecke aller Fahrzeuge

Tabelle 5.7: Anzahl der Fahrten zu den Schaufeln (S1:S2:S3:S4:S5:S6)

Strategie \ Fahrzeuge	15	20	25
Wartezeit	<b>60:62:1:1:1:52</b>	<b>44:72:22:9:25:78</b>	<b>77:89:31:12:32:72</b>
Zykluszeit	<b>86:97:1:0:0:1</b>	<b>92:93:45:1:13:32</b>	<b>96:98:64:1:14:54</b>
Leerlaufzeit	<b>61:63:8:16:16:19</b>	<b>75:80:19:25:25:26</b>	<b>91:91:37:29:32:31</b>
Zufällig	<b>69:66:9:9:15:9</b>	<b>69:77:13:24:24:16</b>	<b>97:92:15:17:16:18</b>
	30	35	40
Wartezeit	<b>80:84:38:26:48:90</b>	<b>95:95:57:25:57:84</b>	<b>96:96:59:47:72:91</b>
Zykluszeit	<b>102:103:75:3:12:74</b>	<b>101:102:79:4:18:85</b>	<b>106:105:80:6:37:93</b>
Leerlaufzeit	<b>98:98:51:41:45:36</b>	<b>99:99:61:49:45:48</b>	<b>100:100:68:57:52:63</b>
Zufällig	<b>95:100:20:35:31:26</b>	<b>103:106:14:34:25:21</b>	<b>108:107:20:45:32:21</b>

Tabelle 5.8: Kraftstoffverbrauch aller Fahrzeuge

Strategie \ Fahrzeuge	15	20	25	30	35	40
Wartezeit	7044	9934	11914	13790	15021	17483
Zykluszeit	6798	8547	11839	13826	14993	16539
Leerlaufzeit	6912	9527	12901	15534	14958	17241
Zufällig	6623	8353	10720	11970	15223	16463

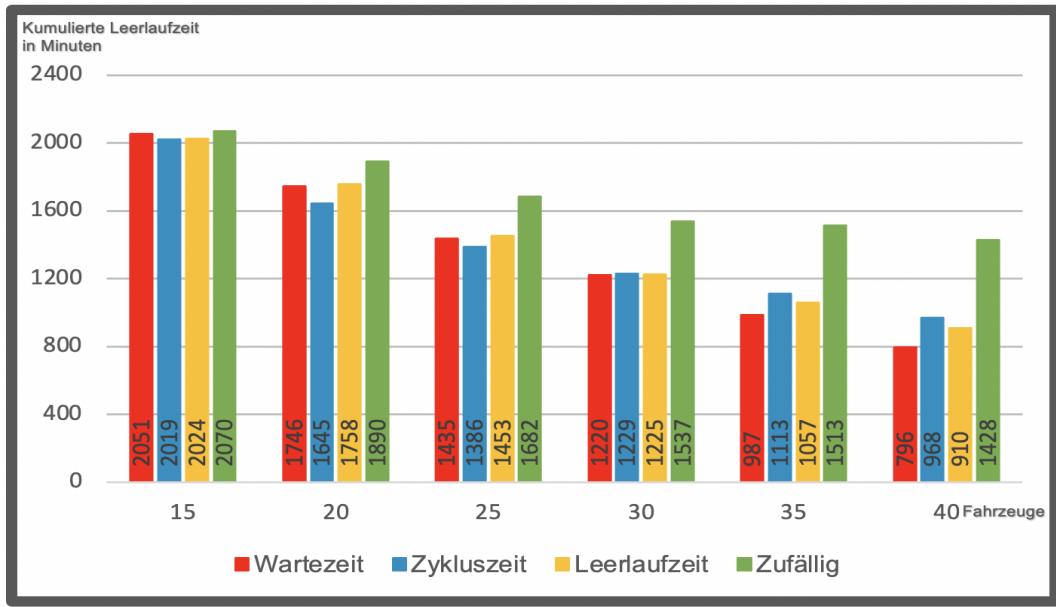


Abbildung 5.9: Kumulierte Leerlaufzeiten aller Schaufeln

Da das Planungssystem in diesem Anwendungsfall in jeder Iteration mehrere Kombinationen betrachten muss, benötigt Shop mehr Zeit die Pläne zu generieren (Tabelle 5.9). In Anwendungsbeispiel 1 gibt es z.B. 5 Schaufeln und 2 Entladestellen. Das entspricht  $5 \times 2 = 10$  Kombinationen hinsichtlich der Fahrtenplanung. In Anwendungsszenario 2 gibt es 6 Schaufeln und 6 Entladestellen, also bestehen 36 verschiedene Kombinationen, in jedem Planungsschritt einem Fahrzeug eine neue Aufgabe zuzuweisen (unabhängig von den spezifischen Einschränkungen der Fahrzeuge und der Transportladung). Daher wird auch bei mehreren Fahrzeugen, die parallel geplant werden müssen, mehr Zeit benötigt. Die Berechnungszeiten sind in Sekunden angegeben.

Tabelle 5.9: Dauer der Plansuche

Strategie \ Fahrzeuge	15	20	25	30	35	40
Wartezeit	16	30	53	69	100	135
Zykluszeit	16	36	61	95	108	123
Leerlaufzeit	17	32	53	84	125	142
Zufällig	84	151	ca. 4 min	ca. 5-6 min	ca. 6 min	ca. 7 min

Insgesamt haben die Ergebnisse in beiden Anwendungsszenarien gezeigt, dass die Strategien in dem Modell bei unterschiedlichen Flottengrößen verschiedene Ergebnisse liefern. Ob eine Strategie effizienter funktioniert, hängt von der Situation und den Anforderungen des Bergwerks ab. Außerdem sind die Ergebnisse der zufallsbasierten Strategie nicht auf einer Ebene mit den anderen Ergebnissen zu vergleichen. Der

Betrachtungszeitraum ist zu groß und die Simulation würde zu lange dauern, um eine ausreichende Anzahl an zufälligen Plänen zu generieren, die vergleichbar gute bzw. bessere Ablaufpläne erzeugen. Beispielsweise könnte das Planungssystem mit der zufallsbasierten Suche (tausende) Pläne für eine einstündige Schicht erzeugen und dadurch messbarer gemacht werden.



# Kapitel 6

## Zusammenfassung

Abschließend werden in diesem Kapitel die Ergebnisse dieser Arbeit zusammengefasst und verschiedene Verbesserungsvorschläge für das Planungssystem gemacht.

### 6.1 Fazit

Im Rahmen des Projekts ARTUS möchte die vorliegende Arbeit einen neuen Ansatz zur Entwicklung von intelligenten Flottenmanagementsystemen in hybriden Bergwerken präsentieren. Dafür wurde mit Hilfe der Hierarchischen-Task-Netzwerk-Planung und SHOP3 ein Planungssystem entwickelt, dass dazu beitragen kann, die Produktivität der Fahrzeugflotten zu erhöhen und die Kosten zu senken.

Die Einsatzplanung von autonomen Fahrzeugflotten in hybriden Bergwerken ist eine vielseitige und komplexe Aufgabe. Das vorgestellte Modell bietet eine Grundlage, die Ablaufplanung der Fahrzeugflotten unter Berücksichtigung verschiedener Restriktionen für den Transportprozess zu optimieren. Obwohl die temporäre Planung von Multiagentensystemen eine rechenintensive Aufgabe ist, haben die Ergebnisse gezeigt, dass das Modell imstande ist, Ablaufpläne für größere Fahrzeugflotten und komplexe Planungsprobleme zu generieren. Darüber hinaus ist das System skalierbar und lässt sich auch auf heterogene Fahrzeugflotten anwenden. Ziel des Planungssystems ist es nicht robuste und kontinuierliche, also auf dynamische Ereignisse reagierende [vgl. Estlin et al., 2000], Pläne für die Fahrzeuge zu generieren. Das Planungssystem ist als eine Entscheidungsunterstützung (High-Level-Control) zu sehen, dass unter Berücksichtigung des Gesamtprozesses den Fahrzeugen ihre nächsten Ziele

zuweist. Hierfür werden auf Basis der aktuellen Blockierungen und Wartezeiten an den Ressourcen, den Einschränkungen der Fahrzeuge selbst, den Nebenbedingungen des Transportprozesses sowie den durchschnittlichen Operationszeiten (Fahrtzeit, Beladezeit usw.) optimierte Aufgabenzuweisungen für die Fahrzeuge bestimmt.

Die Ergebnisse der Testdurchläufe im fünften Kapitel dienen zur Simulation des Planungssystems und zur empirischen Bewertung der Strategien. Die Szenarien haben gezeigt, dass die Ergebnisse der Strategien, abhängig von den unterschiedlichen Umgebungen und Flottengrößen, variieren können.

## 6.2 Ausblick

Das Modell könnte in zweierlei Hinsicht verbessert werden. Zum einen könnten besser bessere Strategien entwickelt werden. So wird in Kapitel 2.1 eine Methode namens „m-Trucks-für-1-Schaufel“ vorgestellt, die den Schaufeln Prioritäten zuordnet. In dem entwickelten Planungssystem ist die Strategie der längsten Leerlaufzeit die einzige Strategie, die die Produktivität der Schaufeln berücksichtigt, indem die Fahrzeuge gleichmäßig zu den Schaufeln verteilt werden. Beispielsweise könnte bei fest vorgegebenen Produktionszielen für jede Schaufel und einem Job-Shop-Algorithmus [vgl. Mueller et al., 2019], der den Prozessfortschritt der Schaufeln laufend überwacht und Produktionsverspätungen minimiert, globalere Handlungsdirektiven für die Fahrzeuge berechnet werden. Außerdem ist die Heuristik der „Minimierung der Schaufelsättigung“ (Kapitel 2.1), bei der die Fahrzeuge in gleichen Zeitintervallen zu den Schaufeln geschickt werden, eine interessante Methode die Effizienz zu steigern und bei großen Fahrzeugflotten Wartezeiten zu vermeiden. Neben den heuristischen Strategien kann die zufallsbasierte Plansuche für kurzfristige Betrachtungszeiträume deutlich mehr Pläne generieren und dadurch bessere Abläufe finden. Die zufallsbasierte Suche und die heuristischen Strategien könnten beispielsweise zu einer intelligenteren Strategie kombiniert werden. Zum anderen ist das Modell für die reale Umwelt zu abstrakt. Denn in den engen Schächten Untertage können sich nur begrenzt Fahrzeuge aufhalten. In dem vorgestellten Modell haben die Warteschlangen keine Kapazitäten bzw. das System plant das Warten der Fahrzeuge an den Ressourcen unabhängig voneinander. Es wird zudem mit einer durchschnittlichen Fahrtzeit gerechnet. Wie bereits in Kapitel 4 erwähnt, gibt es in

den Bergwerken einspurige Transportstraßen (Rampen), die in die Grube bzw. Unterage führen. Durch erhöhte Verkehrsaufkommen können dort Verzögerungen entstehen, wenn ein Fahrzeug in einer Einbuchtung warten muss bis ein Anderes vorbei gefahren ist. Abbildung 6.1 zeigt eine Grafik, wie das Modell erweitert werden könnte.

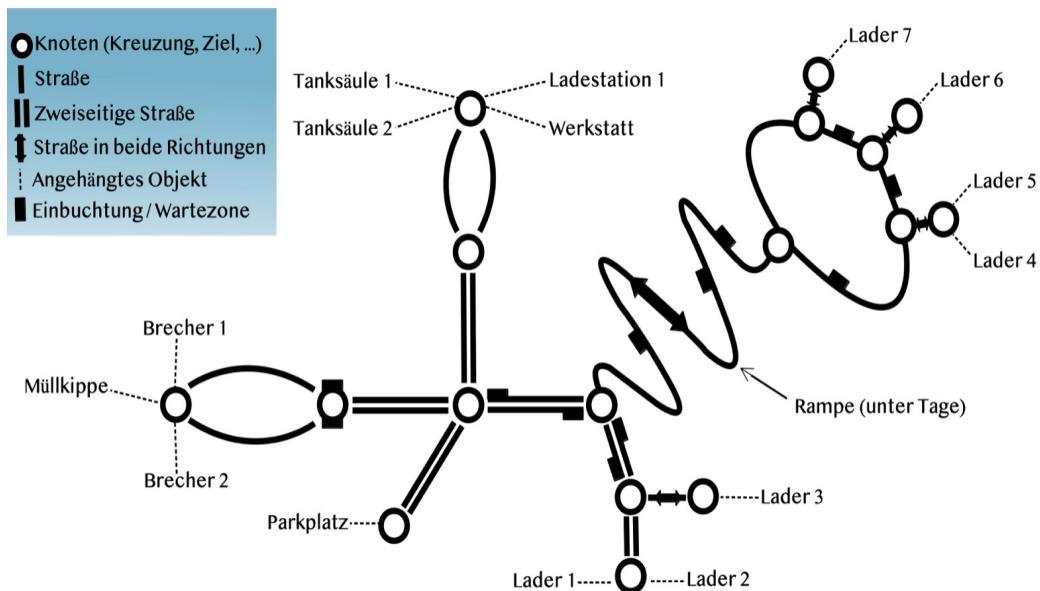


Abbildung 6.1: Grafik zur Modellerweiterung

In der Grafik des erweiterten Modells werden die Wartebereiche als zu reservierende Ressourcen in die Planung einbezogen. Außerdem werden die Strecken in Abschnitte unterteilt, sodass das Planungssystem das Verkehrsaufkommen berücksichtigen kann.



# Anhang A

## Implementierung

### A.1 Problembeschreibung

```
(truck-class c1 25 250 0.15 5 10 15 (f3 f4))
(truck-class c2 35 350 0.15 6 12 24 (s4 s5 s6 f1 f2))
(truck c1 truck1 0.0 g1 250 0 0)
(truck c1 truck2 0.0 g1 250 0 0)
(truck c1 truck3 0.0 g1 250 0 0)
(truck c1 truck4 0.0 g1 250 0 0)
(truck c1 truck5 0.0 g1 250 0 0)
(truck c2 truck6 0.0 g1 350 0 0)
(truck c2 truck7 0.0 g1 350 0 0)
(truck c2 truck8 0.0 g1 350 0 0)
(truck c2 truck9 0.0 g1 350 0 0)
(truck c2 truck10 0.0 g1 350 0 0)
(search-configuration 480 cycle)
(queue 0)
(distance 0)
(consumption 0 0)
(source s1 ((4 500) (2 1000000) ((c1 4) (c2 5)) 0 0 0 ((-1 0)))
(source s2 ((4 500) (2 1000000) ((c1 4) (c2 5)) 0 0 0 ((-1 0)))
(source s3 ((3 1000000) ((c1 4) (c2 5)) 0 0 0 ((-1 0)))
(source s4 ((1 1000000) ((c1 4)) 0 0 0 ((-1 0)))
(source s5 ((1 1000000) ((c1 4)) 0 0 0 ((-1 0)))
(source s6 ((1 1000000) ((c1 4)) 0 0 0 ((-1 0)))
(goal g0 0 ((c1 3) (c2 3)) ((1 2)) 0 0 ((-1 0)))
(goal g1 0 ((c1 3) (c2 3)) ((1 2)) 0 0 ((-1 0)))
(goal g2 0 ((c1 3) (c2 3)) ((1 2)) 0 0 ((-1 0)))
(goal g3 0 ((c1 3) (c2 3)) ((1 (3)) 0 0 ((-1 0)))
(goal w1 0 ((c1 3) (c2 3)) ((4)) 0 0 ((-1 0)))
(goal w2 0 ((c1 3) (c2 3)) ((4)) 0 0 ((-1 0)))
(filling f1 ((c1 4)) ((-1 0)))
(filling f2 ((c1 4)) ((-1 0)))
(filling f3 ((c2 5)) ((-1 0)))
(filling f4 ((c2 5)) ((-1 0)))
(routes ((s1 ((g0 (3 ((c1 17) (c2 18)))) 
(g1 (3 ((c1 17) (c2 18))))
```

```

(g2 (3 ((c1 17) (c2 18))))
(w1 (3 ((c1 17) (c2 18))))
(w2 (3 ((c1 17) (c2 18))))))
(s2 ((g0 (3 ((c1 17) (c2 18))))
      (g1 (3 ((c1 17) (c2 18))))
      (g2 (3 ((c1 17) (c2 18))))
      (w1 (3 ((c1 17) (c2 18))))
      (w2 (3 ((c1 17) (c2 18))))))
(s3 ((g3 (0.5 ((c1 2) (c2 2))))))
(s4 ((g0 (3 ((c1 17) (c2 18))))
      (g1 (5.5 ((c1 30))))
      (g2 (5.5 ((c1 30))))
      (g3 (2.5 ((c1 14))))))
(s5 ((g0 (3 ((c1 17) (c2 18))))
      (g1 (5.5 ((c1 30))))
      (g2 (5.5 ((c1 30))))
      (g3 (2.5 ((c1 14))))))
(s6 ((g0 (3 ((c1 17) (c2 18))))
      (g1 (5.5 ((c1 30))))
      (g2 (5.5 ((c1 30))))
      (g3 (2.5 ((c1 14))))))
(g0 ((s1 (3 ((c1 13) (c2 14))))
      (s2 (3 ((c1 13) (c2 14))))
      (s3 (3 ((c1 12) (c2 13)))))
      (s4 (5.5 ((c1 22))))
      (s5 (5.5 ((c1 22))))
      (s6 (5.5 ((c1 22))))
      (f1 (0.5 ((c1 2))))
      (f2 (0.5 ((c1 2))))
      (f3 (0.5 ((c2 2))))
      (f4 (0.5 ((c2 2))))))
(g1 ((s1 (3 ((c1 13) (c2 14))))
      (s2 (3 ((c1 13) (c2 14))))
      (s3 (3 ((c1 12) (c2 13)))))
      (s4 (5.5 ((c1 22))))
      (s5 (5.5 ((c1 22))))
      (s6 (5.5 ((c1 22))))
      (f1 (0.5 ((c1 2))))
      (f2 (0.5 ((c1 2))))
      (f3 (0.5 ((c2 2))))
      (f4 (0.5 ((c2 2))))))
(g2 ((s1 (3 ((c1 13) (c2 14))))
      (s2 (3 ((c1 13) (c2 14))))
      (s3 (3 ((c1 12) (c2 13)))))
      (s4 (5.5 ((c1 22))))
      (s5 (5.5 ((c1 22))))
      (s6 (5.5 ((c1 22))))
      (f1 (0.5 ((c1 2))))
      (f2 (0.5 ((c1 2)))))))

```

```

(f3 (0.5 ((c2 2))))
(f4 (0.5 ((c2 2)))))

(g3 ((s1 (0.5 ((c1 1.5) (c2 1.5))))
      (s2 (0.5 ((c1 1.5) (c2 1.5))))
      (s3 (0.5 ((c1 1.5) (c2 1.5))))
      (s4 (2.5 ((c1 10)))))
      (s5 (2.5 ((c1 10))))
      (s6 (2.5 ((c1 10))))
      (f1 (3 ((c1 18))))
      (f2 (3 ((c1 18))))
      (f3 (3 ((c2 19))))
      (f4 (3 ((c2 19)))))

(w1 ((s1 (3 ((c1 13) (c2 14))))
      (s2 (3 ((c1 13) (c2 14))))
      (s3 (3 ((c1 13) (c2 14))))
      (s4 (3 ((c1 22))))
      (s5 (3 ((c1 22))))
      (s6 (3 ((c1 22))))
      (f1 (0.5 ((c1 2))))
      (f2 (0.5 ((c1 2))))
      (f3 (0.5 ((c2 2))))
      (f4 (0.5 ((c2 2)))))

(w2 ((s1 (3 ((c1 13) (c2 14))))
      (s2 (3 ((c1 13) (c2 14))))
      (s3 (3 ((c1 13) (c2 14))))
      (s4 (3 ((c1 22))))
      (s5 (3 ((c1 22))))
      (s6 (3 ((c1 22))))
      (f1 (0.5 ((c1 2))))
      (f2 (0.5 ((c1 2))))
      (f3 (0.5 ((c2 2))))
      (f4 (0.5 ((c2 2)))))

(f1 ((s1 (3.5 ((c1 15))))
      (s2 (3.5 ((c1 15))))
      (s3 (3.5 ((c1 15))))
      (s4 (6 ((c1 23))))
      (s5 (6 ((c1 23))))
      (s6 (6 ((c1 23))))))

(f2 ((s1 (3.5 ((c1 15))))
      (s2 (3.5 ((c1 15))))
      (s3 (3.5 ((c1 15))))
      (s4 (6 ((c1 23))))
      (s5 (6 ((c1 23))))
      (s6 (6 ((c1 23))))))

(f3 ((s1 (3.5 ((c2 16))))
      (s2 (3.5 ((c2 16))))
      (s3 (3.5 ((c2 16))))))

(f4 ((s1 (3.5 ((c2 16))))
      (s2 (3.5 ((c2 16))))))

```

**A.2 Domäne**

```
(Defdomain Mine
;----- AXIOMS -----
;(:- (global-time ?global-time) ((bagof ?time (time ?truck ?time) ?list) (assign ?global-time (maximum '?list))))
;(:- (empty ?source) ((ore-at ?source ?blocks) (eval (not (if '?blocks 1)))))
;(:- (goal-reached) ((forall (?source) (source ?source) (empty ?source))))
;(:- (goal-reached) ((ore-at g1 100)))
;----- AXIOMS -----

;----- OPERATORS -----
(:op (!drive-to ?truck ?old-loc ?new-loc ?start ?duration)
:delete ((truck ?class ?truck ?start ?old-loc ?old-tank ?ore-type ?payload) (distance ?old-distance))
:add ((truck ?class ?truck ?end ?new-loc ?new-tank ?ore-type ?payload) (distance ?new-distance))
:precond ((truck ?class ?truck ?start ?old-loc ?old-tank ?ore-type ?payload)
(truck-class ?class ?max-payload ?tank-size ?refill-time ?consumption-idle ?consumption-unloaded
?consumption-loaded ?unachievable-goals)
(assign ?end (eval (+ ?start ?duration)))
(assign ?consumption (if (eq '?payload '0) (* (/ ?duration 60) ?consumption-unloaded)
(* (/ ?duration 60) ?consumption-loaded)))
(assign ?new-tank (- ?old-tank ?consumption))
(distance ?old-distance)
(routes ?routes)
(assign ?distance (car (get_by_key '?new-loc (get_by_key '?old-loc '?routes))))
(assign ?new-distance (+ ?old-distance ?distance)))
:cost 0
)

(:op (!queueing ?truck ?loc ?start ?duration)
:delete ((truck ?class ?truck ?start ?loc ?old-tank ?rock-type ?payload) (queue ?old-queue))
:add ((truck ?class ?truck ?end ?loc ?new-tank ?rock-type ?payload) (queue ?new-queue))
:precond ((truck ?class ?truck ?start ?loc ?old-tank ?rock-type ?payload)
(truck-class ?class ?max-payload ?tank-size ?refill-time ?consumption-idle ?consumption-unloaded
?consumption-loaded ?unachievable-goals)
(assign ?new-tank (- ?old-tank ?consumption-idle))
(assign ?end (eval (+ ?start ?duration)))
(queue ?old-queue)
(assign ?new-queue (+ ?old-queue ?duration)))
:cost 0
)

(:op (!load ?truck ?source ?start ?duration)
:delete ((truck ?class ?truck ?start ?source ?old-tank ?old-ore-type ?old-payload)
(source ?source ?old-ore-stack ?times-list ?old-idle-time-stamp ?old-total-utilization
?old-total-utilization-time ?old-blocked-list))
:add ((truck ?class ?truck ?end ?source ?new-tank ?new-ore-type ?new-payload)
(source ?source ?new-ore-stack ?times-list ?end ?new-total-utilization ?new-total-utilization-time
?new-blocked-list))
:precond ((truck ?class ?truck ?start ?source ?old-tank ?old-ore-type ?old-payload)
(source ?source ?old-ore-stack ?times-list ?old-idle-time-stamp ?old-total-utilization
?old-total-utilization-time ?old-blocked-list)
(truck-class ?class ?max-payload ?tank-size ?refill-time ?consumption-idle
?consumption-unloaded ?consumption-loaded ?unachievable-goals)
(assign ?new-tank (- ?old-tank ?consumption-idle))
(assign ?end (+ ?start ?duration))
(assign ?ore-transfer-data (get_ore ?max-payload '?old-ore-stack))
(assign ?new-ore-stack (nth 1 '?ore-transfer-data))
(assign ?new-ore-type (nth 0 (car '?ore-transfer-data)))
(assign ?new-payload (nth 1 (car '?ore-transfer-data)))
(assign ?new-total-utilization (+ ?old-total-utilization 1))
(assign ?new-total-utilization-time (+ ?old-total-utilization-time ?duration))
(assign ?new-blocking (append (list '?start) (list '?duration)))
(assign ?new-blocked-list (append '?old-blocked-list (list '?new-blocking))))
:cost 0)
```

```

(:op (!unload ?truck ?goal ?start ?duration)
:delete ((truck ?class ?truck ?start ?goal ?old-tank ?old-ore-type ?old-payload)
(goal ?goal ?old-ore-value ?times-list ?old-blending-order ?old-total-utilization
?old-total-utilization-time ?old-blocked-list))
:add ((truck ?class ?truck ?end ?goal ?new-tank 0 0)
(goal ?goal ?new-ore-value ?times-list ?new-blending-order ?new-total-utilization
?new-total-utilization-time ?new-blocked-list))
:precond ((truck ?class ?truck ?start ?goal ?old-tank ?old-ore-type ?old-payload)
(goal ?goal ?old-ore-value ?times-list ?old-blending-order ?old-total-utilization
?old-total-utilization-time ?old-blocked-list)
(truck-class ?class ?max-payload ?tank-size ?refill-time ?consumption-idle ?consumption-unloaded
?consumption-loaded ?unachievable-goals)
(assign ?new-tank (- ?old-tank ?consumption-idle))
(assign ?end (+ ?start ?duration))
(assign ?new-ore-value (+ ?old-ore-value ?old-payload))
(assign ?new-blending-order (append (cdr '?old-blending-order) (list (car '?old-blending-order))))
(assign ?new-total-utilization (+ ?old-total-utilization 1))
(assign ?new-total-utilization-time (+ ?old-total-utilization-time ?duration))
(assign ?new-blocking (append (list '?start) (list '?duration)))
(assign ?new-blocked-list (append '?old-blocked-list (list '?new-blocking))))
:cost 0
)

(:op (!refuel ?truck ?station ?start ?duration)
:delete ((truck ?class ?truck ?start ?station ?old-tank ?ore-type ?payload)
(consumption ?n ?old-consumption)
(filling ?station ?times-list ?old-blocked-list))
:add ((truck ?class ?truck ?end ?station ?new-tank ?ore-type ?payload)
(consumption ?new-n ?new-consumption)
(filling ?station ?times-list ?new-blocked-list))
:precond ((filling ?station ?times-list ?old-blocked-list)
(truck ?class ?truck ?start ?station ?old-tank ?ore-type ?payload)
(truck-class ?class ?max-payload ?new-tank ?refill-time ?consumption-idle
?consumption-unloaded ?consumption-loaded ?unachievable-goals)
(assign ?end (+ ?start ?duration))
(consumption ?n ?old-consumption)
(assign ?new-n (+ ?n 1))
(assign ?fill-amount (- ?new-tank ?old-tank))
(assign ?new-consumption (+ ?old-consumption ?fill-amount))
(assign ?new-blocking (append (list '?start) (list '?duration)))
(assign ?new-blocked-list (append '?old-blocked-list (list '?new-blocking))))
:cost 0
)

(:op (!!clear-blocking ?time)
:precond (delete_old_blockations ?time forall)
:cost 0
)
;----- OPERATORS END -----

```

```

;----- METHODS -----
;----- SOME DEBUGGING HELPER METHODS -----
(:method (print-current-state) ((eval (print-current-state))) ())
(:method (print-current-tasks) ((eval (print-current-tasks))) ())
(:method (print-current-plan) ((eval (print-current-plan))) ())
;----- SOME DEBUGGING HELPER METHODS END -----


;----- LOOP -----
(:method (haulage-loop)

goal-not-reached-and-pick-next-truck
((bagof ?time
  (truck ?class ?truck ?time ?loc ?tank ?ore-type ?payload)
  ?list)
 (assign ?time (car (sort '?list #'<)))
 ;(not (goal-reached))
 (truck ?class ?truck ?time ?loc ?tank ?ore-type ?payload)
 (search-configuration ?time-limit ?heuristic)
 (eval (<= ?time ?time-limit)))
 (:ordered (!!clear-blocking ?time)
   (dispatch ?truck ?heuristic)
   (haulage-loop))

goal-reached-end-loop
()
()

)
;----- LOOP-END -----


;----- DISPATCH -----
(:method (dispatch ?truck ?heuristic)

refuel
(:sort-by ?waiting-time ((truck ?class ?truck ?time ?loc ?tank ?ore-type ?payload)
  (truck-class ?class ?max-payload ?tank-size ?refill-time ?consumption-idle
    ?consumption-unloaded ?consumption-loaded ?unachievable-goals)
  (eval (< ?tank (* ?refill-time ?tank-size)))
  (filling ?station ?times-list ?filling-blocked-list)
  (eval (not (member '?station '?unachievable-goals)))
  (assign ?refill-duration (get_by_key '?class '?times-list))
  (routes ?routes)
  (assign ?drive-time (get_by_key '?class (nth 1 (get_by_key '?station
    (get_by_key '?loc '?routes))))))
  (assign ?arrival-time (+ ?time ?drive-time))
  (assign ?start-time (get_start_time '?arrival-time '?refill-duration
    '?filling-blocked-list))
  (assign ?waiting-time (- ?start-time ?arrival-time)))
  (:ordered (!drive-to ?truck ?loc ?station ?time ?drive-time)
    (queueing ?truck ?station ?arrival-time ?waiting-time)
    (!refuel ?truck ?station ?start-time ?refill-duration)))

haulage-w
; heuristic: wait
((eval (eq '?heuristic 'wait)))
(:ordered (haulage-waiting ?truck source)
  (haulage-waiting ?truck goal))

```

```

;haulage-i
; heuristic: idle
((eval (eq '?heuristic 'idle)))
(:ordered (haulage-idle ?truck)
          (haulage-waiting ?truck goal))

; haulage-c
; heuristic: cycle
((eval (eq '?heuristic 'cycle)))
(:ordered (haulage-cycle ?truck))

;haulage-r
; heuristic: random
((eval (eq '?heuristic 'random)))
(:ordered (haulage-random ?truck))

)

;----- DISPATCH-END -----
;----- DISPATCH-WAITING -----
(:method (haulage-waiting ?truck ?destination)

haulage
(:sort-by ?waiting-time ((truck ?class ?truck ?time ?old-loc ?tank ?ore-type ?payload
                           (truck-class ?class ?max-payload ?tank-size ?refill-time ?consumption-idle
                           ?consumption-unloaded ?consumption-loaded ?unachievable-goals)
                           (?destination ?loc ?ore-stack ?times-list ?data ?total-utilization
                           ?total-utilization-time ?blocked-list)
                           (eval (not (member '?loc '?unachievable-goals)))
                           (eval (if (eq '?destination 'goal) (if (member '?ore-type (car '?data)) 1 1))
                           (routes ?routes)
                           (assign ?drive-time (get_by_key '?class (nth 1 (get_by_key '?loc
                           (get_by_key '?old-loc '?routes))))))
                           (assign ?arrival-time (+ ?time ?drive-time))
                           (assign ?un-loading-time (get_by_key '?class '?times-list))
                           (assign ?start-time (get_start_time '?arrival-time '?un-loading-time
                           '?blocked-list))
                           (assign ?waiting-time (- ?start-time ?arrival-time))
                           (assign ?task (if (eq '?destination 'source) '!load '!unload))))
                           (:ordered (!drive-to ?truck ?old-loc ?loc ?time ?drive-time)
                           (queueing ?truck ?loc ?arrival-time ?waiting-time)
                           (?task ?truck ?loc ?start-time ?un-loading-time))

)
;----- DISPATCH-WAITING-END -----

```

```

;----- DISPATCH-IDLE -----
(:method (haulage-idle ?truck)

haulage
(:sort-by ?idle ((truck ?class ?truck ?time ?old-loc ?tank ?ore-type ?payload)
                  (truck-class ?class ?max-payload ?tank-size ?refill-time ?consumption-idle
                  ?consumption-unloaded ?consumption-loaded ?unachievable-goals)
                  (source ?loc ?ore-stack ?times-list ?idle ?total-utilization
                  ?total-utilization-time ?blocked-list)
                  (eval (not (member '?loc '?unachievable-goals)))
                  (routes ?routes)
                  (assign ?drive-time (get_by_key '?class (nth 1 (get_by_key '?loc
                  (get_by_key '?old-loc '?routes))))))
                  (assign ?arrival-time (+ ?time ?drive-time))
                  (assign ?loading-time (get_by_key '?class '?times-list))
                  (assign ?start-time (get_start_time '?arrival-time '?loading-time '?blocked-list))
                  (assign ?waiting-time (- ?start-time ?arrival-time)))
                  (:ordered (!drive-to ?truck ?old-loc ?loc ?time ?drive-time)
                  (queueing ?truck ?loc ?arrival-time ?waiting-time)
                  (!load ?truck ?loc ?start-time ?loading-time)))]
;----- DISPATCH-IDLE-END -----


;----- DISPATCH-CYCLE -----
(:method (haulage-cycle ?truck)

haulage
(:sort-by ?cycle-time ((truck ?class ?truck ?time ?old-loc ?tank ?ore-type ?payload)
                      (truck-class ?class ?max-payload ?tank-size ?refill-time ?consumption-idle
                      ?consumption-unloaded ?consumption-loaded ?unachievable-goals)
                      (source ?source ?ore-stack ?times-list ?data ?total-utilization
                      ?total-utilization-time ?blocked-list)
                      (eval (not (member '?source '?unachievable-goals)))
                      (routes ?routes)
                      (assign ?drive-time (get_by_key '?class (nth 1 (get_by_key '?source
                      (get_by_key '?old-loc '?routes))))))
                      (assign ?arrival-time (+ ?time ?drive-time))
                      (assign ?loading-time (get_by_key '?class '?times-list))
                      (assign ?start-time (get_start_time '?arrival-time !?loading-time '?blocked-list))
                      (assign ?waiting-time (- ?start-time ?arrival-time))
                      (assign ?leaving-time (+ ?start-time ?loading-time))
                      (goal ?goal ?ore-value ?times-list2 ?current-ore-types ?total-utilization2
                      ?total-utilization-time2 ?blocked-list2)
                      (eval (not (member '?goal '?unachievable-goals)))
                      (assign ?new-ore-type (car (car '?ore-stack)))
                      (eval (member '?new-ore-type (car '?current-ore-types)))
                      (assign ?drive-time2 (get_by_key '?class (nth 1 (get_by_key '?goal
                      (get_by_key '?source '?routes))))))
                      (assign ?arrival-time2 (+ ?leaving-time ?drive-time2))
                      (assign ?unloading-time (get_by_key '?class '?times-list2))
                      (assign ?start-time2 (get_start_time '?arrival-time2
                      '?unloading-time '?blocked-list2))
                      (assign ?waiting-time2 (- ?start-time2 ?arrival-time2))
                      (assign ?cycle-time (+ ?start-time2 ?unloading-time)))
                      (:ordered (!drive-to ?truck ?old-loc ?source ?time ?drive-time)
                      (queueing ?truck ?source ?arrival-time ?waiting-time)
                      (!load ?truck ?source ?start-time ?loading-time)
                      (!drive-to ?truck ?source ?goal ?leaving-time ?drive-time2)
                      (queueing ?truck ?goal ?arrival-time2 ?waiting-time2)
                      (!unload ?truck ?goal ?start-time2 ?unloading-time)))]
;----- DISPATCH-CYCLE-END -----

```

```

;----- DISPATCH-RANDOM -----
(:method (haulage-random ?truck)

haulage
(:first ((truck ?class ?truck ?time ?old-loc ?tank ?ore-type ?payload)
(truck-class ?class ?max-payload ?tank-size ?refill-time ?consumption-idle
?consumption-unloaded ?consumption-loaded ?unachievable-goals)
(routes ?routes)
(bagof ?current-ore-types
((goal ?goal ?ore-value ?times-list ?ore-types ?total-utilization
?total-utilization-time ?blocked-list)
(eval (not (member '?goal '?unachievable-goals)))
(assign ?current-ore-types (car '?ore-types))
?current-ore-types-stack-list)
(assign ?current-ore-types-list (get_ore_types '?current-ore-types-stack-list))
(bagof ?source
((source ?source ?ore-stack ?times-list ?idle ?total-utilization
?total-utilization-time ?blocked-list)
(eval (not (member '?source '?unachievable-goals)))
(eval (member (car (car '?ore-stack)) '?current-ore-types-list)))
?sources-list)
(assign ?source (nth (random (length '?sources-list)) '?sources-list))
(source ?source ?ore-stack ?times-list ?idle ?total-utilization ?total-utilization-time ?blocked-list)
(assign ?drive-time (get_by_key '?class (nth 1 (get_by_key '?source (get_by_key '?old-loc '?routes)))))
(assign ?arrival-time (+ ?time ?drive-time))
(assign ?loading-time (get_by_key '?class '?times-list))
(assign ?start-time (get_start_time '?arrival-time '?loading-time '?blocked-list))
(assign ?waiting-time (- ?start-time ?arrival-time))
(assign ?leaving-time (+ ?start-time ?loading-time))
(bagof ?goal
((goal ?goal ?ore-value ?times-list2 ?current-ore-types ?total-utilization2
?total-utilization-time2 ?blocked-list2)
(eval (not (member '?goal '?unachievable-goals)))
(eval (member (car (car '?ore-stack)) (car '?current-ore-types))))
?goals-list)
(assign ?goal (nth (random (length '?goals-list)) '?goals-list))
(goal ?goal ?ore-value ?times-list2 ?current-ore-types ?total-utilization2
?total-utilization-time2 ?blocked-list2)
(assign ?drive-time2 (get_by_key '?class (nth 1 (get_by_key '?goal (get_by_key '?source '?routes)))))
(assign ?arrival-time2 (+ ?leaving-time ?drive-time2))
(assign ?unloading-time (get_by_key '?class '?times-list2))
(assign ?start-time2 (get_start_time '?arrival-time2 '?unloading-time '?blocked-list2))
(assign ?waiting-time2 (- ?start-time2 ?arrival-time2))
(assign ?cycle-time (+ ?start-time2 ?unloading-time)))
(:ordered (!drive-to ?truck ?old-loc ?source ?time ?drive-time)
(queueing ?truck ?source ?arrival-time ?waiting-time)
(!load ?truck ?source ?start-time ?loading-time)
(!drive-to ?truck ?source ?goal ?leaving-time ?drive-time2)
(queueing ?truck ?goal ?arrival-time2 ?waiting-time2)
(!unload ?truck ?goal ?start-time2 ?unloading-time))
)
;----- DISPATCH-RANDOM-END -----
;----- QUEUEING -----
(:method (queueing ?truck ?loc ?arrival ?wait)
queue
((eval (> ?wait 0)))
(!queueing ?truck ?loc ?arrival ?wait))
not-queue
((eval (=< ?wait 0))) ())
;----- QUEUEING-END -----
;----- METHODS END -----
))

```

```

(defun get_by_key (key list)
  (loop for x in list do
    (if (eq key (car x))
        (return-from get_by_key (nth 1 x)))))

(defun delete_old_blockations (time l)
  (let ((new))
    (dolist (x l)
      (when (or (and (= (nth 0 x) -1) (= (nth 1 x) 0)) (< time (+ (nth 0 x) (nth 1 x))))
        (setq new (append new (list x)))))
    (sort new #'< :key #'car)))

(defun get_ore (p l)
  (let ((b (car l)))
    (if (<= (nth 1 b) p)
        (if (> (length (cdr l)) 0)
            (append (list b) (list (cdr l)))
            (list b))
        (if (> (length (cdr l)) 0)
            (let* ((new_block_value (- (nth 1 b) p))
                   (new_block (append (list (nth 0 b)) (list new_block_value)))
                   (new_l (append (list new_block) (cdr l)))
                   (new_payload (list (car b) p)))
                (append (list new_payload) (list new_l)))
            (let* ((new_block_value (- (nth 1 b) p))
                   (new_block (append (list (nth 0 b)) (list new_block_value)))
                   (new_payload (list (car b) p)))
                (append (list new_payload) (list (list new_block)))))))
        (let* ((e (+ s d))
               (start (car x))
               (end (+ (car x) (nth 1 x))))
          (if (> end s)
              (if (and (= start s) (= end e))
                  (setq s (+ (car x) (car (cdr x))))
                  (if (and (and (> end s) (<= end e)) (and (>= start s) (< start e)))
                      (setq s (+ (car x) (car (cdr x))))
                      (if (and (<= start s) (<= e end))
                          (setq s (+ (car x) (car (cdr x))))
                          (if (and (> end s) (<= end e))
                              (setq s (+ (car x) (car (cdr x))))
                              (if (and (>= start s) (< start e))
                                  (setq s (+ (car x) (car (cdr x)))))))))))) s)

(defun get_start_time (s d l)
  (dolist (x l)
    (let ((e (+ s d))
          (start (car x))
          (end (+ (car x) (nth 1 x))))
      (if (> end s)
          (if (and (= start s) (= end e))
              (setq s (+ (car x) (car (cdr x))))
              (if (and (and (> end s) (<= end e)) (and (>= start s) (< start e)))
                  (setq s (+ (car x) (car (cdr x))))
                  (if (and (<= start s) (<= e end))
                      (setq s (+ (car x) (car (cdr x))))
                      (if (and (> end s) (<= end e))
                          (setq s (+ (car x) (car (cdr x))))
                          (if (and (>= start s) (< start e))
                              (setq s (+ (car x) (car (cdr x)))))))))))) s)

(defun get_ore_types (l)
  (setq new (apply #'append l))
  (setq new (remove-duplicates new :test #'equal)) new)

(defparameter *problem-definition-vehicles*
  (with-open-file (str "/Users/kiliankramer/Desktop/vehicles10")
    (loop :for x = (read str nil nil) :while x :collect x)))
(defparameter *problem-definition-resources*
  (with-open-file (str "/Users/kiliankramer/Desktop/resources-new")
    (loop :for x = (read str nil nil) :while x :collect x)))
(defparameter *task-list* '(:ordered (haulage-loop)))
(make-problem 'problem (append *problem-definition-vehicles* *problem-definition-resources*) *task-list*)

(find-plans 'problem :which :first :verbose :plans :time-limit 60)

```



# Literatur

- [1] Stéphane Alarie und Michel Gamache. “Overview of Solution Strategies Used in Truck Dispatching Systems for Open Pit Mines”. In: *International Journal of Surface Mining, Reclamation and Environment* 16.1 (2002), S. 59–76. DOI: 10.1076/ijsm.16.1.59.3408. eprint: <https://doi.org/10.1076/ijsm.16.1.59.3408>. URL: <https://doi.org/10.1076/ijsm.16.1.59.3408>.
- [2] Ron Alford, Gregor Behnke, Daniel Höller, Pascal Bercher, Susanne Biundo und David W. Aha. “Bound to Plan: Exploiting Classical Heuristics via Automatic Translations of Tail-Recursive HTN Problems”. In: *Proceedings of the Twenty-Sixth International Conference on International Conference on Automated Planning and Scheduling*. ICAPS16. London, UK: AAAI Press, 2016, S. 20–28. ISBN: 1577357574.
- [3] M. Åstrand. “Short-term Underground Mine Scheduling: Constraint Programming in an Industrial Application”. Diss. Mai 2018.
- [4] Marc Asunción, Luis Castillo, Juan Fdez-Olivares, Óscar García-Pérez, Antonio González-Muñoz und Francisco Palao. “SIADEX: An interactive knowledge-based planner for decision support in forest fire fighting”. In: *AI Commun.* 18 (Jan. 2005), S. 257–268.
- [5] D.M. Bajany, L. Zhang und X. Xia. “An Optimization Approach for Shovel Allocation to Minimize Fuel Consumption in Open-pit Mines: Case of Heterogeneous Fleet of Shovels.” In: *IFAC-PapersOnLine* 52.14 (2019). 18th IFAC Symposium on Control, Optimization and Automation in Mining, Mineral and Metal Processing, MMM 2019, S. 207–212. ISSN: 2405-8963. DOI: <https://doi.org/10.1016/j.ifacol.2019.09.196>. URL: <http://www.sciencedirect.com/science/article/pii/S2405896319308316>.
- [6] G Behnke, D Höller, Pascal Bercher, S Biundo, Damien Pellier, Humbert Fiorino und R Alford. “Hierarchical Planning in the IPC”. In: Juli 2019.
- [7] G. Behnke, D. Höller und Susanne Biundo-Stephan. “totSAT - Totally-Ordered Hierarchical Planning Through SAT”. In: *AAAI*. 2018.
- [8] Pascal Bercher, Gregor Behnke, Daniel Höller und Susanne Biundo. “An Admissible HTN Planning Heuristic”. In: *Proceedings of the 26th International Joint Conference on Artificial Intelligence*. IJCAI’17. Melbourne, Australia: AAAI Press, 2017, S. 480–488. ISBN: 9780999241103.
- [9] Michael Brenner. “Multiagent Planning with Partially Ordered Temporal Plans.” In: *In Proceedings of IJCAI03*. 2003.

- [10] J. Benndorf C.R.O. Mariz A. Prior. "Recoverable resource estimation mixing different quality of data". In: *Mining Goes Digital: Proceedings of the 39th International Symposium 'Application of Computers and Operations Research in the Mineral Industry'*. Bd. 720. 1st Edition. London: Taylor Francis, 2019, S. 530–535. ISBN: 9780429320774. DOI: 10.1145/378239.379017.
- [11] Patarawan Chaowasakoo, Heikki Seppälä, Heikki Koivo und Quan Zhou. "Digitalization of mine operations: Scenarios to benefit in real-time truck dispatching". In: *International Journal of Mining Science and Technology* 27 (Jan. 2017). DOI: 10.1016/j.ijmst.2017.01.007.
- [12] Wesley Cox, Tim French, Mark Reynolds und Lyndon While. "A Genetic Algorithm for Truck Dispatching in Mining". In: *GCAI 2017. 3rd Global Conference on Artificial Intelligence*. Hrsg. von Christoph Benzmüller, Christine Lisetti und Martin Theobald. Bd. 50. EPiC Series in Computing. EasyChair, 2017, S. 93–106. DOI: 10.29007/n11t. URL: <https://easychair.org/publications/paper/3PFP>.
- [13] F. Dvorák, R. Barták, A. Bit-Monnot, F. Ingrand und M. Ghallab. "Planning and Acting with Temporal and Hierarchical Decomposition Models". In: *2014 IEEE 26th International Conference on Tools with Artificial Intelligence*. 2014, S. 115–121. DOI: 10.1109/ICTAI.2014.27.
- [14] Niklas Eén und Niklas Sörensson. "An Extensible SAT-solver". In: *Theory and Applications of Satisfiability Testing*. Hrsg. von Enrico Giunchiglia und Armando Tacchella. Berlin, Heidelberg: Springer Berlin Heidelberg, 2004, S. 502–518. ISBN: 978-3-540-24605-3.
- [15] Kutluhan Erol, James Hendler und Dana Nau. "UMCP: A Sound and Complete Procedure for Hierarchical Task-Network Planning". In: *Proceedings of the International Conference on AI Planning Systems* (Apr. 2003).
- [16] Daniel Espinoza, Marcos Goycoolea, Eduardo Moreno und Alexandra Newman. "MineLib: a library of open pit mining problems". In: *Annals of Operations Research* 206.1 (Juli 2013), S. 93–114. DOI: 10.1007/s10479-012-1258-3. URL: <https://ideas.repec.org/a/spr/annopr/v206y2013i1p93-11410.1007-s10479-012-1258-3.html>.
- [17] Richard E. Fikes und Nils J. Nilsson. "Strips: A new approach to the application of theorem proving to problem solving". In: *Artificial Intelligence* 2.3 (1971), S. 189–208. ISSN: 0004-3702. DOI: [https://doi.org/10.1016/0004-3702\(71\)90010-5](https://doi.org/10.1016/0004-3702(71)90010-5). URL: <http://www.sciencedirect.com/science/article/pii/0004370271900105>.
- [18] M. Fox und D. Long. "PDDL2.1: An Extension to PDDL for Expressing Temporal Planning Domains". In: *Journal of Artificial Intelligence Research* 20 (Dez. 2003), S. 61–124. ISSN: 1076-9757. DOI: 10.1613/jair.1129. URL: <http://dx.doi.org/10.1613/jair.1129>.
- [19] Ilche Georgievski und Marco Aiello. "HTN planning: Overview, comparison, and beyond". In: *Artificial Intelligence* 222 (2015), S. 124–156. ISSN: 0004-3702. DOI: <https://doi.org/10.1016/j.artint.2015.02.002>. URL: <http://www.sciencedirect.com/science/article/pii/S0004370215000247>.
- [20] Malik Ghallab u. a. "PDDL - The Planning Domain Definition Language". In: (Aug. 1998).

- [21] Robert P. Goldman und Ugur Kuter. "Hierarchical Task Network Planning in Common Lisp: the case of SHOP3". Genova, Italy: ELSAA, 2019, S. 73–80. ISBN: 978-2-9557474-3-8. URL: <https://european-lisp-symposium.org/static/proceedings/2019.pdf>.
- [22] P. E. Hart, N. J. Nilsson und B. Raphael. "A Formal Basis for the Heuristic Determination of Minimum Cost Paths". In: *IEEE Transactions on Systems Science and Cybernetics* 4.2 (1968), S. 100–107. DOI: 10.1109/TSSC.1968.300136.
- [23] Malte Helmert. "The Fast Downward Planning System". In: *CoRR* abs/1109.6051 (2011). arXiv: 1109.6051. URL: <http://arxiv.org/abs/1109.6051>.
- [24] Joerg Hoffmann. "FF: The Fast-Forward Planning System". In: *AI Magazine* 22.3 (Sep. 2001), S. 57. DOI: 10.1609/aimag.v22i3.1572. URL: <https://ojs.aaai.org/index.php/aimagazine/article/view/1572>.
- [25] Daniel Höller, Gregor Behnke, Pascal Bercher, Susanne Biundo, Humbert Fiorino, Damien Pellier und Ron Alford. "HDDL: An Extension to PDDL for Expressing Hierarchical Planning Problems". In: *Proceedings of the AAAI Conference on Artificial Intelligence* 34.06 (Apr. 2020), S. 9883–9891. DOI: 10.1609/aaai.v34i06.6542. URL: <https://ojs.aaai.org/index.php/AAAI/article/view/6542>.
- [26] Daniel Höller, Pascal Bercher, Gregor Behnke und Susanne Biundo. "On Guiding Search in HTN Planning with Classical Planning Heuristics". In: *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence, IJCAI-19*. International Joint Conferences on Artificial Intelligence Organization, Juli 2019, S. 6171–6175. DOI: 10.24963/ijcai.2019/857. URL: <https://doi.org/10.24963/ijcai.2019/857>.
- [27] Daniel Höller, Pascal Bercher, Gregor Behnke und Susanne Biundo. "On Guiding Search in HTN Planning with Classical Planning Heuristics". In: *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence, IJCAI-19*. International Joint Conferences on Artificial Intelligence Organization, Juli 2019, S. 6171–6175. DOI: 10.24963/ijcai.2019/857. URL: <https://doi.org/10.24963/ijcai.2019/857>.
- [28] Bozo Kolonja, David R. Kalasky und Jan M. Mutmansky. "Optimization of Dispatching Criteria for Open-Pit Truck Haulage System Design Using Multiple Comparisons with the Best and Common Random Numbers". In: *Proceedings of the 25th Conference on Winter Simulation*. WSC '93. Los Angeles, California, USA: Association for Computing Machinery, 1993, S. 393–401. ISBN: 078031381X. DOI: 10.1145/256563.256671. URL: <https://doi.org/10.1145/256563.256671>.
- [29] Guangwei Liu und Chai Senlin. "Optimizing Open-Pit Truck Route Based on Minimization of Time-Varying Transport Energy Consumption". In: *Mathematical Problems in Engineering* 2019 (Aug. 2019), S. 1–12. DOI: 10.1155/2019/6987108.
- [30] Alan K. Mackworth. "Consistency in networks of relations". In: *Artificial Intelligence* 8.1 (1977), S. 99–118. ISSN: 0004-3702. DOI: [https://doi.org/10.1016/0004-3702\(77\)90007-8](https://doi.org/10.1016/0004-3702(77)90007-8). URL: <http://www.sciencedirect.com/science/article/pii/0004370277900078>.

- [31] Luis Montiel und Roussos Dimitrakopoulos. "Optimizing mining complexes with multiple processing and transportation alternatives: An uncertainty-based approach". In: *European Journal of Operational Research* 247.1 (2015), S. 166–178. ISSN: 0377-2217. DOI: <https://doi.org/10.1016/j.ejor.2015.05.002>. URL: <http://www.sciencedirect.com/science/article/pii/S0377221715003720>.
- [32] M. W. Moskewicz, C. F. Madigan, Y. Zhao, L. Zhang und S. Malik. "Chaff: engineering an efficient SAT solver". In: *Proceedings of the 38th Design Automation Conference (IEEE Cat. No.01CH37232)*. 2001, S. 530–535. DOI: 10.1145/378239.379017.
- [33] Mohan Munirathinam und Jon C. Yingling. "A review of computer-based truck dispatching strategies for surface mining operations". In: *International Journal of Surface Mining, Reclamation and Environment* 8.1 (1994), S. 1–15. DOI: 10.1080/09208119408964750. eprint: <https://doi.org/10.1080/09208119408964750>. URL: <https://doi.org/10.1080/09208119408964750>.
- [34] D. Nau, Y. Cao, A. Lotem und Hector Muñoz-Avila. "SHOP: Simple Hierarchical Ordered Planner". In: *IJCAI*. 1999.
- [35] D. S. Nau, T. C. Au, O. Ilghami, U. Kuter, J. W. Murdock, D. Wu und F. Yaman. "SHOP2: An HTN Planning System". In: *Journal of Artificial Intelligence Research* 20 (Dez. 2003), S. 379–404. ISSN: 1076-9757. DOI: 10.1613/jair.1141. URL: <http://dx.doi.org/10.1613/jair.1141>.
- [36] Nicoleta Neagu, Klaus Dorer und Monique Calisti. "Solving Distributed Delivery Problems with Agent-Based Technologies and Constraint Satisfaction Techniques." In: Jan. 2006, S. 159–160.
- [37] E. Pednault. "ADL: Exploring the Middle Ground Between STRIPS and the Situation Calculus". In: *KR*. 1989.
- [38] Abdeldjalil Ramoul, Damien Pellier, Humbert Fiorino und Sylvie Pesty. "Grounding of HTN Planning Domain". In: *International Journal on Artificial Intelligence Tools* 26 (Okt. 2017), S. 1760021. DOI: 10.1142/S0218213017600211.
- [39] Stuart Russell und Peter Norvig. *Artificial Intelligence: A Modern Approach*. 3rd. USA: Prentice Hall Press, 2009. ISBN: 0136042597.
- [40] Earl D. Sacerdoti. "Planning in a hierarchy of abstraction spaces". In: *Artificial Intelligence* 5.2 (1974), S. 115–135. ISSN: 0004-3702. DOI: [https://doi.org/10.1016/0004-3702\(74\)90026-5](https://doi.org/10.1016/0004-3702(74)90026-5). URL: <http://www.sciencedirect.com/science/article/pii/0004370274900265>.
- [41] Earl D. Sacerdoti. "The Nonlinear Nature of Plans". In: *Proceedings of the 4th International Joint Conference on Artificial Intelligence - Volume 1*. IJCAI'75. Tbilisi, USSR: Morgan Kaufmann Publishers Inc., 1975, S. 206–214.
- [42] David. E. Smith Sara Bernardini. "Developing Domain-Independent Search Control for EUROPA2". In: (2007).
- [43] Dominik Schreiber, Damien Pellier, Humbert Fiorino und Toma´s Balyo. "Tree-REX: SAT-Based Tree Exploration for Efficient and High-Quality HTN Planning". In: *Proceedings of the International Conference on Automated Planning and Scheduling* 29.1 (Juli 2019), S. 382–390. URL: <https://ojs.aaai.org/index.php/ICAPS/article/view/3502>.
- [44] Ji-Ae Shin und Ernest Davis. "Continuous Time in a SAT-Based Planner." In: Jan. 2004, S. 531–536.

- [45] H. Simonis, P. Charlier, P. Kay und Cosytec Sa. *TACT — an integrated transportation problem solved with CHIP*. Techn. Ber. COSYTEC SA, 4, rue Jean Rostrand; F-91893 Orsay Cedex, 1996.
- [46] Bartłomiej Skawina. “Load-Haul-Dump operations in underground mines”. Diss. Mai 2019.
- [47] Larissa Statsenko und Noune Melkoumian. “Modeling Blending Process at Open-Pit Stockyards: A Northern Kazakhstan Mining Company Case Study”. In: Jan. 2014, S. 1017–1027. ISBN: 978-3-319-02677-0. DOI: 10.1007/978-3-319-02678-7\_98.
- [48] A. Tate, B. Drabble und R. Kirby. “O-Plan2: an Open Architecture for Command, Planning and Control”. In: 2006.
- [49] Austin Tate. *PROJECT PLANNING USING A HIERARCHIC NON-LINEAR PLANNER*. Techn. Ber. 1976.
- [50] Victor A. Temeng, Francis O. Otuonye und James O. Frendewey Jr. “Real-time truck dispatching using a transportation algorithm”. In: *International Journal of Surface Mining, Reclamation and Environment* 11.4 (1997), S. 203–207. DOI: 10.1080/09208119708944093. eprint: <https://doi.org/10.1080/09208119708944093>. URL: <https://doi.org/10.1080/09208119708944093>.
- [51] DAVID E. WILKINS. “Can AI planners solve practical problems?” In: *Computational Intelligence* 6.4 (1990), S. 232–246. DOI: <https://doi.org/10.1111/j.1467-8640.1990.tb00297.x>. eprint: <https://onlinelibrary.wiley.com/doi/pdf/10.1111/j.1467-8640.1990.tb00297.x>. URL: <https://onlinelibrary.wiley.com/doi/abs/10.1111/j.1467-8640.1990.tb00297.x>.
- [52] Håkan L. S. Younes und Michael L. Littman. *PPDDL 1.0: An extension to PDDL for expressing planning domains with probabilistic effects*. Techn. Ber.
- [53] Günther Zäpfel und Roland Braune. “Maschinenbelegung mit Genetischen Algorithmen”. In: *WiSt - Wirtschaftswissenschaftliches Studium* 35 (Jan. 2006), S. 566–570. DOI: 10.15358/0340-1650-2006-10-566.