

Neue Strategien zur Lösung von Isomorphieproblemen

Dem Fachbereich Mathematik der
Universität Bayreuth
zur Erlangung des akademischen Grades eines
Dr. rer. nat.

eingereichte Dissertation

von
Herrn Dipl.-Math. Matthias Koch
aus
Würzburg

1. Gutachter:
2. Gutachter:

Tag der Einreichung:
Tag des Kolloquiums:

Danke Leute

Summary

Deciding whether two objects with a given structure are identical is a common problem in mathematics, just as in other areas such as chemistry or electronic design automation. For instance, deciding whether two graphs can be made equal by renumbering their nodes is a classical isomorphism problem. In electronic design automation, effectively telling whether a small given logic circuit is implemented in a big circuit board is a common task.

Some of these problems are easy to solve, while others seem to have no efficient algorithmical solution. The subgraph isomorphism problem, for example, is known to be NP-complete and thus difficult to solve. The same goes for the graph isomorphism problem, which might be NP-complete, although this has not been proven.

Homomorphisms of group actions have turned out to be one of the most powerful concepts to solve those isomorphism problems. The Homomorphism Principle by Reinhard Laue established the mathematical basis of many efficient algorithms. However, this approach seemed futile in situations where no homomorphism in the desired direction exists.

Bernd Schmalz discovered a solution to this problem by using a homomorphism in the opposite of the usual direction. Schmalz named this technique „Leiterspiel“ after the children’s game „Snakes and Ladders“ and developed a very successful breadth first search algorithm applying this technique.

As long as all intermediate results of the Snakes and Ladders algorithm can be stored in a fast working memory, the algorithm is very efficient. But since the number of generated intermediate results grows very quickly, Snakes and Ladders was limited to problems with intermediate results fitting into the working memory. Until now, there appeared to be no alternative to storing all intermediate results, since Snakes and Ladders accesses them in no predictable order.

The most important result of the present thesis is a new mathematical approach providing an efficient way of calculating all branches where a given intermediate result is needed. This led to the development of three novel algorithms, the Depthfirst StroLL Algorithm, the Breadthfirst StroLL Algorithm and the Leiterspiel Light Algorithm which are independent of the storage of

all intermediate results. These algorithms have very modest working memory requirements but still show the same efficiency as the original Snakes and Ladders algorithm.

For every Snakes and Ladders algorithm, a series of homomorphisms, the so-called ladder, is needed. The mathematical foundation for the new algorithms are the so-called strong ladders, which require further conditions to the homomorphisms. This thesis establishes the mathematical basis of these ladders clearly and formulates proof for each conclusion.

In order to use the algorithms presented in this work, specific ladders tailored to the problem must be found. A construction strategy presented in this thesis helps the user to build the required ladders. Finally, the three new algorithms presented in this thesis were implemented in the C++ programming language. Along with Schmalz' original Snakes and Ladders, they were tested on their comparative performance and memory requirements.

Zusammenfassung

In vielen Bereichen der Mathematik, sowie in der Chemie und beim automatisierten Entwurf elektronischer Systeme, muss für zwei verschiedenen Strukturbeschreibungen festgestellt werden, ob die beschriebenen Objekte identisch sind. Ein klassisches Isomorphieproblem ist zum Beispiel die Fragestellung, ob zwei verschiedene Graphen durch eine Umbenennung ihrer Knotenmengen identisch werden können. Beim automatisierten Entwurf elektronischer Systeme taucht ein ähnliches Problem auf, wenn für zwei unterschiedliche elektronische Schaltpläne einer logischen Schaltung festgestellt werden soll, ob die beiden Schaltungen sich logisch identisch verhalten.

Einige dieser Problemstellungen sind leicht zu lösen, während für andere wiederum kein effizienter Lösungsalgorithmus bekannt ist. Beim Teilgraphenisomorphieproblem zum Beispiel konnte nachgewiesen werden, dass es in der Klasse der NP vollständigen Probleme enthalten ist und aufgrund dessen nur schwer zu lösen ist. Ähnliches gilt für das Graphenisomorphieproblem, bei dem jedoch bisher nicht festgestellt werden konnte, ob es in der Klasse der NP vollständigen Probleme liegt.

Bei vielen Isomorphieproblemen hat sich gezeigt, dass Homomorphismen von Gruppenoperationen ein mächtiges Werkzeug zur Lösung dieser Probleme sein können. Das von Reinhard Laue gefundene Homomorphieprinzip ist die mathematische Grundlage von vielen effizienten Algorithmen zur Lösung dieser Isomorphieprobleme. In einigen Situationen, in denen kein Homomorphismus in der gewünschten Richtung existiert, schien diese Technik jedoch nicht anwendbar.

Bernd Schmalz entdeckte jedoch eine Technik, bei der ein Homomorphismus in der umgekehrten zu der sonst üblichen Richtung verwendet wird. Er benannte seine „Leiterspiel“-Technik nach einem Spiel für Kinder und entwickelte einen sehr erfolgreichen Algorithmus in Breitensuche, der diese Technik verwendet. Solange alle Zwischenergebnisse, die bei der Bearbeitung einer gegebenen Problemstellung berechnet werden, im Hauptspeicher gehalten werden können, ist dieser Algorithmus außerordentlich effizient. Da die Anzahl der Zwischenergebnisse jedoch meist sehr schnell anwächst, ist der ursprüngliche Leiterspiel Algorithmus nur bei den Problemstellungen anwendbar, bei denen die

Zwischenergebnisse im Hauptspeicher Platz finden. Bis zur Veröffentlichung dieser Arbeit schien es keine Alternative zu geben, bei der die Zwischenergebnisse nicht im Speicher gehalten werden müssen, da es nicht möglich war vorherzusagen, in welchen Verzweigungen des Algorithmus ein zuvor berechnetes Zwischenergebnis benötigt wird.

Das wichtigste Ergebnis der vorliegenden Arbeit ist die mathematische Beschreibung einer neuen Vorgehensweise, mit der im Voraus berechnet werden kann, in welchen Zweigen des Algorithmus ein gegebenes Zwischenergebnis benötigt wird. Diese Technik ermöglichte drei neue Algorithmen, den Breadthfirst StroLL Algorithmus, den Depthfirst StroLL Algorithmus und den Leiterspiel Light Algorithmus, bei denen die Zwischenergebnisse nicht mehr während der gesamten Laufzeit im Speicher gehalten werden müssen. Diese Algorithmen haben im Gegensatz zum ursprünglichen Algorithmus von Schmalz einen geringen Speicherbedarf, während die Effizienz des ursprünglichen Algorithmus erhalten bleibt.

Für alle Leiterspielalgorithmen wird eine Menge Homomorphismen benötigt, die durch eine sogenannte Leiter beschrieben werden. Die Basis dieser neuen Algorithmen sind die sogenannten starken Leitern, das sind Leitern, bei denen die verwendeten Homomorphismen weitere Bedingungen erfüllen müssen. In dieser Arbeit wird die mathematische Grundlage dieser Leitern klar herausgearbeitet und alle Schlussfolgerungen werden mit Beweisen belegt.

Um die in dieser Arbeit vorgestellten Algorithmen anwenden zu können, müssen speziell auf das entsprechende Problem zugeschnittene starke Leitern bereitgestellt werden. Daher wurde in dieser Arbeit ein Weg beschrieben, wie die entsprechenden Leitern berechnet werden können. Weiterhin wurden die drei Algorithmen, die im Rahmen dieser Arbeit gefunden und beschrieben wurden, in der Programmiersprache C++ implementiert. Um die Effizienz dieser drei neuen Algorithmen unter Beweis zu stellen wurden diese mit dem ursprünglichen Algorithmus von Schmalz in Bezug auf ihre Speicheranforderungen und ihre Laufzeit verglichen.

Inhaltsverzeichnis

1	Für den eiligen Leser	1
2	Motivation	3
3	Grundlagen	9
3.1	Sätze und Definitionen	9
3.2	Homomorphieprinzip	14
3.3	Ordnungstreue Erzeugung	15
3.4	Doppelnebenklassen	18
3.5	Split Lemma	20
4	Das Leiterspiel nach Schmalz	27
4.1	Voraussetzungen	28
4.2	Fusionierende Abbildung	29
4.3	Splitting-Orbits	30
4.3.1	Voraussetzungen	30
4.3.2	Vorgehen beim Splitting-Orbits	31
4.4	Fusing Orbits	31
4.4.1	Voraussetzungen	31
4.4.2	Bestimmung der Bahnrepräsentanten	32
4.4.3	Bestimmung der fusionierenden Abbildung und der Sta- bilisatoren	32
5	Der Kanonisierungsalgorithmus am Beispiel eines Würfels	35
5.1	Drehungen eines Würfels	35
5.2	Fragestellung	36
5.3	Kanonische Färbung und Totalordnung	38

5.4	Stabilisatoren	39
5.5	Homomorphismen	39
5.6	Konstruktionspfade	40
5.7	Vorbereitungen	41
5.8	Erweiterung des Stabilisators	43
5.9	Menge der Pfade	44
5.10	Schnittmengen von Stabilisatoren	44
5.11	Fusionierende Drehung	46
5.12	Tiefensuche	49
5.13	Abschneiden von Ästen	51
5.14	Ergebnisse	53
6	Mathematische Grundlagen der Algorithmen	57
6.1	Grundbegriffe	58
6.1.1	Erweiterungsfunktion	58
6.1.2	Ordnungsrelationen	59
6.1.3	Kanonizitätsprädikat	61
6.2	Konstruktionspfade	63
6.3	Ordnungstreue Abbildungen	65
6.4	Blöcke aus Konstruktionspfaden	67
6.5	Starke Untergruppenleiter	69
6.6	Urbilder von G -Homomorphismen	77
7	Kanonizitätstests für Doppelnebenklassen	81
7.1	Würfelfärbungen und Rechtsnebenklassen	83
7.2	Breadthfirst StroLL Algorithmus	85
7.2.1	Überblick	85
7.2.2	Bestimmung der zu untersuchenden Nebenklassen	86
7.2.3	Ein Isomorphismus von Gruppenoperationen	86
7.2.4	Eine weitere Untergruppenleiter	87
7.2.5	Inklusions-Homomorphismus	87
7.2.6	Fusionierende Abbildung	88
7.2.7	Splitting-Orbits	89
7.2.8	Fusing-Orbits	91
7.2.9	Varianten des Algorithmus	93

7.3	Depthfirst StroLL Algorithmus	94
7.3.1	Überblick	94
7.3.2	Vorbereitungen	96
7.3.3	Splitting Orbits	101
7.3.4	Fusing Orbits	111
7.3.5	Pseudocode	123
7.4	Leiterspiel Light Algorithmus	127
7.4.1	Überblick	127
7.4.2	Vorbereitung	129
7.4.3	Splitting Orbits	133
7.4.4	Fusing Orbits	134
7.4.5	Berechnung des Stabilisators von $A_k q$	136
7.5	Teilgraphenisomorphieproblem und Doppelnebenklassen	137
7.6	Unterprogramme des Kanonizitätstests	140
7.6.1	Identitäten und inverse Homomorphismen	140
7.6.2	ReduceStab	144
7.6.3	Canonizer	148
7.6.4	ExtendGroup	150
7.6.5	FindOrbitRep	151
7.6.6	FindSmallestPath	151
8	Analyse des Laufzeitverhaltens	155
8.1	Kurzbeschreibung der Algorithmen	155
8.1.1	Leiterspielalgorithmus von Schmalz	155
8.1.2	Leiterspiel-Light	156
8.1.3	Depthfirst StroLL Algorithmus	156
8.1.4	Breadthfirst StroLL Leiterspiel	157
8.2	Vergleich der Eigenschaften	157
8.2.1	Der Beispielgraph	158
8.2.2	Voraussetzungen	159
8.2.3	Ergebnisse	160
8.3	Vergleich der Leistungsfähigkeit	163
8.3.1	Voraussetzungen	164
8.3.2	Ergebnisse	164
8.3.3	Ergebnis des Vergleichs	167

9 Ausblick	169
10 Bemerkungen	171
10.1 Kontrolliere folgendes	171
10.1.1 Sprachlich	171
10.1.2 Inhaltlich	173
10.1.3 Fertig	173
10.2 Allgemeines	174

Abbildungsverzeichnis

2.1	Vitamin B_{12} <small>Quelle: Wikipedia</small>	3
3.1	zwei schlichte Graphen auf acht Knoten	22
5.1	Die 24 Drehungen eines Würfels	36
5.2	Würfel mit weiß markierten Ecken	37
5.3	Würfel mit dem Färbungstupel $(2, w, 5, w, 7, w, 4, s)$	39
5.4	Beispiel einer Abbildung unter dem Fusing-Homomorphismus . .	40
5.5	Beispiel einer Abbildung unter dem Splitting-Homomorphismus	40
5.6	der kleinste Pfad zum Würfel in Abbildung 5.2	42
5.7	Würfel des Pfades ρ und deren Stabilisatoren	42
5.8	eine Untergruppe des Stabilisators	44
5.9	alle Pfade zum Würfel in Abbildung 5.2	45
5.10	Beispiel eines Pfades	47
5.11	Dieser Würfel soll an den Pfad angehängt werden	47
5.12	Würfel mit fusionierender Drehung	47
5.13	wende fusionierende Drehung an	48
5.14	Stabilisator des Bahnrepräsentanten	48
5.15	Bahn des Stabilisators	49
5.16	Würfel mit fusionierender Drehung	49
5.17	Pfad mit Darstellung der fusionierenden Drehungen	50
5.18	neuer erweiterter Stabilisator	51
5.19	Abschneiden von Pfaden	52
5.20	Würfel mit Stabilisator	52
5.21	Abschneiden von Ästen	53
5.22	Abschneiden von Ästen	54
5.23	Anteil der im Algorithmus betrachteten Würfelfärbungen	55

7.1	Kommutatives Diagramm	84
7.2	Homomorphismen zwischen den Leitern	95
7.3	Bahnen von H (rechte Seite) in der Menge $\mathcal{S} \cap \Lambda_i$	119
7.4	Bijektion zwischen den Urbildmengen von $A_{i+1}aq$ und $E_{i+1}a$	135
7.5	schlichte Graphen auf acht und auf vier Knoten	137
7.6	Vereinigungsgraph \mathcal{C}	138
8.1	Graph aus 10 Knoten und 25 Kanten	158
8.2	Anzahl aller durchlaufenen Nebenklassen	160
8.3	Bahnlängen aller Nebenklassen $A_i p$ unter der Gruppe B	161
8.4	Bedarf an Arbeitsspeicher	161
8.5	Bedarf an Rechenzeit	162
8.6	Bedarf an Rechenzeit	165
8.7	Bedarf an Rechenzeit pro Graph	166
8.8	Bedarf an Arbeitsspeicher	166
8.9	Durchschnittlicher Speicherbedarf pro Nebenklasse	167
8.10	Leiterspiel nach Schmalz, Ergebnisse tabellarisch	167
8.11	Leiterspiel-Light, Ergebnisse tabellarisch	168

Kapitel 1

Für den eiligen Leser

In Kapitel 2 wird der Leser anhand von Beispielen an das Thema dieser Dissertation herangeführt. Dieses Kapitel kann von allen Lesern, die bereits motiviert sind, übersprungen werden.

In Kapitel 3 wird die mathematische und algorithmische Basis dieser Arbeit gelegt. Neben grundlegenden mathematischen Sätzen und Definitionen wird das Homomorphieprinzip und die Ordnungstreue Erzeugung eingeführt. Mit den Algorithmen dieser Arbeit lassen sich Repräsentanten von Doppelnebenklassen berechnen. Der Zusammenhang zwischen diesen Doppelnebenklassen und anderen Isomorphieproblemen, wie zum Beispiel dem Graphenisomorphieproblem, wird in Kapitel 3.5 erläutert. Dieses Kapitel ist absolut notwendig für das Verständnis dieser Arbeit.

In Kapitel 4 wird das Leiterspiel nach Schmalz beschrieben. Da alle Algorithmen dieser Arbeit auf dem ursprünglichen Leiterspiel von Schmalz aufbauen, ist dieses Kapitel sehr hilfreich, aber nicht notwendig für das weitere Verständnis.

In Kapitel 5 wird einer der Algorithmen dieser Arbeit anhand eines einfachen Beispiels vorgeführt. Hier werden dem Leser die Ideen, die hinter den drei neuen Leiterspielalgorithmen dieser Arbeit stehen, näher gebracht. Anhand von verschieden gefärbten Würfeln werden dem Leser die Konzepte und der Ablauf von einem der neuen Algorithmen vorgestellt. Die mathematischen

Konzepte werden dabei anschaulich skizziert, aber nicht genauer ausgeführt.

Kapitel 6 stellt den mathematischen Kern dieser Arbeit dar. Hier werden neue Begriffe wie starke Untergruppenleitern und Konstruktionspfade definiert. Weiterhin werden die Ordnungen auf den Nebenklassenmengen und das im Folgenden verwendete Kanonizitätsprädikat festgelegt. Aufbauend auf diesen Definitionen werden eine Vielzahl von Sätzen bewiesen, die sowohl zur Konstruktion starker Leitern als auch für die später beschriebenen Algorithmen notwendig sind. Die mathematischen Sätze und die Definition der starken Leitern werden in Kapitel 6.5 beschrieben. Diese sind schon allein unter dem gruppentheoretischen Blickwinkel sehr interessant, da diese auch als Beweisgrundlage für weitere Sätze der Gruppentheorie dienen können.

In Kapitel 7 werden drei verschiedene, bisher unbekannte Algorithmen beschrieben. Diese dienen dazu, die Kanonizität einer gegebenen Nebenklasse zu untersuchen und den Stabilisator dieser Nebenklasse zu berechnen. Weiterhin wird in Kapitel 7.1 auch gezeigt, wie diese Algorithmen zur Lösung von weiteren Isomorphieproblemen, wie dem Graphenisomorphieproblem verwendet werden können. Auch das Teilgraphenisomorphieproblem kann mit jedem dieser drei Algorithmen gelöst werden. In Kapitel 7.5 wird beschrieben, wie das Teilgraphenisomorphieproblem auf ein Doppelnebenklassenproblem abgebildet werden kann. Diese drei Algorithmen sind zusammen mit den starken Leitern die zentralen Ergebnisse dieser Arbeit.

In Kapitel 8 wird anhand einer Beispielrechnung die Effizienz der drei Algorithmen aus Kapitel 7 mit der Effizienz des ursprünglichen Leiterspielalgorithmus von Schmalz verglichen. Dabei werden nicht nur die Laufzeit sondern auch der Speicherbedarf der Algorithmen untersucht.

In Kapitel 9 wird angesprochen, welche neuen Fragen diese Arbeit aufgeworfen hat und in welchen Richtungen noch weiterer Forschungsbedarf besteht.

Kapitel 2

Motivation

In diesem Kapitel soll das Thema der vorliegenden Arbeit vorgestellt und das Interesse des Lesers geweckt werden. Daher wird an manchen Stellen auf eine ausführlichere Darstellung der angesprochenen Themen verzichtet.

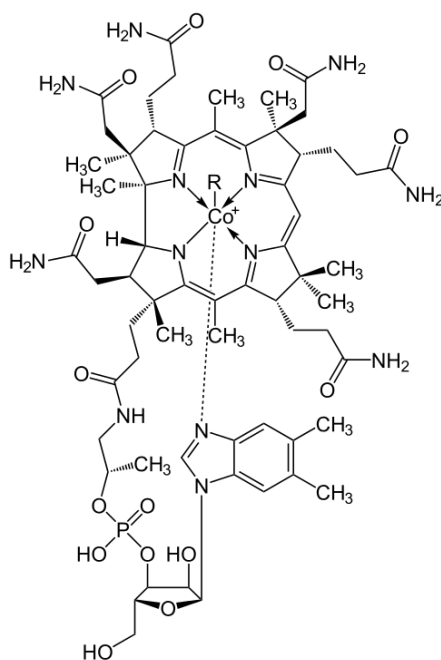


Abbildung 2.1: Vitamin B_{12} Quelle: Wikipedia

Stellen Sie sich vor, Sie sind Chemiker und haben gerade ein sehr interessantes Molekül gefunden. Möglicherweise hat das Molekül überraschende chemische Eigenschaften, lässt sich zur Herstellung von Wasserstoff verwenden oder

es verhindert das Wachstum von Tumoren. Dann möchten Sie natürlich wissen, ob dieses Molekül bereits von anderen Chemikern untersucht wurde, ob weitere Eigenschaften des Moleküls bekannt sind und ob bereits ein Patent auf diesen Stoff angemeldet wurde. Viele Moleküle haben sogenannte Trivialnamen, das sind Bezeichnungen, die von der Struktur des Moleküls unabhängig sind, die sich aber unter Chemikern durchgesetzt haben, wie zum Beispiel der Name Glaubersalz für Natriumsulfat. Wenn das Molekül bisher noch nicht bekannt war, so besitzt es natürlich auch noch keinen Trivialnamen. Daher benötigen Sie für das Molekül eine international anerkannte Bezeichnung, mit Hilfe derer sie in Datenbanken und im Internet nach ihrem Molekül suchen können.

Um eine eindeutige Bezeichnung für ein kleines Molekül zu finden, genügt es, ein paar einfache Regeln anzuwenden, die jeder Chemiker während des Studiums erlernt hat. Für größere und kompliziertere Moleküle existieren international anerkannte Regelwerke, die eine eindeutige Bezeichnung jedes Moleküls ermöglichen. Natürlich gibt es auch Software, die in der Lage ist, zu jedem Molekül die international anerkannte Bezeichnung zu finden. Für den Mathematiker fällt die Aufgabe, einem Molekül einen eindeutigen Namen zu geben in den Bereich der Isomorphieprobleme. Auch die Aufgabe, zu zwei verschiedenen Strukturbeschreibungen festzustellen, ob es sich dabei um das selbe Molekül handelt, ist ein Isomorphieproblem.

In der Biochemie und Medizin werden auch zunehmend Computer eingesetzt, um Moleküle zu finden, die auf Grund ihrer besonderen Eigenschaften zum Beispiel als Medikamente eingesetzt werden können [3, 23]. Häufig wird ein Stoff gesucht, dessen Struktur ganz speziell auf ein Protein abgestimmt ist, so dass sich dieser Stoff an das Protein anheften kann. Der gesuchte Stoff muss an ein oder mehreren Stellen wie ein genaues Gegenstück zu dem entsprechenden Protein passen. Jedes Protein hat mehrere Andockstellen, an die sich der Stoff anheften kann. Zu jeder Andockstelle des Proteins kann das passende Gegenstück berechnet werden. Anschließend kann eine Datenbank daraufhin untersucht werden, ob bereits einen Stoff darin gespeichert wurde, der das zuvor berechnete Gegenstück als Teilstruktur enthält. Die Fragestellung, ob ein gegebener Stoff das gesuchte Gegenstück, das genau zur Andockstelle passt, enthält, ist auch wieder ein Isomorphieproblem.

Isomorphieprobleme tauchen jedoch nicht nur in der Chemie auf, sondern kommen in ganz unterschiedlichen Zusammenhängen vor. Es existieren viele Objekte wie zum Beispiel logische Schaltungen in Field Programmable Gate Arrays oder abstrakte Objekte wie zum Beispiel Graphen, für die es schwierig ist, eine eindeutige Darstellung zu finden. Um festzustellen, ob zwei verschiedene Beschreibungen das selbe Objekt bezeichnen, muss in vielen Fällen ein Isomorphieproblem gelöst werden. Gerade in der Mathematik existieren viele Objekte, bei denen nur unter großem Aufwand festgestellt werden kann, ob zwei verschiedene Beschreibungen das selbe Objekt bezeichnen. Insbesondere bei der Konstruktion von diskreten Objekten, wie Codes, t -Designs, Graphen, Arks und vielen geometrischen Objekten, müssen meist Isomorphieprobleme gelöst werden [10, 9, 8, 19].

Bei der Lösung von Isomorphieproblemen muss vor allen Dingen die Effizienz des verwendeten Algorithmus gewährleistet werden. Denn viele Isomorphieprobleme gehören zur Klasse der NP -vollständigen Probleme, die berühmt und berüchtigt sind, weil sie so schwer zu lösen sind. Schwer zu lösen sind diese Probleme nicht deshalb, weil kein Lösungsweg bekannt wäre, sondern weil der Lösungsweg derart umfangreiche Berechnungen erfordert, dass viele Probleme auch auf einem Rechencluster nicht in akzeptabler Zeit durchgeführt werden können. Daher gibt es viele mathematische Fragestellungen, die allein deshalb nicht gelöst sind, weil ein Computer zur Lösung des Problems Wochen, Monate oder sogar Jahre rechnen müsste.

Die meisten professionellen Algorithmen, mit denen sich Isomorphieprobleme lösen lassen, zerlegen ein gegebenes Problem in immer kleinere Teilprobleme, die einzeln leichter zu lösen sind. Die Lösung jedes dieser Teilprobleme kann dann zur Lösung des nächst größeren Problems verwendet werden. Laue [12] war der erste, der diese Strategie auf der Basis von Homomorphismen von Gruppenoperationen formulierte. Sein Homomorphieprinzip kann vermutlich als die Grundlage aller erfolgreicher Algorithmen zur Lösung von Isomorphieproblemen angesehen werden. Denn schon jede Invariante, die zur Klassifikation von Strukturen verwendet wird, kann als ein Homomorphismus von Grup-

penoperationen von der Menge der Strukturen in die Menge der Invarianten betrachtet werden. Und jeder Homomorphismus von Gruppenoperationen, bei dem die Bahnen in der Bildmenge alle einelementig sind, kann als Abbildung einer Objektmenge auf eine Menge von Invarianten betrachtet werden. Dieses Homomorphieprinzip wird in ?? beschrieben.

Ein erstaunliches Beispiel für die Effizienz des Homomorphieprinzips ist zum Beispiel der von Meringer [14] und Grüner [5] entwickelte Graphengenerator *Gradpart*. In einem Graphen wird die Anzahl der Kanten, die an einem Knoten zusammentreffen, als Knotengrad dieses Knotens bezeichnet. Sind die Knotengrade aller Knoten vorgegeben, so können mit dem Programm *Gradpart* alle Graphen konstruiert werden, deren Knoten die entsprechenden Knotengrade besitzen. Genauer gesagt werden die Graphen implizit durch eine Art Konstruktionsanleitung beschrieben, so dass nicht jeder Graph einzeln konstruiert werden muss. Der Graphengenerator *Gradpart* ist so effizient, dass dieser zum Auffinden und Konstruieren eines Graphen im Schnitt weniger als einen CPU-Takt benötigt.

Eines der bekanntesten Programme zur Lösung von Isomorphieproblemen ist das von McKay geschriebene Programm *nauty* [13]. *Nauty* ist gleichzeitig auch eines der schnellsten Programme zur Lösung von Isomorphieproblemen in Bezug auf Graphen. Leider ist *nauty* ausschließlich zur Lösung des Graphenisomorphieproblems anwendbar. Daher sehen sich viele Leute gezwungen, die Objekte, für die sie gerne einen Isomorphietest durchführen würden, in Graphen umzuwandeln. Um die Objekte auf Isomorphie testen zu können, muss daher zu jedem einzelnen Objekttyp eine Strategie entworfen werden, wie die zu untersuchenden Objekte so auf Graphen abgebildet werden können, dass die Struktur der Objekte erhalten bleibt [16].

Ein auf Doppelnebenklassen basierender Isomorphietest ist hingegen viel universeller einsetzbar, da das seit langem bekannte Split-Lemma, das in Kapitel 3.5 beschrieben ist, eine sehr vielseitige Methode bereitstellt, wie ein Isomorphieproblem in ein Doppelnebenklassen-Isomorphieproblem überführt werden kann. Mit Hilfe des Split-Lemmas können beispielsweise auch alle Gra-

phenisomorphieprobleme auf Doppelnebenklassen-Isomorphieprobleme abgebildet werden. In Satz 6.5.2 wird die Konstruktion einer starken Untergruppenleiter beschrieben, die bei allen Algorithmen dieser Arbeit benötigt wird. Diese starke Untergruppenleiter ermöglicht es, zu den meisten Isomorphieproblemen auf Doppelnebenklassen einen Isomorphietest durchzuführen.

Weiterhin können Doppelnebenklassen zur Lösung vieler gruppentheoretischer Probleme eingesetzt werden, die mit den Algorithmen aus der vorliegenden Arbeit auf einheitliche Weise gelöst werden können. Ein Beispiel für ein schwieriges Problem, das mit Hilfe von Doppelnebenklassen gelöst werden kann, ist die Berechnung der Schnittmenge zweier Untergruppen einer Gruppe.

McKay, der bereits zuvor angesprochene Urheber des Programmes *nauty*, gibt in [13] einen Überblick über die bekannten Ansätze zur Lösung von Isomorphieproblemen. Dabei wirft er die Frage nach den Zusammenhängen zwischen den verschiedenen Algorithmen zur Lösung von Isomorphieproblemen auf und grenzt verschiedene Techniken voneinander ab, indem er auf die Unterschiede im Speicherbedarf und die Möglichkeit zur Parallelisierung eingeht. In der vorliegenden Arbeit wird einer dieser Ansätze, der Leiterspielalgorithmus von Schmalz [18], mit der ordnungstreuen Erzeugung von Read [15] und Faradžev [4] verbunden. Dies ermöglicht es, den Speicherbedarf des Leiterspielalgorithmus auf ein Minimum zu reduzieren und bahnt damit den Weg für einen parallelisierbaren Leiterspielalgorithmus. Damit geht die vorliegende Arbeit einen Schritt weiter in Richtung einer vereinheitlichten, auf dem Homomorphieprinzip basierenden Beschreibung aller Techniken zur Lösung von Isomorphieproblemen. Als nächsten Schritt in Richtung einer vereinheitlichten Beschreibung und in Richtung effizienterer Algorithmen sehe ich eine dynamischere Verwendung der Untergruppenleitern an. Statt eine fest vorgegebene Leiter zu verwenden, könnte der nächste „Leiterschritt“ von der Struktur der verwendeten Untergruppen abhängig gemacht werden und die Schrittgröße dem Problem angepasst werden. Um die Verwendung von dynamischen Leitern möglich zu machen, besteht aber noch weiterer Forschungsbedarf.

Kapitel 5

Der Kanonisierungsalgorithmus am Beispiel eines Würfels

In diesem Kapitel sollen der Ablauf und die Ideen hinter dem in den folgenden Kapiteln beschriebenen Kanonizitätstests vorgestellt werden. Der Schwerpunkt dieses Kapitels liegt auf einer intuitiven und anschaulichen Beschreibung des Kanonizitätstests. Daher wurde an vielen Stellen auf eine exakte Beschreibung der Vorgehensweise verzichtet. Eine genaue und mathematisch fundierte Darstellung des Algorithmus folgt im Kapitel 7.

Der Zusammenhang zwischen den Würfelfärbungen, die in diesem Kapitel zur Darstellung des Kanonizitätstests verwendet werden und den Nebenklassen, die in Kapitel 6 und 7 zur Beschreibung der Algorithmen verwendet werden, ist in Kapitel 7.1 erläutert.

5.1 Drehungen eines Würfels

Es gibt genau 24 Möglichkeiten einen Würfel im Raum zu drehen, so dass jede Ecke nach der Drehung wieder an einer Position zum Liegen kommt, an der sich auch vor der Drehung eine Ecke befunden hat. Die 24 Positionen, in denen sich ein Würfel nach einer Drehung befinden kann, werden in Abbildung 5.1 veranschaulicht. Die „identische Drehung“, bei der der Würfel unverändert bleibt, wurde in die Menge der Drehungen mit aufgenommen. Denn das vorliegende Beispiel soll die mathematische Beschreibung des Algorithmus vorbereiten, in der die Menge der Drehungen als Gruppe betrachtet wird. Dabei nimmt die

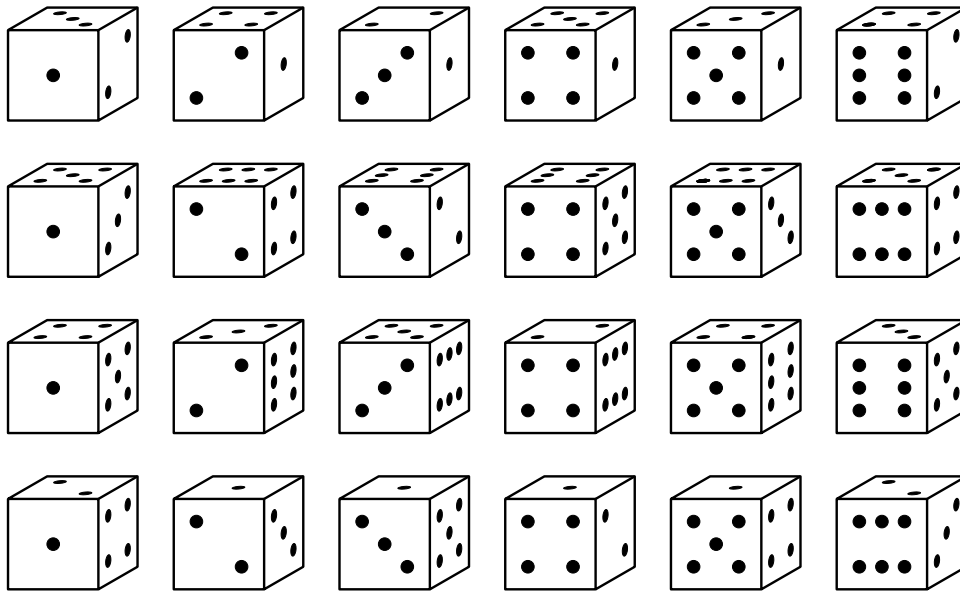


Abbildung 5.1: Die 24 Drehungen eines Würfels

identische Drehung den Platz des neutralen Elementes der Gruppe ein.

Die in Abbildung 5.1 dargestellten Würfelaugen dienten nur der Veranschaulichung, wenn im weiteren Text von Würfeln die Rede ist, so sind immer Würfel ohne Augen gemeint. Mit Drehungen sind im Folgenden immer Drehungen des Würfels im dreidimensionalen Raum gemeint, bei denen Ecken wieder auf Ecken abgebildet werden.

5.2 Fragestellung

In Abbildung 5.2 ist ein Würfel dargestellt, bei dem vier der acht Ecken mit weißen Kreisen markiert wurden. Um die Positionen der markierten Ecken angeben zu können, wurde jede Eckposition mit einer Zahl markiert. Diese Positionsangaben bleiben bei den Drehungen unverändert, das heißt, vor wie nach jeder Drehung wird die vordere, obere, linke Ecke immer als Ecke Nummer eins bezeichnet.

Werden verschiedene Drehungen auf den Würfel in Abbildung 5.2 angewendet, so stellt sich die Frage, bei welchen Drehungen die weißen Ecken wieder

an einer Position landen, an der sich bereits vor der Drehung eine weiße Ecke befunden hat.

Weiterhin stellt sich die Frage, wie viele Möglichkeiten es gibt, vier Ecken eines Würfels weiß zu markieren, wenn nur diejenigen Würfelfärbungen als verschieden betrachtet werden, die durch keine Drehung aufeinander abgebildet werden können.

Zu einer gegebenen Würfelfärbung kann die Menge aller Würfelfärbungen berechnet werden, die durch Drehungen aufeinander abgebildet werden. Diese Menge wird auch die Bahn dieser Würfelfärbung unter der Operation der Gruppe aller Würfeldrehungen genannt.

Wenn aus einer beliebigen Bahn eine Würfelfärbung ausgewählt wurde, die repräsentativ für die ganze Bahn stehen soll, so wird diese ausgewählte Färbung auch kanonischer Repräsentant genannt. In Kapitel 5.3 wird eine Methode vorgestellt, mit der festgelegt werden kann, welche Färbung aus jeder Bahn als kanonisch bezeichnet werden soll.

Wurden bei der Auswahl der kanonischen Repräsentanten gewisse Voraussetzungen erfüllt, so ist der beschriebene Algorithmus dazu geeignet, folgende Fragestellungen zu beantworten:

1. Bei welchen Drehungen landet jede der weiß gefärbten Ecken des Würfels in Abbildung 5.2 wieder auf einer Ecke, die bereits vor der Drehung weiß gefärbt war? (Berechnung des Stabilisators)

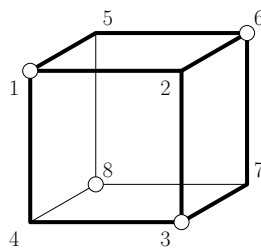


Abbildung 5.2: Würfel mit weiß markierten Ecken

2. Ist der Würfel in Abbildung 5.2 der kanonische Repräsentant seiner Bahn? (Kanonizitätstest)

3. Welche Bahnen von Würfelfärbungen gibt es, bei denen vier Ecken des Würfel weiß gefärbt wurden? (vollständige Konstruktion bis auf Isomorphie)

Im folgenden Beispiel soll vorgeführt werden, wie sich die Fragen mit der Nummer eins und zwei mit Hilfe des beschriebenen Algorithmus beantworten lassen.

5.3 Kanonische Färbung und Totalordnung

Im Folgenden wird eine Totalordnung auf der Menge der Würfelfärbungen festgelegt. Dies ermöglicht es, Würfelfärbungen anhand dieser Ordnungsrelation miteinander zu vergleichen. Um die Beschreibungen nicht unnötig kompliziert zu machen, wird die Sprechweise „Färbung \mathcal{A} ist größer/kleiner/gleich Färbung \mathcal{B} “ verwendet, wenn zwei Färbungen anhand dieser Ordnungsrelation miteinander verglichen werden.

Die in Abbildungen 5.2 und 5.3 angegebenen Zahlen weisen jeder Position, an der sich eine Ecke des Würfel befindet, eine Zahl zu. Diese Positionsangaben ermöglichen es, eine Würfelfärbung durch Angabe der markierten Würfecken zu charakterisieren. Um eine eindeutige Beschreibung jeder Würfelfärbung zu ermöglichen, wird eine einheitliche Beschreibung durch sogenannte Färbungstupel eingeführt.

Im vorliegende Beispiel werden nur schwarze, weiße und ungefärbte Ecken verwendet. Die Farbe Schwarz wird mit dem Buchstaben s und die Farbe weiß mit dem Buchstaben w abgekürzt. Im Färbungstupel sollen zuerst alle Ecken, die mit weißer Farbe markiert wurden und erst anschließend die Ecken, die mit schwarzer Farbe markiert wurden, aufgezählt werden. Um die Farbe dieser Ecke zu verdeutlichen, soll jeweils die Position der Ecke gefolgt vom Buchstaben w , beziehungsweise s geschrieben werden. Dabei sollen die Positionen aller gleichfarbig markierten Ecken immer der Reihenfolge nach aufgezählt werden.

In Abbildung 5.2 sind die Ecken an den Positionen 1, 3, 6 und 8 weiß gefärbt, daher entspricht die Färbung dem Tupel $(1, w, 3, w, 6, w, 8, w)$.

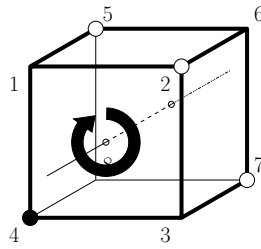


Abbildung 5.3: Würfel mit dem Färbungstupel $(2, w, 5, w, 7, w, 4, s)$

Bei dem Würfel in Abbildung 5.3 hingegen sind die Ecken an den Positionen 2, 5 und 7 weiß und die Ecke an Position 4 schwarz gefärbt, daher entspricht die Färbung dem Tupel $(2, w, 5, w, 7, w, 4, s)$.

Anhand dieses Färbungstupels kann jetzt eine Totalordnung auf der Menge der Würfel­färbungen angegeben werden. Für alle untersuchten Würfel­färbungen wird die Ordnung so festgelegt, dass sie der lexikographischen Ordnung der jeweiligen Färbungstupel entspricht.

Aus jeder Bahn soll genau diejenige Färbung als kanonisch gelten, deren Färbungstupel kleiner ist als das aller anderen Färbungen derselben Bahn.

5.4 Stabilisatoren

Zu jedem beliebig markierten Würfel bildet die Menge aller Drehungen, bei denen das Färbungstupel vor der Drehung das selbe ist wie nach der Drehung, eine Gruppe. Diese Gruppe wird der Stabilisator dieses markierten Würfels genannt.

In diesem Beispiel wird unter anderem gezeigt, wie der Stabilisator der Würfel­färbung aus Abbildung 5.2 berechnet werden kann.

5.5 Homomorphismen

In diesem Beispiel werden zwei verschiedene Abbildungen verwendet. Die erste Abbildung hat eine Definitionsmenge, die aus allen Würfeln besteht, bei denen zwischen einer und acht Ecken farblich markiert sind, davon genau eine

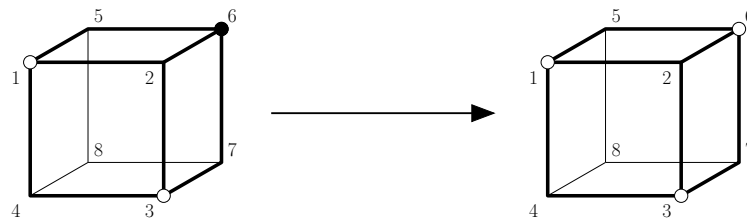


Abbildung 5.4: Beispiel einer Abbildung unter dem Fusing-Homomorphismus

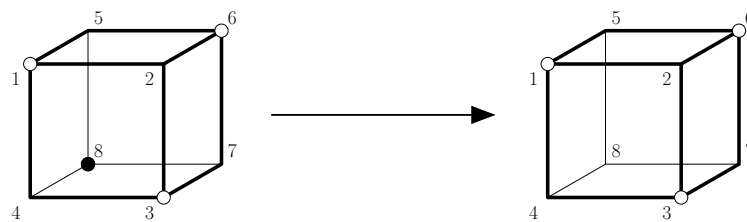


Abbildung 5.5: Beispiel einer Abbildung unter dem Splitting-Homomorphismus

schwarz und alle anderen weiß. Jede dieser Würfelfärbungen wird auf einen Würfel abgebildet, der an denselben Ecken markiert wurde wie der Würfel im Urbild, dessen Markierungen aber alle weiß sind. Diese Abbildung wird im Folgenden auch Fusing-Homomorphismus genannt.

Die zweite Abbildung hat dieselbe Definitionsmenge wie die erste und zwar die Menge aller Würfel, bei denen zwischen einer und acht Ecken farblich markiert sind, wobei genau eine Ecke schwarz und alle anderen weiß markiert sind. Jeder Würfel aus dieser Definitionsmenge wird durch die zweite Abbildung auf einen Würfel abgebildet, bei dem genau diejenigen Ecken, die im Urbild weiß gefärbt waren, auch wieder weiß sind, dessen übrige Ecken aber alle unmarkiert bleiben. Diese Abbildung wird im Folgenden auch Splitting-Homomorphismus genannt.

5.6 Konstruktionspfade

Ein Konstruktionspfad ist eine Folge von Würfelfärbungen, die beim völlig unmarkierten Würfel beginnt und die folgenden Bedingungen erfüllt: Für je zwei

aufeinander folgende Würfelfärbungen des Tupels, von denen die Erste eine schwarz markierte Ecke enthält, muss gelten, dass die Zweite gleich dem Bild der Ersten unter dem Fusing-Homomorphismus ist. Für je zwei aufeinander folgende Würfelfärbungen des Tupels, von denen die Erste keine schwarz markierte Ecke enthält, muss gelten, dass die Erste gleich dem Bild der Zweiten unter dem Splitting-Homomorphismus ist. In Abbildung 5.6 ist ein Beispiel für einen Konstruktionspfad zu sehen.

Unter Verwendung der Totalordnung auf der Menge der Würfelfärbungen kann jetzt eine Totalordnung auf der Menge der Konstruktionspfade definiert werden. Im Folgenden soll genau derjenige von zwei gegebenen Konstruktionspfaden als kleiner gelten, dessen Tupel lexikographisch kleiner ist als das des anderen Pfades.

Konstruktionspfade werden im folgenden Text auch kurz als Pfade bezeichnet.

5.7 Vorbereitungen

Als Vorbereitung für den Algorithmus ist es erforderlich, den kleinsten Pfad zu dem in Abbildung 5.2 gegebenen Würfel zu finden. Dieser kleinste Pfad ist in Abbildung 5.6 dargestellt und wird im Folgenden mit ρ bezeichnet. Dieser kleinste Pfad kann leicht mit Hilfe von dem in Abschnitt 7.6.6 beschriebenen Algorithmus gefunden werden, daher soll an dieser Stelle nicht weiter darauf eingegangen werden.

Als weiterer Vorbereitungsschritt muss für den Pfad ρ geprüft werden, ob jede Komponente außer der letzten eine kanonische Würfelfärbung ist. Ist einer der Würfel des Pfades nicht kanonisch, so sind auch die darauf folgenden Würfel des Pfades ρ nicht kanonisch. Dies kann in ähnlicher Weise zu der Beweisführung von Lemma 6 auf Seite 66 nachgewiesen werden, auf den Nachweis soll jedoch an dieser Stelle verzichtet werden.

Wenn sich an dieser Stelle bereits herausstellt, dass eine der Komponenten des Pfades ρ nicht kanonisch ist, so braucht der in diesem Beispiel betrachtete Algorithmus nicht durchgeführt werden, da ja das gesuchte Ergebnis bereits

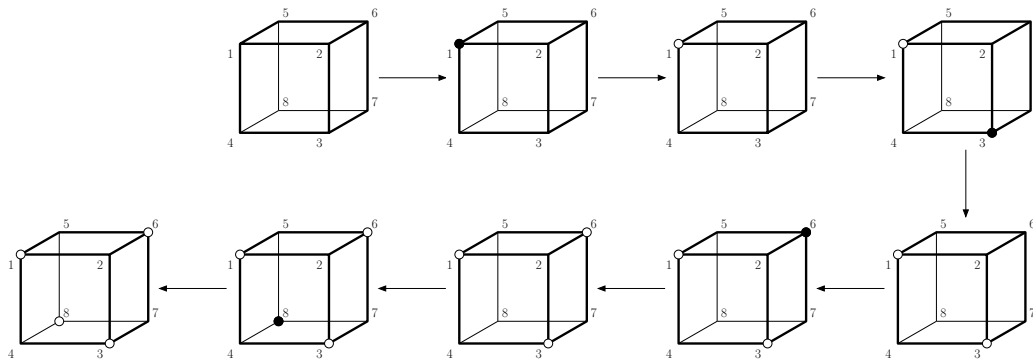


Abbildung 5.6: der kleinste Pfad zum Würfel in Abbildung 5.2

fest steht. Die Überprüfung, ob jede außer der letzten Komponente des Pfades ρ kanonisch ist, kann durch einen rekursiven Aufruf des hier beschriebenen Algorithmus erfolgen. Daher soll auch auf diesen Schritt nicht weiter eingegangen werden. Die Überprüfung der letzten Komponente des Pfades ρ wird anschließend ausführlich durchgeführt.

In Zusammenhang mit der Überprüfung, ob die Komponenten des Pfades ρ kanonisch sind, können die zugehörigen Stabilisatoren aller dieser Komponenten berechnet werden. Die Stabilisatoren aller außer der letzten Würfelfärbung des Pfades ρ werden im weiteren Verlauf des Algorithmus noch benötigt. Die Berechnung des Stabilisators der letzten Komponente des Pfades ρ wird im Folgenden ausführlich durchgeführt.

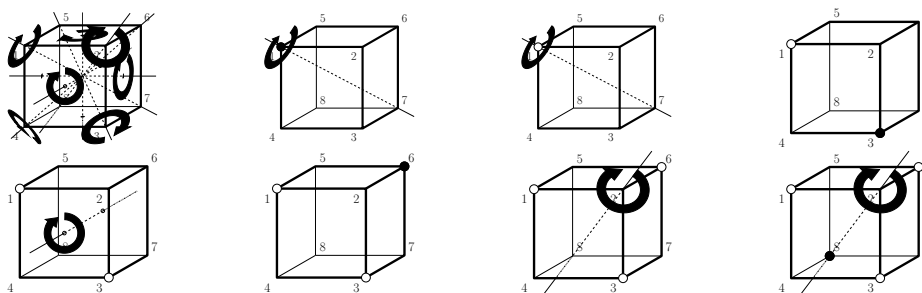


Abbildung 5.7: Würfel des Pfades ρ und deren Stabilisatoren

5.8 Erweiterung des Stabilisators

Der Splitting-Homomorphismus besitzt eine sehr interessante Eigenschaft, wie sich anhand der Stabilisatoren zweier Würfel im Bild und im Urbild des Homomorphismus zeigen läßt. Der Würfel im Urbild und der Würfel im Bild des Splitting-Homomorphismus unterscheiden sich nur in der Färbung einer einzigen Ecke. Diese Ecke ist im Urbild schwarz und im Bild ungefärbt.

Jede Drehung, die im Stabilisator des Urbildes liegt, muss auch im Stabilisator des Bildes liegen. Denn jede Drehung, die das Urbild auf sich selbst abbildet, muss immer auch alle weißen Ecken aufeinander abbilden.

Der Fusing-Homomorphismus verhält sich genauso in Bezug auf die Stabilisatoren in Bild und Urbild. Eine Würfelfärbung im Definitionsbereich und die entsprechende Würfelfärbung im Bild des Fusing-Homomorphismus unterscheiden sich auch wieder nur in einer einzigen Ecke. Die Ecke, die beim Würfel im Urbild schwarz gefärbt ist, ist bei der Würfelfärbung im Bild weiß. Jede Drehung, die die schwarz und die weiß gefärbten Ecken des Würfels im Urbild auf sich selbst abbildet, muss auch die weiß gefärbten Ecken des Würfels im Bild aufeinander abbilden. Daher muss jede Drehung aus dem Stabilisator einer gegebenen Würfelfärbung auch im Stabilisator der entsprechenden Würfelfärbung im Bild enthalten sein.

Diese Erkenntnis kann dazu benutzt werden, eine Untergruppe des Stabilisators der Würfelfärbung aus Abbildung 5.2 zu bestimmen. Die Abbildung zwischen dem vorletzten Würfel des Pfades ρ und dem letzten Würfel des Pfades ist ein Fusing-Homomorphismus. Daher muss jede Drehung aus dem Stabilisator der vorletzten Würfelfärbung des Pfades ρ auch im gesuchten Stabilisator der letzten Würfelfärbung enthalten sein.

Die Drehungen, die in Abbildung 5.8 skizziert sind, werden aus dem Stabilisator der vorletzten Würfelfärbung in den Stabilisator der letzten Würfelfärbung des Pfades ρ übernommen.

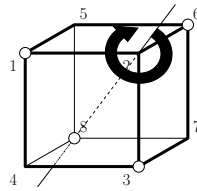


Abbildung 5.8: eine Untergruppe des Stabilisators

5.9 Menge der Pfade

In Abbildung 5.9 sind alle Pfade dargestellt, die vom Würfel ohne gefärbte Ecken zu dem zu untersuchenden Würfel aus Abbildung 5.2 führen. Diese Pfade sollen im weiteren Verlauf des Algorithmus in Tiefensuche durchlaufen werden.

Nur ein Bruchteil dieser Pfade wird jedoch dazu beitragen, neue Stabilisatoren des Würfels in Abbildung 5.2 zu finden. Um zu überprüfen, ob der Würfel in Abbildung 5.2 kanonisch ist, wird auch nur eine kleine Teilmenge der dargestellten Pfade benötigt. Um zu vermeiden, dass alle diese Pfade trotzdem betrachtet werden müssen, werden Kriterien benötigt, die es ermöglichen, Pfade von der Untersuchung auszuschließen, die das Ergebnis nicht beeinflussen.

5.10 Schnittmengen von Stabilisatoren

Beginnend mit dem Pfad, der nur aus einem ungefärbten Würfel besteht, sollen die Pfade in Tiefensuche durchlaufen werden, indem der bestehende Pfad schrittweise um weitere Würfelfärbungen verlängert wird. Zu jeder Würfelfärbung, die an einen bestehenden Pfad angehängt werden soll, wird weiterhin eine besondere Untergruppe des Stabilisators dieses Würfels berechnet. Diese Untergruppe ermöglicht es, Pfadmengen von der Untersuchung auszuschließen und so die zu untersuchende Menge von Pfaden zu verkleinern.

Die gesuchte Gruppe ist eine Untergruppe des Stabilisators der letzten Würfelfärbung des betrachteten Pfades. Diese besteht genau aus der Menge aller Drehungen, die zusätzlich auch im aktuell bekannten Stabilisator des Würfels in Abbildung 5.2 enthalten sind.

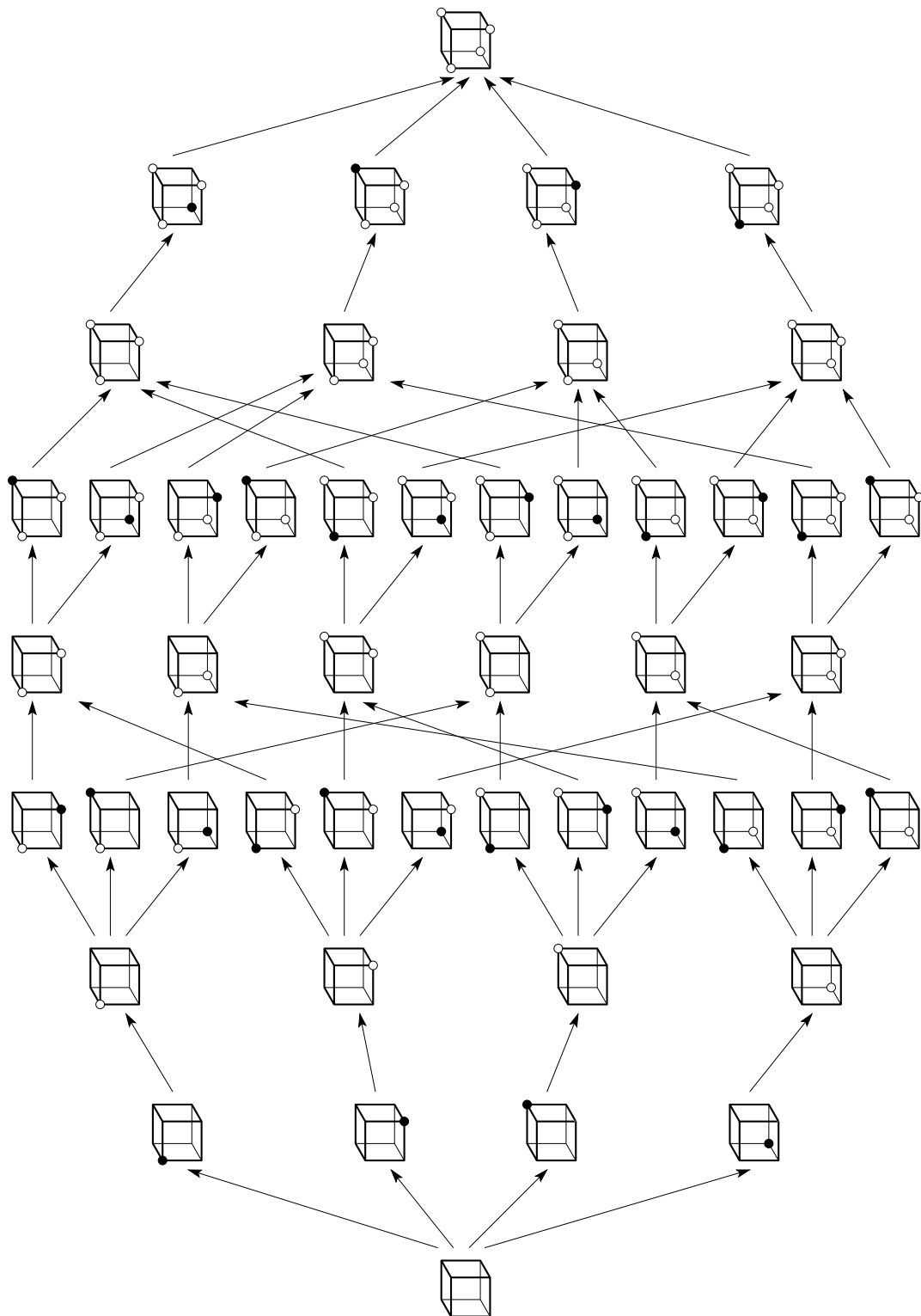


Abbildung 5.9: alle Pfade zum Würfel in Abbildung 5.2

Diese Menge ist identisch mit dem Stabilisator des betrachteten Würfels in der Gruppe aller Drehungen des bisher bekannten Stabilisators des Würfels in Abbildung 5.2. Daher kann zur Berechnung dieser Untergruppe ein rekursiver Aufruf des hier in diesem Beispiel vorgestellten Algorithmus verwendet werden. Im Folgenden soll zu Gunsten der Übersicht nicht mehr weiter auf die Berechnung dieser Untergruppen eingegangen werden.

5.11 Fusionierende Drehung

Um das Abschneiden von Pfaden zu ermöglichen und den gesuchten Stabilisator zu berechnen, muss zu jeder Würfelfärbung, die an einen bestehenden Pfad angehängt werden soll, eine sogenannte fusionierende Drehung bestimmt werden. Als fusionierende Drehungen werden diejenigen Drehungen bezeichnet, die eine gegebene Würfelfärbung auf den kanonischen Repräsentanten dieser Bahn abbilden.

Bei der Bestimmung einer fusionierenden Drehung können zwei Fälle unterschieden werden. Enthält die Würfelfärbung, um die ein gegebener Pfad verlängert werden soll, keine schwarze Ecke, so braucht die fusionierende Drehung nicht explizit berechnet werden. Denn die fusionierende Drehung der letzten Würfelfärbung des noch nicht verlängerten Pfades ist bereits eine der gesuchten fusionierenden Drehungen zu der Würfelfärbung, die an den Pfad angehängt werden soll. Dies ergibt sich als direkte Folge aus den Eigenschaften des Fusing-Homomorphismus.

Andernfalls soll anhand eines Beispiels erläutert werden, wie eine fusionierende Drehung gefunden werden kann:

Der in Abbildung 5.10 gegebene Pfad soll um den in Abbildung 5.11 dargestellten Würfel verlängert werden.

Die fusionierende Abbildung bildet die untersuchte Würfelfärbung immer auf eine Würfelfärbung ab, die dieselbe Anzahl von schwarz und weiß gefärbten Ecken besitzt. Denn jede Drehung eines Würfels, dessen Ecken farblich

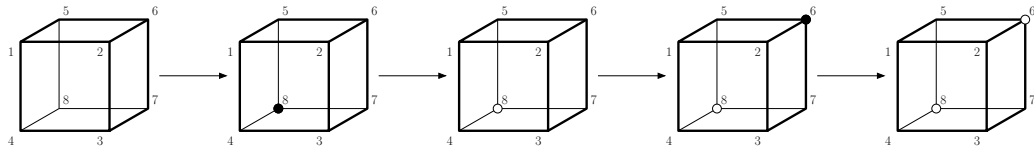


Abbildung 5.10: Beispiel eines Pfades

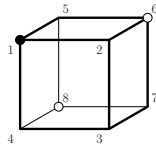


Abbildung 5.11: Dieser Würfel soll an den Pfad angehängt werden

markiert wurden, lässt die Anzahl der schwarz und weiß markierten Ecken unverändert.

Zu jeder Würfelfärbung des untersuchten Pfades existiert genau eine Würfelfärbung des Pfades ρ , die dieselbe Anzahl von schwarz und weiß gefärbten Ecken besitzt. Es kann davon ausgegangen werden, dass jede Würfelfärbung des noch nicht verlängerten Pfades in der Bahn einer Würfelfärbung des Pfades ρ liegt. Die Pfade, bei denen dies nicht der Fall ist, werden gesondert betrachtet und die Bestimmung der fusionierenden Abbildung ist in diesen Fällen nicht notwendig.

Da jede, außer der letzten Würfelfärbung des Pfades ρ nach Voraussetzung kanonisch sein muss, kann davon ausgegangen werden, dass eine Drehung, die die untersuchte Würfelfärbung auf eine Würfelfärbung des Pfades ρ abbildet, eine der gesuchten fusionierenden Abbildungen ist.

Unter Anwendung der fusionierenden Drehung der letzten Würfelfärbung des noch nicht verlängerten Pfades kann diese letzte Würfelfärbung auf den

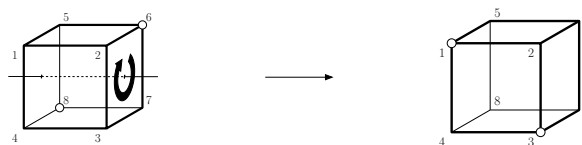


Abbildung 5.12: Würfel mit fusionierender Drehung

entsprechend markierten Würfel des Pfades ρ abgebildet werden. Diese fusionierende Drehung kann als bekannt vorausgesetzt werden und ist im Falle des Pfades aus Abbildung 5.10 eine Drehung um 180 Grad um die in Abbildung 5.12 dargestellte Achse. Nach Voraussetzung wurde der Stabilisator dieser Würfelfärbung des Pfades ρ bereits bestimmt. Der Stabilisator dieses Würfels aus dem Pfad ρ ist in Abbildung 5.14 skizziert.

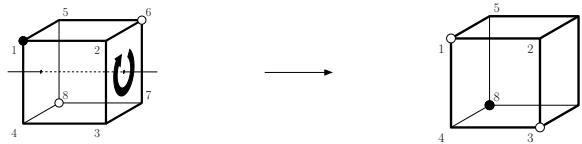


Abbildung 5.13: wende fusionierende Drehung an

Anschließend soll diese Drehung auf den Würfel in Abbildung 5.11 angewendet werden, wie dies in Abbildung 5.13 dargestellt ist. Jede der Drehung des Stabilisators aus Abbildung 5.14 soll anschließend auf den in Abbildung 5.13 auf der rechten Seite dargestellten Würfel angewendet werden. Dies führt zu der Menge von Würfelfärbungen, die in Abbildung 5.15 dargestellt ist.

Aus dieser Menge von Würfelfärbungen wird die, mit dem kleinsten Färbungstupel ausgewählt. Diese Würfelfärbung ist bereits die gesuchte kanonische Würfelfärbung aus der Bahn der Färbung aus Abbildung 5.11. Jede Drehung, die den ursprünglichen Würfel aus Abbildung 5.11 auf diesen ausgewählten Würfel abbildet, ist eine der gesuchten fusionierenden Drehungen. Eine dieser fusionierenden Drehungen ist in Abbildung 5.16 skizziert.

Wird, wie in Abbildung 5.16, die fusionierende Drehung eines Würfels auf diesen Würfel selbst angewandt und ist die Würfelfärbung, die sich daraus ergibt, kleiner als die Würfelfärbung des entsprechenden Würfels des Pfades ρ ,

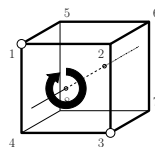


Abbildung 5.14: Stabilisator des Bahnrepräsentanten

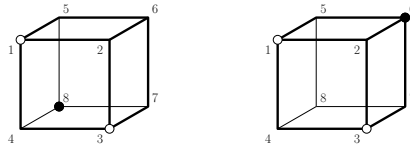


Abbildung 5.15: Bahn des Stabilisators

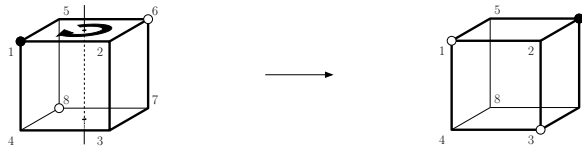


Abbildung 5.16: Würfel mit fusionierender Drehung

so ist der Würfel am Ende des Pfades ρ nicht kanonisch.

Dies kann auf ähnliche Weise wie in Lemma 6 bewiesen werden, darauf soll jedoch an dieser Stelle verzichtet werden. Ist das Färbungstupel des Würfels, der sich aus der Anwendung der fusionierenden Drehung eines Würfels auf diesen Würfel selbst ergibt, größer als das Färbungstupel des entsprechenden Würfels des Pfades ρ , so braucht keiner der Pfade, die diesen Würfel enthalten, untersucht werden. Denn diese Pfade sind weder für die Überprüfung der Kanonizität noch für die Berechnung des Stabilisators des Würfels in Abbildung 5.2 von Bedeutung.

5.12 Tiefensuche

Die in Abbildung 5.9 dargestellten Pfade werden jetzt in Tiefensuche durchlaufen. Beginnend mit dem Pfad ρ , der nicht weiter zur Berechnung des Stabilisators und der Überprüfung der Kanonizität beiträgt, wird der nächstmögliche Pfad untersucht. Dies ist der Pfad, der als rechte Würfelfolge in Abbildung 5.17 dargestellt ist. Die linke Würfelfolge in dieser Abbildung stellt den Pfad ρ dar.

Bei dem in Abbildung 5.17 dargestellten Pfad bildet die fusionierende Drehung des vorletzten Würfels des rechten Pfades diesen auf den vorletzten Würfel des rechten Pfades ab. Da der Fusing-Homomorphismus den jeweils vorletzten Würfel beider Pfade auf den gemeinsamen letzten Würfel abbildet, liegt

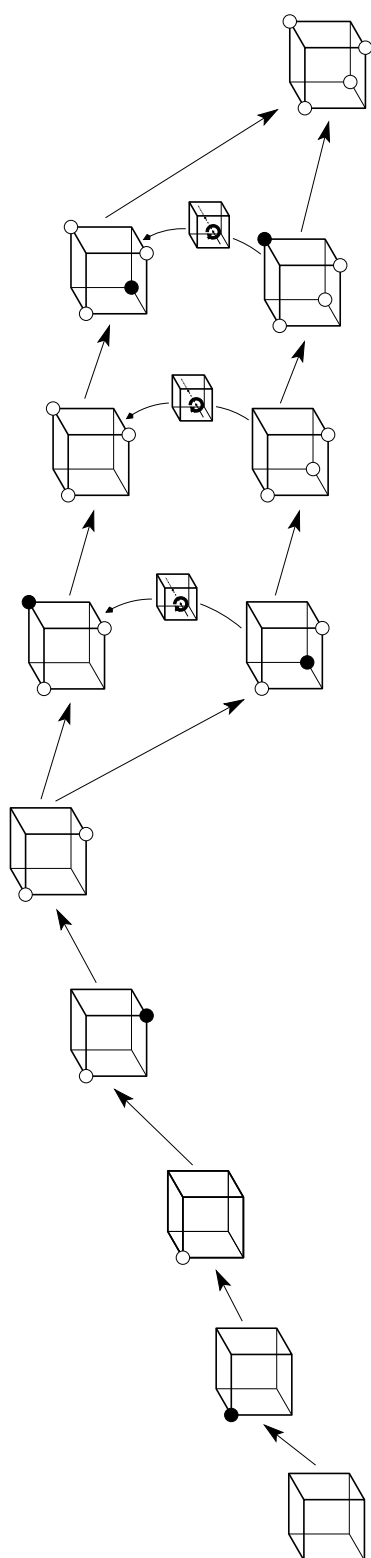


Abbildung 5.17: Pfad mit Darstellung der fusionierenden Drehungen

die fusionierende Drehung im gesuchten Stabilisator des letzten Würfels des Pfades ρ .

Der bisher bekannte Stabilisator des letzten Würfels des Pfades kann daher so erweitert werden, dass dieser zusätzlich alle Drehungen enthält, die sich als Kombinationen aus Drehungen des bisherigen Stabilisators und der fusionierenden Drehung des vorletzten Würfels des Pfades ergeben. Wie diese Gruppe am geschicktesten berechnet werden kann, soll an dieser Stelle jedoch nicht erklärt werden. Die Beschreibung einer geeigneten Methode ist in Abschnitt 7.6.4 zu finden.

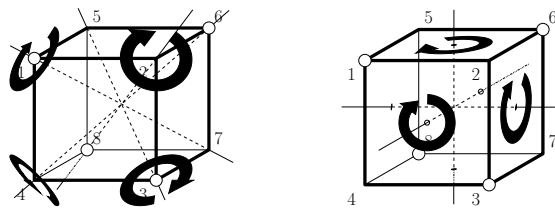


Abbildung 5.18: neuer erweiterter Stabilisator

Der neue, um die fusionierende Drehung erweiterte Stabilisator des untersuchten Würfels, ist anhand der beiden Würfel in 5.18 dargestellt. Die Drehungen um die vier in 5.18 auf der linken Seite dargestellten Achsen sind Drehungen um 120 Grad, während die Drehungen um die auf der rechten Seite dargestellten Achsen Drehungen um 180 Grad sind. Der neue Stabilisator besteht aus insgesamt 12 Drehungen und ist damit nach wie vor kleiner als der Stabilisator des unmarkierten Würfels, der aus 24 Drehungen besteht.

5.13 Abschneiden von Ästen

Der nächste Pfad, der bei der Tiefensuche betrachtet wird, enthält den in Abbildung 5.19 auf der rechten Seite unten dargestellten Würfel. Die fusionierende Drehung ist in Form eines kleinen Würfels angedeutet. Bei dieser fusionierenden Drehung ist bemerkenswert, dass diese sowohl im neuen, erweiterten Stabilisator des letzten Würfels des Pfades ρ enthalten ist als auch in dem in

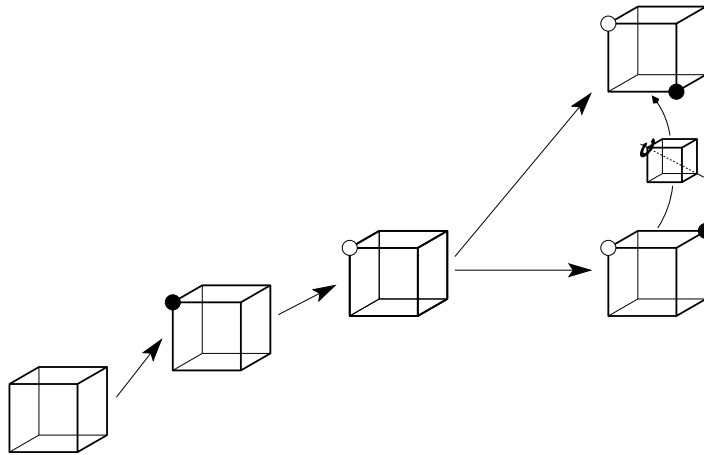


Abbildung 5.19: Abschneiden von Pfaden

Abbildung 5.20 dargestellten Stabilisator des vorangehenden Würfels des Pfades. Das heißt, die zum vorangehenden Würfel berechnete Schnittmenge von Stabilisatoren, die in Kapitel 5.10 beschrieben wurde, enthält die fusionierende Drehung des darauf folgenden Würfels des Pfades.

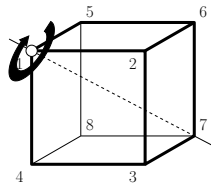


Abbildung 5.20: Würfel mit Stabilisator

In solchen Fällen kann die weitere Untersuchung aller Pfade, die den in Abbildung 5.19 auf der rechten Seite unten dargestellten Würfel enthalten, abgebrochen werden. Keiner dieser Pfade kann zur Erweiterung des Stabilisators oder zur Überprüfung der Kanonizität beitragen. Denn jeder Pfad, der durch diesen Würfel verläuft, besteht ausschließlich aus Würfelfärbungen, deren kanonische Bahnrepräsentanten bereits untersucht wurden.

Dies gilt ganz ähnlich für den nächstmöglichen Pfad, der nicht durch diesen Würfel verläuft. Denn dieser Pfad enthält den in 5.21 unten rechts dargestellten Würfel. Für diesen Würfel gilt genauso, dass die fusionierende Drehung in der Schnittmenge des Stabilisators des vorangehenden Würfels und des letzten

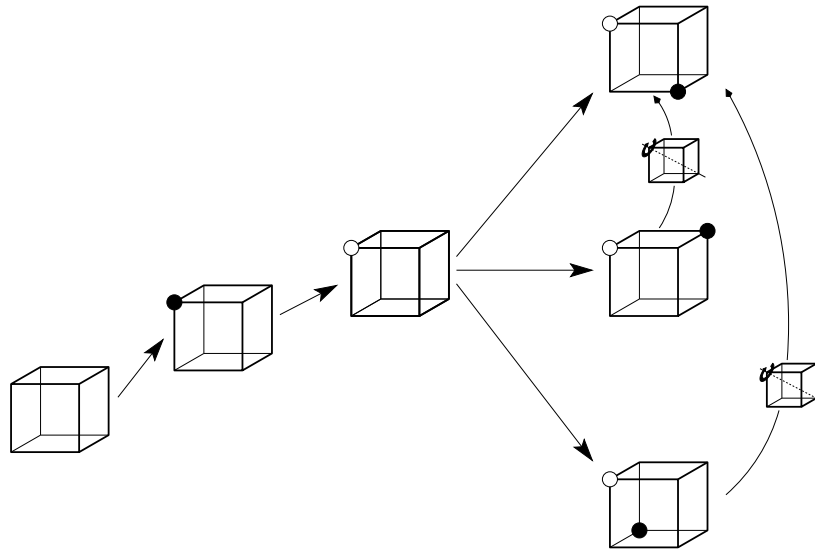


Abbildung 5.21: Abschneiden von Ästen

Würfels im Pfad ρ enthalten ist. Daher wird auch hier wieder die Untersuchung aller Pfade, die durch diesen Würfel verlaufen, abgebrochen.

Auf der rechten Seite in Abbildung 5.22 sind alle möglichen Würfelfärbungen dargestellt, die an der zweiten Position eines Pfades stehen können. Zu drei der vier Würfel wurde eine fusionierende Drehung skizziert, die sowohl im Stabilisator des vorangehenden unmarkierten Würfels als auch im Stabilisator der letzten Würfelfärbung des Pfades ρ enthalten ist. Daher kann, wie bereits zuvor erläutert wurde, jeder Pfad, der durch einen dieser drei Würfel verläuft, auf einen bereits untersuchten Pfad abgebildet werden.

5.14 Ergebnisse

Durch die Abschneidetechnik des hier vorgestellten Algorithmus konnten viele der in Abbildung 5.9 dargestellten Pfade bereits frühzeitig abgeschnitten werden. Dies ist in Abbildung 5.23 anschaulich dargestellt. Während der Tiefensuche wurde kein Würfel gefunden, dessen Färbungstupel kleiner ist als das des entsprechend gefärbten Würfels im Pfad ρ . Daraus kann gefolgert werden, dass der ursprüngliche in Abbildung 5.2 dargestellte Würfel kanonisch ist.

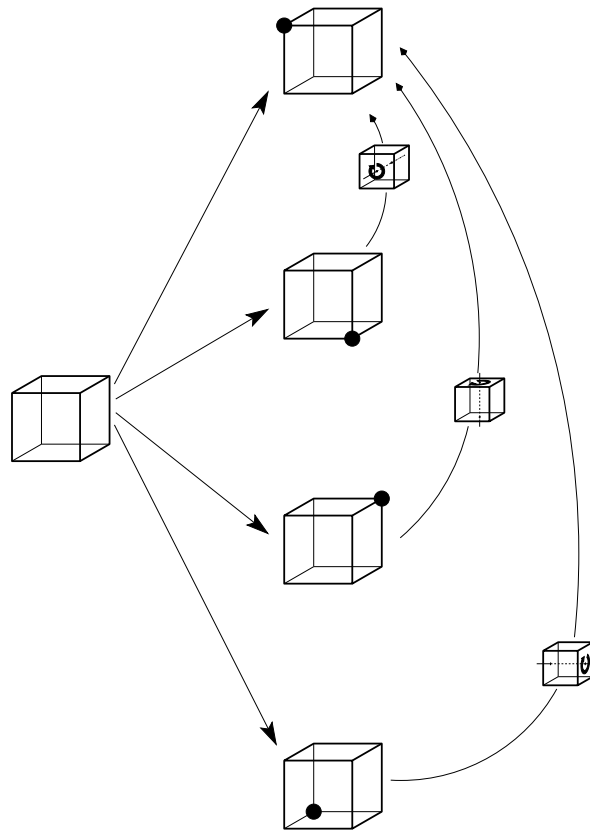


Abbildung 5.22: Abschneiden von Ästen

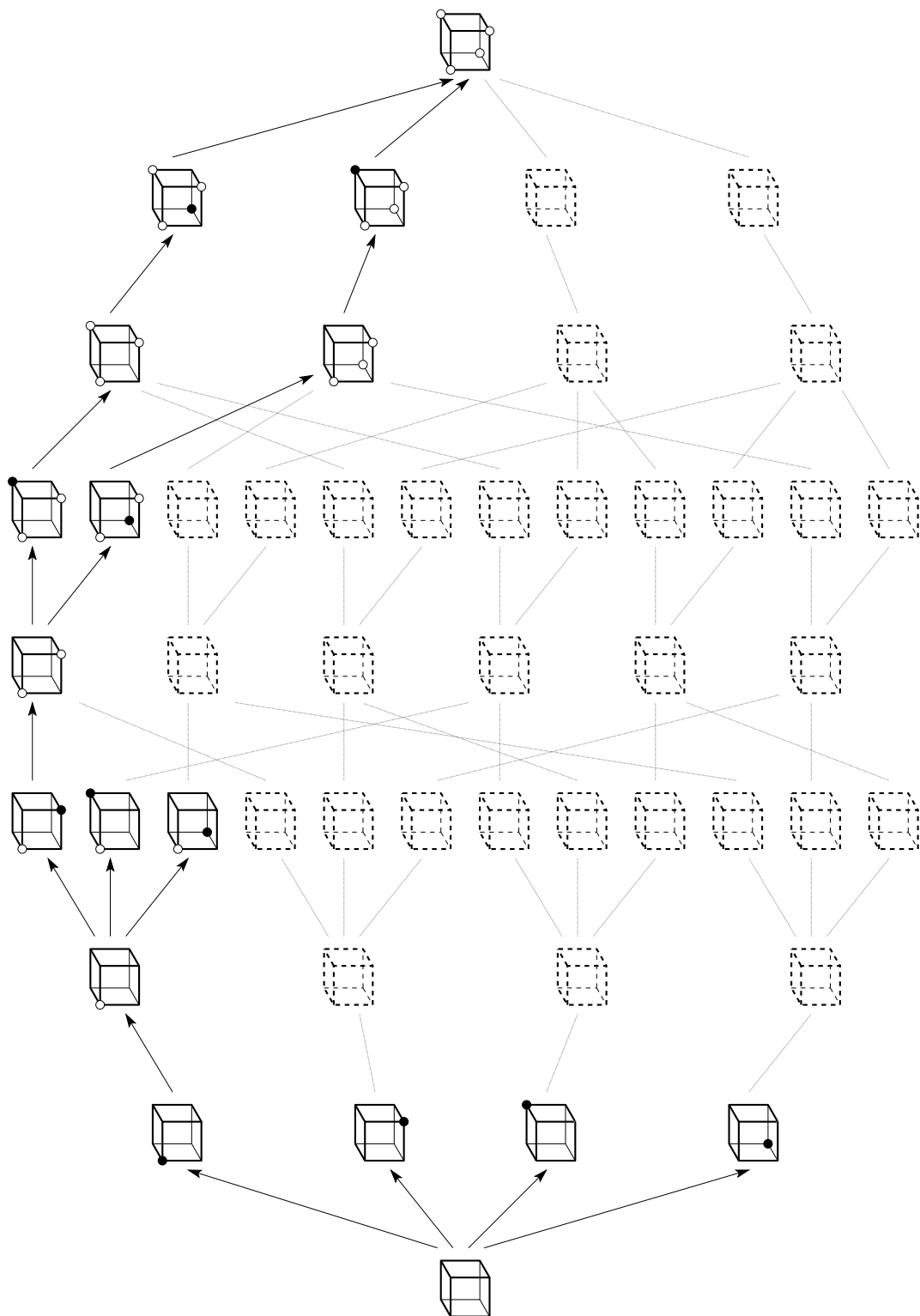


Abbildung 5.23: Anteil der im Algorithmus betrachteten Würfelfärbungen

Nachdem der Algorithmus vollständig durchlaufen wurde, ist die zuletzt berechnete Untergruppe des Stabilisators der letzten Würfelfärbung des Pfades ρ maximal. Diese wurde durch sukzessives Erweitern aus dem Stabilisators der vorletzten Würfelfärbung berechnet. Daher ist diese Gruppe identisch mit dem Stabilisators der ursprünglich untersuchten Würfelfärbung, der letzten Komponente des Pfades ρ .

Kapitel 7

Kanonizitätstests für Doppelnebenklassen

Die Fragestellung, ob zwei diskrete Strukturen „gleichgestaltig“ sind, wird in der Mathematik Isomorphieproblem genannt. Gleichgestaltig bedeutet in diesem Zusammenhang, dass eine strukturerhaltende Abbildung, ein sogenannter Homomorphismus, zwischen den beiden Strukturen existiert. Diese Art von Problem ist meist nicht leicht zu lösen.

Ein prominenter Fall ist das Isomorphieproblem auf der Menge der Graphen, von dem nicht bekannt ist, ob es zur Klasse der *NP*-vollständigen Probleme gehört. Ein weiteres Problem, das auch zu den Isomorphieproblemen gehört, ist das Teilgraphenisomorphieproblem, bei dem entschieden werden muss, ob ein gegebener Graph ein Teilgraph eines anderen Graphen ist. Von diesem Problem ist bekannt, dass es zur Klasse der *NP*-vollständigen Probleme gehört und daher sehr schwer zu lösen ist.

Unter Verwendung des Split Lemmas 3.5.1 können viele dieser Isomorphieprobleme auf eine Problemstellung aus dem Bereich der Doppelnebenklassen zurückgeführt werden. Dies ermöglicht es, denselben Lösungsansatz zur Lösung unterschiedlichster Probleme anzuwenden.

Der ursprüngliche Algorithmus des Leiterspiels, wie er von Schmalz [17] entwickelt wurde, ist außerordentlich effizient bei der Konstruktion von Doppelnebenklassen. Die Effizienz dieses Ansatzes ist jedoch an die Voraussetzung geknüpft, dass der Zugriff auf die im Speicher liegenden Zwischenergebnis-

se sehr schnell erfolgen kann. Da die Anforderungen an die Größe des Speicherplatzes jedoch enorm sind, bildet dies bei umfangreicheren Berechnungen häufig den limitierenden Faktor. Sobald im Arbeitsspeicher kein Platz mehr vorhanden ist und auf die viel langsamere Festplatte zugegriffen werden muss, stellt sich die Frage nach einer alternativen Strategie.

In diesem Kapitel werden drei Algorithmen beschrieben, die auf dem Leiterspiel aufbauen, aber mit sehr viel weniger Speicherplatz auskommen. Unter Verwendung der ordnungstreu erzeugten Read [15] können auch diese Algorithmen zur Konstruktion von Doppelnebenklassen eingesetzt werden. Das Anwendungsfeld dieser neuen Algorithmen ist jedoch wesentlich breiter, da es mit jedem der drei Algorithmen möglich ist, für eine einzelne Nebenklasse zu prüfen, ob diese kanonisch ist. In Abschnitt 7.6.3 wird weiterhin beschrieben, wie jeder dieser Kanonizitätstests abgewandelt werden kann, um zu einer gegebenen Nebenklasse den kanonischen Repräsentanten der Bahn dieser Nebenklasse zu berechnen. Beides ist mit der ursprünglichen Version des Leiterspiels so nicht möglich.

Der Breadthfirst Strong Ladders Leiterspiel (Breadthfirst StroLL) Algorithmus basiert genau wie der ursprüngliche Leiterspiel Algorithmus von Schmalz auf einer Breitensuche, bei der alle Zwischenergebnisse im Speicher gehalten werden müssen. Der Speicherbedarf des Depthfirst StroLL ist trotzdem wesentlich geringer als der des ursprünglichen Leiterspiel Algorithmus, da es die starken Leitern ermöglichen, die Anzahl der zu berechnenden Nebenklassen auf ein Minimum zu reduzieren. Im Gegensatz zum ursprünglichen Leiterspiel Algorithmus nach Schmalz kann mit diesem Algorithmus der kanonische Repräsentant zu einer einzelnen Nebenklasse bestimmt werden.

Mit dem Depthfirst Strong Ladders Leiterspiel (Depthfirst StroLL) Algorithmus ist es möglich, für eine einzelne Nebenklasse in Tiefensuche zu bestimmen, ob diese kanonisch ist. Dieser Algorithmus hat einen sehr geringen Speicherbedarf und ist auch zur Lösung von sehr großen Isomorphieproblemen geeignet, die sich aufgrund ihres Speicherbedarfs nicht mit dem Breadthfirst StroLL berechnen lassen.

Der Leiterspiel Light Algorithmus ist ein kleiner schneller Algorithmus, mit dem eine einzelne Nebenklasse daraufhin überprüft werden kann, ob diese kanonisch ist. Wie auch der Depthfirst StroLL hat auch das Leiterspiel Light nur einen sehr geringen Speicherbedarf. Das Leiterspiel Light zeichnet sich insbesondere durch seine Effizienz bei der Lösung von kleineren Isomorphieproblemen aus.

7.1 Würfelfärbungen und Rechtsnebenklassen

Die in diesem Kapitel beschriebenen Algorithmen behandeln ausschließlich Isomorphieprobleme in Verbindung mit Nebenklassen. Der Zusammenhang zwischen den Würfelfärbungen aus Kapitel 5 und den Nebenklassen, die in den Beschreibungen der Algorithmen in diesem Kapitel verwendet werden, wird durch Lemma 1 und das Split Lemma 3.5.1 hergestellt.

Eine Färbung der Ecken eines Würfels, bei der jede Ecke entweder schwarz, weiß oder ungefärbt ist, kann wie folgt beschrieben werden: Die Würfecken werden mit den Zahlen 1 bis 8 gekennzeichnet. Anschließend kann die Färbung als Abbildung $g : \{1, \dots, 8\} \longrightarrow \{\text{schwarz}, \text{weiß}, \text{ungefärbt}\}$ von der Menge der Ecken in die Menge der Farben dargestellt werden.

Die Symmetrische Gruppe S_8 operiert auf der Menge $\{1, \dots, 8\}$. Jede Permutation kann als bijektive Abbildung $f : \{1, \dots, 8\} \longrightarrow \{1, \dots, 8\}$ dargestellt werden. Dann operiert die S_8 auf der Menge aller Würfelfärbungen durch Komposition $g \circ f$ der beiden jeweiligen Abbildungen.

In Anlehnung an Lemma 1 wird die S_8 in den weiteren Ausführungen mit G bezeichnet. Nach Lemma 1 existiert zu jeder Bahn von G auf der Menge der Würfelfärbungen und jedem Element ω aus dieser Bahn ein G -Isomorphismus, durch den jede Würfelfärbung aus dieser Bahn auf eine Rechtsnebenklasse abgebildet werden kann. Dieser G -Isomorphismus ermöglicht es, jede Situation, die im Beispiel aus Kapitel 5 anhand von Würfeln beschrieben wurde, auf eine entsprechende Situation im Zusammenhang mit Nebenklassen abzubilden.

Jede Bahn von G in der Menge der Würfelfärbungen, ist eine Teilmenge, die aus allen Färbungen besteht, die dieselbe Anzahl schwarz und weiß gefärbter Ecken besitzen. Die Gruppe aller Drehungen des Würfels im Raum ist eine Untergruppe der S_8 die mit B bezeichnet wird. Im Beispiel aus Kapitel 5 operiert diese Untergruppe B auf jeweils einer der Bahnen der Gruppe G in der Menge der Würfelfärbungen.

Da B eine Untergruppe von G ist, ist der durch ein Bahnelement ω festgelegte G -Isomorphismus nach Lemma 1 auch ein B -Isomorphismus. Daher ist der Stabilisator jeder Würfelfärbung in der Gruppe B identisch mit dem Stabilisator der entsprechenden Nebenklasse im Bild.

Es bezeichne Ω_1 und Ω_2 zwei Bahnen der Gruppe G in der Menge der Würfelfärbungen und $\psi : \Omega_1 \longrightarrow \Omega_2$ einen B -Homomorphismus. Zu jedem Element ω aus einer der Mengen Ω_1 und Ω_2 bezeichne $\varphi_\omega : \omega^G \longrightarrow G_\omega \backslash G$ den in Lemma 1 beschriebenen G -Isomorphismus zum Element ω , so dass $\varphi_\omega(\omega^g) = G_\omega g$ für alle $g \in G$ gilt. Weiterhin bezeichne $\pi : G_\omega \backslash G \longrightarrow G_{\psi(\omega)} \backslash G$ für alle $\omega \in \Omega_1$ einen G -Homomorphismus, mit $\pi_\omega(G_\omega g) = G_{\psi(\omega)} g$ für alle $g \in G$. Dann kommutiert das Diagramm in Abbildung 7.1.

$$\begin{array}{ccc}
 \Omega_1 & \xrightarrow{\varphi_\omega} & G_\omega \backslash G \\
 \downarrow \psi & & \downarrow \pi_\omega \\
 \Omega_2 & \xrightarrow{\varphi_{\psi(\omega)}} & G_{\psi(\omega)} \backslash G
 \end{array}$$

Abbildung 7.1: Kommutatives Diagramm

Auf diese Weise können alle anhand von Würfelfärbungen beschriebenen Situationen und alle dazu verwendeten Homomorphismen von Gruppenoperationen aus Kapitel 5 auf entsprechende Situationen in Zusammenhang mit Nebenklassen abgebildet werden.

benen Algorithmus berechnet werden.

Wenn die Nebenklasse $E_i a$ nicht der ausgewählten Nebenklasse entspricht, so muss die Nebenklasse $E_i a$ nicht weiter untersucht werden. Andernfalls wird aus der Nebenklasse $E_i a$ die Nebenklasse $E_{i+1} a$ berechnet und die Suche nach dem kleinsten Pfad wird mit dem Block $\Delta_{\{i+1,k\}}^{aq}$ fortgesetzt.

7.4.5 Berechnung des Stabilisators von $A_k q$

Unter der Voraussetzung, dass $A_k q$ der kanonische Repräsentant der Bahn $A_k q^B$ ist, soll mit dem Leiterspiel-Light Algorithmus auch der Stabilisator C_k der Nebenklasse $A_k q$ unter der Operation der Gruppe B berechnet werden.

Wenn A_k eine Untergruppe von A_{k-1} ist, so ist der Stabilisator C_k nach dem Homomorphieprinzip 3.2.1 eine Untergruppe des Stabilisators C_{k-1} , dessen Berechnung in Abschnitt 7.4.2 auf Seite 130 beschrieben wurde. Daher kann C_k in diesem Fall mit dem Unterprogramm *ReduceStab* aus Abschnitt 7.6.2 berechnet werden.

Ist A_{k-1} hingegen eine Untergruppe von A_k , so ist nach dem Homomorphieprinzip 3.2.1 der Stabilisator C_{k-1} eine Untergruppe des Stabilisators von $A_k q$. Im Fundamentallema 1 wurde ein G -Isomorphismus beschrieben, der für alle $g \in G$ den Block $\Delta_{\{k-1,k\}}^g$ auf die Nebenklassen $A_{k-1} g$ abbildet. Durch diesen G -Isomorphismus wird jeder Block $\Delta_{\{k-1,k\}}^{aq} \in \Lambda_{k-1}$ auf die Nebenklasse $A_{k-1} aq$ abgebildet. Zu jedem Block $\Delta_{\{k-1,k\}}^{aq}$ aus der Menge Λ_{k-1} wird während des Leiterspiel Light Algorithmus ein fusionierendes Element $b \in B$ berechnet, so dass $A_{k-1} aqb = A_{k-1} p$ ist. Aus dem Homomorphieprinzip folgt, dass jedes dieser fusionierenden Elemente im Stabilisator C_k liegen muss. Denn sowohl $\Delta_{\{k-1,k\}}^{aq}$ als auch $\Delta_{\{k-1,k\}}^p$ liegen im Urbild des Blockes $\Delta_{\{k\}}^q$ unter dem G -Homomorphismus $\varphi(\{k-1,k\}, \{k\})$. Der Stabilisator von $A_k q$ ist aufgrund der beschriebenen Isomorphie zwischen den Blöcken und den Nebenklassen identisch mit dem Stabilisator der Blockes $\Delta_{\{k\}}^q$. Unter Verwendung des Satzes von Lagrange 3.1.1 und des Homomorphieprinzips kann gezeigt werden, dass die kleinste Gruppe, die sowohl den Stabilisator C_k als auch alle fusionierenden Elemente von Blöcken aus der Menge Λ_{k-1} enthält, der volle Stabilisator von

$A_k q$ sein muss.

Diese kleinste Gruppe, die sowohl den Stabilisator C_k als auch alle fusionierenden Elemente von Blöcken aus der Menge Λ_{k-1} enthält, kann mit Hilfe des Unterprogramms *ExtendGroup* aus Abschnitt 7.6.4 berechnet werden.

7.5 Teilgraphenisomorphieproblem und Doppelnebenklassen

Es seien zwei Graphen \mathcal{A} und \mathcal{B} auf endlich vielen Knoten gegeben und es soll die Frage beantwortet werden, ob der Graph \mathcal{A} einen Teilgraphen enthält, der isomorph zum Graphen \mathcal{B} ist. Diese Problemstellung wird Teilgraphenisomorphieproblem genannt und kann auf ein Doppelnebenklassenproblem abgebildet werden. Auf diese Weise kann das Teilgraphenisomorphieproblem mit Hilfe von jedem der drei in Kapitel 7 beschriebenen Algorithmen gelöst werden. Das Teilgraphenisomorphieproblem ist insbesondere deshalb interessant, weil die Spezialfälle Cliquensuche und Hamiltonkreisproblem und daher auch das Teilgraphenisomorphieproblem selbst zur Klasse der NP-vollständigen Probleme gehören.

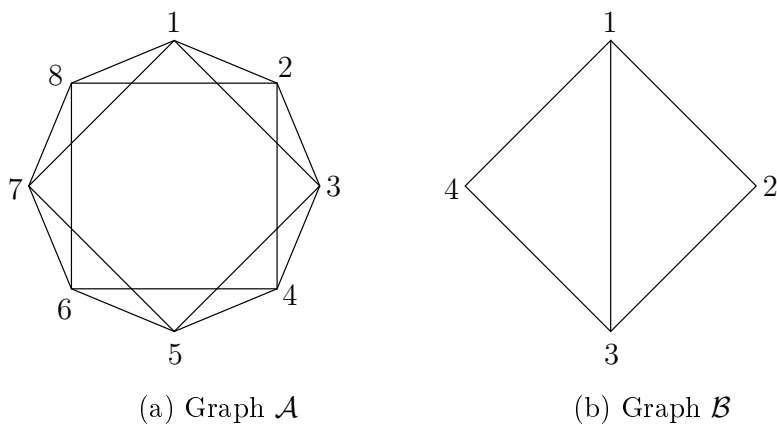


Abbildung 7.5: schlichte Graphen auf acht und auf vier Knoten

Es bezeichne (V_1, E_1) den Graphen \mathcal{A} und $k = |E_1|$ die Anzahl der Kanten des Graphen \mathcal{A} . Es bezeichne (V_2, E_2) den Graphen \mathcal{B} und $i = |E_2|$ die Anzahl

der Kanten des Graphen \mathcal{B} . Die beiden Graphen \mathcal{A} und \mathcal{B} können zu einem einzigen Graphen \mathcal{C} vereinigt werden, der die beiden ursprünglichen Graphen als disjunkte Teilgraphen enthält, wie in Abbildung 7.6 dargestellt. Dieser Vereinigungsgraph \mathcal{C} besteht aus $m = |V_1| + |V_2|$ Knoten und $i + k$ Kanten.

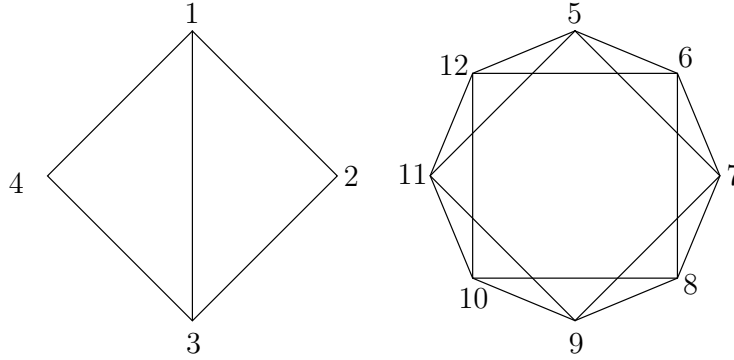


Abbildung 7.6: Vereinigungsgraph \mathcal{C}

Der Graph \mathcal{C} ist isomorph zu einem Teilgraph des vollständigen Graphen auf m Knoten. Es bezeichne (V', E') einen der Teilgraphen im vollständigen Graphen auf m Knoten, der isomorph zum Graphen \mathcal{C} ist. Der Graph (V', E') kann so in zwei disjunkte Teilgraphen zerlegt werden, dass der eine Teil isomorph zum Graphen \mathcal{A} und der andere isomorph zum Graphen \mathcal{B} ist. Die $n = \binom{m}{2}$ Kanten des vollständigen Graphen sollen jetzt so auf die Zahlen 1 bis n abgebildet werden, dass die Kanten aus E' die Nummern 1 bis $i + k$ erhalten. Dabei sollen die Kanten der beiden disjunkten Teilgraphen so auf die Nummern 1 bis $i + k$ abgebildet werden, dass die Kanten des Teilgraphen von (V', E') , der isomorph zum Graphen \mathcal{A} ist, auf die Zahlen 1 bis k abgebildet werden. Weiterhin soll gelten, dass die Kanten des Teilgraphen von (V', E') , der isomorph zum Graphen \mathcal{B} ist, auf die Nummern $k + 1$ bis $i + k$ abgebildet werden.

Die weiteren Schritte laufen analog zum Beispiel in Kapitel 3.5 und werden daher nur sehr knapp erläutert. Die Gruppe S_m operiert durch Permutation der Knotennummerierungen auf dem vollständigen Graphen. Analog zu Kapitel 3.5 operiert die Gruppe S_m auf der Menge der Kanten, indem jede Kante (v_1, v_2) von jedem Element $g \in S_m$ auf die Kante (v_1^g, v_2^g) abgebildet wird.

Daher existiert ein Gruppenmonomorphismus φ von S_m auf eine Untergruppe B der S_n und ein Homomorphismus von Gruppenoperationen, so dass für alle Elemente $g \in S_m$ und alle Kanten ω des vollständigen Graphen $\omega^g = \omega^{\varphi(g)}$ gilt.

Es bezeichne

- λ_i die Partition $\{\{1, \dots, i\}, \{i+1, \dots, n\}\}$,
- λ_k die Partition $\{\{1, \dots, k\}, \{k+1, \dots, n\}\}$,
- G die Gruppe S_n ,
- A_i den Stabilisator der Partition λ_i in G ,
- A_k den Stabilisator der Partition λ_k in G und
- $g \in S_n$ die Permutation $\begin{pmatrix} 1 & 2 & \dots & n-k & n-k+1 & n-k+2 & \dots & n \\ k+1 & k+2 & \dots & n & 1 & 2 & \dots & k \end{pmatrix}$.

Es bezeichne Ω die Menge aller Teilgraphen des gegebenen vollständigen Graphen, die aus m Knoten und i Kanten bestehen. Weiterhin bezeichne $\omega \in \Omega$ den Teilgraphen, bestehend aus den Kanten mit den Nummern 1 bis i . Die Gruppe G operiert durch Permutation der Kanten transitiv auf der Menge Ω und durch Rechtsmultiplikation auf der Menge $A_i \backslash G$. Nach Lemma 1 existiert zum Teilgraphen ω ein G -Isomorphismus, der für alle $h \in G$ den Teilgraphen $\omega^h \in \Omega$ auf die Nebenklasse $A_i h$ abbildet.

Wenn die Menge $\{A_i h \mid h \in A_k\}$ eine Nebenklasse enthält, die in der Bahn von $A_i g$ unter der Operation der Gruppe B liegt, so enthält der Graph \mathcal{A} einen Teilgraphen, der isomorph ist zum Graphen \mathcal{B} . Ob dies der Fall ist, kann mit Hilfe des Algorithmus aus Abschnitt 7.6.3 und einem der drei Algorithmen, die in den Kapitel 7.2 bis 7.4 beschrieben werden, berechnet werden.

Wenn das Kanonizitätsprädikat so gewählt wird, dass $A_i g$ die kanonische Nebenklasse ihrer Bahn ist, so würde jede Nebenklasse der Menge $\{A_i h \mid h \in A_k\}$, die in der Bahn von $A_i g$ liegt, auf diesen kanonischen Repräsentanten abgebildet. Auf diese Weise kann leicht überprüft werden, ob ein Teilgraph von \mathcal{A} isomorph zum Graphen \mathcal{B} ist.

7.6 Unterprogramme des Kanonizitätstests

7.6.1 Identitäten und inverse Homomorphismen

Um die Isomorphietests durchführen zu können, die in Kapitel 7.2 bis 7.4 beschrieben wurden, muss ein Algorithmus bereitgestellt werden, mit dem überprüft werden kann, ob zwei Nebenklassen identisch sind.

Für manche Gruppen kann dieses Problem sehr leicht gelöst werden, im Allgemeinen ist dies jedoch für zwei Nebenklassen einer Untergruppe in einer endlich erzeugten Gruppe nicht entscheidbar. Daher existiert auch keine berechenbare Funktion, mit der die Laufzeit zur Lösung dieses Problems für eine allgemeine, endlich erzeugte Gruppe abgeschätzt werden könnte.

Bei den in Kapitel 7 und Kapitel 7.2 vorgestellten Algorithmen wird jedoch vorausgesetzt, dass der Gruppenindex zweier aufeinander folgender Gruppen der Leiter immer endlich sein muss. Unter dieser Bedingung können die Nebenklassen der kleineren Gruppe in der größeren mit dem Todd-Coxeter Algorithmus [22] berechnet werden. Die Tabellen, die bei der Durchführung dieses Algorithmus erstellt werden, ermöglichen es, für die entsprechenden Nebenklassen zu entscheiden, ob diese identisch sind oder nicht. Ein weiterer Vorteil des Todd-Coxeter Algorithmus liegt darin, dass dieser auch gleichzeitig dazu verwendet werden kann, die Urbilder einer gegebenen Nebenklasse unter den entsprechenden Homomorphismen des Leiterspiels zu berechnen.

Es sei G eine Gruppe und (A_1, \dots, A_n) eine starke Untergruppenleiter von G nach A_n . Für je zwei aufeinander folgende Gruppen A_i, A_{i+1} der Leiter soll der Gruppenindex zwischen diesen beiden Gruppen endlich sein. Für alle $i < n$ und alle Leitergruppen A_i und A_{i+1} muss entweder $A_i \leq A_{i+1}$ oder $A_{i+1} \leq A_i$ gelten. Zur Vereinfachung der Schreibweise soll die kleinere der beiden Gruppen im Folgenden mit U und die größere mit V bezeichnet werden. Die Gruppe G operiert sowohl auf der Menge $U \backslash G$ als auch auf der Menge $V \backslash G$ durch Rechtsmultiplikation. In Kapitel 3.4 wurde bereits gezeigt, dass die Abbildung $\varphi : U \backslash G \rightarrow V \backslash G$, mit $\varphi(Ug) = Vg$ für alle $g \in G$, ein G -Homomorphismus ist.

Da U eine Untergruppe von V ist, können unter Verwendung des Todd-

Coxeter Algorithmus alle Nebenklassen von U in V berechnet werden. Diese Menge von Nebenklassen ist die Urbildmenge der Nebenklasse V unter der Abbildung φ . Da G auf der Menge $V \setminus G$ transitiv operiert, ist für alle $g \in G$ die Menge $\{Uvg \mid v \in V, \}$ die Urbildmenge von $Vg \in V \setminus G$ unter der Abbildung φ .

Angenommen es existiert eine weitere Nebenklasse Uh im Urbild von Vg , die nicht in der Menge $\{Uvg \mid v \in V, \varphi(Uv) = V\}$ enthalten ist. Dann müsste, da φ ein G -Homomorphismus ist, Uhg^{-1} im Urbild von V liegen, denn es gilt $\varphi(Uhg^{-1}) = \varphi(Uh)g^{-1} = Vg^{-1} = V$. Daraus folgt aber sofort, dass $hg^{-1} \in V$ ist und die Nebenklasse $Uhg^{-1}g$ in der Menge $\{Uvg \mid v \in V\}$ enthalten sein muss. Umgekehrt muss für jede Nebenklasse Uvg mit $\varphi(Uv) = V$ auch gelten, dass $\varphi(Uvg) = Vg$ ist.

Nachdem die Nebenklassen von U in V einmal berechnet wurden, können daher die Urbildmengen für alle $Vg \in V \setminus G$ leicht bestimmt werden. Zwei Nebenklassen Ug_1 und Ug_2 sind genau dann identisch, wenn das Element $g_1g_2^{-1}$ in U enthalten ist. Liegen zwei Nebenklassen Ug_1 und Ug_2 im selben Urbild einer Nebenklasse Vg , so kann mit Hilfe der Tabellen, die bei der Durchführung des Todd-Coxeter Algorithmus angelegt werden, überprüft werden, ob $g_1g_2^{-1}$ in U enthalten ist. Sind die Nebenklassen $\varphi(Ug_1)$ und $\varphi(Ug_2)$ hingegen verschieden, so wird wiederum eine Methode benötigt, um dies festzustellen.

Es sei $i \leq n$ und es seien $A_i g_1$ und $A_i g_2$ zwei Nebenklassen, für die festgestellt werden soll, ob diese identisch sind. Dies ist genau dann der Fall, wenn $g_1g_2^{-1} \in A_i$ ist. Um die Schreibweise zu vereinfachen, wird im Folgenden für alle $j \leq i$ die Gruppe $A_j \cap A_i$ mit E_j bezeichnet werden. Das Element $g_1g_2^{-1}$ wird mit g bezeichnet.

Ist $A_i \leq A_{i-1}$, so soll zuerst festgestellt werden, ob $g \in A_{i-1}$ ist. Ist $g \notin A_{i-1}$, so muss auch $g \notin A_i$ gelten. Ist $g \in A_{i-1}$, so kann mit Hilfe der Tabellen, die bei der Durchführung des Todd-Coxeter Algorithmus berechnet wurden, festgestellt werden, ob $g \in A_i$ ist.

Ist $A_i \geq A_{i-1}$ so kann folgende Beobachtung angewendet werden, um her-

auszufinden ob $g \in A_i$ ist. Angenommen es gilt $g \in A_i$ und es existiert ein $a \in A_i$, so dass $ag \in A_j$ ist, dann existiert auch ein $e \in E_j$, so dass $eag \in A_{j+1}$ gilt. Denn aus $g \in A_i$, $a \in A_i$ und $ag \in A_j$ folgt, dass $ag \in E_j$ ist. Dann existiert aber auch ein $e \in E_j$, so dass $eag \in E_{j+1}$ und damit auch $eag \in A_{j+1}$ ist. Angenommen es gilt $g \notin A_i$, dann existiert ein $j < i-1$ und ein $a \in A_i$, so dass $ag \in A_j$ ist und für alle $e \in E_j$ gilt, dass $eag \notin A_{j+1}$ ist. Denn angenommen es existieren Elemente $e, a \in A_i$, so dass $eag \in A_{i-1}$ ist, dann muss auch $g \in A_i$ sein.

Diese Beobachtung ermöglicht es, schrittweise zu überprüfen, ob zwei Nebenklassen identisch sind. Unter Verwendung des Todd-Coxeter Algorithmus können für alle $j < n$ mit $E_j \geq E_{j+1}$ die Nebenklassen von E_{j+1} in E_j berechnet werden. Diese Nebenklassen werden im folgenden Algorithmus benötigt. Unter Verwendung der beim Todd-Coxeter Algorithmus angefertigten Tabellen können dann auch die erforderlichen Identitätsvergleiche des folgenden Algorithmus durchgeführt werden. Der Algorithmus gibt genau dann den Wert *true* zurück, wenn g in der Gruppe A_i enthalten ist.

Pseudocode Membership Test

Input: Index i
 (A_1, \dots, A_n) $\backslash\backslash$ Subgroup Ladder
 $g \in G$ $\backslash\backslash$ check if $g \in A_i$
Output: Bool

```

1  GroupElement  $a \leftarrow 1_G$ 
2  foreach (  $j \in \{1, \dots, i-1\}$  )                       $\backslash\backslash$ go through in increasing order
3      if (  $A_j \leq A_{j+1}$  )
4          continue
5      foreach (  $E_{j+1}e \in \{E_{j+1}h \mid h \in E_j\}$  )
6          if (  $ea g \notin A_{j+1}$  )
7              continue
8          endif
9           $a \leftarrow ea$ 
10     endforeach
11     if (  $ag \notin A_{j+1}$  )
12         return false
13     endif
14 endforeach
15 return true

```

Der Todd-Coxeter Algorithmus muss nicht bei jeder Identitätsprüfung und bei jeder Berechnung des Inversen erneut durchgeführt werden. Es genügt vielmehr, wenn der Algorithmus nur einmal für die entsprechenden Gruppen ausgeführt wird und die Ergebnisse gespeichert werden. Die Gruppen, für die der Todd-Coxeter Algorithmus durchgeführt werden muss, lassen sich allein anhand der gegebenen Untergruppenleiter bestimmen. Daher braucht der Todd-Coxeter Algorithmus für eine gegebene Leiter nur ein einziges Mal für die jeweiligen Gruppen ausgeführt werden. Danach können beliebig viele Problemstellungen zu dieser gegebenen Leiter bearbeitet werden, ohne dass der Algorithmus erneut ausgeführt werden müsste.

Bei gegebener Untergruppenleiter (A_1, \dots, A_n) muss der Algorithmus für je zwei aufeinander folgende Gruppen der Leiter einmal ausgeführt werden. Weiterhin muss der Todd-Coxeter Algorithmus für jedes $1 < i \leq n$ mit $A_{i-1} \leq A_i$

und jedes $j < i$ durchgeführt werden. Wenn $A_j \leq A_{j+1}$ ist, so müssen die Nebenklassen der Gruppe $E_j = A_i \cap A_j$ in der Gruppe $E_{j+1} = A_i \cap A_{j+1}$ berechnet werden. Wenn $A_j \geq A_{j+1}$ ist, so müssen die Nebenklassen der Gruppe $E_{j+1} = A_i \cap A_{j+1}$ in der Gruppe $E_j = A_i \cap A_j$ berechnet werden. Daher muss der Todd-Coxeter Algorithmus für jede Untergruppenleiter (A_1, \dots, A_n) maximal $n^2 + n$ Mal ausgeführt werden.

Für viele spezielle Gruppen existieren effizientere Methoden, um das Urbilder einer Nebenklassen unter einem der G -Homomorphismen zu berechnen oder um zu überprüfen, ob zwei Nebenklassen identisch sind. Auf diese Methoden einzugehen würde aber den Rahmen sprengen. Daher soll an dieser Stelle nur noch auf den Membership Test von Sims [20] verwiesen werden, der für viele Permutationsgruppen geeignet ist. Dieser Membership Test wurde durch Jerrum [7] und Cooperman, Finkelstein und Purdom [1] weiterentwickelt.

7.6.2 ReduceStab

Das Unterprogramm *ReduceStab* hat die Aufgabe, zu gegebenen Untergruppen A, C einer Gruppe G und einer gegebenen Rechtsnebenklasse Ag den Stabilisator D dieser Nebenklasse in der Gruppe C zu bestimmen. Genauer gesagt wird eine Menge von Elementen aus der Gruppe C gesucht, die den Stabilisator von Ag in C erzeugen.

Existiert eine starke Untergruppenleiter (A_1, \dots, A_n) von G nach A und sind die Gruppenindizes aller aufeinander folgenden Gruppen der Leiter klein und insbesondere endlich, so kann der in Abschnitt 7.6.3 vorgestellte Kanonisierungsalgorithmus zur Berechnung des Stabilisators eingesetzt werden. Wenn aufgrund der gegebenen Problemstellung klar ist, dass der Gruppenindex $(C : D)$ des gesuchten Stabilisators D in der Gruppe C klein ist, dann gibt es Algorithmen, die in dieser Situation wesentlich effizienter arbeiten.

Beim Leiterspielalgorithmus taucht folgende Situation häufig auf. Es seien B eine Untergruppe von G und U, V zwei aufeinander folgende Gruppen der Untergruppenleiter des Leiterspielalgorithmus. Dann ist die Abbildung $\varphi : U \backslash G \rightarrow V \backslash G$ ein B -Homomorphismus, mit $\varphi(Ug) = Vg$ für alle $g \in G$. Bezeichnet C den Stabilisator einer Nebenklasse $Vh \in V \backslash G$ unter der Operati-

on der Gruppe B , dann muss, wie in Kapitel 3.4 gezeigt wurde, der Stabilisator D der Nebenklasse Uh gleich $U \cap B$ sein. Nach dem Homomorphieprinzip ?? ist der Stabilisator D der Nebenklasse Uh auch eine Untergruppe von C , daher gilt $D = U \cap C$.

Üblicherweise hängt die Laufzeit der Algorithmen zur Berechnung des Stabilisators von der Größe des Gruppenindex $(C : D)$ ab. Der Gruppenindex $(C : D)$ kann aber maximal so groß sein, wie der Gruppenindex $(V : U)$. Denn für die Menge $UC = \{uc \mid u \in U, c \in C\}$ gilt $|UC| = \frac{|U||C|}{|U \cap C|}$. Mit dem Satz von Lagrange 3.1.1 gilt

$$(V : D) = (V : C)(C : D) = (V : U)(U : D)$$

Daraus folgt

$$\frac{(V : C)}{(U : D)} = \frac{(V : U)}{(C : C \cap U)} = \frac{|V||C \cap U|}{|U||C|} = \frac{|V|}{|UC|} \geq 1$$

Daher sollte bei der Berechnung der Untergruppenleiter, die für den Leiterspielalgorithmus verwendet werden soll, darauf geachtet werden, dass die Gruppenindizes zwischen aufeinander folgenden Gruppen der Leiter immer klein sind.

Das Untergruppenlemma nach Schreier kann dazu verwendet werden, den Bahnenalgorithmus so zu erweitern, dass dieser auch die Erzeuger des gesuchten Stabilisators berechnet.

Lemma 21. (*Schreier*)

Es sei G eine Gruppe, S ein Erzeugendensystem von G und U eine Untergruppe von G . Es sei $\mathcal{T} \subseteq G$ ein minimales Repräsentantensystem aller Rechtsnebenklassen von U in G , das bedeutet:

$$\forall g \in G \exists! t \in \mathcal{T} : Ug = Ut$$

Weiterhin sei eine Abbildung $\phi : G \rightarrow \mathcal{T}$ gegeben, mit der Eigenschaft, dass für alle $g \in G$ und $t = \phi(g)$ gilt, dass $Ug = Ut$ ist.

Dann ist die Menge $\{tsr \mid t \in \mathcal{T}, s \in S, r = \phi(ts)^{-1}\}$ ein Erzeugendensystem der Gruppe U .

Beweis. (in Anlehnung an den Beweis in [6])

Es sei u ein beliebiges Element aus der Untergruppe U . Es sei $x_0 = \phi(1_G)$ und $v = x_0^{-1}ux_0 \in U$. Da S ein Erzeugendensystem ist, läßt sich v als die Verknüpfung endlich vieler Elemente $v = s_1 \dots s_n$ mit $s_1, \dots, s_n \in S$ schreiben. Für alle $i \leq n$ sei $b_i = s_1 \dots s_i$ und $x_i = \phi(b_i)$. Da v in U liegt, gilt auch $x_n = \phi(v) = \phi(1_G)$. Dann ist für alle $i < n$ das Element $x_{i+1} = \phi(b_{i+1}) = \phi(b_i s_{i+1}) = \phi(x_i s_{i+1})$. Es gilt:

$$\begin{aligned} u &= x_0 v x_0^{-1} = x_0 s_1 \dots s_n x_n^{-1} \\ u &= x_0 s_1 (x_1^{-1} x_1) s_2 (x_2^{-1} x_2) \dots s_{n-1} (x_{n-1}^{-1} x_{n-1}) s_n x_n^{-1} \\ u &= (x_0 s_1 x_1^{-1}) (x_1 s_2 x_2^{-1}) \dots (x_{n-1} s_n x_n^{-1}) \end{aligned}$$

Für alle $i \leq n$ gilt aber, dass $x_{i-1} s_i x_i^{-1} = x_{i-1} s_i \phi(x_{i-1} s_i)^{-1}$ ist. \square

Operiert die Gruppe G auf einer Menge Ω , so kann mit dem einfachen Bahnenalgorithmus die Bahn eines Elementes δ unter der Operation einer Gruppe C berechnet werden. Dem Bahnenalgorithmus wird eine Menge S von Erzeugern der Gruppe C und das Element δ übergeben. Ist die Bahn von δ endlich, so berechnet der Bahnenalgorithmus daraus die Menge Δ , die Bahn des Elementes δ .

Beim erweiterten Bahnenalgorithmus wird zusätzlich ein Abbildung ϕ berechnet. Die Abbildung ϕ wird so konstruiert, dass nach Durchführung des Algorithmus für alle $\omega \in \Delta$ gilt, dass $\delta^{\phi(\omega)} = \omega$ ist. Die Menge E enthält am Ende eine Menge von Erzeugern, so dass für den gesuchten Stabilisator $C_\delta = \langle E \rangle$ gilt.

Um den Bahnenalgorithmus zur Bestimmung des Stabilisators verwenden zu können, wird ein Algorithmus benötigt, mit dessen Hilfe feststellbar ist, ob zwei Nebenklassen Ag_1 und Ag_2 identisch sind. In Abschnitt 7.6.1 wurden mehrere Algorithmen beschrieben, die dazu geeignet sind.

Bahnenalgorithmus

<i>Input:</i>	Set S	$\backslash \backslash C = \langle S \rangle$
	SetElement δ	
<i>Referenced :</i>	Set Δ	$\backslash \backslash \Delta = \{\delta\}$
	$\phi : \Delta \rightarrow C$	$\backslash \backslash \phi(\delta) = 1_C$
	Set E	$\backslash \backslash \langle E \rangle \leq C_\delta$

```

1  foreach (  $\delta \in \Delta$  )
2      foreach (  $g \in S$  )
3           $\omega \leftarrow \delta^g$ 
4          if (  $\omega \notin \Delta$  )
5               $\Delta \leftarrow \Delta \cup \omega$ 
6               $\phi(\omega) \leftarrow \phi(\delta)g$ 
7          else
8               $E \leftarrow E \cup \{\phi(\delta)g\phi(\omega)^{-1}\}$ 
9          endif
10     endforeach
11 endforeach
12 exit

```

Die Anzahl der Erzeuger in der Menge E kann mitunter sehr groß werden, was insbesondere dann ein Nachteil sein kann, wenn diese Erzeuger wieder als Eingabe für einen weiteren Durchlauf des Bahnenalgorithmus verwendet werden sollen. Denn die Laufzeit des Bahnenalgorithmus wächst linear mit der Anzahl der Erzeuger. Daher sollte der Bahnenalgorithmus mit anderen Algorithmen kombiniert werden, die in der Lage sind, die Anzahl der Erzeuger zu reduzieren.

Die Suche nach Methoden, mit denen die Anzahl der Erzeuger reduziert werden kann, bildet einen eigenen Forschungsbereich. Denn im Allgemeinen ist schon allein das Wortproblem für endlich repräsentierte Gruppen nicht entscheidbar. Daher ist die Frage, ob zwei Gruppenelemente $s_{i_1}s_{i_2}\dots s_{i_n}$ und $s_{j_1}s_{j_2}\dots s_{j_m}$ mit $s_{i_1}, s_{i_2}, \dots, s_{i_n}, s_{j_1}, s_{j_2}, \dots, s_{j_m} \in S$ das selbe Gruppenelement bezeichnen, im Allgemeinen nicht berechenbar. Mit Hilfe des Knuth-Bendix

Vervollständigungsverfahren kann jedoch in manchen Fällen eine kleinere Erzeugermenge gefunden werden [21].

Im Folgenden soll noch auf einen Spezialfall eingegangen werden, für den ein besonders effizienter Algorithmus existiert. Es sei Δ eine kleine endliche Menge und C eine Permutationsgruppe auf der Menge Δ . Diese Technik kann immer dann angewendet werden, wenn zum gesuchten Stabilisator der Nebenklasse Ag ein Element $\delta \in \Delta$ existiert, so dass C_δ identisch mit dem gesuchten Stabilisator ist.

Der Schreier-Sims Algorithmus [20] kann zur Berechnung eines starken Labelled Branchings zur Gruppe C eingesetzt werden, einer Datenstruktur zur Speicherung einer Permutationsgruppe. In [2] beschreiben Cooperman und Finkelstein zwei verschiedene Algorithmen, um einen Basiswechsel in einem starken Labelled Branching durchzuführen. Insbesondere der Basiswechsel vom Typ „Las Vegas“ ist sehr effizient und eignet sich hervorragend zur Berechnung dieser Stabilisatoren. Indem das Element $\delta \in \Delta$ durch einen Basiswechsel an die erste Position der Basis des Labelled Branchings permutiert wird, kann der Stabilisator von δ anschließend mühelos berechnet werden.

7.6.3 Canonizer

◆ *hier wurden viele Änderungen vorgenommen, daher muss man hier nochmal korrekturlesen*

Es sei eine Gruppe G gegeben und es sei (A_1, \dots, A_n) eine starke Untergruppenleiter von G nach A_n . Zu jeder Gruppe A_i der Leiter soll die Menge der Nebenklassen von A_i in G mit $\Omega_i = A_i \backslash G$ bezeichnet werden. Für alle $i \leq n$ sei auf der Menge Ω_i eine Ordnung gegeben, welche die in Kapitel 6.1 geforderte Bedingung an die Ordnungen der Nebenklassen erfüllt. Es sei B eine weitere Untergruppe von G , die durch Rechtsmultiplikation auf jeder der Mengen $\Omega_1, \dots, \Omega_n$ operiert.

Beim Aufruf der Methode *Canonizer* soll zu einer Nebenklasse $A_k q$ sowohl der kanonische Bahnrepräsentant unter der Operation der Gruppe B bestimmt werden als auch der Stabilisator dieses kanonischen Bahnrepräsentanten be-

rechnet werden.

◆ *ergänze noch Beschreibungen der Eingabeparameter: k und A_kq*

Optional kann der Methode *Canonizer* eine Untergruppe des Stabilisators der Nebenklasse A_iq übergeben werden, die es ermöglicht, die Berechnungen zu beschleunigen.

Jeder der drei in Kapitel 7 vorgestellten Algorithmen kann leicht so abgewandelt werden, dass dieser zu einer gegebene Nebenklasse A_kq nicht nur überprüft, ob diese Nebenklasse kanonisch ist, sondern auch den entsprechenden kanonischen Bahnrepräsentanten und dessen Stabilisator berechnet. Ist die gegebene Nebenklasse A_kq kanonisch, so kann der zugehörige Stabilisator bereits mit Hilfe von jedem der drei Algorithmen aus Kapitel 7 berechnet werden, ohne dass dazu Änderungen an den Algorithmen notwendig wären. Sobald aber zur Nebenklasse A_kq ein Element $b \in B$ gefunden wurde, für das $A_kqb \prec A_kq$ gilt, werden bei jedem der drei in Kapitel 7 beschriebenen Algorithmen alle weiteren Berechnungen abgebrochen und es wird das Ergebnis „ A_kq nicht kanonisch“ zurückgegeben.

Anhand des depthfirst StroLL Algorithmus aus Kapitel 7.3 soll beschrieben werden, welche Anpassungen notwendig sind, um einen der drei Algorithmen aus Kapitel 7 zur Berechnung des kanonischen Repräsentanten und dessen Stabilisator einzusetzen. Die erste Änderung am depthfirst StroLL Algorithmus ist, dass zusätzlich zu der Information, dass die untersuchte Nebenklasse A_kq nicht kanonisch ist, ein beliebiges Element $b \in B$ zurückgegeben wird, für das $A_kqb \prec A_kq$ ist. Weiterhin soll die zu diesem Zeitpunkt bekannte Untergruppe des Stabilisators der Nebenklasse A_kq zurückgegeben werden.

Nach Satz 3.1.2 existiert ein Homomorphismus von Gruppenoperationen, der jede Nebenklasse auf ihren Stabilisator abbildet. Daher ist der Stabilisator der Nebenklasse A_kqb gleich dem b -konjugierten Stabilisator der Nebenklasse A_kq . Wenn der so abgewandelte depthfirst StroLL Algorithmus abbricht, weil eine kleinere Nebenklasse A_kqb gefunden wurde, so kann aus der bekannten Untergruppe C_k des Stabilisators der Nebenklasse A_kq die Untergruppe $b^{-1}C_kb$ berechnet werden, die eine Untergruppe des Stabilisators der Neben-

klasse A_kqb ist. Anschließend kann der depthfirst StroLL Algorithmus erneut aufgerufen werden, nur diesmal unter Eingabe der Nebenklasse A_kqb und des Stabilisators $b^{-1}C_kb$. Das Ergebnis dieses Aufrufs ist dann entweder eine noch kleinere Nebenklasse A_kqb' zusammen mit einer eventuell vergrößerten Untergruppe des Stabilisators der Nebenklasse A_kqb oder die Nebenklasse A_kqb ist bereits kanonisch. Wenn eine kleinere Nebenklasse gefunden wurde, so wird die beschriebene Vorgehensweise solange wiederholt, bis der kanonische Repräsentant gefunden wurde. In diesem Fall wird der depthfirst StroLL Algorithmus den vollen Stabilisator dieser Nebenklasse A_kqb bereitstellen.

Da eine Totalordnung auf der Menge der Nebenklassen Ω_k gegeben ist und bei jedem der Aufrufe des depthfirst StroLL Algorithmus entweder der kanonische Repräsentant gefunden wird oder eine kleinere, als die bisher bekannte Nebenklasse der Bahn A_kq^B zurückgegeben wird, führt diese Vorgehensweise zu einem Algorithmus, der zu jeder Nebenklasse $A_i g$ den kanonischen Bahnrepräsentanten und dessen Stabilisator berechnet. Die Effizienz des ursprünglichen depthfirst StroLL Algorithmus bleibt dabei weitgehend erhalten.

7.6.4 ExtendGroup

In diesem Kapitel werden verschiedene Methoden beschrieben, wie eine Untergruppe U einer Gruppe B um einen Erzeuger b erweitert werden kann. Genauer gesagt soll die kleinste Untergruppe V von B berechnet werden, die sowohl U als auch den Erzeuger b enthält.

Die Lösung dieses Problems hängt stark davon ab, welche Datenstruktur zum Speichern der Gruppe verwendet werden soll. Im einfachsten Fall, ist bereits ein Erzeugendensystem zur Gruppe U gegeben und es soll lediglich ein Erzeugendensystem der Gruppe V bestimmt werden. Dafür sind keine weiteren Berechnungen notwendig, sondern es genügt, das Element b zur Menge der Erzeuger von U hinzuzufügen. Denn diese Menge ist bereits ein Erzeugendensystem der gesuchten Gruppe V .

Ein Nachteil dieser Methode ist, dass das Erzeugendensystem auf diese Weise sehr groß werden kann. Bei den in Kapitel 7 beschriebenen Algorithmen wurde vorausgesetzt, dass der Index je zweier in der Leiter aufeinander folgender

Gruppen endlich sein muss. Daher kann auch bei den Erweiterungsproblemen, die im Zusammenhang mit den beschriebenen Algorithmen auftauchen, davon ausgegangen werden, dass der Gruppenindex $(V : U)$ endlich ist. In [22] wird eine allgemeine Methode beschrieben, die es erlaubt, die Anzahl der Erzeuger zu reduzieren. Abhängig von den Eigenschaften der untersuchten Gruppe, existiert eine ganze Reihe weiterer Verfahren, deren Beschreibung an dieser Stelle jedoch den Rahmen sprengen würde.

Ist die Gruppe B eine Permutationsgruppe und sollen U und V in Form von Labelled Branchings nach Jerrum [7] gespeichert werden, so kann der Algorithmus von Cooperman, Finkelstein und Purdom [1] zur Berechnung eines starken Erzeugendensystems verwendet werden. Dies ermöglicht es gleichzeitig, die Anzahl der Erzeuger unter Kontrolle zu halten.

◆ *Referenz zu Cube einfügen? Sift?*

7.6.5 FindOrbitRep

Im Unterprogramm *FindOrbitRep* soll zu einer Nebenklasse Ag und gegebener Gruppe B der kleinste Repräsentant der Bahn Ag^B gefunden werden. Ist die Bahn Ag^B sehr lang, so ist diese Aufgabe schwer zu lösen und es müssen Kanonisierungsalgorithmen wie der in Abschnitt 7.6.3 beschriebene eingesetzt werden. Ist die Bahn Ag^B hingegen kurz, so kann wie in Abschnitt 7.6.2 der Bahnenalgorithmus zur Lösung eingesetzt werden. Der Bahnenalgorithmus konstruiert dann alle Nebenklassen, die in der Bahn Ag^B liegen.

Unter diesen Nebenklassen kann dann unter Verwendung eines Kanonizitätsprädikates die kanonische identifiziert werden. Ist auf der Menge der Nebenklassen A_g^B eine Totalordnung gegeben, so kann zum Beispiel die kleinste Nebenklasse dieser Bahn als die kanonische ausgezeichnet werden.

7.6.6 FindSmallestPath

Es sei G eine Gruppe und (A_1, \dots, A_n) eine starke Untergruppenleiter von G nach A_n . Die Gruppenindizes zwischen je zwei aufeinander folgenden Gruppen der Untergruppenleiter seien klein und insbesondere auch endlich. Es sei ein

$k \leq n$ und eine Nebenklasse $A_k q$ mit $q \in G$ gegeben. In der in Abschnitt 7.3.2 beschriebenen Vorbereitungsphase des Leiterspielalgorithmus aus Kapitel 7 soll der kleinste Pfad $(A_1 p, \dots, A_k p) \in P_k$ berechnet werden, dessen letzte Komponente $A_k p$ gleich der gegebenen Nebenklasse $A_k q$ ist. Die dabei zugrunde gelegte Ordnung ist die in Kapitel 6.2 beschriebene Ordnung auf der Menge P_k , der Pfade der Länge $k - 1$.

Jeder Pfad aus der Menge P_k , dessen letzte Komponente gleich $A_k q$ ist, liegt im Block $\Delta_{\{k\}}^q$. Da $A_1 = G$ ist und die erste Komponente aller Pfade aus der Menge P_k daher gleich G ist, ist der Block $\Delta_{\{1,k\}}^q$ identisch mit dem Block $\Delta_{\{k\}}^q$. Dieser Block kann in kleinere Blöcke zerlegt werden, so dass jeweils alle Pfade, deren zweite Komponenten identisch sind, in dem jeweils selben Block liegen.

Nach Lemma 16 ist die Urbildmenge des Blockes $\Delta_{\{1,k\}}^q$ unter der Abbildung $\varphi_{(\{1,2,k\},\{1,k\})}$ gleich $\{\Delta_{\{1,2,k\}}^{aq} \mid a \in A_k\}$. Die Ordnung auf der Menge der Blöcke wurde so definiert, dass der kleinste Block der Menge $\{\Delta_{\{1,2,k\}}^{aq} \mid a \in A_k\}$ genau derjenige ist, der den kleinsten Pfad enthält. Nach Lemma 18 bilden die Blöcke im Urbild von $\Delta_{\{1,k\}}^q$ unter dem G -Homomorphismus $\varphi_{(\{1,2,k\},\{1,k\})}$ eine Partition dieses Blockes. Daher ist der gesuchte kleinste Pfad $(A_1 p, \dots, A_k p)$ des Blockes $\Delta_{\{1,k\}}^q$ im kleinsten Block des Urbilds enthalten.

Der kleinste Block aus der Menge $\{\Delta_{\{1,2,k\}}^{aq} \mid a \in A_k\}$ ist nach Lemma 6 derjenige Block, dessen Pfade die kleinste zweite Komponente besitzen. Die Menge $\{A_2 a q \mid a \in A_k\}$ enthält alle Nebenklassen, die als zweiten Komponenten in einem Pfad eines Blockes aus der Menge $\{\Delta_{\{1,2,k\}}^{aq} \mid a \in A_k\}$ vorkommen. Wenn der Gruppenindex $(A_1 : A_2)$ klein ist, kann die Menge $\{\Delta_{\{1,2,k\}}^{aq} \mid a \in A_k\}$ einfach durchlaufen werden, um den kleinsten darin enthaltenen Block zu finden. Da der Pfad $(A_1 p, \dots, A_k p)$ in diesem Block enthalten sein muss, kann auf diese Weise auch die zweite Komponente des Pfades $(A_1 p, \dots, A_k p)$ bestimmt werden.

Im Folgenden wird ein iteratives Verfahren beschrieben, mit dem die übrigen Komponenten des Pfades $(A_1 p, \dots, A_k p)$ in aufsteigender Reihenfolge bestimmt werden können. Bei der Bestimmung der jeweils nächsten Kompo-

nente werden zwei Fälle unterschieden, je nachdem ob die darauf folgende Untergruppe der Leiter eine Untergruppe der Vorherigen ist oder nicht. Bei der Bestimmung der $i + 1$ -ten Komponente wird vorausgesetzt, dass im vorangehenden Iterationsschritt ein Element $g \in G$ bestimmt wurde, so dass $(A_1p, \dots, A_kp) \in \Delta_{\{i,k\}}^g$ gilt. Bezeichnet $\Delta_{\{1,2,k\}}^{aq}$ den Block, der im ersten Schritt als kleinster Block im Urbild bestimmt wurde, so setze für den folgenden Iterationsschritt $g = aq$.

Es sei $A_i \geq A_{i+1}$ und die i -te Komponente des Pfades (A_1p, \dots, A_kp) sei bereits bekannt. Es bezeichne $g \in G$ ein Element, für das $(A_1p, \dots, A_kp) \in \Delta_{\{i,k\}}^g$ gilt. Unter Verwendung von Lemma 16 kann die Urbildmenge des Blockes $\Delta_{\{i,k\}}^g$ unter dem G -Homomorphismus $\varphi_{(\{i,i+1,k\}, \{i,k\})}$ berechnet werden. Dies ist die Menge $\{\Delta_{\{i,i+1,k\}}^{ag} \mid a \in A_i \cap A_k\}$. Die Gruppe $E_i = A_i \cap A_k$ braucht dabei für jede Untergruppenleiter nur ein einziges Mal bestimmt und anschließend gespeichert werden. Die Berechnung der Gruppe E_i wurde bereits in Kapitel Abschnitt 7.6.1 beschrieben.

Die Ordnung auf der Menge der Blöcke wurde so definiert, dass der gesuchte kleinste Pfad im kleinsten Block der Urbildmenge $\{\Delta_{\{i,i+1,k\}}^{ag} \mid a \in E_i\}$ enthalten ist. Der kleinste Block der Menge $\{\Delta_{\{i,i+1,k\}}^{ag} \mid a \in E_i\}$ ist nach Lemma 6 derjenige Block, dessen Pfade die kleinste $i + 1$ -te Komponente besitzen. Ist der Gruppenindex $(A_i : A_{i+1})$ klein, so ist auch die Menge $\{A_{i+1}ag \mid a \in E_i\}$ klein und die Nebenklassen aus dieser Menge können leicht durchlaufen werden. Dabei kann die kleinste Nebenklasse $A_{i+1}ag$ aus dieser Menge ermittelt werden. Die $i + 1$ -te Komponente des Pfades (A_1p, \dots, A_kp) muss dann gleich $A_{i+1}ag$ sein und der gesuchte Pfad liegt im Block $\Delta_{\{i+1,k\}}^{ag}$. Im folgenden Iterationsschritt wird vorausgesetzt, dass ein Element $h \in G$ bestimmt wurde, für das $(A_1p, \dots, A_kp) \in \Delta_{\{i+1,k\}}^h$ gilt. Bezeichnet $\Delta_{\{i+1,k\}}^{aq}$ den Block, der den Pfad (A_1p, \dots, A_kp) enthält, so setze für den folgenden Iterationsschritt $h = aq$.

Ist $A_i \leq A_{i+1}$ und ist die i -te Komponente des Pfades (A_1p, \dots, A_kp) bereits bekannt, so kann die darauf folgende Pfadkomponente leicht bestimmt werden. Es bezeichne $g \in G$ wieder ein Element, für das $(A_1p, \dots, A_kp) \in \Delta_{\{i,k\}}^g$ gilt.

Aufgrund der Definition der Erweiterungsfunktion Υ_i in Kapitel 6.1 muss die $i+1$ -te Komponente des Pfades gleich $A_{i+1}g$ sein. Denn das Bild der i -ten Komponente $A_i g$ unter der Erweiterungsfunktion Υ_i ist die einelementige Menge $\{A_{i+1}g\}$. Da die Definition eines Pfades vorschreibt, dass die $i+1$ -te Komponente in dieser einelementigen Menge enthalten sein muss, kann die $i+1$ -te Komponente sofort festgelegt werden. Für den darauf folgenden Iterationsschritt wird wieder ein Element $h \in G$ benötigt, für das $(A_1 p, \dots, A_k p) \in \Delta_{\{i+1, k\}}^h$ gilt. Für das Element g ist diese Voraussetzung bereits erfüllt.

Auf diese Weise kann der gesuchte Pfad innerhalb von $k-2$ Schritten gefunden werden. Sind die zuvor angesprochenen Gruppenindizes der Leitergruppen klein, so kann jeder der einzelnen Schritte mit geringem Aufwand durchgeführt werden.

Kapitel 8

Analyse des Laufzeitverhaltens

In diesem Kapitel werden die Eigenschaften der drei in dieser Arbeit neu vorgestellten Leiterspielalgorithmen anhand von konkreten Rechenbeispielen untersucht.

In Kapitel 8.2 wird jeder der drei Algorithmen auf ein konkretes Kanonisierungsproblems angesetzt. Daraufhin werden die Algorithmen in Bezug auf ihre Laufzeit, ihren Bedarf an Hauptspeicher und die Anzahl der berechneten Nebenklassen zur Lösung des Kanonisierungsproblems verglichen.

In Kapitel 8.3 wird der Leiterspiel-Light Algorithmus mit dem ursprünglichen Leiterspielalgorithmus von Schmalz verglichen. Beide Algorithmen werden unabhängig voneinander zur Konstruktion aller schlichten Graphen auf bis zu 9 Knoten eingesetzt. Dabei werden die Algorithmen in Bezug auf ihre Laufzeit und ihren Bedarf an Hauptspeicher miteinander verglichen.

8.1 Kurzbeschreibung der Algorithmen

8.1.1 Leiterspielalgorithmus von Schmalz

Der ursprüngliche Leiterspielalgorithmus von Schmalz wurde in Kapitel 4 vorgestellt. Dieser Algorithmus zeichnet sich insbesondere dadurch aus, dass mit diesem Algorithmus auch schwierigste Isomorphieprobleme gelöst werden können.

Mit dem Leiterspielalgorithmus von Schmalz kann zu gegebenen Gruppen G , A und B , mit $A \leq G$ und $B \leq G$, die Menge aller Doppelnebenklassen

$A \backslash G / B = \{AgB \mid g \in G\}$ bestimmt werden. Genauer gesagt wird zu jeder Doppelnebenklasse AgB eine Rechtsnebenklasse Agb mit $b \in B$ bestimmt, die die entsprechende Doppelnebenklasse repräsentiert. Der Repräsentant einer Doppelnebenklasse AgB kann jedoch nicht einzeln bestimmt werden, sondern nur zusammen mit allen anderen Repräsentanten von Doppelnebenklassen aus der Menge $A \backslash G / B$.

8.1.2 Leiterspiel-Light

Der Leiterspiel-Light Algorithmus wurde in Kapitel 7.4 beschrieben. Es seien G , A und B drei Gruppen, mit $A \leq G$ und $B \leq G$ und g ein Element aus der Gruppe G . Dieser Algorithmus ermöglicht es, zu jeder Doppelnebenklasse AgB ein Element $b \in B$ und eine Rechtsnebenklasse Agb zu bestimmen, die diese Doppelnebenklasse repräsentiert.

Um diesen Algorithmus einsetzen zu können, müssen mehrere Voraussetzungen erfüllt sein. Es wird eine Untergruppenleiter (A_1, \dots, A_n) von G nach A benötigt, die nach den in Kapitel 6.5 beschriebenen Kriterien eine starke Untergruppenleiter ist. Weiterhin wird für alle $i \leq n$ eine Totalordnung auf der Menge $A_i \backslash G$ der Rechtsnebenklassen der Gruppe A_i benötigt, die den in Abschnitt 6.1.2 genannten Kriterien genügt.

In vielen Fällen ist der Leiterspiel-Light Algorithmus ein schneller, einfach zu implementierender Algorithmus, der zur Lösung von kleineren Doppelnebenklassen Isomorphieproblemen verwendet werden kann. Dieser Algorithmus eignet sich in Kombination mit dem im Anschluss beschriebenen Depthfirst StroLL Algorithmus auch zur Lösung von großen Doppelnebenklassen Isomorphieproblemen.

8.1.3 Depthfirst StroLL Algorithmus

Der Depthfirst StroLL Algorithmus wurde in Abschnitt 7.3.2 bis 7.3.5 beschrieben. Genau wie der Leiterspiel-Light Algorithmus ermöglicht es dieser Algorithmus, zu jeder Doppelnebenklasse AgB ein Element $b \in B$ und eine Rechtsnebenklasse Agb zu bestimmen, die diese Doppelnebenklasse repräsentiert.

Der Depthfirst StroLL benötigt dieselben Voraussetzungen wie das Leiterspiel

Light, eignet sich im Gegensatz zu diesem jedoch auch zur Lösung von großen Doppelnebenklassen Isomorphieproblemen. Dieser Algorithmus ermöglicht es, große Isomorphieprobleme in mehrere kleinere zu zerlegen, die dann anschließend rekursiv gelöst werden können. In der Praxis hat sich herausgestellt, dass das Leiterspiel Light wesentlich effizienter ist bei der Lösung von kleineren Isomorphieproblemen. Daher empfiehlt es sich, kleinere Isomorphieprobleme, die beim Depthfirst StroLL in den Rekursionsschritten auftauchen, immer mit Hilfe des Leiterspiel Light Algorithmus zu lösen.

8.1.4 Breadthfirst StroLL Leiterspiel

Der Breadthfirst StroLL Algorithmus wurde in Kapitel 7.2 beschrieben. Dieser Algorithmus ermöglicht es, genau wie der Depthfirst StroLL und das Leiterspiel Light, zu einer einzelnen gegebenen Doppelnebenklasse eine repräsentierende Rechtsnebenklasse zu berechnen.

Die Voraussetzungen für diesen Algorithmus sind identisch zu denen des Depthfirst StroLL und des Leiterspiel Light Algorithmus. Im Gegensatz zu diesen Algorithmen benötigt der Breadthfirst StroLL Algorithmus jedoch deutlich mehr Arbeitsspeicher. Unter den drei Algorithmen Leiterspiel-Light, Depthfirst StroLL und Breadthfirst StroLL ist dieser Algorithmus dem ursprünglichen Algorithmus von Schmalz am ähnlichsten.

8.2 Vergleich der Eigenschaften

In diesem Kapitel werden die drei neuen Algorithmen dieser Arbeit miteinander verglichen. Anhand eines Beispiels wird gezeigt, wie unterschiedlich die drei Algorithmen in Bezug auf ihre Laufzeit, ihren Bedarf an Arbeitsspeicher und die Anzahl der durchlaufenen Nebenklassen sind. Dieser Vergleich dient jedoch ausdrücklich nicht dazu, die Algorithmen hinsichtlich ihrer Leistungsfähigkeit zu vergleichen. Die Leistungsfähigkeit des neuen Ansatzes wird in Kapitel 8.3 unter Beweis gestellt.

Der Depthfirst StroLL arbeitet rekursiv, große Problemstellungen werden in kleinere Probleme zerlegt, die anschließend wieder mit einem Kanonisierungsalgorithmus gelöst werden müssen. Wird der Depthfirst StroLL auch wieder zur Lösung dieser kleineren Probleme eingesetzt, so muss insgesamt eine sehr

große Zahl sehr kleiner Kanonizitätsprobleme mit diesem Algorithmus gelöst werden. Da jedoch kleine Isomorphieprobleme mit dem Depthfirst StroLL nur sehr langsam gelöst werden können, führt dies zu einem insgesamt langsamen Algorithmus.

In diesem Kapitel sollen jedoch die Unterschiede der drei Algorithmen im Vordergrund stehen, daher wurde darauf verzichtet, die verschiedenen Algorithmen miteinander zu kombinieren. In der Praxis hingegen sollten beim Depthfirst StroLL die kleinen Isomorphieprobleme aus den Rekursionsschritten immer mit einem anderen Algorithmus wie zum Beispiel dem Leiterspiel-Light gelöst werden.

8.2.1 Der Beispielgraph

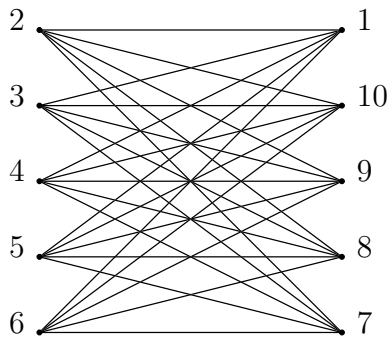


Abbildung 8.1: Graph aus 10 Knoten und 25 Kanten

Um die drei Algorithmen vergleichen zu können, wird jedem der Drei die Aufgabe gestellt, den Graphen aus Abbildung 8.1 zu kanonisieren. Nach dem Split Lemma aus Kapitel 3.5 kann der Graph aus Abbildung 8.1 auf eine Rechtsnebenklasse abgebildet werden. Dieser Schritt wurde bereits in dem Beispiel auf Seite 21 beschrieben. Dort wurde auch beschrieben, wie die Gruppen G , A und B zur Doppelnebenklassenmenge $A \backslash G / B$, der Bildmenge der Split-Lemma Abbildung, bestimmt werden.

Wurde der Graph unter der Abbildung des Split Lemmas auf eine Nebenklasse Ag abgebildet, so kann mit jedem der drei Algorithmen überprüft werden, ob Ag die kanonische Nebenklasse in ihrer Bahn $Ag^B = \{Agb | b \in B\}$ ist. Bei jedem der drei Testfälle wird das selbe Kanonizitätsprädikat zugrunde gelegt,

das in Abschnitt 8.2.2 erläutert wird.

In einem weiteren Schritt wird dann das Urbild der kanonischen Nebenklasse unter der Abbildung des G -Isomorphismus des Split Lemmas berechnet. Das Kanonizitätsprädikat auf der Menge der Graphen kann so gewählt werden, dass das Urbild jeder kanonischen Nebenklasse auch der kanonische Repräsentant der Bahn des betreffenden Graphen ist. Auf diese Weise kann mit jedem der drei Algorithmen die Kanonizität des Graphen in Abbildung 8.1 überprüft werden.

8.2.2 Voraussetzungen

Der vollständige Graph auf 10 Knoten besitzt genau 45 Kanten. Um den Graphen aus Abbildung 8.1 auf eine Doppelnebenklasse abzubilden, wird die Gruppe G gleich der Symmetrischen Gruppe S_{45} gewählt, entsprechend dem Beispiel auf Seite 21.

Bei allen drei Algorithmen wurde folgende starke Untergruppenleiter zugrunde gelegt:

$$\begin{aligned} A_1 &= G \\ A_{2k} &= S_{(\{1, \dots, k\}, \{k+1, \dots, 45\})} \quad \forall 1 \leq k \leq 45 \\ A_{2k+1} &= S_{(\{1, \dots, k\}, \{k+1\}, \{k+2, \dots, 45\})} \quad \forall 1 \leq k < 45 \end{aligned}$$

Auf der Menge der Gruppenelemente von G ist durch die lexikographische Ordnung eine Totalordnung gegeben. Entsprechend dieser Ordnung kann für alle Untergruppe U von G eine Totalordnung auf der Menge $U \backslash G$ festgelegt werden: Zu je zwei gegebenen Elementen $g_1, g_2 \in G$ soll $Ug_1 \preceq Ug_2$ gelten, genau dann, wenn das kleinste Element g'_1 aus der Menge $\{ug_1 \mid u \in U\}$ kleiner ist als das kleinste Element der Menge $\{ug_2 \mid u \in U\}$. Das Kanonizitätsprädikat wurde so gewählt, dass für alle Elemente $g \in G$ und Untergruppen U und B von G genau die nach dieser Ordnung kleinste Nebenklasse in der Bahn $Ug^B = \{Ugb \mid b \in B\}$ als kanonisch gelten soll.

Sämtliche Berechnungen wurden auf einem einzelnen Prozessor des Linux-Clusters der Universität Bayreuth durchgeführt. Die verwendete CPU ist ein Intel Xeon Prozessor vom Typ E5520 ausgestattet mit 24 Gigabyte 1066 MHz DDR3 RAM Modulen.

8.2.3 Ergebnisse

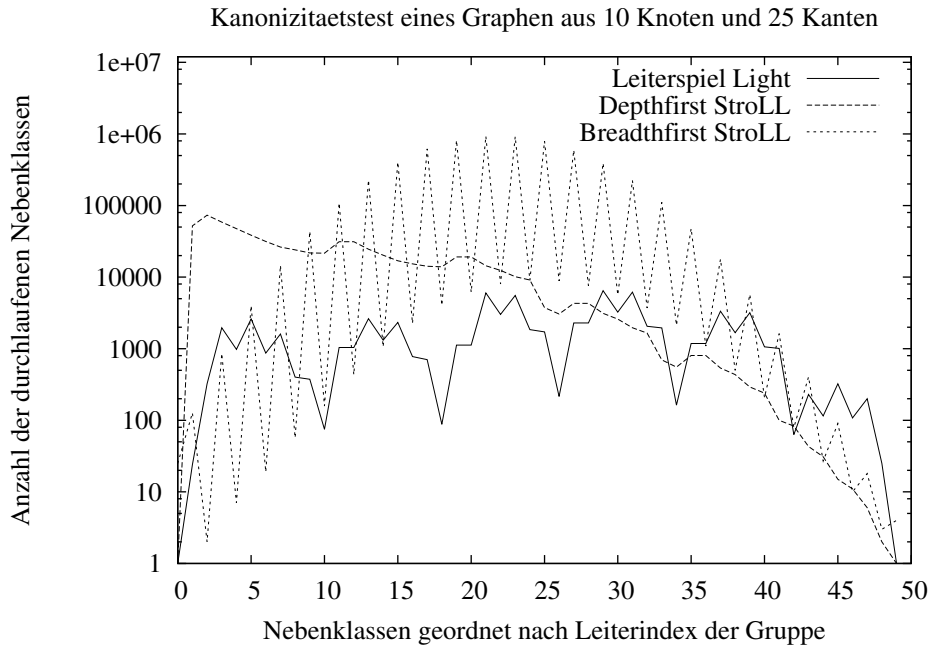


Abbildung 8.2: Anzahl aller durchlaufenen Nebenklassen

In Abbildung 8.2 ist für jeden der drei Algorithmen die Anzahl der durchlaufenen Rechtsnebenklassen dargestellt. Es bezeichne (A_1, \dots, A_{90}) die starke Untergruppenleiter, die bei den Berechnungen der Algorithmen verwendet wird. Zu jeder der durchlaufenen Rechtsnebenklassen existiert ein $g \in G$ und eine Gruppe E , so dass die Rechtsnebenklasse als Eg geschrieben werden kann. Zu dieser Gruppe E existieren wiederum zwei Indizes $i, k \in \{1, \dots, 50\}$ mit $i < k$, so dass $E = A_i \cap A_k$ ist. Für jeden der Algorithmen wurden die während des Kanonizitätstests durchlaufenen Nebenklassen gezählt und entsprechend dem kleineren der beiden Indizes aufgeschlüsselt. Die Anzahl der durchlaufenen Nebenklassen wurde dann entsprechend diesem Index auf der x-Achse in Abbildung 8.2 dargestellt. Auf diese Weise entstehen drei sehr unterschiedliche Nebenklassen Verteilungen, die wiederum Aufschluss über die Arbeitsweise des jeweiligen Algorithmus geben.

Beim Leiterspiel-Light Algorithmus wird zuerst ein Pfad $\rho = (A_1p, \dots, A_{50}p)$ berechnet, der kleinste Pfad zu der gegebenen Nebenklasse $A_{50}p$, für die überprüft werden soll, ob diese kanonisch ist. Für jeden Index $i \leq 50$ entspricht

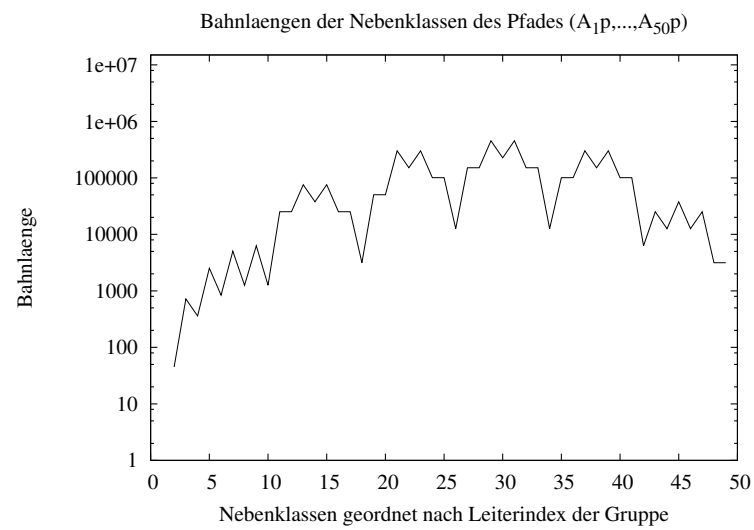


Abbildung 8.3: Bahnlängen aller Nebenklassen A_{ip} unter der Gruppe B

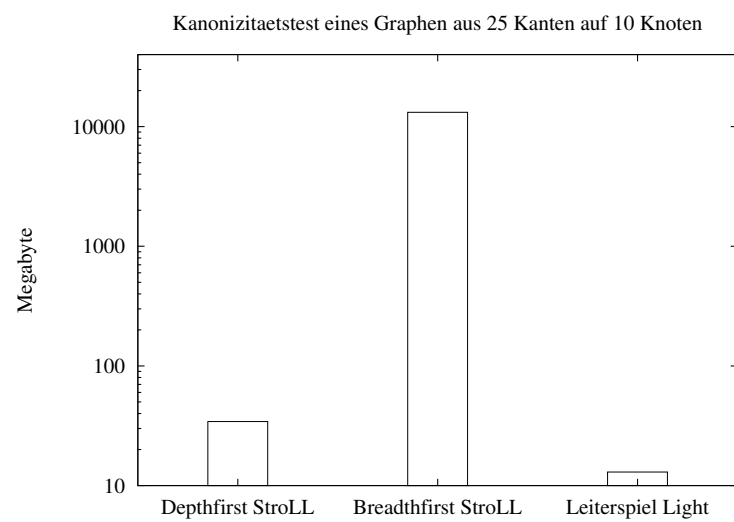


Abbildung 8.4: Bedarf an Arbeitsspeicher

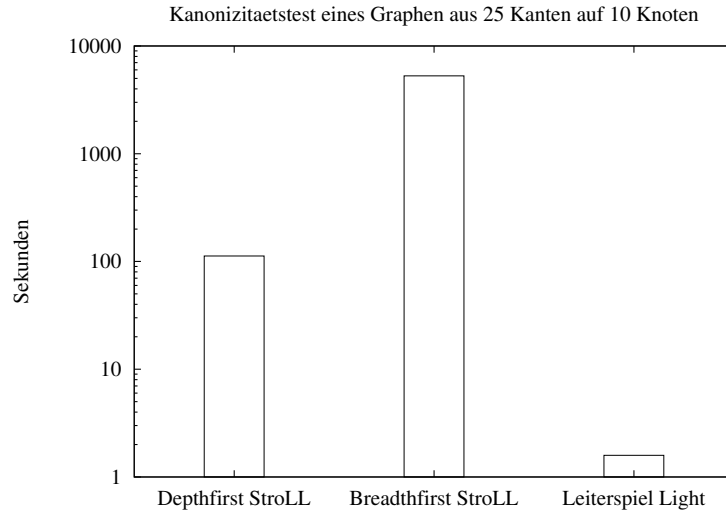


Abbildung 8.5: Bedarf an Rechenzeit

die Anzahl der durchlaufenen Nebenklassen, bis auf eine geringe, programmiertechnisch bedingte Abweichung, genau der Größe der Schnittmenge der beiden Mengen $\{A_i p a \mid a \in A_{50}\}$ und $\{A_i p b \mid b \in B\}$. Die Mächtigkeit der Menge $\{A_i p b \mid b \in B\}$ ist daher entscheidend für die Anzahl der zu durchlaufenden Nebenklassen. Diese kann wie folgt berechnet werden:

$$|\{A_i p b \mid b \in B\}| = \frac{|B|}{|B \cap p^{-1} A_i p|}$$

Der Nenner im Bruch auf der rechten Seite ist identisch mit der Größe des Stabilisators der Nebenklasse $A_i p$ in B . Daher übt die Größe dieses Stabilisators einen entscheidenden Einfluss auf die Anzahl der zu durchlaufenden Nebenklassen aus. In Abbildung 8.3 ist für alle Indizes i von 1 bis 50 die Mächtigkeit der Menge $\{A_i p b \mid b \in B\}$ dargestellt. Die Korrelation zwischen der Abbildung 8.3 und der Nebenklassenverteilung des Leiterspiel-Light Algorithmus in Abbildung 8.2 ist leicht zu erkennen.

Die rekursive Lösungsstrategie des Depthfirst StroLL führt dazu, dass sehr viele Nebenklassen mit kleinem Leiterindex durchlaufen werden. Im Vergleich zu den beiden anderen Algorithmen werden die Nebenklassen mit großem Leiterindex etwas seltener durchlaufen. In der Verteilung der Nebenklassen des Depthfirst StroLL in Abbildung 8.2 ist auch ein geringer Einfluss der Bahnlängen aus Abbildung 8.3 erkennbar. Dieser ist jedoch viel kleiner als beim

Leiterspiel-Light Algorithmus.

Die Sägezähne, die in der Nebenklassenverteilung des Breadthfirst StroLL in Abbildung 8.2 zu sehen sind, können auf die unterschiedlichen Größen der Gruppen der Leiter zurückgeführt werden. Da sich bei der gegebenen Leiter die Splitting- und Fusingschritte abwechseln, entsteht dieses Sägezahnmuster, das auch in den Darstellungen der beiden anderen Algorithmen zu erkennen ist.

8.3 Vergleich der Leistungsfähigkeit

In diesem Kapitel werden die Eigenschaften des Leiterspiel Algorithmus von Schmalz mit dem Leiterspiel-Light Algorithmus verglichen. Die beiden Algorithmen werden hinsichtlich ihres Bedarfs an Rechenzeit und ihres Bedarfs an Arbeitsspeicher untersucht.

Bezeichnet G eine Gruppe und A und B zwei Untergruppen von G , so kann der Algorithmus von Schmalz zur Konstruktion aller kanonischen Repräsentanten von Doppelnebenklassen $A \backslash G / B$ eingesetzt werden. Es ist mit diesem Algorithmus jedoch nicht möglich, den kanonischen Repräsentanten einer einzelnen Doppelnebenklasse zu bestimmen, ohne die übrigen kanonischen Repräsentanten zu berechnen.

Der Leiterspiel-Light Algorithmus hingegen ist ausschließlich dazu in der Lage, zu einer einzelnen Doppelnebenklasse den kanonischen Repräsentanten zu bestimmen. Dieser Algorithmus kann jedoch in Kombination mit der ordnungstreuen Erzeugung zur Berechnung der kanonischen Repräsentanten aller Doppelnebenklassen $A \backslash G / B$ eingesetzt werden.

Für den Vergleich wurde daher ein Rechenbeispiel gewählt, bei dem mit dem Algorithmus von Schmalz alle kanonischen Repräsentanten von Doppelnebenklassen $A \backslash G / B$ berechnet werden und das Leiterspiel-Light in Kombination mit der ordnungstreuen Erzeugung zur Konstruktion der selben kanonischen Repräsentanten eingesetzt wird.

Nach dem Split Lemma aus Kapitel 3.5 kann die Aufgabe, alle schlichten Graphen auf n Knoten bis auf Isomorphie zu konstruieren, auf ein Doppelnebenklassenproblem abgebildet werden. In dem Beispiel zum Split Lemma auf Seite 21 wurde bereits beschrieben, wie ein Graph auf eine Nebenklasse

abgebildet werden kann. Dort wurde auch beschrieben, wie die Gruppen G , A und B zur Doppelnebenklassenmenge $A \backslash G / B$, der Bildmenge der Split-Lemma Abbildung, bestimmt werden. Jede Doppelnebenklasse entspricht einem Graphenisomorphietypen, daher muss zur Konstruktion aller schlichten Graphen auf n Knoten außer der Berechnung der Doppelnebenklassen nahezu kein zusätzlicher Aufwand betrieben werden. Daher eignet sich dieses Beispiel besonders gut für einen Vergleich der unterschiedlichen Algorithmen.

An dieser Stelle soll betont werden, dass die im Beispiel konstruierten Graphenmengen längst bekannt sind und dieses Problem nur aufgrund seiner Anschaulichkeit ausgewählt wurde.

8.3.1 Voraussetzungen

Zum Vergleich der beiden Algorithmen wurden alle schlichten Graphen auf 4, 5, 6, 7, 8 und 9 Knoten generiert. Der vollständige Graph auf n Knoten besitzt genau $m = n \cdot (n - 1) / 2$ Kanten. Zur Konstruktion aller Graphen auf n Knoten wurde, entsprechend dem Beispiel auf Seite 21, die Gruppe G gleich der Symmetrischen Gruppe S_m gewählt.

Für die Konstruktion der Graphen auf n Knoten wurde folgende starke Untergruppenleiter verwendet:

$$\begin{aligned} A_1 &= G \\ A_{2k} &= S_{(\{1, \dots, k\}, \{k+1, \dots, m\})} \quad \forall 1 \leq k \leq m \\ A_{2k+1} &= S_{(\{1, \dots, k\}, \{k+1\}, \{k+2, \dots, m\})} \quad \forall 1 \leq k < m \end{aligned}$$

Den Berechnungen wurde das selbe Kanonizitätsprädikat zugrunde gelegt, das bereits in Abschnitt 8.2.2 beschrieben wurde. Sämtliche Berechnungen wurden auf einem einzelnen Prozessor des Linux-Clusters der Universität Bayreuth durchgeführt. Die verwendete CPU ist ein Intel Xeon Prozessor vom Typ E5520 ausgestattet mit 24 Gigabyte 1066 MHz DDR3 RAM Modulen.

8.3.2 Ergebnisse

Wie in Abbildung 8.6 zu sehen ist, benötigen der Algorithmus von Schmalz und das Leiterspiel-Light nahezu die selbe Rechenzeit zur Berechnung aller schlichten Graphen auf bis zu neun Knoten. In Abbildung 8.7 ist die Laufzeit jedes der beiden Algorithmen geteilt durch die Anzahl der konstruierten

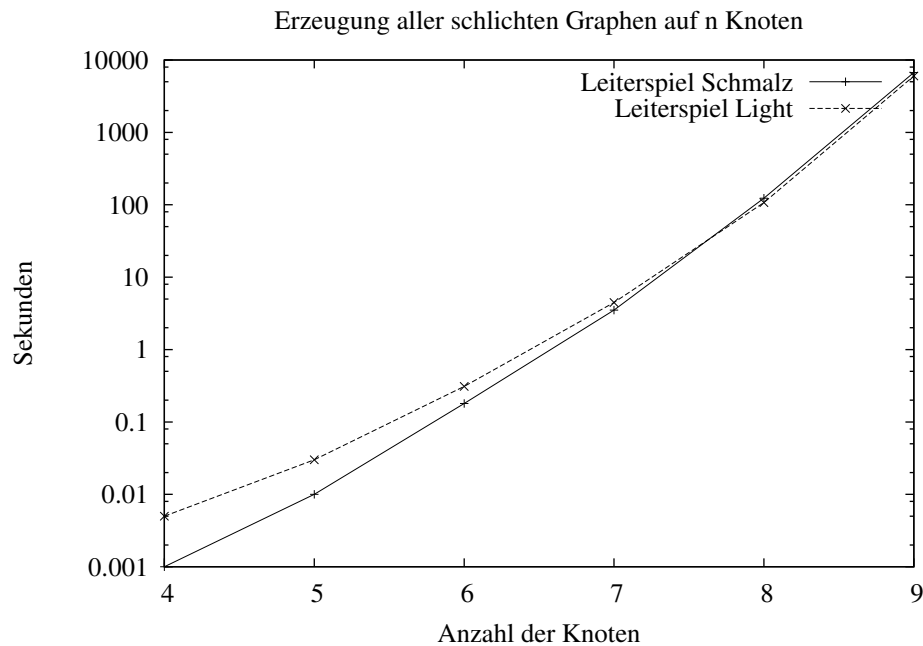


Abbildung 8.6: Bedarf an Rechenzeit

Graphen dargestellt. Dies ermöglicht es, die Unterschiede in der Laufzeit der beiden Algorithmen noch etwas genauer darzustellen.

Der Vergleich des Leiterspiels von Schmalz mit dem Leiterspiel-Light zeigt insbesondere, dass das Leiterspiel von Schmalz einen um mehrere Größenordnungen größeren Bedarf an Arbeitsspeicher hat. Die Graphen auf 10 Knoten konnten mit dem Leiterspiel von Schmalz nicht mehr berechnet werden, da schon bei der Konstruktion aller einfachen Graphen auf 9 Knoten 34 Gigabyte Speicher benötigt werden. Wie unterschiedlich der Speicherbedarf der beiden Algorithmen ist, kann sehr deutlich anhand der Abbildung 8.8 gezeigt werden. Dabei ist insbesondere zu beachten, dass die y-Achse der Graphik in Abbildung 8.8 eine logarithmische Skalierung besitzt.

Der Leiterspiel-Light Algorithmus konstruiert die gesuchten Graphen in Tiefsuche unter Verwendung der ordnungstreuen Erzeugung. Daher müssen immer nur sehr wenige kanonische Nebenklassen und Stabilisatoren gleichzeitig im Speicher gehalten werden. Beim Leiterspiel von Schmalz hingegen werden alle kanonischen Nebenklassen zusammen mit ihren Stabilisatoren gleichzeitig im Speicher gehalten. Aufgrund der hohen Anzahl von Nebenklassen, die zur

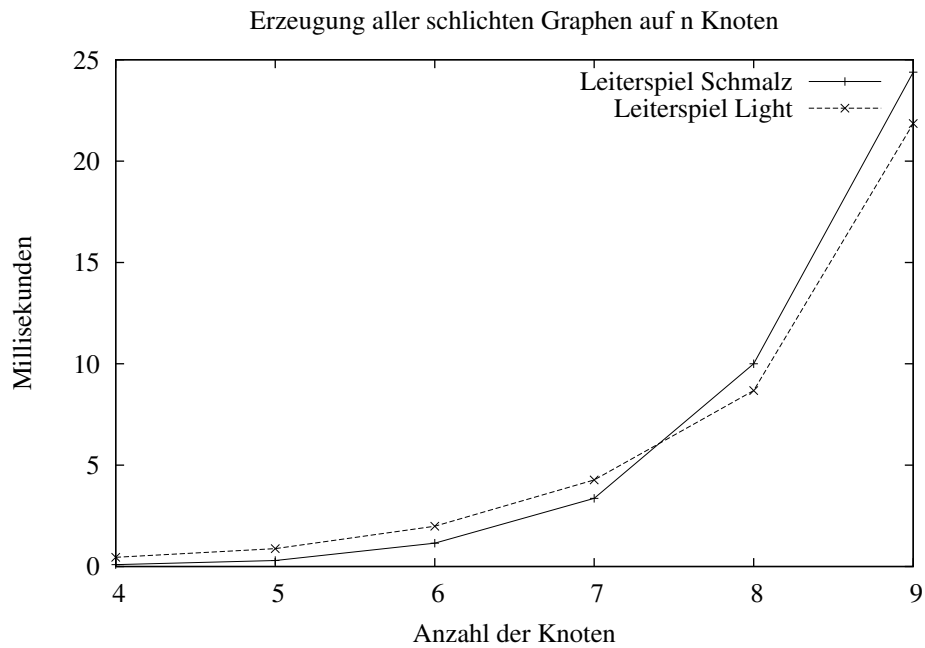


Abbildung 8.7: Bedarf an Rechenzeit pro Graph

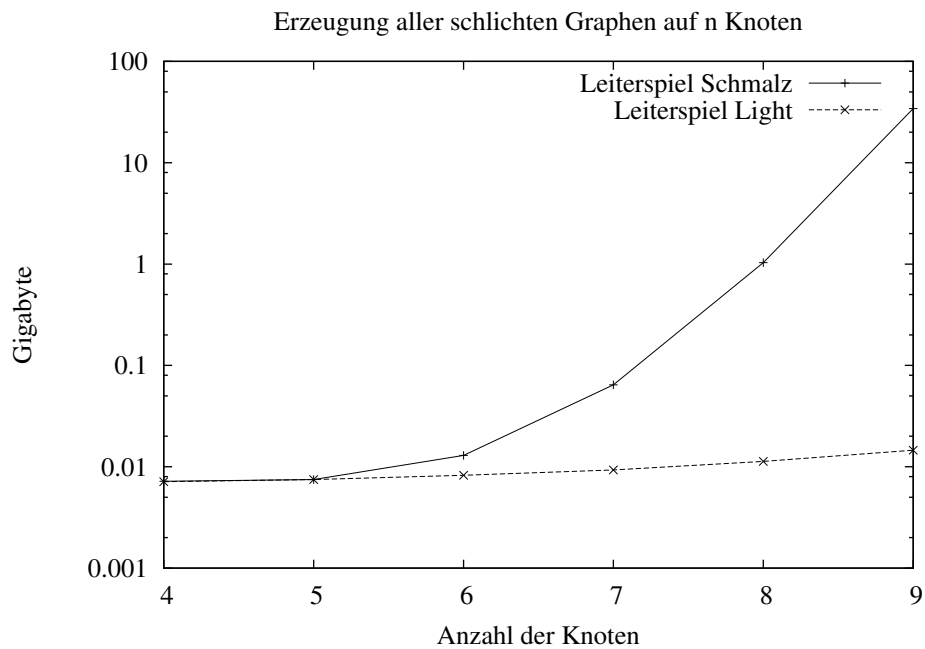


Abbildung 8.8: Bedarf an Arbeitsspeicher

Konstruktion der Graphen benötigt werden, ergibt sich dieser enorme Speicherbedarf. Wird beim Algorithmus von Schmalz der Speicherbedarf durch die Anzahl der konstruierten Nebenklassen geteilt, so ergibt sich das Verhältnis, das in Abbildung 8.9 dargestellt ist.

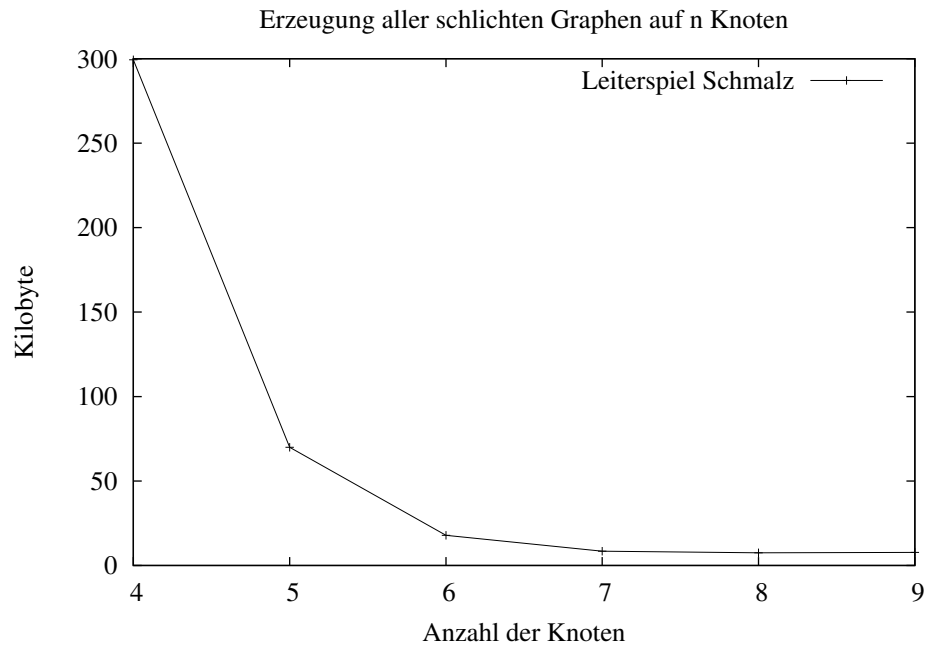


Abbildung 8.9: Durchschnittlicher Speicherbedarf pro Nebenkasse

#Knoten	CPU-Zeit	#Nebenklassen	#Graphen	Hauptspeicher
4	0.00	24	11	7.184 MByte
5	0.01	107	34	7.488 MByte
6	0.18	727	156	12.96 MByte
7	3.51	7607	1044	64.33 MByte
8	123.51	137465	12346	1.032 GByte
9	6698.10	4422499	274668	34.18 GByte

Abbildung 8.10: Leiterspiel nach Schmalz, Ergebnisse tabellarisch

8.3.3 Ergebnis des Vergleichs

Die Einsatzmöglichkeiten des ursprünglichen Leiterspiel Algorithmus werden vor allem durch die Größe des zur Verfügung stehenden Hauptspeichers be-

#Knoten	CPU-Zeit	#Nebenklassen	#Graphen	Hauptspeicher
4	0.00	296	11	7.120 MByte
5	0.03	2878	34	7.488 MByte
6	0.31	29750	156	8.240 MByte
7	4.46	365344	1044	9.296 MByte
8	107.18	75000593	12346	11.296 MByte
9	6002.34	4	274668	14.560 MByte

Abbildung 8.11: Leiterspiel-Light, Ergebnisse tabellarisch

schränkt. Zu jeder Doppelnebenklasse AgB wird eine Nebenklasse Agb mit $b \in B$ bestimmt, die als kanonischer Repräsentant dieser Doppelnebenklasse fungiert. Und zu jeder dieser kanonischen Nebenklassen Agb muss das Bild dieser Nebenklasse unter der fusionierenden Abbildung und der Stabilisator $B_{Agb} = B \cap b^{-1}g^{-1}Agb$ dieser Nebenklasse unter der Operation der Gruppe B gespeichert werden. Daher wächst beim ursprünglichen Leiterspielalgorithmus mit jeder konstruierten Doppelnebenklasse der Bedarf an Hauptspeicher.

Werden die Doppelnebenklassen hingegen mit dem Leiterspiel-Light Algorithmus in ordnungstreuer Erzeugung konstruiert, so wird dazu nur eine sehr geringe Menge an Hauptspeicher benötigt. Bei der Konstruktion von schlichten Graphen bis 9 Knoten ist der Leiterspiel-Light Algorithmus genauso so schnell wie der ursprüngliche Leiterspiel Algorithmus von Schmalz.

Kapitel 9

Ausblick

Homomorphismen von Gruppenoperationen werden nicht nur zur Lösung von Isomorphieproblemen verwendet. Viele Beweise gruppentheoretischer Aussagen und viele Algorithmen der Gruppentheorie beruhen auf Homomorphismen von Gruppenoperationen. Untergruppenleitern können als Beschreibung einer Kette von Homomorphismen von Gruppenoperationen auf Nebenklassenmengen betrachtet werden. Die starken Untergruppenleitern, die in dieser Arbeit zum ersten Mal beschrieben wurden, sind ein neues Werkzeug in der Gruppentheorie, das sowohl für Beweise in der Gruppentheorie als auch zur Beschreibung von Algorithmen verwendet werden kann.

Der Unterschied zwischen den bisher verwendeten und den starken Untergruppenleitern besteht in der Transitivität der Gruppenoperation auf der Menge der Pfade. Dieses Konzept kann ohne weiteres auch auf andere Gruppenoperationen als die Rechtsmultiplikation und auch auf andere Objektmengen als Nebenklassen angewendet werden. Diese Verallgemeinerung ermöglicht weitere Abwandlungen des Leiterspielalgorithmus. Insgesamt sind die in dieser Arbeit beschriebenen Algorithmen als Prototypen zu betrachten, um die Tragfähigkeit dieser neuen Konzepte zu zeigen. Indem die beschriebenen Algorithmen miteinander kombiniert wurden, die Ordnungstreue Erzeugung auch auf den Breadthfirst StroLL Algorithmus angewendet wurde und das Konzept der starken Leitern ausgeweitet wurde, sind inzwischen weitere Algorithmen hinzugekommen, die alle auf dem Konzept der starken Leitern beruhen.

Weiterer Forschungsbedarf besteht daher darin, Varianten der Algorithmen zu implementieren und hinsichtlich ihrer Effizienz zu untersuchen. Neben der Effizienz sollte insbesondere auch die Parallelisierung dieser Algorithmen untersucht werden. In Kapitel 7.3 wurde zu allen wesentlichen Teilen des Algorithmus auch der Pseudocode angegeben. Dies ist als Hilfestellung an die Leser gedacht und soll diese dazu motivieren, die beschriebenen Algorithmen zu implementieren und anzuwenden. Anhand der Kürze des Pseudocodes soll auch verdeutlicht werden, dass eine Implementierung der Algorithmen trotz der Schwierigkeit der Thematik eine durchaus überschaubare Aufgabe bleibt.

Invarianten können als spezielle Homomorphismen von Gruppenoperationen betrachtet werden. Bei vielen der erfolgreichsten Algorithmen zur Lösung von Isomorphieproblemen werden auf geschickte Weise Invarianten berechnet, mit Hilfe derer sich die Effizienz der Algorithmen mitunter drastisch verbessern lässt. Dieser Aspekt wurde bei den Algorithmen dieser Arbeit weitgehend außer Acht gelassen um den Fokus auf das Wesentliche zu beschränken. Auch hier besteht weiterer Forschungsbedarf um herauszufinden, welche Invarianten sich zur Unterscheidung von Doppelnebenklassen eignen. In diesem Zusammenhang soll auch insbesondere auf die Möglichkeit hingewiesen werden, dynamische Leitern zur Lösung von Isomorphieproblemen einzusetzen. Mit dynamischen Leitern sind Leitern gemeint, bei denen die Eigenschaften der betrachteten Doppelnebenklasse einen Einfluss darauf haben, welcher Homomorphismus von Gruppenoperationen im darauf folgenden Leiterschritt angewendet werden soll.

Literaturverzeichnis

- [1] COOPERMAN, G. ; FINKELSTEIN, L. ; PURDOM, P. W. Jr.: Fast Group Membership Using a Strong Generating Test for Permutation Groups. In: *Proceedings of the Third Conference on Computers and Mathematics*. New York, NY, USA : Springer-Verlag New York, Inc., 1989. – ISBN 0-387-97019-3, 27–36
- [2] COOPERMAN, Gene ; FINKELSTEIN, Larry: New methods for using Cayley graphs in interconnection networks. In: *Discrete Applied Mathematics* 37?38 (1992), Nr. 0, 95 - 118. [http://dx.doi.org/http://dx.doi.org/10.1016/0166-218X\(92\)90127-V](http://dx.doi.org/http://dx.doi.org/10.1016/0166-218X(92)90127-V). – DOI [http://dx.doi.org/10.1016/0166-218X\(92\)90127-V](http://dx.doi.org/10.1016/0166-218X(92)90127-V). – ISSN 0166-218X
- [3] DAHIYAT, Bassil I. ; MAYO, Stephen L.: De novo protein design: fully automated sequence selection. In: *Science* 278 (1997), S. 82–87
- [4] FARADŽEV, I.A.: Constructive enumeration of combinatorial objects. In: *Problèmes combinatoires et théorie des graphes. Orsay 1976, Colloq. int. CNRS, No.260*. Colloques internationaux C.N.R.S, 1978, S. 131–135
- [5] GRÜNER, Thomas: *Ein neuer Ansatz zur rekursiven Erzeugung von schlichten Graphen*, Universität Bayreuth, Diplomarbeit, 1995
- [6] HALL, M.: *The Theory of Groups*. AMS Chelsea Pub., 1976 (AMS Chelsea Publishing Series). – ISBN 9780821819678
- [7] JERRUM, Mark: A compact representation for permutation groups. In: *Journal of Algorithms* 7 (1986), Nr. 1, 60 - 78. [http://dx.doi.org/http://dx.doi.org/10.1016/0196-6774\(86\)90038-6](http://dx.doi.org/http://dx.doi.org/10.1016/0196-6774(86)90038-6). – DOI [http://dx.doi.org/10.1016/0196-6774\(86\)90038-6](http://dx.doi.org/10.1016/0196-6774(86)90038-6). – ISSN 0196-6774

- [8] KIERMAIER, Michael ; KOCH, Matthias ; KURZ, Sascha: 2-arcs of maximal size in the affine and the projective Hjelmslev plane over \mathbb{Z}_{25} . In: *Advances in Mathematics of Communications* 5 (2014), Januar 17, Nr. 2, 287–301. <http://arxiv.org/abs/1401.4340>. – Comment: 20 pages, 3 tables
- [9] KOCH, M.: *Anwendung von Konstruktionsalgorithmen in der diskreten Geometrie*, Universität Bayreuth, Diplomarbeit, 2006
- [10] KURZ, S.: *Konstruktion und Eigenschaften ganzzahliger Punktmengen*, Bayreuth. Math. Schr. 76. Universität Bayreuth, Diss., 2006
- [11] LAUE, R.: Construction of combinatorial objects: A tutorial. In: *Bayreuther Math. Schr.* 43 (1993), S. 53–96
- [12] LAUE, Reinhard: *Zur Konstruktion und Klassifikation endlicher auflösbarer Gruppen*. Bayreuther Mathematische Schriften 9, 304 S., 1982
- [13] MCKAY, Brendan D.: Isomorph-free exhaustive generation. In: *J. Algorithms* 26 (1998), Nr. 2, S. 306–324. <http://dx.doi.org/10.1006/jagm.1997.0898>. – DOI 10.1006/jagm.1997.0898
- [14] MERINGER, Markus: *Erzeugung regulärer Graphen*, Universität Bayreuth, Diplomarbeit, 1996
- [15] READ, R. C.: Every one a winner or how to avoid isomorphism search when cataloguing combinatorial configurations. In: *Ann. Discrete Math.* 2 (1978), S. 107–120
- [16] ROYLE, Gordon F.: An orderly algorithm and some applications in finite geometry. In: *Discrete Mathematics* 185 (1998), Nr. 1/3, 105 - 115. [http://dx.doi.org/http://dx.doi.org/10.1016/S0012-365X\(97\)00167-2](http://dx.doi.org/http://dx.doi.org/10.1016/S0012-365X(97)00167-2). – DOI [http://dx.doi.org/10.1016/S0012-365X\(97\)00167-2](http://dx.doi.org/10.1016/S0012-365X(97)00167-2). – ISSN 0012–365X
- [17] SCHMALZ, Bernd: *Computerunterstützte Konstruktion von Doppelnebenklassenrepräsentanten mit Anwendung auf das Isomorphieproblem der Graphentheorie*, Universität Bayreuth, Diplomarbeit, 1989

- [18] SCHMALZ, Bernd: Verwendung von Untergruppenleitern zur Bestimmung von Doppelnebenklassen. In: *Bayreuther Math. Schr.* Bd. 31. Universität Bayreuth, 1990, S. 109–143
- [19] SCHMALZ, Bernd: *Bayreuther Math. Schr.* Bd. 41: *t-Designs zu vorgegebener Automorphismengruppe*. Universität Bayreuth, 1992. – Dissertation, Universität Bayreuth,
- [20] SIMS, Charles C.: Computation with Permutation Groups. In: *Proceedings of the Second ACM Symposium on Symbolic and Algebraic Manipulation*. New York, NY, USA : ACM, 1971 (SYMSAC '71), 23–28
- [21] SIMS, Charles C.: *Computation with finitely presented groups*. Cambridge,, England, New York : Cambridge University Press, 1994 (Encyclopedia of mathematics and its applications). <http://opac.inria.fr/record=b1082972>. – ISBN 0–521–43213–8
- [22] TODD, J. A. ; COXETER, H. S. M.: A practical method for enumerating cosets of a finite abstract group. In: *Proc. Edinb. Math. Soc., II. Ser.* 5 (1936), S. 27–34. <http://dx.doi.org/10.1017/S0013091500008221>. – DOI 10.1017/S0013091500008221. – ISSN 0013–0915; 1464–3839/e
- [23] VERLINDE, Christophe L. ; HOL, Wim G.: Structure-based drug design: progress, results and challenges. In: *Structure* 2 (1994), Nr. 7, S. 577–587