Algorithmik und Komplexität OBDD-repräsentierter Graphen

Dissertation

zur Erlangung des Grades eines

Doktors der Naturwissenschaften

der Universität Dortmund

am Fachbereich Informatik

von

Daniel Sawitzki

Dortmund

2006

Tag der mündlichen Prüfung:	29.11.2006
Dekan:	Prof. Dr. Peter Buchholz
Gutachter:	PD Dr. Martin Sauerhoff, Prof. Dr. Ingo Wegener

Zusammenfassung

Ordered Binary Decision Diagrams (OBDDs) werden in vielen praktischen Anwendungsgebieten erfolgreich als Datenstruktur zur kompakten Repräsentation boolescher Funktionen eingesetzt. Auch sehr große Graphen werden in Bereichen wie CAD und Model Checking oft implizit durch boolesche Funktionen und OBDDs dargestellt. In dieser Dissertation interessieren wir uns für grundlegende graphtheoretische Probleme auf OBDD-repräsentierten Graphen und loten die Möglichkeiten entsprechender OBDD-basierter Algorithmen aus.
Zum einen werden neue Algorithmen vorgestellt und ihre Eigenschaften im Hinblick auf das Entwurfsziel sublinearer Heuristiken analysiert. Zum anderen werden Grenzen des Ansatzes durch komplexitätstheoretische Härteresultate und konkrete untere Schranken aufgezeigt.

Danksagung

Mein besonderer Dank gebührt Ingo Wegener für die Ermöglichung meiner Zeit am Lehrstuhl 2 und seine Betreuung. Darüber hinaus danke ich meinen Kollegen Oliver Giel, André Gronemeier, Thomas Hofmeister, Martin Sauerhoff und Detlef Sieling für fachliche Diskussionen und das Korrekturlesen der Forschungspapiere, die dieser Dissertation zugrunde liegen. Allen Mitarbeiterinnen und Mitarbeitern des Lehrstuhls danke ich für die freundliche und motivierende Arbeitsatmosphäre.
Schließlich danke ich Christian Asche, Michael Gregorius und Andrea Schmitz für das Korrekturlesen des Manuskripts.

Inhaltsverzeichnis

1 Einleitung

1.1 Motivation

Graphen spielen eine wichtige Rolle bei der Modellierung von Informationen und Problemen in vielen verschiedenen Anwendungsgebieten. Sie werden als abstrakte Darstellungsform für Beziehungen zwischen Objekten oder Personen verwendet und wurden intensiv in Mathematik und Informatik untersucht. Die Objekte/Personen werden dabei Knoten genannt und Beziehungen zwischen Knoten heißen Kanten. Letztere können gerichtet oder ungerichtet sein. Bei sogenannten gewichteten Graphen werden die Kanten in eine Zahlenmenge abgebildet, um Kosten, Distanzen, Kapazitäten etc. darzustellen.

Es gibt einige fundamentale Probleme auf Graphen, auf die sich viele Fragestellungen aus praktischen Anwendungen zurückführen lassen und die daher in besonderem Maße und unabhängig von konkreten Anwendungen untersucht wurden. Dazu gehören verschiedene Varianten von Kürzeste-Wege-Problemen auf gewichteten Graphen, bei denen kürzeste Wege zwischen speziellen oder allen Knotenpaaren berechnet werden sollen. Ein anderes Beispiel ist die Berechnung maximaler Flüsse zwischen einem Quellenknoten q und einem Senkenknoten s. Dabei hat jede Kante eine beschränkte Flusskapazität und jeder Knoten außer q und s muss die gleiche Menge Fluss aufnehmen, die er auch abgibt. Auch hier gibt es viele Varianten und Verallgemeinerungen. Als drittes populäres Graphproblem sei hier das Zuordnungsproblem genannt, bei dem eine möglichst große Kantenmenge berechnet werden soll, sodass jeder Knoten zu höchstens einer Kante inzident ist.

Diese drei Probleme besitzen effiziente Algorithmen, die stets optimale Ergebnisse in einer polynomiellen Anzahl von Rechenschritten liefern. Viele andere wichtige Graphprobleme einschließlich einiger Varianten der drei oben genannten sind als NP-vollständig bekannt und es besteht keine Hoffnung auf die Existenz solcher effizienten Algorithmen. Lehrbücher wie Cormen u. a. (2001) geben einen Überblick über das weite Feld der Graphprobleme und entsprechende Algorithmen.

Typischerweise werden Graphen $G = (V, E)$ mit Knotenmenge $V = \{v_0, \ldots, v_{N-1}\}$ und Kantenmenge $E \subseteq V^2$ der Kardinalität M durch Adjazenzlisten oder Adjazenzmatrizen dargestellt. Im ersten Fall wird für jeden Knoten eine Liste der Knoten verwaltet, zu denen er adjazent ist. Diese Darstellung hat die Größe $\Theta(N + M)$. Im zweiten Fall handelt es sich um eine $N \times N$-Matrix, die in Zeile i und Spalte j eine 1 enthält wenn $(v_i, v_j) \in E$ und ansonsten eine 0. Diese Darstellung hat die Größe $\Theta(N^2)$. Zudem gibt es zahlreiche Varianten dieser grundlegenden Datenstrukturen für spezielle Fälle wie

spärliche Graphen mit sehr wenigen Kanten. Da Graphen $\Omega(N^2)$ Kanten besitzen können, sind Adjazenzmatrizen im Allgemeinen eine minimale Darstellung. Adjazenzlisten sind dann kompakter, wenn wesentlich weniger als N^2 Kanten vorhanden sind. Um Graphen kompakter repräsentieren zu können, müssen diese also spezielle Eigenschaften haben, die in der Graphentheorie ausgiebig untersucht worden sind (Brandstädt u. a., 1999). So lassen sich etwa die N Knoten eines Intervallgraphen durch N Intervalle $[a_i, b_i]$ mit $a_i, b_i \in \{1, \ldots, 2N\}$ darstellen, die $2N$ $\mathcal{O}(\log N)$-Bitzahlen entsprechen. Die Wahl der Datenstruktur hängt aber oft von der Herkunft des Graphen und von den gewünschten algorithmischen Eigenschaften ab: Während wir die Existenz einer Kante in einer Matrix in konstanter Zeit ablesen können, erlaubt die listenbasierte Darstellung die Ausgabe aller zu einem Knoten adjazenten Knoten in linearer Zeit.

Im Forschungsgebiet der effizienten Algorithmen gelten traditionell Laufzeiten als effizient, die durch Polynome möglichst kleinen Grades in der Eingabegröße beschränkt sind. Heutzutage gibt es aber Bereiche, in denen sogar lineare Laufzeit zur Lösung von Graphproblemen nicht mehr praktikabel ist, da die Graphen sehr groß sind oder die Probleme sehr oft oder mit speziellen Echtzeitrestriktionen gelöst werden müssen. Beispiele sind hier die Analyse von Verkehrs- und Kommunikationsnetzwerken, Probleme in der Bioinformatik und Bilderkennung in Echtzeit. In weiteren Gebieten kann die Graphgröße per Definition exponentiell in einer natürlichen Problemgröße sein, sodass selbst die Darstellung der Graphen im internen Speicher heutiger Rechner mit klassischen Datenstrukturen nicht mehr möglich ist. Wichtige Beispiele sind hier Entwurf und Analyse sequenzieller Schaltkreise (CAD und Model Checking), deren Zustandsgraph exponentielle Größe bez. der Größe des Schaltkreisspeichers hat. In eine aus der Graphentheorie bekannten Klasse mit wesentlich kompakterer Darstellung lassen sich die Graphen auch nicht a priori einordnen.

Derartigen Herausforderungen begegnet man in der Praxis oft mit anwendungsspezifischen Heuristiken. So werden beispielweise in Holzer u. a. (2004) Kombinationen verschiedener bewährter Kürzeste-Wege-Heuristiken experimentell untersucht, unter anderem der zielgerichteten Suche und des Mehrebenenansatzes. Bei Ersterer werden zusätzliche Potenziale zu den Kantenkosten addiert, die die vom einem Startknoten beginnende Suche nach einem Zielknoten im Rahmen des bekannten Dijkstra-Algorithmus in die richtige Richtung lenken soll. Sind die Kantenkosten euklidische Abstände, so kann als Potenzial z. B. der euklidische Abstand zum Zielknoten verwendet werden und Knoten mit geringerem Abstand werden bei der Suche bevorzugt. Beim Mehrebenenansatz hingegen wird eine Sequenz von Kürzeste-Wege-Anfragen durch eine Vorverarbeitung beschleunigt, bei der der Graph zunächst in eine Hierarchie von Teilgraphen zerlegt wird. Nun werden abkürzende Kanten zwischen den Repräsentantenknoten einer Ebene hinzugefügt, deren Kosten denen von kürzesten Wegen entsprechen. Schließlich reicht es für eine konkrete Anfrage aus, einen relativ kleinen Teilgraphen von Repräsentanten einer hohen Ebene zu betrachten. Obwohl solche Techniken die Laufzeit nicht in jedem Fall auf sublineare Größenordnung beschleunigen können und zusätzlichen Verwaltungsaufwand erfordern, sind sie auf Pro-

bleminstanzen aus der Praxis sehr erfolgreich, da diese oft nicht nur sehr groß, sondern auch regelmäßig und von einfacher Struktur sind.

Natürlich gibt es auch Ansätze, um mit der Größenbeschränkung des internen Speichers umzugehen. Dazu gehören vor allem sogenannte externe Algorithmen, die sich eines zusätzlichen großen, aber langsamen Speichers (externes Speichermedium) oder im Allgemeinen einer Speicherhierarchie bedienen. Nur ein kleinerer Teil der gesamten Daten kann sich jeweils im internen Speicher befinden und Teilprobleme müssen lokal gelöst werden. Dabei soll vor allem die Anzahl der Zugriffe auf das langsame externe Speichermedium (I/O-Operationen) minimiert werden. Vielseitig einsetzbare Methoden wurden von Chiang u. a. (1995), Eppstein u. a. (1997), Abello u. a. (2002) und Munagala und Ranade (1999) vorgestellt. Dehne u. a. (1999) nutzen Ansätze aus dem Entwurf paralleler Algorithmen zur Minimierung der I/O-Operationen im Szenario externer Algorithmen. Mit externen Verfahren zum Schlüsselproblem der Graphtraversierung befassen sich Buchsbaum u. a. (2000).

Beim Umgang mit sehr großen Zustandsgraphen in den Bereichen CAD und Model Checking sind seit langem sogenannte implizite (auch symbolisch genannte) Graphdarstellungen durch Datenstrukturen für boolesche Funktionen etabliert. Die sogenannte charakteristische Funktion χ_G eines Graphen G bildet dabei im einfachsten ungerichteten und ungewichteten Fall binär codierte Paare (x, y) von Knotennummern genau dann auf 1 ab, wenn die entsprechenden Knoten in G adjazent sind. Die Darstellung von χ_G soll strukturierte Graphen sehr kompakt darstellen und gleichzeitig möglichst gute algorithmische Eigenschaften haben. Als guter Kompromiss in vielen Fällen haben sich Ordered Binary Decision Diagrams (OBDDs) durchgesetzt (Bryant, 1985, 1986; Wegener, 2000, 2004). Bei dieser graphbasierten Funktionsdarstellung handelt es sich um eingeschränkte Branchingprogramme, bei denen die Berechnung eines Funktionswertes durch Traversierung der Graphdarstellung erfolgt. Mit Variablen markierte Knoten werden je nach Belegung der Variable über eine von zwei möglichen Kanten verlassen, bis schließlich ein Werteterminal erreicht wird. Entscheidend für die Kompaktheit bei strukturierten Funktionen ist, dass mehrfach vorkommende gleiche Subfunktionen nur einmal dargestellt werden. Bei OBDDs dürfen Variablenmarkierungen auf Pfaden im OBDD-Graphen jedoch nur gemäß einer speziellen Variablenpermutation, der Variablenordnung besucht werden. Dadurch wird die Existenz von effizienten Algorithmen für alle wichtigen funktionalen Operationen möglich (z. B. binäre Synthese, Erfüllbarkeit und Gleichheitstest).

Auf der anderen Seite ist die Worst-case-Größe eines OBDDs nicht größer als die trivialer Funktionsdarstellungen wie z. B. Wertetabellen. Erstmals ist es Nunkesser und Woelfel (2005) gelungen, die OBDD-Größe spezieller aus der Graphentheorie bekannter Graphklassen zu untersuchen. So haben Cographen, P_4-spärliche Graphen und P_4-erweiterbare Graphen OBDDs der Größe $\mathcal{O}(N \log N)$ und Intervallgraphen OBDDs der Größe $\mathcal{O}(N^{3/2} / \log^{3/4} N)$. Meer und Rautenbach (2006) konnten das Ergebnis für Cographen auf $\mathcal{O}(\log N)$ verbessern und die gleiche Schranke für Graphen mit konstanter Tree-Width zeigen. Bei konstanter Clique-Width und einer weiteren speziellen

Struktureigenschaft können die Autoren eine obere Schranke von $\mathcal{O}(N)$ zeigen. In dieser Dissertation werden die Resultate von Meer und Rautenbach insofern interessant sein, als dass sie von OBDDs mit sogenannter gemischter Variablenordnung ausgehen, die zudem konstante vollständige Breite haben. Damit fallen sie in eine Kategorie von Graphen, für die in Kapitel 5 effiziente Laufzeit eines Kürzeste-Wege-Algorithmus bewiesen wird.

Zu den Pionierarbeiten über die hardwareorientierte Anwendung von OBDDs gehören Bryant (1986), Burch u. a. (1992) und Coudert (1995). Eine Übersicht geben hier Hachtel und Somenzi (1996) und Wegener (2000). Ein zentrales Graphproblem im Kontext der Verifikation sequenzieller Schaltkreise ist die Berechnung der Knoten, die von einem Startknoten im Zustandsgraph erreichbar ist (Erreichbarkeitsanalyse). Es kamen viele weitere Anwendungen von OBDDs hinzu, darunter ganzzahlige Optimierung (Becker u. a., 2005), Algorithmisches Lernen (Gavalda und Guijarro, 1995) und Relationale Algebra (Berghammer und Neumann, 2005). Außerdem wurden Branchingprogramme für Funktionen mit ganzzahligen Ausgaben vorgestellt, darunter multiterminale BDDs (Fujita u. a., 1997), ADDs (Bahar u. a., 1993), EVBDDs (Lai u. a., 1994), *BMDs (Bryant und Chen, 1997) und K*BMDs (Drechsler u. a., 1996). Unabhängig vom Einsatz als Datenstruktur in effizienten Algorithmen wurden Branchingprogramme auch intensiv in der Komplexitätstheorie untersucht. Untere Schranken für die Branchingprogrammgröße implizieren untere Schranken für die Speicherplatzkomplexität von sequenziellen Berechnungen. Außerdem implizieren Tradeoff-Resultate für Tiefe und Größe von Branchingprogrammen Zeit-Platz-Tradeoffs für sequenzielle Maschinen (Beame u. a., 2001; Borodin und Cook, 1982).

Von besonderem Interesse im Kontext dieser Dissertation sind erste Arbeiten der Boulder-Gruppe um Hachtel und Somenzi, die sich nicht mehr nur auf Varianten der Erreichbarkeitsanalyse und andere spezielle, anwendungsnahe Probleme beschränken, sondern sich mit weiteren abstrakten graphtheoretischen Problemen beschäftigen. In Hachtel u. a. (1994) wird die Minimierung des Energieverbrauchs von Schaltwerken auf die Berechnung maximaler gewichteter Matchings in implizit repräsentierten Graphen zurück geführt. In Hachtel und Somenzi (1997) wird ein OBDD-basiertes Verfahren zur Berechnung maximaler Flüsse in sogenannten 0-1-Graphen vorgestellt, in denen alle Kanten die gleiche Kapazität besitzen. Die Berechnung starker Zusammenhangskomponenten wird in Touati u. a. (1995), Xie und Beerel (1999) und Bloem u. a. (2000) behandelt. Diese Arbeiten zogen auch die Aufmerksamkeit eines theoretischer ausgerichteten Publikums auf sich und motivierten, sich anwendungsunabhängig mit dem theoretischen Potenzial impliziter und insbesondere OBDD-basierter Algorithmen auseinanderzusetzen. Bisher wurden die Verfahren nämlich nur heuristisch eingesetzt und ihre praktische Nützlichkeit durch Experimente auf realen Instanzen belegt. Eine Ausnahme bildet die Laufzeitanalyse eines speziellen Verifikationsverfahrens für Wallace-Tree-Multiplikationsschaltkreise mithilfe von *BMDs (Keim u. a., 2003). In anderen Arbeiten wird die Laufzeit anhand der Anzahl der ausgeführten funktionalen OBDD-Operationen geschätzt, da Methoden für eine Gesamtanalyse un-

ter Einbeziehung der OBDD-Größen fehlten. Dazu gehören Hojati u. a. (1993), aber auch die neueren Arbeiten von Gentilini und Policriti (2003) und Gentilini u. a. (2003).

Die letzten beiden Arbeiten beschäftigen sich mit Zweizusammenhangskomponenten und starken Zusammenhangskomponenten. Oft werden nicht einmal alle OBDD-Operationen gezählt, sondern nur Folgen von Quantifizierungen über Variablen, die Knotennummern codieren. Einzelne binäre Synthesen etwa werden vernachlässigt. Dieses Vorgehen wird mit der Beobachtung gerechtfertigt, dass solche Quantifizierungsfolgen über $\Theta(\log N)$ Variablen (oft Image-Operationen genannt) die Laufzeit besonders dominieren. Zudem bewegt sich der Entwurf impliziter Algorithmen oft in einem Szenario, in dem außer den Variablenvektoren x und y für die beiden Endknoten einer Kante keine weiteren booleschen Hilfsvariablen erlaubt sind. Dadurch wird die Worst-case-Größe der OBDDs beschränkt, die prinzipiell exponentiell in der Anzahl der Variablen sein kann. Andererseits erhebt diese Einschränkung den Entwurf von impliziten Algorithmen mit wenigen Operationen für viele Probleme erst in den Bereich des nicht Trivialen. Mit Hilfe von zusätzlichen Hilfsvariablen lassen sich Probleme wie die implizite Berechnung starker Zusammenhangskomponenten durch $\mathrm{polylog}(N)$ OBDD-Operationen sehr einfach lösen. Dabei kommt allerdings der Ansatz des iterativen Quadrierens zum Einsatz, der auf praktischen Instanzen aber oft eingeschränkten Verfahren mit einer höheren Anzahl von OBDD-Operationen unterlegen ist. Kritisch ist an dieser Motivation aber zu sehen, dass es andererseits auch keine theoretischen oder praktischen Belege dafür gibt, dass das Verbot von Zusatzvariablen bei Algorithmen wie denen von Gentilini, Piazza und Policriti zu effizientem Laufzeitverhalten führt. Ihr recht komplizierter Entwurf spricht erfahrungsgemäß für eher große OBDDs.

Wir werden uns in dieser Dissertation daher von derartig eingeschränkten Modellen lösen und so einerseits effizientes Laufzeitverhalten impliziter Algorithmen auf speziellen strukturierten Instanzen als andererseits auch allgemeine komplexitätstheoretische Härteresultate präsentieren. Die letztgenannten Ergebnisse bauen teilweise auf Ergebnissen von Feigenbaum u. a. (1998) auf, die erste PSPACE-Härte-Resultate für einfache Graphprobleme auf OBDD-repräsentierten Graphen vorgestellt haben. Vergleichbare Ergebnisse sind seit langem für andere kompakte Darstellungen wie Schaltkreise bekannt (Balcázar und Lozano, 1989; Galperin und Wigderson, 1983; Papadimitriou und Yannakakis, 1986).

1.2 Ziele

Die Forschungen, deren Ergebnisse diese Dissertation dokumentiert, hatten also das allgemeine Ziel, implizite, insbesondere OBDD-basierte Algorithmen für verschiedene grundlegende Graphprobleme von einem theoretischen Standpunkt aus zu beleuchten. Bewertungsmaßstab sollte im einfachsten Fall die Anzahl der OBDD-Operationen sein. Im Hinblick auf praktische Verwendbarkeit sollten aber auch verstärkt die Gesamtlaufzeit und der Gesamtspeicherplatz unter Einbeziehung der Analyse auftretender

OBDD-Größen betrachtet werden. In Fällen, in denen dennoch keine theoretischen Ergebnisse erzielt werden konnten, sollten auch experimentelle Untersuchungen stattfinden – allerdings nicht auf praktischen Benchmark-Instanzen, sondern auf angemessenen zufälligen Instanzen, die eine Schätzung des asymptotischen Verhaltens zulassen. Um den praktischen Erfolg OBDD-basierter Methoden theoretisch besser zu verstehen, sollten möglichst allgemeine Struktureigenschaften gefunden werden, die zu einem sehr viel effizienteren als dem Worst-case-Verhalten führen.

Wir interessieren uns also im Bereich der OBDD-basierten Heuristiken für Effizienz auf sehr großen aber zugleich sehr strukturierten Graphen. Dabei erschien es zunächst intuitiv sinnvoll zu sein, sich auf Probleme in der Klasse P zu beschränken – schließlich wollen wir sublineare Laufzeit erreichen und erwarten entsprechend den Ergebnissen von Feigenbaum u. a. (1998), dass Probleme für kompakte Eingabedarstellungen eher schwieriger werden. In Kapitel 7 wird sich diese Intuition erhärten.

Als effizient verstehen wir in dieser Dissertation primär (im Unterschied zum klassischen Effizienzbegriff) polylogarithmische Laufzeit bzw. polylogarithmischen Speicherplatz in Bezug auf die Knotenanzahl N. Dies ist nicht zuletzt durch den traditionsreichen OBDD-Anwendungsbereich der Hardwareverifikation mithilfe von Zustandsgraphen exponentieller Größe motiviert. Das Ziel beim Entwurf eines impliziten Algorithmus ist also, auf möglichst vielen interessanten strukturierten Instanzen einen entsprechend als effizient definierten Ressourcenverbrauch zu erzielen. Ein einfaches kanonisches Vorgehen ist dabei, die Anzahl der OBDD-Operationen in allen Fällen polylogarithmisch zu beschränken, während entsprechende obere Schranken für Größen der OBDDs, die der Algorithmus erzeugt, nur unter speziellen Strukturannahmen über die Eingabeinstanz bewiesen werden können. Als weniger ambitioniertes, zweitrangiges Ziel werden wir allgemeine sublineare Laufzeiten in Bezug auf die Graphgröße betrachten, mit denen wir uns bei experimentellen Untersuchungen zur Nützlichkeit OBDD-basierter Kürzeste-Wege- und Stundenplanalgorithmen zufriedengeben werden.

Im Zusammenhang mit impliziten Graphdarstellungen und -algorithmen sind außerdem die Variablenordnung eines OBDDs, die wesentlich die OBDD-Größe beeinflussen kann, sowie Verfahren zur effizienten Gewinnung einer OBDD-Darstellung aus einer traditionellen Graphdarstellung von Relevanz. Beide Aspekte werden in dieser Dissertation nicht näher betrachtet. Wir werden entweder von einer speziellen gemischten Variablenordnung ausgehen, die angenehme algorithmische Eigenschaften besitzt, oder aber untere Schranken möglichst für alle Variablenordnungen zeigen.

Abschließend seien drei Vorarbeiten genannt, denen die gleichen Zielsetzungen zugrunde lagen wie dieser Dissertation. In Sawitzki (2002) werden implizite Algorithmen für die Flussmaximierung in 0-1-Graphen untersucht. Die Ergebnisse beschränken sich im Wesentlichen auf die Anzahl der OBDD-Operationen sowie auf Experimente. In Woelfel (2003) wird ein impliziter Algorithmus für topologisches Sortieren vorgestellt, der stets mit einer polylogarithmischen Anzahl von OBDD-Operationen auskommt. Zudem werden Analysetechniken für die Gesamtlaufzeit OBDD-basierter

Algorithmen auf Graphen mit speziell strukturierten charakteristischen Funktionen vorgestellt und eingesetzt, um den Sortieralgorithmus auf einfachen Gittergraphen zu analysieren. Tatsächlich wird hier das erste Mal die Effizienz eines impliziten Algorithmus für ein graphentheoretisches Problem theoretisch bewiesen. Schließlich werden in Zeng (2005) verschiedene Varianten impliziter Heuristiken zur Berechnung von Graphfärbungen und minimalen Spannbäumen vorgestellt und teils theoretisch mit den gleichen Methoden wie in Abschnitt 3.8, teils experimentell auf ihre Nützlichkeit auf verschiedenen Arten von Graphinstanzen hin untersucht.

1.3 Überblick

Es folgt ein Überblick über die Inhalte dieser Dissertation. Kapitel 2 versorgt die Leserin und den Leser mit Grundlagen und Notationsvereinbarungen zu Graphen, Graphproblemen, Branchingprogrammen im Allgemeinen, OBDDs im Speziellen, impliziter Graphdarstellung sowie spezieller relevanter boolescher Funktionen. Während grundlegende Konzepte der theoretischen Informatik wie die $\mathcal{O}$-Notation oder die Definition der Komplexitätsklassen NP und PSPACE vorausgesetzt werden (siehe etwa Cormen u. a. (2001)), geben wir hier eine kurze Einführung in das Konzept der Festparameteralgorithmen (FPT-Algorithmen).

Schließlich kommen wir in Kapitel 3 zu ersten Resultaten im Bereich des Entwurfs neuer impliziter Graphalgorithmen. In der Pionierarbeit von Hachtel und Somenzi (1997) benötigen die Autoren zur Berechnung eines maximalen Flusses in einem implizit repräsentierten 0-1-Graphen mindestens $\Omega(D \log N)$ OBDD-Operationen für den Graphdurchmesser D, um ein Niveaunetzwerk oder einen Fluss verbessernden Pfad zu berechnen. Da wir aber im Allgemeinen polylogarithmische Größenordnungen anstreben, benutzen wir den bereits angesprochenen Ansatz des iterativen Quadrierens, um diese beiden Module eines Flussmaximierungsalgorithmus auf polylog(N) OBDD-Operationen zu verbessern. Schließlich wird auch ein heuristisches Flussverbesserungsmodul vorgestellt, das in günstigen Fällen auch mehrere verbessernde Pfade simultan berechnet. Der Gesamtalgorithmus benötigt auf Worst-case-Instanzen dennoch $\Omega(N \log^2 N)$ OBDD-Operationen. In Kapitel 7 werden wir sehen, dass die Existenz oder Nichtexistenz eines wesentlich besseren impliziten Algorithmus ein offenes Problem aus dem Bereich der parallelen Algorithmen lösen würde. Der Algorithmus ist eine Weiterentwicklung eines in Sawitzki (2002) vorgestellten Verfahrens.

Im zweiten Teil des Kapitels betreten wir erstmalig das Gebiet der Gesamtlaufzeitanalyse impliziter Algorithmen und beweisen ähnlich wie in Woelfel (2003), dass durch iteratives Quadrieren maximale Flüsse auf Gittergraphen effizient berechnet werden können. Obwohl die Analysetechnik ebenfalls auf multivariaten Thresholdfunktionen basiert, verwenden wir doch wesentlich flexiblere Methoden, die zudem auch einfachere Beweise besitzen. So fassen wir eine konstante Anzahl von OBDD-Operationen als Festparameteralgorithmus für den Parameter der vollständigen OBDD-Breite auf,

der sich im Falle der Gittergraphen als konstant erweist. Vollständige OBDDs dürfen keine Variablentests auf einem Weg zu einem Senkenknoten überspringen; ihre Breite ist durch die maximale Anzahl von OBDD-Knoten mit gleicher Variablenmarkierung definiert. Nach diesen asymptotischen Ergebnissen berichten wir abschließend, wann das Verfahren von Hachtel und Somenzi (1997) auch in Experimenten geschlagen wird.

In Kapitel 4 wenden wir uns vorübergehend von Algorithmenentwurf und -analyse ab und schaffen Grundlagen, die wir in späteren Kapiteln benötigen. In der OBDD-Theorie (siehe z. B. Wegener (2000)) wurde intensiv untersucht, inwieweit sich die einschränkende Variablenordnung auf die Repräsentationskraft von OBDDs auswirkt. Konkret sind polynomielle obere Schranken für die OBDD-Größe wichtiger Funktionen ebenso von Interesse wie exponentielle untere Schranken, die möglichst für jede denkbare Variablenordnung gelten. Solche unteren Schranken sind z. B. für die grundlegenden Funktionen des mittleren Ausgabebits der ganzzahligen Multiplikation (MUL), der Indirect-Storage-Access-Funktion (ISA) sowie der Hidden-Weighted-Bit-Funktion (HWB) bekannt. Ein Vorgehen zum Beweisen unterer Schranken für die Laufzeit impliziter Graphalgorithmen ist nun, solche als schwer bekannten Funktionen in Eingabeinstanzen hineinzucodieren. Leider existiert keine intuitive Korrespondenz zwischen der Definition entsprechender Funktionen und der Natur charakteristischer Funktionen, wie sie im Rahmen impliziter Algorithmen Anwendung finden. Dies gilt jedoch für den sogenannten Graphen einer Funktion, der sich leichter in Instanzen codieren lässt. Es stellt sich nun die Frage, ob existierende exponentielle Schranken für die OBDD-Größe grundlegender Funktionen stets auf den entsprechenden Graphen einer Funktion übertragen werden können. Anhand eines Gegenbeispiels zeigen wir, dass dies leider nicht der Fall ist. Jedoch gelingen derartige Übertragungen für die genannten Funktionen MUL, ISA und HWB. Diese Ergebnisse sind unabhängig von ihrer Anwendung in der Algorithmenanalyse auch interessante Antworten auf bisher offene natürliche Fragen.

In Kapitel 5 wenden wir uns wieder einer klassischen Menge von Graphproblemen zu, nämlich den Kürzeste-Wege-Problemen. Zunächst verfolgen wir wie in Kapitel 3 das Ziel, polylogarithmische Algorithmen zu entwickeln. Für das All-Pairs-Shortest-Paths-Problem existiert ein nahe liegendes Verfahren, das mithilfe des iterativen Quadrierens das OBDD aller Kürzeste-Wege-Distanzen durch $\mathcal{O}(\log^2 N)$ OBDD-Operationen berechnet. Zunächst ist jedoch unklar, welche Eigenschaften die dabei erzeugten OBDDs besitzen. So präsentieren wir einen alternativen Algorithmus, der nur für strikt positive Kantenlängen richtig arbeitet. Wir zeigen, dass dieser Algorithmus polylogarithmisch bez. $N \cdot B$ für die maximale Kantenlänge B arbeitet, wenn sowohl Eingabe- als auch Ausgabe-OBDD konstante vollständige Breite für eine spezielle natürliche Variablenordnung besitzen. Allgemein kann das Verfahren auch als Festparameteralgorithmus für den Parameter der vollständigen OBDD-Breite aufgefasst werden. Damit haben wir uns erstmalig von einer speziellen Graphinstanz (wie im Falle der Flussmaximierung) gelöst und ein Laufzeitergebnis für eine ganze Klasse von Eingaben bewiesen. Der Ab-

schluss dieser Eigenschaft unter verschiedenen Graphoperationen soll die Nützlichkeit eines derartigen Festparameteralgorithmus unterstreichen.

Schließlich stellt sich die Frage, ob nicht auch einfache Iterative-Quadrieren-Ansätze, die auch mit Nullkanten umgehen können, die gleichen Festparametereigenschaften besitzen. Dies widerlegen wir durch Konstruktion einer pathologischen Instanz ohne Nullkanten, die derartige Algorithmen zur Erzeugung von OBDDs exponentieller Größe zwingt, obwohl Eingabe- und Ausgabe-OBDDs konstante Breite haben. Dabei wird der Graph der Multiplikation in die Instanz codiert und eine entsprechende untere Schranke aus Kapitel 4 benutzt. Dieses Ergebnis unterstützt somit unser Verbot von Nullkanten.

Im zweiten Teil des Kapitels verfolgen wir einen anderen Ansatz: Statt spezialisierte Algorithmen zu entwickeln, die asymptotisch sehr effizient auf sehr strukturierten Instanzen sind, fragen wir uns, wie sich wohl implizite Äquivalente zu klassischen Algorithmen für das Single-Source-Shortest-Paths-Problem im Durchschnitt verhalten. Wir betrachten dabei die Ansätze von Dijkstra und Bellman-Ford. Da sie ohne iteratives Quadrieren funktionieren, benötigen sie $\Omega(N)$ OBDD-Operationen. Dennoch erhoffen wir uns von ihnen, dass sie auf zufälligen Graphen im Durchschnitt kompaktere OBDDs erzeugen, die hoffentlich zu relativ effizientem, wenn auch nicht polylogarithmischem Verhalten führen. Wir werden also sowohl Dijkstra-inspirierte als auch Bellman-Ford-inspirierte implizite Algorithmen vorstellen und in Experimenten auf verschiedenen Arten von (größtenteils randomisiert erzeugten) Graphen anwenden. Dabei bestätigt sich die Vermutung, dass der Dijkstra-Ansatz überlegen im Platzbedarf ist, während der Bellman-Ford-Ansatz eine bessere Laufzeit aufweist: Das explorative Verhalten beim Dijkstra-Ansatz führt zu strukturierten Zwischenergebnissen und kleinen OBDDs, es müssen jedoch alle Knoten einzeln betrachtet werden. Der Bellman-Ford-Ansatz hingegen relaxiert Kanten parallel durch wenige OBDD-Operationen, was trotz unstrukturierterer Zwischenfunktionen und größeren OBDDs oft zu einer besseren Gesamtlaufzeit führt. So kann in einigen Fällen ein wesentlich effizienteres Verhalten beobachtet werden, als es einfache obere Schranken zunächst fürchten lassen. Tatsächlich scheinen die beiden Algorithmen also für Anwendungsfälle geeignet zu sein, in denen eine vertretbare Laufzeit auf OBDD-repräsentierten Instanzen erzielt werden soll, über die keine weiteren Struktureigenschaften bekannt sind. Für theoretische Analysen auf zufälligen Graphen fehlen momentan noch entsprechende Methoden.

Bei der algorithmischen Entwurfsmethode des Dynamischen Programmierens werden aus Lösungen für Teilprobleme eines Problems effizient Lösungen für größere Teilprobleme zusammengesetzt. Dies führt zu insgesamt effizienten Algorithmen, wenn es nur polynomiell viele verschiedene Teilprobleme gibt. Im Kontrast dazu würden rekursiv arbeitende Algorithmen unter Umständen Probleme mehrfach lösen und somit exponentiell schlechter sein. Typischerweise werden die Werte bisheriger Teillösungen in einer Tabelle verwaltet. Zeiger auf Tabelleneinträge, die in die Berechnung eines Eintrags eingeflossen sind, ermöglichen, im Nachhinein einen Pfad/Baum von Teilpro-

blemen zu traversieren und eine konkrete optimale Lösung zu konstruieren. Ein bekanntes Beispiel aus der Bioinformatik ist der Smith-Waterman-Algorithmus, der eine optimale Ausrichtung zweier Zeichenketten A und B der Längen N und M bez. eines anwendungsabhängigen Abstandsmaßes in Zeit und Platz $\mathcal{O}(N \cdot M)$ ermittelt (Smith und Waterman, 1981). Kapitel 6 lag die Idee zugrunde, eine derartige Tabelle eines Dynamische-Programmieren-Algorithmus und ihre Zeiger als Graph aufzufassen und somit den Kontext klassischer graphtheoretischer Probleme zu erweitern. Schließlich können entsprechende Tabellen polynomieller Größe trotzdem bereits zu groß für einen sinnvollen praktischen Einsatz sein, sodass eine Ausnutzung eventueller Regelmäßigkeiten durch OBDD-basierte Heuristiken wünschenswert ist. Genau dies ist der Fall beim Algorithmus von Leung (1982) zur Berechnung von Stundenplänen mit beschränkten Bearbeitungszeiten: Der Algorithmus berechnet eine Zuordnung einer Menge von N Aufgaben, die nur k verschiedene Bearbeitungszeiten besitzen, auf m Maschinen, sodass die maximale Bearbeitungszeit der einzelnen Maschinen (auch Makespan genannt) minimiert wird. Betrachtet man k als konstant, so ist die entsprechende Laufzeit $\mathcal{O}\left(N^{2(k-1)} \log m\right)$ polynomiell. Dennoch ist die Tabelle der Größe $N^{k-1} \log m$ zu groß, um das Verfahren für praktisch interessante Werte k tatsächlich anwenden zu können. Wir stellen eine OBDD-basierte Version von Leungs Algorithmus vor und verfolgen ähnliche Ziele wie bei den impliziten Versionen der klassischen Kürzeste-Wege-Verfahren in Kapitel 5: Durch Experimente auf zufällig erzeugten Instanzen mit Parametern, die für das Gebiet der Stundenplanberechnung (Scheduling) typisch sind, belegen wir die Überlegenheit der OBDD-Heuristik für großes N.

Während im uneingeschränkten Szenario impliziter Algorithmen, in dem im Unterschied zu beispielsweise Gentilini und Policriti (2003) und Gentilini u. a. (2003) auch Zusatzvariablen erlaubt sind, viele Probleme durch polylog(N) funktionale Operationen lösbar sind, konnten wir einen derartigen Algorithmus nicht für die Flussmaximierung in 0-1-Graphen finden. Die Beobachtung, dass eine solche Operation die Werte einer boolesche Funktion für alle Variablenbelegungen „gleichzeitig“ verarbeitet, führt zum Vergleich impliziter Algorithmen mit parallelen Algorithmen. So folgt aus einem Simulationsalgorithmus für implizite Algorithmen durch parallele im ersten Teil von Kapitel 7, dass P-vollständige Probleme unter der etablierten Annahme P $\neq$ NC auch keine impliziten Algorithmen mit einer polylogarithmischen Anzahl von funktionalen Operationen haben. Zu der großen Bandbreite der als P-vollständig bekannten Probleme zählt auch die Berechnung maximaler Flüsse. Es ist jedoch noch ein offenes Problem, ob die Flussmaximierung in 0-1-Graphen P-vollständig oder in NC enthalten ist. Somit würde ein entsprechender impliziter Algorithmus dieses offenbar schwierige Problem in der Komplexitätstheorie lösen. Greenlaw u. a. (1995) geben einen Überblick über das Gebiet der NC-Algorithmen und P-Vollständigkeits-Theorie.

Die OBDD-Breite ist ein erstes kanonisches strukturelles Merkmal auf der Suche nach geeigneten Eigenschaften realer Instanzen, die möglicherweise dem praktischen Erfolg von OBDD-Heuristiken zugrunde liegen. Obwohl in dieser Dissertation polynomielle Laufzeiten impliziter Algorithmen auf speziellen Instanzen mit konstan-

ter vollständiger OBDD-Breite nachgewiesen werden, gelingt dies nur in Einzelfällen und unter starken Annahmen. So stellt sich die Frage, ob Graphprobleme auf OBDD-repräsentierten Graphen nicht trotz starker Breitenbeschränkungen schwierig bleiben. Dies führt im zweiten Teil von Kapitel 7 zu Erweiterungen des PSPACE-Härte-Ergebnisses von Feigenbaum u. a. (1998) zum Nachweis der Nichtexistenz entsprechender Festparameteralgorithmen für viele grundlegende Graphprobleme unter der Annahme P $\neq$ PSPACE. Die betrachteten Parameter sind dabei die vollständigen OBDD-Breiten von Eingabe- und Ausgabe-OBDD für beliebige Variablenordnungen, während der FPT-Algorithmus für das eingeschränkte APSP-Problem in Kapitel 5 von speziellen Variablenordnungen ausgeht. Für keines der betrachteten Probleme existiert in diesem Modell ein Festparameteralgorithmus bez. der Eingabe-OBDD-Breite. Unter weiterer Einbeziehung des Ausgabe-OBDDs bleibt die Frage für manche Probleme offen, für viele können die Negativresultate jedoch angepasst werden.

Schließlich wird im dritten Teil von Kapitel 7 die Platzkomplexität OBDD-basierter Graphprobleme betrachtet. Zunächst wird ein einfaches Schema vorgestellt, mit dem viele grundlegende Graphprobleme auf polynomiellem Platz bez. des Maximums von Eingabe- und Ausgabe-OBDD-Größe gelöst werden können. Dieses Schema kann auch als Festparameteralgorithmus bez. des Maximums der entsprechenden OBDD-Breiten aufgefasst werden. Es ist im Falle der betrachteten Entscheidungsprobleme sogar nur von Größe bzw. Breite des Eingabe-OBDDs abhängig. Interessanter wird es jedoch bei den betrachteten grundlegenden Such- und Optimierungsproblemen. Dort gelingt es in keinem einzigen Fall, einen Festparameteralgorithmus in alleiniger Abhängigkeit von der vollständigen Eingabe-Breite anzugeben. Für Zusammenhangs-Zerlegungs-Probleme, Kürzeste-Wege-Probleme, das Flussmaximierungsproblem, das Topologische Sortieren und eingeschränkte Varianten der Erreichbarkeitsanalyse und des Transitiven Abschlusses gelingt es sogar, die Nichtexistenz derartiger Algorithmen zu beweisen. In diesem Kontext wird auch die erste exponentielle untere Schranke für die OBDD-Größe des höchsten Bits der ganzzahligen Multiplikation für eine spezielle Variablenordnung bewiesen. Abschließend geben wir in Abschnitt 7.4 einen Überblick über die erzielten Komplexitätsresultate.

Zu guter Letzt sprechen wir in Kapitel 8 die verschiedenen Ergebnisse der Dissertation kurz zusammenfassend an, ziehen ein Fazit und geben einen Ausblick auf mögliche fortsetzende Betrachtungen. So wird es sich also gezeigt haben, dass selbst unter starken Einschränkungen keine effizienten Algorithmen für die meisten der betrachteten Probleme existieren. Dies scheint dem großen praktischen Erfolg OBDD-basierter Heuristiken zu widersprechen und wir bieten zwei Erklärungen an, um diesen scheinbaren Widerspruch aufzulösen.

1.4 Veröffentlichungen und Eigenanteil des Doktoranden

Diese Dissertation basiert im Wesentlichen auf bereits veröffentlichten Forschungsergebnissen, die im Rahmen des DFG-Schwerpunktprogramms 1126, „Algorithmik großer and komplexer Netzwerke“, Projekt We 1066/10, „Effiziente Algorithmen für Probleme auf implizit, insbesondere durch BDDs beschriebenen Netzwerken“, entstanden sind. Für die damit verbundene finanzielle Unterstützung sei an dieser Stelle ausdrücklich gedankt. Alle Arbeiten wurden vom Doktoranden selbstständig verfasst. Es folgt eine Auflistung.

Sawitzki, D. (2004a): A symbolic approach to the all-pairs shortest-paths problem, in: *Graph-Theoretic Concepts in Computer Science 2004*, Band 3353 von *Lecture Notes in Computer Science*, S. 154–167, Springer

Auf dieser Arbeit basiert Abschnitt 5.1.

Sawitzki, D. (2004b): Experimental studies of symbolic shortest-path algorithms, in: *International Workshop on Experimental and Efficient Algorithms 2004*, Band 3059 von *Lecture Notes in Computer Science*, S. 482–497, Springer

Auf dieser Arbeit basiert Abschnitt 5.4.

Sawitzki, D. (2004c): Implicit flow maximization by iterative squaring, in: *SOFSEM 2004: Theory and Practice of Computer Science*, Band 2932 von *Lecture Notes in Computer Science*, S. 301–313, Springer

Auf dieser Arbeit basiert Kapitel 3.

Sawitzki, D. (2005a): Lower bounds on the OBDD size of graphs of some popular functions, in: *SOFSEM 2005: Theory and Practice of Computer Science*, Band 3381 von *Lecture Notes in Computer Science*, S. 298–309, Springer

Auf dieser Arbeit basieren Abschnitt 4.3 und Abschnitt 5.3.

Sawitzki, D. (2005b): On symbolic scheduling independent tasks with restricted execution times, in: *International Workshop on Experimental and Efficient Algorithms 2005*, Band 3503 von *Lecture Notes in Computer Science*, S. 277–289, Springer

Auf dieser Arbeit basiert Kapitel 6.

Sawitzki, D. (2006a): Exponential lower bounds on the space complexity of OBDD-based graph algorithms, in: *Latin American Theoretical Informatics 2006*, Band 3887 von *Lecture Notes in Computer Science*, S. 781–792, Springer

Auf dieser Arbeit basiert Abschnitt 7.3.

Sawitzki, D. (2006b): The complexity of problems on implicitly represented inputs, in: *SOFSEM 2006: Theory and Practice of Computer Science*, Band 3831 von *Lecture Notes in Computer Science*, S. 471–482, Springer

Auf dieser Arbeit basieren Abschnitt 7.1 und Abschnitt 7.2.

Sawitzki, D. (2007): Lower bounds on the OBDD size of two fundamental functions' graphs, *Information Processing Letters*, **101**(2), S. 66–71

Auf dieser Arbeit basieren Abschnitt 4.4 und Abschnitt 4.5.

Die Implementationen der experimentell untersuchten Algorithmen sowie die experimentellen Ergebnisdaten stehen unter `http://thefigaro.sourceforge.net` zur Verfügung.

2 Grundlagen

In diesem Kapitel finden die Leserin und der Leser Grundlagen, die später zum Verständnis benötigt werden. Einige grundlegende Definitionen und Erkenntnisse wurden jedoch in den Abschnitten platziert, in denen sie ihre Anwendung finden, um dieses Kapitel übersichtlicher zu gestalten.

In Abschnitt 2.1 führen wir zunächst verschiedene Arten von Graphen ein sowie eine Auswahl an grundlegenden Graphproblemen, deren implizite Versionen in dieser Dissertation eine Rolle spielen. In Abschnitt 2.2 folgt eine kurze Einführung in das Konzept der Festparameteralgorithmen, soweit es in dieser Dissertation verwendet wird. Dann stellen wir in Abschnitt 2.3 mit verschiedenen Branchingprogrammen und schließlich OBDDs Datenstrukturen zur Verfügung, die im Kontext impliziter Graphalgorithmen zur Realisierung funktionaler Darstellungen und Operationen verwendet werden. Anschließend lernen wir in Abschnitt 2.4 zum einen boolesche Funktionen kennen, die als Bausteine mit bekanntermaßen kompakten OBDDs in impliziten Algorithmen Anwendung finden, und zum anderen solche, die keine kompakten OBDDs besitzen und somit beim Beweis unterer Laufzeitschranken nützlich sind. Zu guter Letzt stellen wir in Abschnitt 2.5 die Verbindung zur impliziten Darstellung von Graphen und entsprechenden impliziten Graphalgorithmen her.

2.1 Graphen und ausgewählte grundlegende Graphprobleme

Obwohl von der Vertrautheit der Leserin und des Lesers mit den Grundbegriffen der Graphentheorie ausgegangen wird, beginnen wir mit der formalen Definition von Graphen und der Einführung der für diese Dissertation wichtigsten Konzepte. Sprechen wir nicht explizit von gerichteten oder ungerichteten Graphen, so können die Definitionen oder Aussagen auf beide Fälle bezogen werden und wir verwenden stellvertretend die Notation für gerichtete Graphen.

Definition 2.1.1 (Graphen).

(a) *Ein* gerichteter Graph $G = (V, E)$ *besteht aus einer endlichen Menge* V *von* Knoten *und einer Menge* E *von geordneten Knotenpaaren* $(u, v) \in V^2$*, die* Kanten *genannt werden.*

(b) Ein ungerichteter Graph $G = (V, E)$ *besteht aus einer endlichen Menge V von* Knoten *und einer Menge E von ungeordneten Knotenpaaren $\{u, v\} \subseteq V$, die* Kanten *genannt werden.*

(c) Ein gewichteter Graph $G = (V, E, \phi)$ *ist ein gerichteter oder ungerichteter Graph (V, E) mit* Kantengewichtsfunktion $\phi\colon E \to \mathcal{Z}$*, die die Kanten in eine Zahlenmenge $\mathcal{Z}$ abbildet.*

(d) Ein ungerichteter Graph $G = (V, E)$ mit $E = \{\{u, v\} \mid u, v \in V,\ u \neq v\}$ heißt vollständig.

(e) Eine Schlaufe *ist eine reflexive gerichtete Kante (v, v).*

(f) Ein gerichteter Graph heißt symmetrisch*, wenn $(u, v) \in E \Rightarrow (v, u) \in E$ für alle $u, v \in V$ gilt.*

(g) Ein gerichteter Graph heißt asymmetrisch*, wenn er keine symmetrischen Kantenpaare $(u, v), (v, u)$ enthält.*

(h) Ein Graph $G' = (V', E')$ ist Teilgraph *eines Graphen $G = (V, E)$ wenn $V' \subseteq V$ und $E' = E \cap (V' \times V')$.*

(i) Der Grad $d(v)$ *eines Knotens $v \in V$ ist die Anzahl der Knoten, zu denen er Kanten besitzt (zu denen er* adjazent *ist).*

Wenn nicht anders angegeben, enthält V in diesem Abschnitt $N \in \mathbb{N}$ Knoten $v_0, \ldots, v_{N-1}$ und E hat Kardinalität M.

Definition 2.1.2 (Adjazenz-/Gewichtsmatrix). *Eine* Gewichtsmatrix $A \in \mathcal{Z}^{N \times N}$ *eines gewichteten Graphen $G = (V, E, \phi)$ mit $\phi\colon E \to \mathcal{Z}$ enthält in Position (i, j) mit $(i, j) \in E$ den Eintrag $\phi(i, j)$; für $(i, j) \notin E$ ist $A(i, j)$ ein vereinbarter gesonderter Wert.*

Adjazenzmatrizen *sind die entsprechende Darstellung für ungewichtete Graphen: In diesem Fall definieren wir $A \in \{0, 1\}^{N \times N}$ durch $A(i, j) = 1 \Leftrightarrow (i, j) \in E$.*

Beide Matrizen können offenbar durch N^2 Zahlen dargestellt werden.

Definition 2.1.3 (Adjazenzlisten). *Die* Adjazenzlisten $L_0, \ldots, L_{N-1}$ *eines ungewichteten Graphen $G = (V, E)$ sind ein Vektor von Listen, wobei eine Liste L_i alle Knoten j mit $(i, j) \in E$ enthält. Im Falle gewichteter Graphen speichern wir für jede Kante (i, j) in Liste L_i nicht nur j sondern auch $\phi(i, j)$ ab.*

Offenbar lassen sich die Adjazenzlisten durch insgesamt $\Theta(N+M)$ Zahlen darstellen.

Definition 2.1.4 (Bipartitheit). *Ein ungerichteter Graph $G = (V, E)$ heißt* bipartit*, wenn sich seine Knotenmenge in zwei disjunkte nicht leere Mengen V_1 und V_2 partitionieren lässt, sodass die beiden Endpunkte jeder Kante in verschiedenen dieser beiden Mengen liegen.*

Definition 2.1.5 (Planarität). *Sei $G = (V, E)$ ein Graph. G heißt* planar*, wenn er so in die Ebene $\mathbb{R}^2$ gezeichnet werden kann, dass sich keine Kanten schneiden.*

Definition 2.1.6 (Pfade und Kreise). *Sei $G = (V, E)$ ein Graph.*

(a) Ein Pfad *der* Länge $\ell \in \mathbb{N}$ *in G ist ein Knotenvektor $(v_{\alpha(1)}, \ldots, v_{\alpha(\ell+1)})$, sodass $(v_{\alpha(i)}, v_{\alpha(i+1)}) \in E$ für $i \in \{1, \ldots, \ell\}$ gilt. Für $q = v_{\alpha(1)}$ und $s = v_{\alpha(\ell+1)}$ spricht man von einem q-s-Pfad.*

(b) Ein q-q-Pfad positiver Länge heißt Kreis.

(c) Ein Kreis, auf dem jede Kante in E genau einmal besucht wird, heißt Eulerkreis*.*

Definition 2.1.7 ((Zwei-)Zusammenhang). *Sei $G = (V, E)$ ein ungerichteter Graph.*

(a) G heißt zusammenhängend*, wenn für jedes Paar $(q, s) \in V^2$ ein q-s-Pfad in G existiert.*

(b) G heißt zweizusammenhängend*, wenn für jedes $v \in V$ auch $G' = (V', E')$ mit $V' := V \setminus \{v\}$ und $E' := E \cap \mathcal{P}(V')$ zusammenhängend ist.*

Definition 2.1.8 (Starker Zusammenhang). *Sei $G = (V, E)$ ein gerichteter Graph. G heißt* stark zusammenhängend*, wenn für jedes Paar $(q, s) \in V^2$ ein q-s-Pfad in G existiert.*

Beobachtung 2.1.9. *Ein ungerichteter Graph enthält genau dann einen Eulerkreis, wenn er bis auf unverbundene Knoten zusammenhängend ist und jeder Knoten geraden Grad besitzt.*

Definition 2.1.10 (Bäume und Wälder).

(a) Ein ungerichteter zusammenhängender kreisfreier Graph heißt Baum*.*

(b) Ein gerichteter kreisfreier Graph $G = (V, E)$, in dem alle Knoten von einer Wurzel *$v \in V$ aus erreichbar sind, heißt* gewurzelter Baum*.*

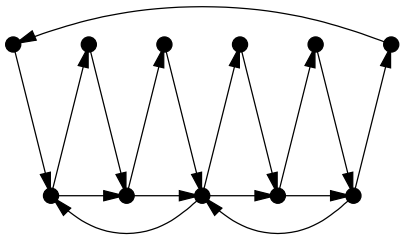
(c) Eine disjunkte Vereinigung von (gewurzelten) Bäumen heißt Wald*.*

Definition 2.1.11 (Matchings). *Sei $G = (V, E)$ ein ungerichteter Graph.*

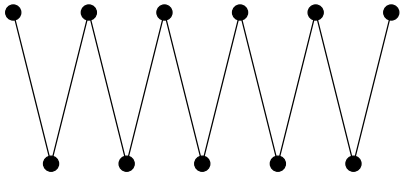
(a) Ein Matching *$M \subseteq E$ in G enthält für jeden Knoten $v \in V$ höchstens eine Kante e mit $e \cap v \neq \emptyset$.*

(b) Ein Matching der Kardinalität $N/2$ heißt perfekt*.*

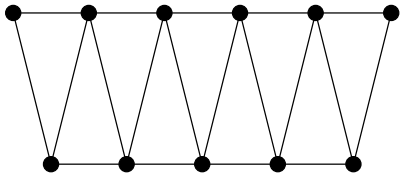
Abbildung 2.1.1 zeigt einige Beispielgraphen zur Veranschaulichung der obigen Grapheigenschaften.



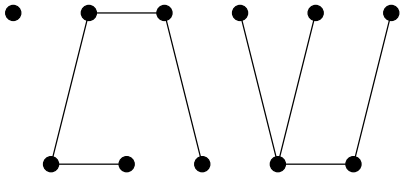
(a) Stark zusammenhängender Graph.



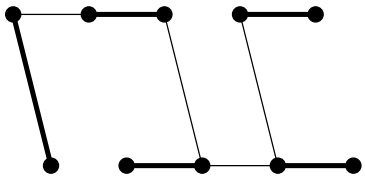
(b) Nicht zweizusammenhängender, bipartiter Graph.



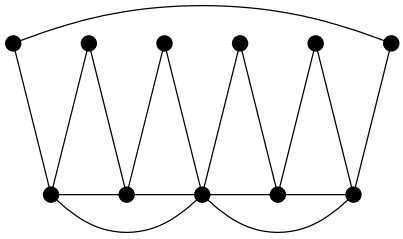
(c) Zweizusammenhängender, nicht bipartiter Graph.



(d) Ein Wald.



(e) Graph mit einem eindeutigen perfekten Matching (in Fettdruck).



(f) Graph mit Eulerkreis.

Abbildung 2.1.1: Einige planare Beispielgraphen.

2.1.1 Ausgewählte Graphprobleme

Wir führen nun grundlegende Probleme auf Graphen ein, die in dieser Dissertation eine Rolle spielen. Ihre Bezeichnungen werden im Folgenden fett gedruckt. Dabei definieren wir im Hinblick auf eine spätere komfortable Graphdarstellung durch boolesche Funktionen Probleme auf gewichteten Graphen direkt für natürliche Gewichte. Natürlich werden in der klassischen Algorithmik auch rationale Kantengewichte betrachtet, die aber bei allen betrachteten Problemen durch Brucherweiterung mit dem kleinsten gemeinsamen Nenner durch natürliche Zahlen ersetzt werden können.

Problem 2.1.12 (q-s-Pfad-Suche). *Gegeben seien ein Graph $G = (V, E)$ und Knoten $q, s \in V$. Existiert ein q-s-Pfad, so gib seine Kanten aus, ansonsten die leere Menge.*

Problem 2.1.13 (q-Erreichbarkeit). *Gegeben seien ein Graph $G = (V, E)$ und ein Knoten $q \in V$. Gib die Menge aller v aus, sodass ein q-v-Pfad in G existiert.*

Problem 2.1.14 (Transitiver Abschluss). *Gegeben sei ein Graph $G = (V, E)$. Gib den Graphen $A = (V, E')$ aus, sodass $(q, s) \in E'$ genau dann gilt, wenn G einen q-s-Pfad enthält.*

Problem 2.1.15 (Kreis-Suche). *Gegeben sei ein Graph G. Gib die Kantenmenge eines beliebigen Kreises in G aus oder, wenn G kreisfrei ist, die leere Menge.*

Problem 2.1.16 (Zusammenhangs-Zerlegung). *Gegeben sei ein ungerichteter Graph G. Gib die Menge der nicht vergrößerbaren zusammenhängenden Teilgraphen von G aus.*

Problem 2.1.17 (Zweizusammenhangs-Zerlegung). *Gegeben sei ein ungerichteter Graph G. Gib die Menge der nicht vergrößerbaren zweizusammenhängenden Teilgraphen von G aus.*

Problem 2.1.18 (Starke-Zusammenhangs-Zerlegung). *Gegeben sei ein gerichteter Graph G. Gib die Menge der nicht vergrößerbaren stark zusammenhängenden Teilgraphen von G aus.*

Problem 2.1.19 (Maximales Matching). *Gegeben sei ein ungerichteter Graph G. Gib ein Matching in G maximaler Kardinalität aus.*

Problem 2.1.20 (Perfekte-Matching-Suche). *Gegeben sei ein ungerichteter Graph G. Falls G ein perfektes Matching besitzt, gib es aus. Ansonsten gib die leere Menge aus.*

Problem 2.1.21 (Flussmaximierung). *Gegeben sei ein gerichteter gewichteter Graph $G = (V, E, \phi)$ mit natürlichen Kantengewichten $\phi(e) \in \mathbb{N}$, den* Kantenkapazitäten, *sowie eine Quelle $q \in V$ und eine Senke $s \in V$.*

Eine Abbildung $F\colon E \to \mathbb{N}$ heißt Fluss*, wenn für alle $v \in V \setminus \{q, s\}$ die Eigenschaft $\sum_{e=(\bullet,v)} F(e) = \sum_{e=(v,\bullet)} F(e)$ gilt. Der* Wert $\mathrm{val}(F)$ *eines Flusses F ist $\sum_{e=(s,\bullet)} F(e)$. Gib einen Fluss F mit maximalem Wert aus.*

Problem 2.1.22 (0-1-Flussmaximierung). *Gegeben sei ein gerichteter Graph $G = (V, E)$ sowie eine Quelle $q \in V$ und eine Senke $s \in V$. Jede Kante in E habe Kapazität 1. Berechne eine Teilmenge $F \subseteq E$ von Kanten, die jeweils einen Fluss von 1 tragen, sodass der entsprechende Flusswert $|\{F \cap (\{q\} \times V)\}|$ maximiert wird.*

Beobachtung 2.1.23. *Problem 2.1.22 ist ein Teilproblem der allgemeinen* **Flussmaximierung***. F entspricht der Kantenmenge einer maximalen Menge kantendisjunkter q-s-Pfade.*

Problem 2.1.24 (Single-Source-Shortest-Paths (SSSP)). *Gegeben sei ein gerichteter gewichteter Graph $G = (V, E, \phi)$ mit natürlichen Kantengewichten $\phi(e) \in \mathbb{N}$, den* Kantenlängen*, sowie eine Quelle $q \in V$. Berechne für jeden Knoten $v \in V$ die minimale Länge $\Delta(v)$ eines q-v-Pfades, falls existent. Die Länge eines Pfades ist dabei die Summe seiner einzelnen Kantenlängen.*

Problem 2.1.25 (Single-Source-Shortest-Paths$^+$ (SSSP$^+$)). *Entspricht dem SSSP-Problem mit $\phi\colon E \to \mathbb{N}^+$.*

Problem 2.1.26 (All-Pairs-Shortest-Paths (APSP)). *Gegeben sei ein gerichteter gewichteter Graph $G = (V, E, \phi)$ mit natürlichen Kantengewichten $\phi(e) \in \mathbb{N}$, den* Kantenlängen*. Berechne für jedes Knotenpaar $(u, v) \in V^2$ die minimale Länge $\Delta(u, v)$ eines u-v-Pfades, falls existent. Die Länge eines Pfades ist dabei die Summe seiner einzelnen Kantenlängen.*

Problem 2.1.27 (All-Pairs-Shortest-Paths$^+$ (APSP$^+$)). *Entspricht dem APSP-Problem mit $\phi\colon E \to \mathbb{N}^+$.*

Im Gegensatz zur üblichen Definition der beiden obigen Kürzeste-Wege-Probleme verlangen wir hier nicht eine Ausgabe der Pfadkanten selbst, um die Probleme so einheitlicher in Kapitel 7 behandeln zu können. In Kapitel 5 werden aber auch implizite Ansätze zur Ermittlung der konkreten Pfadkanten angesprochen.

Problem 2.1.28 (Topologisches Sortieren). *Gegeben sei ein gerichteter kreisfreier Graph $G = (V, E)$. Berechne eine Abbildung $\Delta\colon V \to \mathbb{N}$, sodass $\Delta(v)$ für jedes $v \in V$ die maximale Länge eines in v endenden Pfades in G ist.*

Problem 2.1.29 (Minimaler Spannbaum). *Gegeben sei ein ungerichteter zusammenhängender gewichteter Graph $G = (V, E, \phi)$ mit natürlichen Kantengewichten $\phi(e) \in \mathbb{N}$, den* Kantenkosten*. Berechne einen Baum $T = (V, E', \phi)$, dessen Kosten $\sum_{e \in E'} \phi(e)$ minimal sind. Da T alle Knoten von V verbindet, wird er* Spannbaum *genannt.*

Nach den obigen Such- und Optimierungsproblemen folgen nun Entscheidungsprobleme.

Problem 2.1.30 (Bipartitheit). *Gegeben sei ein ungerichteter Graph $G = (V, E)$. Ist G bipartit?*

Problem 2.1.31 (Planarität). *Gegeben sei ein ungerichteter Graph $G = (V, E)$. Ist G planar?*

Problem 2.1.32 (q-s-Pfad-Existenz). *Gegeben seien ein Graph $G = (V, E)$ und Knoten $q, s \in V$. Existiert ein q-s-Pfad in G?*

Problem 2.1.33 (Kreis-Existenz). *Gegeben sei ein Graph $G = (V, E)$. Enthält G einen Kreis?*

Problem 2.1.34 (Eulerkreis-Existenz). *Gegeben sei ein gerichteter Graph $G = (V, E)$. Existiert ein Eulerkreis in G?*

Problem 2.1.35 (Zusammenhang). *Gegeben sei ein ungerichteter Graph $G = (V, E)$. Ist G zusammenhängend?*

Problem 2.1.36 (Zweizusammenhang). *Gegeben sei ein ungerichteter Graph $G = (V, E)$. Ist G zweizusammenhängend?*

Problem 2.1.37 (Starker Zusammenhang). *Gegeben sei ein gerichteter Graph $G = (V, E)$. Ist G stark zusammenhängend?*

Problem 2.1.38 (Perfekte-Matching-Existenz). *Gegeben sei ein ungerichteter Graph $G = (V, E)$. Existiert ein perfektes Matching in G?*

Die Problemnamen wurden so gewählt, dass eine Korrespondenz zwischen Problemen mit dem Suffix „Suche“ und solchen mit dem Suffix „Existenz“ besteht. Die Existenzvariante wird genau dann mit „Nein“ beantwortet, wenn die Lösung der Suchvariante die leere Menge ist. Bei den Zerlegungsproblemen für Zusammenhang, Zweizusammenhang und starken Zusammenhang lautet zudem die Antwort der entsprechenden Entscheidungsvarianten genau dann „Ja“, wenn die Lösung der Zerlegungsvariante aus genau einer Menge besteht. Diese Zusammenhänge werden in Abschnitt 7.2 hilfreich sein.

Für klassische Darstellungen von Eingaben und Ausgaben wie Adjazenzlisten der Größe $\Theta(N + M)$ oder Adjazenzmatrizen der Größe $\Theta(N^2)$ sind die obigen Probleme effizient lösbar:

Theorem 2.1.39 (Siehe z. B. Cormen u. a. (2001)). *Alle in diesem Abschnitt definierten Probleme besitzen polynomielle Algorithmen bez. der Knotenanzahl N der Eingabeinstanz G.*

2.2 Festparameteralgorithmen

In der theoretischen Informatik wird die Laufzeit- und Speicherplatzkomplexität typischerweise in Bezug auf eine Eingabe-/Problemgröße N gemessen. Je nach Modell ist dies die Länge einer Eingabezeichenkette über einem endlichen Alphabet, eine Anzahl von Eingaberegistern, die ganze Zahlen enthalten, oder eine Anzahl abstrakter Eingabeobjekte (wie z. B. Graphknoten). Die Größe bzw. Schwierigkeit einer Eingabeinstanz kann jedoch bez. verschiedener Aspekte betrachtet werden und somit mehrere Dimensionen haben.

Beim NP-vollständigen Cliquenproblem möchten wir beispielweise wissen, ob ein ungerichteter Graph $G = (V, E)$ einen vollständigen Teilgraphen (Clique) der Größe k besitzt. Dabei ist k typischerweise Teil der Eingabe und trägt höchstens $\mathcal{O}(\log k)$ Zeichen zur Eingabegröße bei. Betrachten wir die Eingabe jedoch zweidimensional als Tupel (G, k) der zweidimensionalen Größe $(|V|, k)$, ergibt sich ein neues Szenario: Wir können die Schwierigkeit des Cliquenproblems nun getrennt nach $|V|$ und k betrachten. Die Motivation des Gebiets der parametrisierten Komplexität und der Festparameteralgorithmen ist, dass in der Praxis eine Problemdimension oft klein ist und wir unter dieser Einschränkung hoffen, auch als schwer bekannte Probleme effizient lösen zu können. Dann wird die kombinatorische Explosion nur durch einige der betrachteten Problemdimensionen verursacht und „große“ Instanzen können trotzdem behandelt werden. Für das Cliquenproblem existieren so Algorithmen, deren Laufzeit polynomiell für konstantes k ist.

Wir formalisieren eine eingeschränkte Variante des Konzepts für unsere Zwecke, während eine detaillierte Einführung in das Gebiet z. B. in Downey und Fellows (1999) nachgelesen werden kann. Im Folgenden sei $\Lambda(I)$ die Länge eines Vektors $I \in M^*$ für den kleenschen Abschluss M^* einer endlichen Menge M.

Definition 2.2.1 (Festparameteralgorithmen (FPT-Algorithmen)).

(a) Sei Γ ein endliches Alphabet. Ein parametrisiertes Problem Π *ist eine Abbildung $\Pi\colon \Gamma^* \times \mathbb{N} \to \mathcal{P}(\Gamma^*)$. Die zweite Komponente k einer Probleminstanz (I, k) wird* Problemparameter *genannt.*

(b) Ein Algorithmus für ein parametrisiertes Problem Π wird Festparameteralgorithmus (FPT-Algorithmus) *genannt, wenn er für ein geeignetes $\alpha\colon \mathbb{N} \to \mathbb{N}$ für jede Eingabe (I, k) mit $N := \Lambda(I)$ eine Lösung $O \in \Pi(I)$ in Zeit $\mathcal{O}(\mathrm{poly}(N) \cdot \alpha(k))$ berechnet.*

Ein FPT-Algorithmus darf also beliebige endliche (insbesondere exponentielle) Laufzeit bez. eines festen Parameters k haben, während er in der klassischen Problemgröße N polynomiell sein muss. Dabei ist die Abkürzung „FPT“ vom englischen „fixed-parameter tractable“ abgeleitet. Vergleichbar mit dem Konzept der klassischen NP-Vollständigkeit ist hier die W[1]-Vollständigkeit. Es gibt aber noch weitere Techniken, um zu beweisen, dass ein Problem unter als anerkannt geltenden Annahmen

keinen FPT-Algorithmus besitzt. Wir werden in Kapitel 7 jedoch stärkere Resultate beweisen, die lediglich von der schwachen Annahme P $\neq$ PSPACE ausgehen.

2.3 Branchingprogramme und OBDDs

Die Grundlagen zu verschiedenen Varianten von Branchingprogrammen in diesem Abschnitt sind genauer in Wegener (2000) nachzulesen.

Im Folgenden sei $\mathbb{B} := \{0,1\}$. Sei $x := (x_0, \ldots, x_{n-1}) \in \mathbb{B}^n$ ein binärer Vektor. Mit $|x|$ bezeichnen wir den *Wert* $\sum_i x_i \cdot 2^i$ von x, mit $||x|| := \sum_i x_i$ hingegen die Anzahl Einsen in x. Die Menge aller booleschen Funktionen $\mathbb{B}^n \to \mathbb{B}^m$ bezeichnen wir mit $B_{n,m}$, und $B_{n,1}$ heiße verkürzt B_n. Sei nun $f \in B_n$ definiert auf einer Menge $X := \{x_0, \ldots, x_{n-1}\}$ boolescher Variablen.

Definition 2.3.1 (Branchingprogramme (BPs)).

(a) Ein auf X definiertes Branchingprogramm *P ist ein gerichteter kreisfreier Graph mit zwei Arten von Knoten:* innere Knoten *und* Senken. *Ein innerer Knoten v ist mit einer Variablen $\ell(v) \in X$ markiert und wird von zwei Kanten verlassen, der* 1-Kante *und der* 0-Kante. *Eine Senke w ist mit einer booleschen Konstante $\ell(w) \in \mathbb{B}$ markiert und wird von keiner Kante verlassen. Ein spezieller Knoten ist als* Quelle *q markiert.*

(b) Eine Belegung $c = (c_0, \ldots, c_{n-1}) \in \mathbb{B}^n$ der Variablen in X definiert einen eindeutigen Berechnungspfad *p_c in P von q zu einer Senke s_c, auf dem innere Knoten mit Variablenmarkierung x_i über ihre c_i-Kante verlassen werden. P repräsentiert die Funktion $f \in B_n$ genau dann, wenn $f(c) := \ell(s_c)$ für alle $c \in \mathbb{B}^n$ gilt.*

(c) Die Größe *von P ist durch die Anzahl von Ps Knoten definiert und wird als* $\mathrm{size}(P)$ *notiert.*

Wir betrachten zwei eingeschränkte Varianten von Branchingprogrammen. Sei $\pi \in X^*$ ein Vektor von Variablen aus X.

Definition 2.3.2 (Stereotype BPs). *Ein auf X definiertes BP P heißt* π-stereotyp, *wenn die Variablensequenz, die auf einem Pfad p von Quelle q zu einer Senke besucht wird, für jedes p eine Teilsequenz von π ist.*

Beobachtung 2.3.3. *Die Knoten eines auf X definierten π-stereotypen BPs P können so in Teilmengen $M_1, \ldots, M_{\Lambda(\pi)}$ partitioniert werden, dass eine Kante ausschließlich von einem M_i- zu einem M_j-Knoten für entsprechende Indizes $i < j$ verläuft und für jedes $i \in \{1, \ldots, \Lambda(\pi)\}$ alle Knoten in M_i mit Variable π_i markiert sind.*

Definition 2.3.4. *Eine der Beobachtung 2.3.3 entsprechende Partitionierung $M_1, \ldots, M_{\Lambda(\pi)}$ der Knoten eines π-stereotypen BPs P nennen wir* π-Einbettung *von P. Die* Breite *einer solchen π-Einbettung definieren wir als* $\max\{|M_1|, \ldots, |M_{\Lambda(\pi)}|\}$.

Definition 2.3.5 (Stereotype Read-once-Branchingprogramme).

(a) Ein auf X definiertes π-stereotypes BP P heißt π-stereotypes Read-once-BP *oder* π-OBDD *(von „Ordered Binary Decision Diagram“), wenn π jede Variable aus X genau einmal enthält. Dann wird π die* Variablenordnung *von P genannt.*

(b) Ein auf X definiertes OBDD P heißt vollständig, *wenn jeder Pfad von Quelle q zu einer Senke Länge n hat.*

Eine Variablenordnung π kann als Bijektion $\{1, \ldots, |X|\} \to X$ interpretiert werden, und für $i < j$ muss auf einem Berechnungspfad $\pi(i)$ vor $\pi(j)$ gelesen werden. Während in der Literatur Variablenordnungen meist Permutationen der Variablenindexmenge sind, haben wir sie hier also als Permutationen der Menge $X = \{x_0, \ldots, x_{n-1}\}$ definiert. So können wir später übersichtlicher mit Funktionen arbeiten, die auf mehreren Subvektoren von Variablen definiert sind. Analog zur Menge Σ_n aller Permutationen $\{1, \ldots, n\} \to \{1, \ldots, n\}$ sei also die Menge Σ_X definiert. Analog zur identischen Abbildung id sei die identische Variablenpermutation $\mathrm{id}(i) := x_{i-1}$ definiert.

Abbildung 2.3.1 zeigt die minimalen id-OBDDs der beiden Beispielfunktionen DSA_4 und $T_{3,6}$, die in Abschnitt 2.4 allgemein definiert werden. Erstere berechnet den Wert der Variablen $y_{|x|}$, Letztere berechnet genau dann eine 1, wenn $\sum_{i=0}^{5} x_i \geq 3$ ist. Bei der grafischen Darstellung von OBDDs werden 1-Kanten üblicherweise durchgehend, 0-Kanten hingegen gestrichelt gezeichnet.

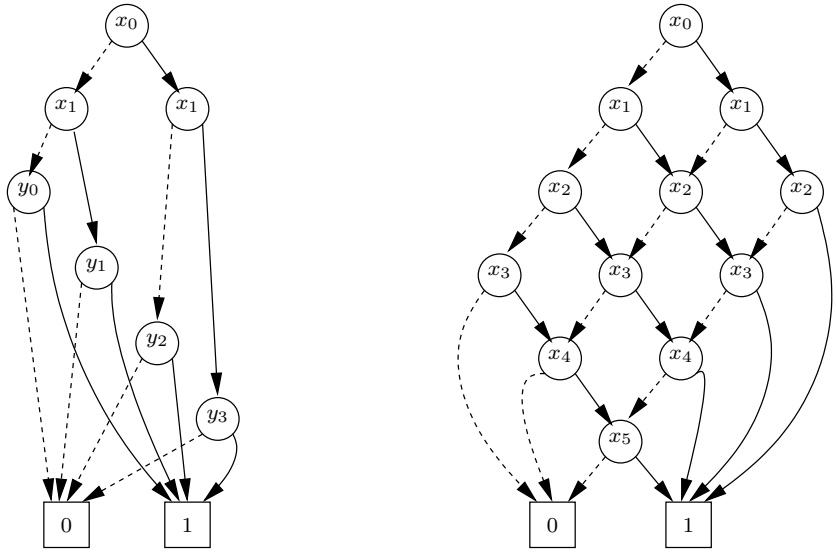
Theorem 2.3.6 (OBDD-Strukturtheorem (Sieling und Wegener, 1993a)). *Für $f \in B_n$ sei S_i die Anzahl verschiedener Subfunktionen $f \mid x_0 := c_0, \ldots, x_{i-1} := c_{i-1}$, die essenziell von x_i abhängen. Sei $S_{i+1} := |f(\mathbb{B}^n)| \in \{1, 2\}$ die Bildgröße von f. Ein minimales* id-OBDD *für f enthält genau S_i innere Knoten, die mit x_i markiert sind, sowie S_{i+1} Senken.*

Jede Subfunktion von f gemäß der Variablenordnung π entspricht also genau einem Knoten im minimalen π-OBDD. Wir werden nun ein Kriterium für die Minimalität von OBDDs kennenlernen.

Theorem 2.3.7 (Minimalität von OBDDs). *Ein π-OBDD hat genau dann minimale Größe, wenn*

1. *alle seine Knoten von seiner Quelle q aus erreichbar sind,*
2. *es keinen inneren Knoten mit gleichem 1- und 0-Nachfolger enthält und*
3. *es kein Paar u, v innerer Knoten gibt, die mit der gleichen Variable markiert sind und identische 1-Nachfolger sowie identische 0-Nachfolger besitzen.*

Es ist sofort klar, dass ein Knoten mit gleichen 1- und 0-Nachfolger überflüssig ist. Ebenso braucht jedes Tripel aus Variable, 1-Nachfolger und 0-Nachfolger nur durch



(a) id-OBDD[DSA_4]. (b) id-OBDD[$T_{3,6}$].

Abbildung 2.3.1: Die OBDDs zweier Beispielfunktionen.

einen inneren Knoten repräsentiert zu werden. Dass diese Eigenschaften jedoch für die Minimalität hinreichend sind, ist nicht sofort einsichtig, folgt aber leicht aus Theorem 2.3.6 (siehe z. B. Theorem 3.3.2 in Wegener (2000)). Dies gilt auch für folgende Eigenschaft:

Theorem 2.3.8 (Eindeutigkeit minimaler OBDDs). *Das minimale π-OBDD einer Funktion $f \in B_n$ ist bis auf Isomorphie eindeutig und wird im Folgenden als π-OBDD$[f]$ notiert. Das minimale vollständige π-OBDD für f ist ebenfalls bis auf Isomorphie eindeutig und wird im Folgenden als π-OBDD$_c[f]$ notiert.*

Theorem 2.3.9 (Siehe Breitbart u. a. (1995)). *Für jedes $f \in B_n$ und jede Variablenordnung π der n Variablen gilt* size(π-OBDD$[f]$) $\leq (2 + o(1))2^n/n$.

Obwohl wir uns von OBDDs eine wesentlich kleinere Größe als $2^n/n$ für strukturierte Funktionen erhoffen, ist diese Darstellung andererseits nie wesentlich größer als eine entsprechende triviale Darstellung wie z. B. eine Funktionswertetabelle. Die BP-Größe (und damit auch OBDD-Größe) zufälliger Funktionen ist jedoch ebenfalls exponentiell.

Theorem 2.3.10 (Siehe Theorem 2.2.2 in Wegener (2000)). *Mindestens $2^{2^n} - 2^{2^n - 2^{n/2}}$ Funktionen in B_n haben mindestens BP-Größe $2^n n^{-1}(1 - o(1))$.*

Seien Prob(a) die Wahrscheinlichkeit eines Ereignisses a und E$[X]$ der Erwartungswert einer Zufallsvariablen X.

Korollar 2.3.11. *Aus Theorem 2.3.10 und der Markoffschen Ungleichung folgt für den Erwartungswert* $\mathrm{E}[S]$ *der minimalen Größe* S *einer gleichverteilt zufälligen Funktion* $f \in B_n$ *und* $t := 2^n n^{-1}(1 - o(1))$

$$\mathrm{E}[S] \geq t \cdot \mathrm{Prob}(S \geq t) = t \cdot \left(1 - 2^{2^{n/2}-2^n}\right) = 2^n n^{-1}(1 - o(1)).$$

Lemma 2.3.12 (Siehe Lemma 2.2.1 in Wegener (2000)). *Höchstens* $sn^s(s+1)^{2s}/s!$ *Funktionen* $f \in B_n$ *haben BPs, die nicht größer als* s *sind.*

Theorem 2.3.13. *Sei* $f \in B_n$ *eine zufällige Funktion mit* $\mathrm{Prob}\,(f(x) = 1) = p$ *für alle* $x \in \mathbb{B}^n$ *und ein konstantes* $p \in]0, 1[$. *Die erwartete Größe eines minimalen BPs für* f *ist* $\Omega(2^{(1-\varepsilon)n})$ *für* $\varepsilon > 0$.

Beweis: Nach der Stirling-Formel ist $s! \geq as^{s+1/2}\mathrm{e}^{-s}$ für eine Konstante a und es folgt

$$\log\left(sn^s(s+1)^{2s}/s!\right) \leq s(\log s + \log n + b) \tag{2.3.1}$$

für eine Konstante b (siehe Beweis von Theorem 2.2.2 in Wegener (2000)).

Wir gehen zunächst von $p \leq 1/2$ aus; dann ist $\mathrm{Prob}(f) \leq (1-p)^{2^n}$ für jedes $f \in B_n$. Sei S die Zufallsvariable der minimalen BP-Größe einer zufälligen Funktion $f \in B_n$. Aus Lemma 2.3.12 und Gleichung (2.3.1) mit $s = 2^{(1-\varepsilon)n}$ folgt so für eine Konstante c und hinreichend großes n

$$\log\left(\mathrm{Prob}\left(S < 2^{(1-\varepsilon)n}\right)\right) \leq 2^{(1-\varepsilon)n}((1-\varepsilon)n + \log n + c) + 2^n \log(1-p) \leq -n.$$

Mit der Markoffschen Ungleichung gilt dann für hinreichend großes n

$$\mathrm{E}[S] \geq 2^{(1-\varepsilon)n} \cdot \mathrm{Prob}\left(S \geq 2^{(1-\varepsilon)n}\right) \geq 2^{(1-\varepsilon)n} \cdot \left(1 - 2^{-n}\right) = \Omega(2^{(1-\varepsilon)n}).$$

Die gleiche Schranke gilt auch für $p > 1/2$, da $f \in B_n$ die gleiche minimale BP-Größe wie $\overline{f}$ hat. □

Vollständige OBDDs dürfen keine Variablentests überspringen. Ersetzen wir jedoch Sprungkanten in allgemeinen OBDDs durch eine Verkettung innerer Knoten, die mit den übersprungenen Variablen markiert sind, entstehen für jeden bisherigen OBDD-Knoten nicht mehr als n neue Knoten. Man kann sich so überlegen, dass minimale vollständige OBDDs Bedingung 2 in Theorem 2.3.7 nicht erfüllen müssen, sich das Minimalitätskriterium aber ansonsten übertragen lässt.

Fakt 2.3.14. *Stets gilt* $\mathrm{size}(\pi\text{-OBDD}_{\mathrm{c}}[f]) \leq (n+1) \cdot \mathrm{size}(\pi\text{-OBDD}[f])$.

Bei der Analyse OBDD-basierter Algorithmen in dieser Dissertation wird die Breite vollständiger OBDDs eine zentrale Rolle spielen.

Beobachtung 2.3.15. *Die π-Einbettung eines minimalen (vollständigen) π-OBDDs P ist eindeutig. Die Breite von P ist dann die maximale Anzahl innerer Knoten, die mit gleicher Variable markiert sind.*

In der Praxis werden meistens viele boolesche Funktionen gemeinsam in einem OBDD repräsentiert und jede Funktion f ist durch einen Zeiger q_f auf ihren Quellenknoten definiert. So müssen gemeinsame Subfunktionen nur einmal repräsentiert werden. Wir werden im Folgenden ebenfalls von so einer gemeinsamen Darstellung ausgehen, die sich *Shared Binary Decision Diagram (SBDD)* nennt.

Schließlich seien noch OBDDs mit *komplementierten Kanten* erwähnt (siehe auch hier Kapitel 3 in Wegener (2000)). Bei dieser Variante erhalten alle 1-Kanten sowie Funktionszeiger ein sogenanntes *Komplementbit*, und es gibt keine 1-Senke, sondern nur eine 0-Senke. Ist die Anzahl gesetzter Komplementbits auf einem Berechnungspfad ungerade, wird der Funktionswert an der Senke negiert. Minimale OBDDs sind somit wieder eindeutig, und eine Funktion $f \in B_n$ und ihr Komplement $\overline{f}$ sind stets gleichzeitig repräsentiert. Es ergibt sich der Zusammenhang $\text{size}(\pi\text{-OBDD}[f, \overline{f}]) = 2 \cdot \text{size}(\pi\text{-OBDD}^*[f])$, wobei π-OBDD$^*[f]$ das minimale π-OBDD mit komplementierten Kanten bezeichne. In Implementationen wird oft mit komplementierten Kanten gearbeitet, da sich das Komplementbit ohne Mehrkosten in Zeigern einbetten lässt. Die theoretischen Betrachtungen dieser Dissertation gehen jedoch nicht auf komplementierte Kanten ein, da sie die OBDD-Größe nicht asymptotisch beeinflussen und die Beweise so übersichtlicher sind.

2.3.1 Effiziente OBDD-Algorithmen für funktionale Operationen

Es ist ein NP-hartes Problem (Bollig und Wegener, 1996), die optimale Variablenordnung zu finden, die zu einem OBDD minimaler Größe für eine Funktion $f \in B_n$ führt. Manche Funktionen haben für jede Variablenordnung exponentielle OBDD-Größe (z. B. Indirect Storage Access), andere haben stets polynomielle Größe (z. B. symmetrische Funktionen wie die Thresholdfunktionen), wieder andere haben bei geschickter Wahl der Variablenordnung exponentiell kleinere OBDDs als bei einer ungeschickten Wahl (z. B. Direct Storage Access). Kapitel 5 in Wegener (2000) gibt hier einen Überblick.

Die Festlegung auf eine Variablenordnung kann uns also exponentielle OBDD-Größe kosten. Auf der anderen Seite ermöglicht sie angenehme algorithmische Eigenschaften, die den effizienten Einsatz von OBDD-basierten impliziten Graphalgorithmen erst möglich machen. Wir definieren grundlegende Probleme auf booleschen Funktionen, die im Folgenden *funktionale Operationen* genannt werden. Dabei sei $f \mid x := c$ die Subfunktion von f für die Konstantsetzung der Variable x durch die Konstante c.

Definition 2.3.16 (Funktionale Operationen). *Seien $f, g \in B_n$ definiert auf der Variablenmenge $X := \{x_0, \ldots, x_{n-1}\}$.*

1. **Auswertung** *Berechne $f(c)$ für eine Variablenbelegung $c \in \mathbb{B}^n$.*

2. **Erfüllbarkeit** *Ist f erfüllbar?*

3. **Anzahl erfüllender Belegungen** *Berechne die Anzahl erfüllender Belegungen für f.*

4. **Ausgabe erfüllender Belegungen** *Gib die Menge $f^{-1}(1)$ aller erfüllenden Belegungen von f aus.*

5. **Menge essenzieller Variablen** *Berechne die Menge der* essenziellen Variablen *aus $x_i \in X$, für die $(f \mid x_i := 0) \neq (f \mid x_i := 1)$ gilt.*

6. **Äquivalenz** *Ist $f = g$?*

7. **Negation** *Berechne $\overline{f}$.*

8. **Subfunktion** *Berechne die Subfunktion $f \mid x := c$ für ein $x \in X$ und ein $c \in \mathbb{B}$.*

9. **Binäre Synthese** *Sei $\otimes \in B_2$ ein binärer Infix-Operator (z. B. $\wedge$, $\vee$ oder $\oplus$). Berechne die Funktion $f \otimes g$.*

10. **Quantifizierung** *Sei $\mathcal{Q} \in \{\exists, \forall\}$ ein Quantor. Berechne die Funktion $(\mathcal{Q}x_i)f$.*

11. **Variablentausch** *Tausche zwei Variablen x_i und x_j für $i < j$, berechne also die Funktion*

$$f'(x_0, \ldots, x_{n-1}) := f(x_0, \ldots, x_{i-1}, x_j, x_{i+1}, \ldots, x_{j-1}, x_i, x_{j+1}, \ldots, x_{n-1}).$$

Außerdem spielt in Praxis und Literatur auch die *ternäre Synthese* $\mathrm{ite}(f, g, h) := (f \wedge g) \vee (\overline{f} \wedge h)$ eine Rolle. Für unsere Zwecke wird sie jedoch keinen wesentlichen Unterschied zur binären Synthese ausmachen und daher hier nicht betrachtet werden. Auch die Ersetzung einer Variablen durch eine andere Funktion wird in dieser Dissertation nicht gesondert betrachtet.

Die binäre Synthese ist in dieser Dissertation von zentraler Bedeutung, da sie als einzige grundlegende Operation neben der Quantifizierung (die jedoch typischerweise mithilfe einer binären Synthese realisiert wird) die OBDD-Größe wesentlich erhöhen kann und daher eine Herausforderung bei der Analyse impliziter Graphalgorithmen darstellt. Wir stellen mit Algorithmus 2.3.1 einen üblichen rekursiven OBDD-Algorithmus für die binäre Synthese vor. Gegeben seien wieder die auf $X := \{x_0, \ldots, x_{n-1}\}$ definierten Funktionen $f, g \in B_n$. Wir können hier o. B. d. A. $\pi = \mathrm{id} = (x_0, \ldots, x_{n-1})$ annehmen. Ein rekursiver Aufruf zur Berechnung von id-OBDD$[f \otimes g]$ wird nun durch $\otimes(q_f, q_g)$ bezeichnet, wobei q_f der Zeiger auf den Quellenknoten der Funktion f im alles enthaltenden SBDD sind. Der Algorithmus verwendet vier Untermodule:

- Der Aufruf terminalfall() prüft, ob der aktuelle rekursive Aufruf $\otimes(f, g)$ ein Terminalfall ist, das Ergebnis also offensichtlich eine konstante Funktion ist. Dies gilt natürlich, wenn f und g konstant sind, aber abhängig von $\otimes$ unter Umständen auch dann, wenn nur f oder g konstant sind (z. B. $f \wedge 0$ oder $f \vee 1$).

- Der Aufruf label(q_f, q_g) berechnet $\min\{\ell(q_f), \ell(q_g)\}$, wobei $\ell(q_f)$ die Variablenmarkierung des Quellenknotens von f ist. Er liefert also den Index der ersten zu testenden Variablen, wenn wir parallel die OBDDs für f und g traversieren wollen.

- Die Computed-Table ist eine Datenstruktur für die Ergebnisse der Teilprobleme $\otimes(q_f, q_g)$, die bereits gelöst wurden. Der Aufruf computed-table(q_f, q_g) liefert den Zeiger auf die Ergebnisfunktion, wenn id-OBDD$[f \otimes g]$ bereits berechnet wurde, und den Nullzeiger nil ansonsten.

- Die Unique-Table ist eine Datenstruktur zur Speicherung aller OBDD-Knoten v als Tripel (i, v_1, v_0), wobei $\ell(v) = x_i$ ist und v_1 und v_0 1- bzw. 0-Nachfolger von v sind. Der Aufruf unique-table(i, v_1, v_0) mit $v_1 \neq v_0$ liefert einen Zeiger auf den entsprechenden Knoten, falls er vorhanden ist. Ist er noch nicht vorhanden, so wird er erzeugt, in den Unique-Table eingefügt und als Zeiger ausgegeben. Für $v_1 = v_0$ wird entsprechend Punkt 2 von Theorem 2.3.7 einfach v_0 ausgegeben.

Algorithmus 2.3.1 berechnet also den von (q_f, q_g) aus erreichbaren Teilgraphen des Produktgraphen der beiden Eingabe-OBDDs. Die Korrektheit im Hinblick auf die zu berechnende Ergebnisfunktion ist offensichtlich. Durch Verwendung des Unique-Table-Wörterbuchs werden die in Theorem 2.3.7 genannten Eigenschaften garantiert, und das Ergebnis ist ein minimales π-OBDD.

Theorem 2.3.17 (Siehe z. B. Abschnitt 3.3 in Wegener (2000)). *Seien die auf X definierten Funktionen $f, g \in B_n$ durch π-OBDD$[f]$ und π-OBDD$[g]$ dargestellt. Die in Definition 2.3.16 eingeführten Operationen 1–3 und 5–11 haben polynomielle Algorithmen auf den minimalen OBDD-Darstellungen von f und g, während Operation 4 lineare Laufzeit $\mathcal{O}(n \cdot |f^{-1}(1)|)$ in der Größe ihrer Ausgabe hat.*

Beweis:

1. **Auswertung** Der Wert $f(c)$ kann in Zeit $\mathcal{O}(n)$ durch Traversierung von π-OBDD$[f]$ entsprechend c berechnet werden.

2. **Erfüllbarkeit** Funktion f ist genau dann erfüllbar, wenn sie nicht die Nullfunktion ist. Letztere hat offensichtlich konstante minimale OBDD-Größe. Es ergibt sich Zeit $\mathcal{O}(1)$.

Algorithmus 2.3.1 : Rekursive binäre OBDD-Synthese $\otimes(q_f, q_g)$.

Eingabe : Zwei OBDDs id-OBDD[f] und id-OBDD[g].
Ausgabe : Das OBDD id-OBDD[$f \otimes g$].

```
1  if terminalfall() then
2      return Ergebnis des Terminalfalls;
3  if computed-table(q_f,q_g) ≠ nil then
4      return computed-table(q_f,q_g);
5  else
6      i := label(q_f,q_g);
7      v_1 := ⊗(q_{f|x_i:=1}, q_{g|x_i:=1});
8      v_0 := ⊗(q_{f|x_i:=0}, q_{g|x_i:=0});
9      v := unique-table(i,v_1,v_0);
10     computed-table(q_f,q_g) := v;
11     return v;
```

3. **Anzahl erfüllender Belegungen** Wir berechnen bottom-up für jeden OBDD-Knoten v die Anzahl $Z(v)$ der erfüllenden Belegungen für die Subfunktion f_v, der v entspricht. Für die 0-Senke s_0 gilt offenbar $Z(s_0) = 0$, für die 1-Senke $Z(s_1) = 2^n$. Für einen x_i-Knoten v mit 0-Nachfolger v_0 und 1-Nachfolger v_1 gilt $Z(v) = (Z(v_0) + Z(v_1))/2$, da die Hälfte aller erfüllenden Belegungen für f_{v_0} bzw. f_{v_1} Variable x_i mit 0 bzw. 1 belegt. Für jeden Knoten fällt konstante Zeit an, ingesamt also $\mathcal{O}(\text{size}(\pi\text{-OBDD}[f]))$.

4. **Ausgabe erfüllender Belegungen** Wir führen eine rekursive Tiefensuche von der 1-Senke s_1 aus durch, nachdem alle Kanten im OBDD reversiert wurden. Ein rekursiver Aufruf $A(v)$ für einen OBDD-Knoten v erhält eine Belegung c mit $f_v(c) = 1$ als Parameter. Sei v ein x_j-Knoten und u ein x_i-Vorgängerknoten von v. Sei $X' \in X$ die Teilmenge von Variablen, die in π zwischen x_i und x_j stehen. In $A(v)$ wird nun für jeden solchen Vorgänger u mit a-Kante (u, v) rekursiv $2^{|X'|}$-mal $A(u)$ aufgerufen mit dem Parameter abc für $b \in \mathbb{B}^{|X'|}$. Erreicht ein Aufruf die Quelle q von f, wird c ausgegeben. So entspricht die Laufzeit der Operation der des Ausgebens aller erfüllender Belegungen der Länge n und beträgt damit $\mathcal{O}(n \cdot |f^{-1}(1)|)$.

5. **Menge essenzieller Variablen** In einem minimalen OBDD kommen genau die Variablen als Markierungen innerer Knoten vor, die essenziell sind. In linearer Zeit $\mathcal{O}(\text{size}(\pi\text{-OBDD}[f]))$ besuchen wir alle inneren Knoten und sammeln die gefundenen Variablenmarkierungen.

6. **Äquivalenz** Werden f und g im gleichen SBDD repräsentiert, müssen nur die Quellenzeiger beider Funktionen in Zeit $\mathcal{O}(1)$ verglichen werden. Ansonsten ist das

Problem in Zeit $\mathcal{O}(\text{size}(\pi\text{-OBDD}[f]) + \text{size}(\pi\text{-OBDD}[g]))$ durch parallele Tiefensuchen durch beide OBDD-Graphen lösbar.

7. Negation In Zeit $\mathcal{O}(\text{size}(\pi\text{-OBDD}[f]))$ durch Suche nach Kanten zu Senken und deren anschließende Änderung auf die jeweils andere Senke.

8. Subfunktion In Zeit $\mathcal{O}(\text{size}(\pi\text{-OBDD}[f]))$ durch Suche aller Zeiger und Kanten auf x_i-Knoten und deren Änderung auf die jeweiligen c-Nachfolger.

9. Binäre Synthese Algorithmus 2.3.1 betrachtet jeden Knoten im erreichbaren Teilgraphen G^* des Produktgraphen beider OBDD-Graphen nur einmal. Realisieren wir die Wörterbücher Computed-Table und Unique-Table mithilfe von Datenstrukturen wie balancierten Bäumen mit logarithmischer Zugriffszeit (siehe z. B. Cormen u. a. (2001)), ergibt sich so die Gesamtlaufzeit

$$\mathcal{O}\left(\text{size}(G^*) \cdot \log(\text{size}(G^*))\right) = \mathcal{O}\left(\text{size}(\pi\text{-OBDD}[f]) \cdot \text{size}(\pi\text{-OBDD}[g]) \cdot \log(\text{size}(\pi\text{-OBDD}[f]) \cdot \text{size}(\pi\text{-OBDD}[g]))\right). \quad (2.3.2)$$

In Implementationen ist eine Realisierung der Wörterbücher mithilfe von Hashingverfahren üblich. In der oberen Schranke für die erwartete Laufzeit entsprechend Gleichung (2.3.2) fallen dann die logarithmischen Faktoren weg. Eine eher unübliche Alternative ist, ohne Verwendung von Wörterbüchern den gesamten Produktgraphen zu erzeugen und das Ergebnis-OBDD anschließend in linearer Zeit zu minimieren. (Letzteres ist möglich, siehe Sieling und Wegener (1993b), aber in dieser Dissertation nicht von Interesse.) Die entsprechende Laufzeit von $\mathcal{O}(\text{size}(\pi\text{-OBDD}[f]) \cdot \text{size}(\pi\text{-OBDD}[g]))$ ist auch die allgemeine optimale obere Schranke für die OBDD-Größe des Ergebnisses. Dies ist zwar asymptotisch besser als Algorithmus 2.3.1, in der Praxis ist G^* aber erfahrungsgemäß wesentlich kleiner als der gesamte Produktgraph.

10. Quantifizierung Es ist $(\exists x_i)f = (f \mid x_i := 0) \vee (f \mid x_i := 1)$ und $(\forall x_i)f = (f \mid x_i := 0) \wedge (f \mid x_i := 1)$. Beide Fälle sind also durch zwei Subfunktionsoperationen und eine binäre Synthese in Zeit $\mathcal{O}((\text{size}(\pi\text{-OBDD}[f]))^2)$ zu lösen. Dies ist auch die allgemeine optimale obere Schranke für die OBDD-Größe des Ergebnisses.

11. Variablentausch Wir führen eine Sequenz von $j - i$ Vertauschungen von in π benachbarten Variablen durch, beginnend mit $j - 1$ und j. Durch Aufzählung aller möglichen lokal auftretenden Situationen (siehe Abbildung 5.7.1 in Wegener (2000)) ist leicht ersichtlich, dass bei Vertauschung zweier benachbarter Variablen x_k und x_ℓ für jeden x_k-Knoten höchstens zwei neue Knoten erzeugt werden. Vorher wurden keine neuen x_k-Knoten erzeugt. Insgesamt verdreifacht sich also die OBDD-Größe höchstens in linearer Zeit $\mathcal{O}(\text{size}(\pi\text{-OBDD}[f])$. (Siehe hier auch Definition 5.7.1 und Theorem 5.7.4 in Wegener (2000).)

□

Beobachtung 2.3.18. *Die OBDD-Größe kann sich lediglich bei den Operationen binäre Synthese, Quantifizierung und Variablentausch vergrößern.*

2.4 Ausgewählte grundlegende boolesche Funktionen

Wir beginnen mit Funktionen, die (teils unter gewissen Einschränkungen) OBDDs polynomieller Größe haben, gefolgt von Funktionen, die exponentielle OBDD-Größe unabhängig von der Variablenordnung verursachen. Letztere sind in späteren Kapiteln für untere Laufzeitschranken impliziter Graphalgorithmen von Interesse.

Definition 2.4.1 (Direct Storage Access (DSA)). *Die* Direct-Storage-Access-Funktion DSA_n *für* $n = 2^m$ *ist auf einer Menge von* m *Variablen* $x_0, \ldots, x_{m-1}$ *und einer Menge von* n *Variablen* $y_0, \ldots, y_n$ *definiert und berechnet den Wert* $y_{|x|}$.

Beobachtung 2.4.2. *Für alle Variablenordnungen, die die* x*-Variablen vor den* y*-Variablen testen, hat* DSA_n *die OBDD-Größe* $2n + 1$ *(siehe Abbildung 2.3.1(a)). Das OBDD besteht aus dem vollständigen binären Baum auf den* x*-Variablen, der schließlich die jeweiligen* y*-Variablen als Blätter hat.*

Definition 2.4.3 (Thresholdfunktionen). *Die* Thresholdfunktion $T_{k,n}$ *für* $k, n \in \mathbb{N}^+$ *ist auf einer Menge von* n *Variablen* $x_0, \ldots, x_{n-1}$ *definiert und berechnet genau dann eine 1, wenn* $\sum_{i=0}^{n-1} x_i \geq k$.

Beobachtung 2.4.4. *Für alle Variablenordnungen hat* $T_{k,n}$ *die OBDD-Größe* $\mathcal{O}(n^2)$. *Für* $1 \leq k \leq n/2$ *beträgt die genaue Größe* $k(n-k+1)+2$. *Die Konstruktion erfolgt entsprechend Abbildung 2.3.1(b).*

Obwohl $T_{k,n}$ keine direkte Rolle in dieser Dissertation spielt, haben wir sie als Einstimmung auf verallgemeinerte Thresholdfunktionen eingeführt. Die kanonische Verallgemeinerung ist wohl, die Variablen-Summanden mit Gewichten zu versehen. Die entsprechenden *allgemeinen Thresholdfunktionen* haben leicht ersichtlich eine OBDD-Größe von $\mathcal{O}(n(|\delta_0| + \cdots + |\delta_{n-1}|))$ für die jeweiligen Gewichte δ_i. Hosaka u. a. (1997) haben eine untere Schranke von $2^{\sqrt{n}/2}$ für die OBDD-Größe einer konkreten allgemeinen Thresholdfunktion bewiesen. Wir gehen nun zu einer weiteren Verallgemeinerung von Woelfel (2003) über.

Definition 2.4.5 (Multivariate Thresholdfunktionen, siehe Woelfel (2003)). *Sei* $f \in B_{kn}$ *auf den Variablenvektoren* $x^{(1)}, \ldots, x^{(k)} \in \mathbb{B}^n$ *definiert. Funktion* f *heißt* k-variate Thresholdfunktion*, wenn es* $W \in \mathbb{N}$, $t \in \mathbb{Z}$ *und* $\delta_1, \ldots, \delta_k \in \{-W, \ldots, W\}$ *gibt, sodass*

$$f(x^{(1)}, \ldots, x^{(k)}) = \left(\sum_{i=1}^{k} \delta_i \cdot |x^{(i)}| \geq t \right).$$

W *heißt* maximales absolutes Gewicht *von* f. *Die entsprechende Klasse von Funktionen wird mit* $\mathbb{T}_{k,n}^W$ *bezeichnet.*

Die Verallgemeinerung besteht also darin, dass nicht nur einzelne Variablen gewichtet werden können, sondern Subvektoren von Variablen, die als Binärzahl interpretiert gewichtet und summiert werden. Die Vereinigung der k-variaten Thresholdfunktionen für alle $k \in \mathbb{N}^+$ nennen wir *multivariate* Thresholdfunktionen. Um genauer auf die OBDD-Größe dieser Funktionenklasse eingehen zu können, führen wir zunächst formal gemischte Variablenordnungen ein.

Definition 2.4.6 (Gemischte Variablenordnungen). *Betrachte Variablenvektoren $x^{(1)}, \ldots, x^{(k)} \in \mathbb{B}^n$ und die Permutation $\tau \in \Sigma_n$. Die Variablenordnung*

$$\left(x^{(1)}_{\tau(1)-1}, x^{(2)}_{\tau(1)-1}, \ldots, x^{(k)}_{\tau(1)-1}, x^{(1)}_{\tau(2)-1}, x^{(2)}_{\tau(2)-1}, \ldots\ldots, x^{(k)}_{\tau(n)-1}\right)$$

heißt k-gemischte τ-Ordnung*, bezeichnet mit $\pi^{\tau}_{k,n}$.*

Die jeweilige Teilmenge der Variablen eines Vektors wird also gemäß der lokalen Permutation τ geordnet, während die Gesamtordnung $\pi^{\tau}_{k,n}$ die Variablen $x^{(1)}_{\tau(i)}, \ldots, x^{(k)}_{\tau(i)}$ für jedes i en bloc liest. Für $\tau = \mathrm{id}$ sprechen wir von einer *natürlichen gemischten* Variablenordnung.

Theorem 2.4.7 (Siehe Woelfel (2003)). *Für $f \in \mathbb{T}^W_{k,n}$ hat $\pi^{\mathrm{id}}_{k,n}$-OBDD$_c[f]$ Breite $\mathcal{O}(kW)$, und es ist* $\mathrm{size}(\pi^{\mathrm{id}}_{k,n}\text{-OBDD}[f]) \leq \mathrm{size}(\pi^{\mathrm{id}}_{k,n}\text{-OBDD}_c[f]) = \mathcal{O}(k^2Wn)$.

Beweis: Da multivariate Thresholdfunktionen eine zentrale Rolle in dieser Dissertation spielen, skizzieren wir den Aufbau des entsprechenden minimalen vollständigen OBDDs. Dazu überlegen wir uns zunächst einen Algorithmus, der die Variablen gemäß $\pi^{\mathrm{id}}_{k,n}$ liest und dem Prinzip der schriftlichen Addition folgt. Schließlich soll das Vorzeichen der Summe

$$s := -t + \sum_{i=1}^{k} \delta_i \cdot |x^{(i)}|$$

berechnet werden, das genau dann negativ ist, wenn $f(x^{(1)}, \ldots, x^{(k)}) = 0$ gilt. Repräsentieren wir s durch $(s_n, \ldots, s_0)$ mit $s_i \in \mathbb{B}$ für $i < n$ und $s_n \in \mathbb{Z}$ und $s = \sum_{i=0}^{n} s_i \cdot 2^i$, so entspricht das Vorzeichen von s dem von s_n. Auch t sei entsprechend durch $(t_n, \ldots, t_0) \in \mathbb{Z} \times \mathbb{B}^n$ repräsentiert. In Schritt i der schriftlichen Addition sind bereits s_{i-1} und der Übertrag c_i bekannt, und wir fahren fort mit

$$\begin{aligned} s_i &:= \left(c_{i-1} + t_i + \sum_{j=1}^{k} \delta_j \cdot x_i^{(j)}\right) \bmod 2 \qquad \text{und} \\ c_i &:= \left\lfloor \left(c_{i-1} + t_i + \sum_{j=1}^{k} \delta_j \cdot x_i^{(j)}\right) /2 \right\rfloor \end{aligned}$$

für $c_{-1} := 0$. So erhalten wir schließlich $S_n = c_{n-1} + t_n$.

Diesen Schritt i teilen wir weiter in die Schritte $(i,1),\ldots,(i,k)$ ein, in denen die Werte c_{i-1}, t_i und $w_j \cdot x_i^{(j)}$ summiert bzw. schließlich modulo 2 gerechnet wird. Da letztlich die Werte $s_0,\ldots,s_{n-1}$ für f nicht von Belang sind, benötigen wir in Schritt (i,j) neben der Eingabe $x_i^{(j)}$ also lediglich den bisherigen Summationszustand, der vor $(i,1)$ genau c_{i-1} ist. Aus diesem Vorgehen kann leicht ein vollständiges OBDD konstruiert werden, dessen $x_i^{(j)}$-Knoten den verschiedenen möglichen Summationszuständen vor Schritt (i,j) entsprechen. Abhängig von $x_i^{(j)}$ wird dann in den nächsten Zustand in Form eines $x_i^{(j+1)}$- bzw. $x_{i+1}^{(1)}$-Knotens gewechselt. Am Ende wird die Eingabe für $s_n \geq 0$ durch Besuch der 1-Senke akzeptiert und sonst durch Besuch der 0-Senke verworfen. Die Breite des vollständigen OBDDs entspricht also der maximalen Anzahl an Summationszuständen, die durch

$$2 \cdot \max_{0 \leq i < n} \left\{ |c_{i-1}| + t_i + \sum_{j=1}^{k} |\delta_j| \cdot x_i^j \right\} + 1 \leq 2 \cdot \max_{0 \leq i < n} \{|c_{i-1}| + t_i + kW\} + 1$$

abgeschätzt werden kann. Aus $c_{-1} = 0$ und $|c_i| \leq |c_{i-1} + kW + 1|/2$ für $1 \leq i < n$ folgt per vollständiger Induktion $|c_i| \leq kW + 1$. Die Breite von $\pi_{k,n}^{\mathrm{id}}$-OBDD$_{\mathrm{c}}[f]$ ist also durch $\mathcal{O}(kW)$ beschränkt. □

Wir weiten den Begriff des Thresholds auf allgemeine Vergleiche aus:

Definition 2.4.8 (Multivariate Vergleichsfunktionen). *Sei $f \in B_{kn}$ auf den Variablenvektoren $x^{(1)},\ldots,x^{(k)} \in \mathbb{B}^n$ definiert. Funktion f heißt* k-variate Vergleichsfunktion*, wenn es $W \in \mathbb{N}$, $t \in \mathbb{Z}$ und $\delta_1,\ldots,\delta_k \in \{-W,\ldots,W\}$ gibt, sodass*

$$f(x^{(1)},\ldots,x^{(k)}) = \left(\sum_{i=1}^{k} \delta_i \cdot |x^{(i)}| \bowtie t \right)$$

für ein $\bowtie \in \{\geq, >, \leq, <, =\}$ gilt. W heißt maximales absolutes Gewicht *von f. Die entsprechende Klasse von Funktionen wird mit $\mathbb{V}_{k,n}^{W}$ bezeichnet.*

Offenbar kann jede k-variate Vergleichsfunktion durch wenige k-variate Thresholdfunktionen realisiert werden:

Beobachtung 2.4.9. *Eine Funktion $f \in \mathbb{V}_{k,n}^{W}$ kann durch eine konstante Anzahl von binären Synthesen und Negationen auf Funktionen in $\mathbb{T}_{k,n}^{W}$ gewonnen werden.*

Somit stellen multivariate Thresholdfunktionen in späteren Kapiteln einen wichtigen Baustein bei der Konstruktion von Funktionen mit kompakten OBDDs dar. Wir definieren eine weitere Bausteinfunktion, die wir jedoch nur in einem Algorithmus verwenden werden (siehe Abschnitt 6.2).

Definition 2.4.10 (Multivariate Modulofunktionen, siehe Woelfel (2003)). *Sei $f \in B_{kn}$ auf den Variablenvektoren $x^{(1)}, \ldots, x^{(k)} \in \mathbb{B}^n$ definiert. Funktion f heißt* k-variate Modulo-M-Funktion *für $M \in \mathbb{Z}$, wenn es $W \in \mathbb{N}$, $t \in \mathbb{Z}$ und $\delta_1, \ldots, \delta_k \in \{-W, \ldots, W\}$ gibt, sodass*

$$f(x^{(1)}, \ldots, x^{(k)}) = \left(\sum_{i=1}^{k} \delta_i \cdot |x^{(i)}| \bmod M = t \right).$$

W heißt maximales absolutes Gewicht *von f. Die entsprechende Klasse von Funktionen wird mit $\mathbb{M}_{k,n}^{M}$ bezeichnet.*

Theorem 2.4.11 (Siehe Woelfel (2003)). *Für $f \in \mathbb{M}_{k,n}^{M}$ hat π-OBDD$_c[f]$ für jede Variablenordnung π Breite $\mathcal{O}(M)$, und es ist* size(π-OBDD$[f]$) $\leq$ size(π-OBDD$_c[f]$) $= \mathcal{O}(kMn)$.

Wir skizzieren die triviale Konstruktion des entsprechenden OBDDs: Die OBDD-Knoten repräsentieren stets einen Zustand $Q \in \{0, \ldots, M-1\}$, der mit 0 initialisiert wird. Nach dem Lesen einer Variablen $x_i^{(j)}$ wird Q durch $(Q + \delta_j \cdot 2^i \cdot x_i^{(j)}) \bmod M$ ersetzt. Schließlich wird für den letzten Zustand $Q = t$ die 1-Senke erreicht, ansonsten die 0-Senke.

Wir wenden uns nun vier Funktionen zu, die bekanntermaßen ungeeignet für die Repräsentation durch OBDDs sind – die OBDD-Größe ist hier für jede Variablenordnung exponentiell in der Variablenanzahl.

Definition 2.4.12 (Indirect Storage Access (ISA)). *Die* Indirect-Storage-Access-Funktion *ISA_n für $n = 2^m$ ist auf einer Menge von m Variablen $x_0, \ldots, x_{m-1}$ und einer Menge von n Variablen $y_0, \ldots, y_{n-1}$ definiert und berechnet den Wert y_α für $\alpha := |y_{|x|} \ldots y_{(|x|+m-1) \bmod n}|$.*

Theorem 2.4.13 (Siehe Breitbart u. a. (1995)). *Die Größe von π-OBDD$[ISA_n]$ ist für jede Variablenordnung π durch $2^{\lfloor n/\log n \rfloor - 1}$ nach unten beschränkt.*

Exemplarisch betrachten wir die Beweisidee für die ISA-Funktion: Wir müssen uns beim ersten und einzigen Lesen der y-Variablen bereits linear viele Werte merken, da sich die Adresse α selbst erst bei diesem Lesevorgang offenbart. Für jede Ordnung der y-Variablen können die x-Variablen so gewählt werden, dass nach dem Lesen der ersten $\lfloor n/\log n \rfloor - 1$ Variablen noch nichts über α bekannt ist.

Definition 2.4.14 (Hidden Weighted Bit (HWB)). *Die* Hidden-Weighted-Bit-Funktion *HWB_n ist auf einer Menge von n Variablen $x_1, \ldots, x_n$ definiert und berechnet den Wert x_α für $\alpha := ||x||$ und $x_0 := 0$.*

Theorem 2.4.15 (Siehe Theorem 4.10.2 in Wegener (2000)). *Die Größe von π-OBDD$[HWB_n]$ ist für jede Variablenordnung π durch $\Omega(2^{n/5})$ nach unten beschränkt.*

Definition 2.4.16 (Multiplikation (MUL)). *Die* Multiplikationsfunktion $MUL_{n,i}$ *ist auf zwei Variablenmengen* $\{x_0, \ldots, x_{n-1}\}$ *und* $\{y_0, \ldots, y_{n-1}\}$ *definiert und berechnet das Bit* z_i *der Binärdarstellung* $z_{2n-1}, \ldots, z_0$ *des Produkts* $|x| \cdot |y|$.

Theorem 2.4.17 (Siehe Woelfel (2001)). *Die Größe von* π-OBDD$[MUL_{n,n-1}]$ *ist für jede Variablenordnung* π *durch* $2^{n/2}/61$ *nach unten beschränkt.*

Definition 2.4.18 (Quadrieren (SQU)). *Die* Quadrierungsfunktion $SQU_{n,i}$ *ist auf einer Variablenmenge* $\{x_0, \ldots, x_{n-1}\}$ *definiert und berechnet das Bit* z_i *der Binärdarstellung* $z_{2n-1}, \ldots, z_0$ *des Quadrats* $|x|^2$.

Theorem 2.4.19 (Siehe Theorem 4.6.2 in Wegener (2000)). *Das OBDD* π-OBDD$[SQU_{n,\alpha}]$ *mit* $\alpha := 3\lfloor n/3 \rfloor - 1$ *und* $n \geq 3$ *hat für jede Variablenordnung* π *exponentielle Größe in* n.

Obwohl die Multiplikation im klassischen algorithmischen Szenario eine einfache Funktion ist, stellt sie im Zusammenhang mit der Verifikation entsprechender Schaltkreise ein schwieriges Problem dar. Wir werden eine schwächere untere Schranke für $MUL_{n,n-1}$ (dem *mittleren Bit der Multiplikation*) später auf eine Variante der Funktion, den Graphen der Multiplikation, übertragen (siehe Abschnitt 4.3) sowie die erste untere Schranke für das höchste Bit der Multiplikation beweisen und anwenden (siehe Abschnitt 7.3). Für den Spezialfall des Quadrierens gelingt eine Übertragung der Schranke nur eingeschränkt.

2.5 Implizite Graphdarstellung und implizite Graphalgorithmen

Wir schlagen nun eine Brücke zur impliziten Repräsentation von Graphen durch boolesche Funktionen und insbesondere OBDDs, um schließlich genauer einfassen zu können, was implizite Graphalgorithmen ausmacht.

2.5.1 Implizite Graphdarstellung

Bei der impliziten Darstellung von Daten soll die explizite Aufzählung einzelner Elemente (z. B. Knoten und Kanten) vermieden werden, um die potenzielle Möglichkeit einer stark sublinearen Darstellungsgröße zu eröffnen. Das in dieser Dissertation betrachtete Vorgehen ist, die Daten stets als Teilmenge M einer Grundmenge $\{0, \ldots, N-1\}^k$ zu modellieren, bei der sich die Datumsgröße in N widerspiegelt, und k konstant ist. Sei also M dementsprechend definiert.

Definition 2.5.1. *Die* charakteristische Funktion $\chi_M \in B_{kn}$ *von* M *mit* $n := \lceil \log N \rceil$ *ist definiert durch*

$$\chi_M(x^{(1)}, \ldots, x^{(k)}) = 1 :\Leftrightarrow (|x^{(1)}|, \ldots, |x^{(k)}|) \in M$$

für $x^{(1)}, \ldots, x^{(k)} \in \mathbb{B}^n$.

Ungerichtete Kanten stellen wir einfach als gerichtete symmetrische Kanten dar. Im Falle eines ungewichteten Graphen $G = (V, E)$ besteht Gs implizite Darstellung aus $\chi_E(x, y) = 1 \Leftrightarrow (v_{|x|}, v_{|y|}) \in E$ für Knotennummern $x, y \in \mathbb{B}^n$ für $n := \lceil \log |V| \rceil$ und wird im Folgenden mit χ_G bezeichnet. Bei Graphen mit Kantengewicht $\phi\colon E \to \{0, \ldots, B-1\}$ erweitern wir die Variablenmenge um $\{a_0, \ldots, a_{m-1}\}$, sodass

$$\chi_G(x, y, a) = 1 \Leftrightarrow (v_{|x|}, v_{|y|}) \in E \wedge \phi(v_{|x|}, v_{|y|}) = |a|$$

für $m = \lceil \log B \rceil$ gilt.

Für $n \neq m$ haben wir damit aber Definition 2.5.1 verletzt. Wir gehen also im Folgenden von $n = m := \lceil \max\{\log |V|, \log B\} \rceil = \log(|V| \cdot B) + \mathcal{O}(1)$ aus. So kann der implizit dargestellte Graph mehr Knoten enthalten als G, und wir führen eine weitere Funktion $\chi_V \in B_n$ ein mit $\chi_V(x) := (v_{|x|} \in V)$, die gemeinsam mit χ_G die implizite Darstellung von G ergibt.

Beobachtung 2.5.2. *Die charakteristische Funktion χ_G eines ungewichteten gerichteten Graphen G ist auf höchstens $2 \log |V| + \mathcal{O}(1)$ booleschen Variablen definiert. Aus Theorem 2.3.9 folgt direkt, dass $\mathcal{O}(|V|^2 / \log |V|)$ eine obere Schranke für die OBDD-Größe von χ_G ist – diese Darstellung ist also nicht wesentlich schlechter als die durch Adjazenzmatrizen. Ebenso ist leicht ersichtlich, dass die π-OBDD-Größe einer Funktion $f \in B_n$ mit $|f^{-1}(1)| = M$ durch $\mathcal{O}(M \cdot n)$ beschränkt ist. Für π-OBDD$[\chi_G]$ folgt also die Schranke $\mathcal{O}(|E| \log |V|)$, die wiederum nicht wesentlich größer als bei einer Adjazenzlistendarstellung ist. Für gewichtete gerichtete Graphen G mit maximalem Gewicht $B - 1$ ist χ_G dann jedoch auf $3 \cdot \max\{\log |V|, \log B\} + \mathcal{O}(1)$ Variablen definiert, und die entsprechenden Schranken sind $\mathcal{O}((\max\{|V|, B\})^3 / \log(|V| \cdot B))$ und $\mathcal{O}(|E| \log(|V| \cdot B))$.*

Andererseits ist χ_G auf mindestens $2 \log |V|$ booleschen Variablen definiert. Laut Korollar 2.3.11 haben schließlich gleichverteilt zufällige gerichtete ungewichtete Graphen erwartete Größe $\Omega(|V|^2 / \log |V|)$. Für den Fall konstanter Kantenwahrscheinlichkeit $p \in]0, 1[$ ergibt sich die entsprechende untere Schranke von $\Omega(|V|^{2-\varepsilon})$. Da in gewichteten Graphen jeder Kante nur ein Kantengewicht zugeordnet wird, existiert keine Bijektion zwischen zufälligen Graphen mit zufälligen Kantengewichten und zufälligen Funktionen auf mindestens $3 \cdot \lceil \max\{\log |V|, \log B\} \rceil$ Variablen. Wir müssen hier also mit den gleichen unteren Schranken wie bei zufälligen ungewichteten Graphen vorliebnehmen.

Leicht ersichtlich gelten diese asymptotischen Ergebnisse ebenso für schlaufenfreie, asymmetrische und ungerichtete Graphen.

2.5.2 Implizite Graphalgorithmen

Wir wenden uns nun dem Begriff des impliziten Graphalgorithmus zu, indem wir zunächst grob allgemeine implizite Algorithmen definieren:

Definition 2.5.3. *Ein* impliziter Algorithmus *ist ein Algorithmus, dessen Eingabe I der Größe N und Ausgabe O unter anderem aus charakteristischen Funktionen von Teilmengen von $\{0, \ldots, N-1\}^k$ für konstantes k bestehen.*

Diese Definition ist durch das heuristische Ziel impliziter Algorithmen motiviert, sublineare Laufzeit auf großen, aber strukturierten Eingaben zu erreichen, indem Eingabe und Ausgabe im Wesentlichen durch charakteristische Funktionen repräsentiert werden; die Ausgabe soll im Wesentlichen durch funktionale Operationen berechnet werden. Jedoch lässt die Definition noch vieles offen: Die Problemgröße N muss nicht der Länge einer Codierung von I entsprechen, sondern kann auch eine abstrakte problemabhängige Größe wie z. B. die Knotenanzahl eines ungewichteten Graphen sein. Bei gewichteten Graphen $G = (V, E, \phi)$ mit Gewichtsschranke $B - 1$ ist N sogar mindestens $|V|B$, während der Faktor B in klassischen Algorithmen höchstens logarithmisch in die Problemgröße eingeht.

Damit letztlich die Anzahl und die Komplexität der funktionalen Operationen für die Gesamteffizienz eines impliziten Algorithmus entscheidend sind, sollten Eingabe und Ausgabe im Wesentlichen durch charakteristische Funktionen dargestellt sein – auch jeder klassische Algorithmus kann schließlich als impliziter aufgefasst werden, der keinen Gebrauch von der Möglichkeit charakteristischer Funktionen macht. Lediglich einzelne ergänzende Informationen wie z. B. die Nummern der Knoten q und s bei der ***q*-*s*-Pfad-Suche** können zwecks komfortabler Notation als einfache Zahlen oder Zeichenketten dargestellt sein.

Zudem legt Definition 2.5.3 noch nicht fest, wie die charakteristischen Funktionen repräsentiert werden sollen oder wie die Eingabe in die Ausgabe überführt werden darf. Wir werden einerseits meistens davon ausgehen, dass OBDDs zur Darstellung boolescher Funktionen verwendet werden, während die Beschreibung konkreter impliziter Algorithmen sowie einige komplexitätstheoretische Resultate andererseits davon unabhängig sein werden.

Im Hinblick auf unser Ziel, polylogarithmische Heuristiken zu entwerfen, sollte vermieden werden, wesentlich mehr als polylog(N) Objekte einzeln zu behandeln. Statt dessen sollte versucht werden, die Behandlung vieler Elemente (z. B. Knoten und Kanten) quasi parallel durch wenige funktionale Operationen zu realisieren, wie wir sie in Definition 2.3.16 eingeführt haben. Von diesen wiederum erhoffen wir uns, dass jede einzelne durch vorteilhafte Eigenschaften von Datenstruktur (z. B. effiziente OBDD-Operationen) und repräsentierter Funktionen (z. B. kleine OBDD-Größe) effizient ist. Es ergibt sich ein zweistufiges Entwurfsparadigma:

1. Entwirf einen impliziten Algorithmus, der „wenige“ funktionale Operationen auf charakteristischen Funktionen ausführt und auch ansonsten nur „wenig“ Laufzeit benötigt. Gewichtet man jede funktionale Operation mit konstanten Laufzeitkosten, so erwünschen wir uns genauer Gesamtlaufzeit polylog(N).

2. Im zweiten Schritt interessieren wir uns für die tatsächliche Laufzeit jeder einzelnen funktionalen Operation. Hat jede Operation (wie bei OBDDs) polynomielle Laufzeit bez. der Operandengröße, so möchten wir die maximale Größe aller repräsentierten charakteristischen Funktionen zumindest für Eingabeinstanzen mit gewissen Einschränkungen begrenzen. Dies kann natürlich wieder zu einer Entwurfsänderung in Schritt 1 zurückführen.

Zur Verdeutlichung dieses Modells und zur Einführung üblicher Schreibweisen stellen wir zwei einfache implizite Graphalgorithmen für das Problem ***q*-Erreichbarkeit** vor. Algorithmus 2.5.1 stellt eine implizite Breitensuche dar. χ_R wird am Ende des Algorithmus die charakteristische Funktion der Menge aller von q erreichbaren Knoten sein. Die Hilfsfunktion NEW stellt die in einer Iteration i neu erreichten Knoten dar, die von q aus auf einem kürzesten Pfad der Länge i erreichbar sind.

Algorithmus 2.5.1 : Impliziter BFS-Ansatz für ***q*-Erreichbarkeit**.

Eingabe : Impliziter Graph $\chi_G(x, y)$, Knoten q.
Ausgabe : Implizite Menge $\chi_R(x)$ der erreichbaren Knoten.

```
1 χ_R(x) := (v_|x| = q);
2 repeat
3 |   NEW(x) := (∃y) [χ_R(y) ∧ χ_G(y, x) ∧ ¬χ_R(x)];
4 |   χ_R(x) := χ_R(x) ∨ NEW(x);
5 until NEW(x) = 0 ;
6 return χ_R(x);
```

Bei in impliziten Algorithmen eingeführten Hilfsfunktionen wie $NEW(x)$ verzichten wir im Sinne einer einfacheren Notation auf die χ-Schreibweise: NEW bezeichnet dann die dargestellte explizite Menge, $NEW(x)$ die charakteristische Funktion. $NEW(x)$ ist also nicht als Funktionswert zu verstehen; x stellt nicht eine Variablenbelegung dar, sondern kennzeichnet die Variablenmenge, auf denen die Funktion definiert ist. Daher werden wir separate Bezeichner für konkrete Belegungen verwenden. Diese Schreibweise hat sich etabliert und eignet sich insbesondere für die übersichtliche Handhabung von Variablenumbenennungen, die weiter unten diskutiert werden.

In jeder Iteration werden die neu gewonnenen Knoten in $NEW(x)$ zu $\chi_R(x)$ hinzugefügt (Zeile 4). Wir betrachten die Berechnung von $NEW(x)$ in Zeile 3 genauer: Ein Knoten $v_{|x|}$ ist genau dann in der Menge neu erreichter Knoten, wenn er bisher noch nicht in R enthalten ist ($\overline{\chi_R(x)}$), jedoch durch eine Kante ($\chi_G(y, x)$), von einem bereits in R enthaltenen Knoten y ($\exists y$, $\chi_R(y)$) erreichbar ist.

Wir haben das Problem ***q*-Erreichbarkeit** mithilfe von funktionalen Operationen gelöst. In jeder Iteration fallen $\mathcal{O}(n) = \mathcal{O}(\log N)$ Operationen an, da $(\exists y)$ eigentlich n einzelnen Quantifizierungen entspricht. Die Anzahl der Iterationen ist jedoch die Länge des längsten Pfades von q zu einem anderen Knoten, die wiederum $\Omega(N)$ sein kann.

Damit bearbeiten wir zwar alle Knoten einer BFS-Ebene gemeinsam durch nur logarithmisch viele Operationen, behandeln die einzelnen Ebenen jedoch sequenziell. Um stets eine polylogarithmische Anzahl von Operationen zu gewährleisten, macht Algorithmus 2.5.2 von der Technik des iterativen Quadrierens Gebrauch. Diese kann als dynamische Programmierung verstanden werden, bei der sich die betrachtete Teilproblemgröße in jeder Iteration verdoppelt, sodass letztlich nur logarithmisch viele Iterationen benötigt werden. Wir werden dieses Paradigma später auch zur Berechnung maximaler Flüsse (Abschnitt 3.3) und kürzester Wege (Abschnitt 5.1) verwenden.

Algorithmus 2.5.2 : Impliziter Iterative-Quadrieren-Ansatz für ***q*-Erreichbarkeit**.

Eingabe : Impliziter Graph $\chi_G(x, y)$, Knoten q.
Ausgabe : Implizite Menge $\chi_R(x)$ der erreichbaren Knoten.

```
1 A(x, y) := (x = y) ∨ χ_G(x, y);
2 repeat
3 |   A'(x, y) := A(x, y);
4 |   A(x, y) := (∃z) [A'(x, z) ∧ A'(z, y)];
5 until A(x, y) = A'(x, y) ;
6 χ_R(x) := A(⟨q⟩_2, x);
7 return χ_R(x);
```

Im Folgenden gebe für ein $v_i \in V$ der Ausdruck $\langle v_i \rangle$ die Nummer i des Knotens an, während $\langle v_i \rangle_2$ die Binärdarstellung von i sei, deren Länge sich jeweils aus dem Kontext erschließt. Wir betrachten Algorithmus 2.5.2: Eigentlich wird hier sogar das Problem **Transitiver Abschluss** gelöst, dessen Lösung $A(x, y)$ alle Knotenpaare $(v_{|x|}, v_{|y|})$ mit $v_{|x|}$-$v_{|y|}$-Pfaden repräsentiert. Daraus wird in Zeile 6 χ_R extrahiert – wir belegen die Startknotenvariablen mit $\langle q \rangle_2$. Doch wie erhalten wir zuvor $A(x, y)$? Wir initialisieren $A(x, y)$ in Zeile 1 so, dass anfangs $v_{|x|}$-$v_{|y|}$-Pfade der Längen 0 und 1 abgedeckt werden. Nun verdoppeln wir die maximale Pfadlänge in jeder Iteration. Deckt $A(x, y)$ bisher Pfade der maximalen Länge i ab, und existiert ein Knoten z, sodass die Existenz entsprechend längenbeschränkter $v_{|x|}$-$v_{|z|}$- und $v_{|z|}$-$v_{|y|}$-Pfade bekannt ist, dann existiert auch ein $2i$-längenbeschränkter x-y-Pfad. Schließlich brechen wir ab, wenn ein Fixpunkt erreicht wurde und keine neuen verbundenen Knotenpaare hinzugefügt wurden. Dies ist offensichtlich nach $\mathcal{O}(\log N)$ Iterationen der Fall. Wieder werden pro Iteration $\mathcal{O}(n) = \mathcal{O}(\log N)$ Operationen ausgeführt, insgesamt nun also $\mathcal{O}(\log^2 N)$.

Beobachtung 2.5.4. *In Algorithmus 2.5.1 und Algorithmus 2.5.2 können wir χ_V außer Acht lassen.*

Die Übertragung der hier betrachteten Graphprobleme ***q*-Erreichbarkeit** und **Transitiver Abschluss** ins implizite Szenario geschah dabei möglichst kanonisch:

Eine Menge R von erreichbaren Knoten soll berechnet werden – hier also deren charakteristische Funktion χ_R. Ein Graph $A = (V, E')$ soll berechnet werden – hier also dessen charakteristische Funktion $A(x, y)$ entsprechend Definition 2.5.1. Dies entspricht zudem der Definition anwendungsspezifischer Probleme in den Bereichen Verifikation (siehe z. B. Abschnitt 13.2 in Wegener (2000)) und Model Checking (siehe z. B. Abschnitt 13.3 in Wegener (2000)). Bei Problemen, die noch keine impliziten Entsprechungen in Anwendungen haben, werden wir ebenfalls versuchen, ihre impliziten Versionen sinnvoll und kanonisch zu definieren.

2.5.3 Weitere Vorgehensweisen bei OBDD-basierten impliziten Algorithmen

Zwei Aspekte der beiden obigen Algorithmen haben wir noch nicht näher im Hinblick auf ihre Realisierung mit OBDDs betrachtet: Die jeweiligen Zeilen 1 enthalten eine Vergleichsfunktion der Art $(x = y)$. Zudem werden die Variablenvektoren stellenweise umbenannt (z. B. in Algorithmus 2.5.2, Zeile 4: $A(x, z)$ statt $A(x, y)$). Die Leserin und der Leser können sich jedoch leicht überlegen, dass der Vergleich $(x = y)$ OBDDs konstanter vollständiger Breite besitzt, wenn die Vektoren x und y mit gemischter Variablenordnung gelesen werden. Die Funktion kann zudem mithilfe von multivariaten Thresholdfunktionen ausgedrückt werden.

Wir wollen im Folgenden davon ausgehen, dass sinnvolle implizite Algorithmen nur $\mathcal{O}(\log N)$ boolesche Variablen insgesamt verwenden, die sie bereits zu Anfang initialisieren. Da wir von der Darstellung der Funktionen in einem SBDD ausgehen, sind also alle Funktionen auf dieser von Anfang an festgelegten Variablenmenge definiert. In der Schreibweise $\chi_G(x, y)$ spiegeln sich jedoch jeweils nur die Variablenvektoren wider, die für die Interpretation der dargestellten Daten relevant sind. Die Hilfsvariablen z spielen z. B. für χ_G alleine keine Rolle. Diese Vereinbarungen stellen keine gravierenden Einschränkungen unseres Modells dar, da sie sowohl von allen impliziten Algorithmen in der Literatur erfüllt werden, als auch die maximale OBDD-Größe entsprechend Theorem 2.3.9 durch $2^{\mathcal{O}(\log N)} = \mathrm{poly}(N)$ polynomiell in der expliziten Problemgröße N beschränken.

Nun zu der Umbenennung: Variablenvektoren können auch als *Argumente* einer charakteristischen Funktion aufgefasst werden und werden im Folgenden auch als solche bezeichnet. Wir können die einzelnen Variablen der Argumente nicht einfach umbenennen, da dies die Variablenordnung verändern würde. Wir betrachten eine Funktion $f \in B_{kn}$, die auf k Argumenten $x^{(1)}, \ldots, x^{(k)} \in \mathbb{B}^n$ definiert ist, und führen eine weitere Operation ein, die im Kontext impliziter Algorithmen benötigt wird. Sei $\rho \in \Sigma_k$.

Definition 2.5.5 (Argumentumordnung). *Die* Argumentumordnung $\mathcal{R}_\rho(f) \in B_{kn}$ *ist definiert durch*

$$\mathcal{R}_\rho(f)\big(x^{(1)}, \ldots, x^{(k)}\big) = f\big(x^{(\rho(1))}, \ldots, x^{(\rho(k))}\big).$$

Umordnungen wie in Algorithmus 2.5.2, Zeile 4, können durch eine Argumentumordnungsoperation realisiert werden. Sei $\tau \in \Sigma_n$.

Theorem 2.5.6. *Das OBDD* $\pi^{\tau}_{k,n}$-OBDD$[\mathcal{R}_\rho(f)]$ *ist nicht größer als* $3^k \operatorname{size}(\pi^{\tau}_{k,n}\text{-OBDD}[f])$ *und kann in Zeit* $\mathcal{O}(k3^k \operatorname{size}(\pi^{\tau}_{k,n}\text{-OBDD}[f]))$ *auf Platz* $\mathcal{O}(3^k \operatorname{size}(\pi^{\tau}_{k,n}\text{-OBDD}[f]))$ *berechnet werden.*

Beweis: Die Umbenennung der Variablenmarkierungen im OBDD geschieht in linearer Zeit. Nun muss die Variablenordnung wieder hergestellt werden. Jede Variable $x_i^{(j)}$ kann zu ihrer neuen Position $\rho(j)$ inmitten des Bitblocks i mithilfe einer Sequenz $S_{i,j}$ von höchstens $k-1$ Vertauschungen benachbarter Variablen gebracht werden (siehe Definition 5.7.1(iii) in Wegener (2000)). Durch eine solche Sequenz wird entsprechend der Operation Variablentausch im Beweis von Theorem 2.3.17 (Punkt 11) lediglich die Anzahl der OBDD-Knoten in Bitblock i höchstens verdreifacht. Die Sequenzen $S_{0,j}, \ldots, S_{n-1,j}$ können also unabhängig betrachtet werden; sie haben nach Theorem 5.7.4 in Wegener (2000) nur lokale Auswirkungen. Für jede Sequenzenfolge $S_{0,j}, \ldots, S_{n-1,j}$ mit $1 \leq j \leq k$ verdreifacht sich also die Gesamtgröße höchstens, und es folgt die Aussage über die letztendliche OBDD-Größe.

Ein entsprechender Algorithmus kann alle Vertauschungen in den Sequenzen $S_{0,j}, \ldots, S_{n-1,j}$ für ein $j \leq k$ en bloc durchführen; diese geschehen aufgrund der Unabhängigkeit der Bitblöcke in linearer Gesamtlaufzeit bez. der resultierenden OBDD-Größe, die wiederum durch die Größe des Endergebnisses $\pi^{\tau}_{k,n}$-OBDD$[\mathcal{R}_\rho(f)]$ beschränkt ist. Somit ergibt sich die Laufzeitaussage. Der Platz ist offensichtlich zu jedem Zeitpunkt durch die Größe des Endergebnisses beschränkt. □

Wir werden im Folgenden zwischen zwei Sichten unterscheiden:

1. Aus Sicht eines OBDD-basierten impliziten Algorithmus wird die Eingabefunktion χ_G für ungewichtete Graphen auf $2n$ Variablen für x- und y-Argument definiert; für gewichtete Graphen entsprechend auf $3n$ Variablen. Die gemischte Variablenordnung liest also alternierend die Variablen der zwei bzw. drei Argumente, sodass die Argumentumordnung effizient realisiert werden kann.

2. Aus Sicht der Algorithmenanalyse wissen wir evtl. mehr als der Algorithmus, nämlich dass die x-, y- und a-Argumente wiederum aus einer konstanten Anzahl von Subvektoren bestehen, die so eine Darstellung von χ_G oder anderen betrachteten Funktionen durch Komposition von multivariaten Thresholdfunktionen möglich machen. Wir werden dieser Sichtweise erstmals in Abschnitt 3.8 begegnen, wo die Knotennummern in Gittergraphen aus Zeilen- und Spaltenkomponenten bestehen.

Diese Sichtweisen werden durch unterschiedliche Argumentbezeichner voneinander getrennt werden; x, y und a bezeichnen stets die Argumente aus Algorithmensicht. Aus

Algorithmensicht ist die gemischte Ordnung der Subvektoren in der lokalen Ordnung τ entsprechend Definition 2.4.6 verborgen, während wir bei einer Analyse die Ordnung aller tatsächlichen Argumente betrachten.

Schließlich wollen wir noch auf die OBDD-Größe der Funktion χ_V eingehen, wenn die Variablen eines Knotenarguments x tatsächlich aus Subvektoren $x^{(1)}, \ldots, x^{(k)}$ bestehen.

Anmerkung 2.5.7. *Typischerweise entstehen die zusätzlichen Knoten der impliziten Darstellung durch Aufrunden der Variablenanzahl oder durch Angleichung der Variablenanzahl zur Codierung von Knoten und Kantengewichten. Dann ist also*

$$\chi_V(x) := \bigwedge_{j=1}^{k} \left(|x^{(j)}| < t_j\right)$$

für Schwellenwerte $t_1, \ldots, t_k \in \mathbb{N}$. Aus Beobachtung 2.4.9 folgt, dass χ_V durch $\mathcal{O}(k)$ binäre Synthesen multivariater Thresholdfunktionen in $\mathbb{T}_{k,n}^{\mathcal{O}(1)}$ erhalten werden kann, die wiederum nach Theorem 2.3.17 (Punkt 9) und Theorem 2.4.7 zu einer $\pi_{k,n}^{\mathrm{id}}$-OBDD-Größe von $(k^2 n)^{\mathcal{O}(k)}$ führen. Bei genauerer Betrachtung lässt sich die Größenschranke aber leicht auf $\mathcal{O}(2^k kn)$ verbessern. Aus späteren Resultaten wird jedoch direkt eine Verbesserung auf $2^{\mathcal{O}(k \log k)} n$ folgen, mit der wir zufrieden sein werden, da k typischerweise konstant ist.

Noch einfacher ist es, wenn zu jedem Argument $x^{(j)} \in \mathbb{B}^n$ einer Knotennummer $x \in \mathbb{B}^{kn}$ lediglich jeweils m weitere Bits hinzugefügt werden – χ_V muss $x_{n+m-1}^{(j)}, \ldots, x_n^{(j)} = 0$ für $j := 1, \ldots, k$ sicherstellen. Dies kann offenbar leicht durch eine Kette von m OBDD-Knoten realisiert werden.

Nach einer derartigen Erzeugung von zusätzlichen Knoten muss auch die Funktion χ_G zu $\chi_G(x, y) \wedge \chi_V(x) \wedge \chi_V(y)$ angepasst werden. Für $k = \mathcal{O}(1)$ wächst die OBDD-Größe von χ_V also um den Faktor $\mathcal{O}(n)$, später werden wir diesen Faktor auf $\mathcal{O}(1)$ verbessern.

Auch wenn zusätzliche isolierte Knoten und die Funktion χ_V für die meisten betrachteten Probleme und Algorithmen in dieser Dissertation keine Rolle spielen werden, sollten sie dennoch als Bestandteil unserer Sichtweise auf allgemeine implizite Algorithmen eingeführt werden.

Obwohl wir die Definition OBDD-basierter Algorithmen in dieser Dissertation nicht grundsätzlich an gemischte Variablenordnungen binden wollen (insbesondere nicht beim Beweis unterer Laufzeitschranken), so gehen wir also im Interesse kompakter Bausteinfunktionen (z. B. $(x = y)$) und effizienter Argumentumordnungen von ihrer Verwendung aus. Wir erlauben impliziten Algorithmen die Verwendung von grundlegenden Hilfsfunktionen, wenn sie für natürliche gemischte Variablenordnungen $\pi_{k,n}^{\mathrm{id}}$ mit konstantem k kleine und effizient zu erzeugende OBDDs besitzen. In der Praxis ist des Weiteren die Anwendung von Heuristiken zur Optimierung der Variablenordnung üblich; dies geschieht typischerweise, wenn die Gesamt-SBDD-Größe einen

gewissen Schwellenwert überschreitet. Die in dieser Dissertation vorgestellten neuen Algorithmen werden davon keinen Gebrauch machen. Andererseits werden unsere unteren Schranken bis auf eine Ausnahme für alle Variablenordnungen gelten und daher nicht von der Verwendung entsprechender Optimierungsverfahren tangiert werden.

3 Maximale Flüsse in 0-1-Graphen

Wir beginnen den Hauptteil der Dissertation mit einem impliziten Algorithmus für die **Flussmaximierung** in 0-1-Graphen – einem der ersten anwendungsunabhängigen graphtheoretischen Probleme, für das solch ein Algorithmus von Hachtel und Somenzi (1997) vorgestellt wurde. Obwohl die Autoren in Experimenten auf dichten Graphen die Laufzeit klassischer Verfahren mit ihrem OBDD-Ansatz schlagen konnten, geht dieser inhärent sequenziell vor: Durch die sukzessive Betrachtung einzelner Graphschichten, deren Anzahl $\Omega(|V|)$ sein kann, wird eine sublineare Gesamtlaufzeit bereits unabhängig von den OBDD-Größen vereitelt. Der in Abschnitt 3.3 vorgestellte Algorithmus umgeht dieses Problem durch die Anwendung von iterativem Quadrieren. Seine grundlegende Idee ist, durch die iterative Verdopplung der Länge konstruierter Pfade nur polylogarithmisch viele funktionale Operationen zu benötigen. Dies kann nicht für alle Eingaben garantiert werden, ist jedoch nicht zwangsweise an einen kleinen Durchmesser der Graphen gebunden.

Für einige Probleme wie die Berechnung des **Transitiven Abschlusses** lassen sich sehr einfach Iterative-Quadrieren-Ansätze angeben (siehe Algorithmus 2.5.2). In der Praxis ist dieser Ansatz jedoch recht unbeliebt, da es sich gezeigt hat, dass die Größe der dabei entstehenden OBDDs den Vorteil der geringeren Operationsanzahl wieder aufhebt und letztlich sequenzielle Verfahren wie Algorithmus 2.5.1 überlegen sind (siehe z. B. Bloem u. a. (2000) und Hojati u. a. (1993)). Im Sinne unseres Zieles, polylogarithmische Laufzeit auf speziell strukturierten Instanzen zu erreichen, verfolgen wir dennoch diesen Ansatz als einzige bekannte Möglichkeit, die Anzahl der OBDD-Operationen wesentlich zu verringern. Der Algorithmus ist eine Weiterentwicklung eines Verfahrens in Sawitzki (2002), dessen Pfadkonstruktion nur dann polylogarithmische Operationsanzahl garantieren kann, wenn die Niveaunetzwerktiefe eine Zweierpotenz ist. Von dieser Bedingung werden wir uns lösen und den Algorithmus dadurch sogar vereinfachen. Schließlich werden wir eine Technik zur Analyse der Gesamtlaufzeit OBDD-basierter Algorithmen vorstellen und nachweisen, dass der vorgestellte Flussalgorithmus tatsächlich wesentlich effizienter als bisherige Verfahren sein kann. Diese Technik wird auch im weiteren Verlauf dieser Dissertation eine zentrale Rolle spielen. Ähnliche Resultate erzielt Woelfel (2003) mit einem Iterative-Quadrieren-Ansatz für implizites **Topologisches Sortieren**.

3.1 Flussmaximierung mit Niveaunetzwerken

Sowohl der Algorithmus von Hachtel und Somenzi (1997) (im Folgenden *HS-Algorithmus* genannt) als auch der im nächsten Abschnitt vorgestellte (im Folgenden *IQ-Algorithmus* genannt) verwenden Niveaunetzwerke. Verschiedene Niveaunetzwerkalgorithmen und ihre Eigenschaften können in Ahuja u. a. (1993) oder Even (1979) nachgelesen werden. Wir werden ihre Funktionsweise nun knapp und ohne Beweise vorstellen. Gemäß Beobachtung 2.1.23 wollen wir für einen Graphen $G = (V, E)$ und $q, s \in V$ die Kantenmenge F einer maximalen Menge kantendisjunkter q-s-Pfade in G berechnen. Es wird zwecks einfacherer Notation typischerweise davon ausgegangen, dass G asymmetrisch ist; dies kann leicht durch Einfügung von höchstens $|E|$ Hilfsknoten erreicht werden.

Niveaunetzwerkalgorithmen beginnen mit einem leeren Fluss $F := \emptyset$, der in *Phasen* verbessert wird. Im entsprechenden *Restnetzwerk* $G^F := (V, E^F)$ sind alle momentanen Flusskanten umgekehrt, also

$$E^F := \big\{(u,v) \mid (u,v) \in (E \setminus F)\big\} \cup \big\{(u,v) \mid (v,u) \in F\big\}. \tag{3.1.1}$$

Pfade in G^F heißen *Restpfade*; *(fluss)verbessernde Pfade* sind Rest-q-s-Pfade. Dabei nennen wir hier vereinfacht die Kantenmenge eines verbessernden Pfades p ebenfalls p. Wir erhalten den *verbesserten Fluss* $F \diamond p$ durch Hinzufügung der *Vorwärtskanten* $p \cap E$ und Entfernung der *Rückwärtskanten* $p \cap (E^F \setminus E)$ (siehe Abbildung 3.1.1):

$$F \diamond p := \big\{(u,v) \mid (u,v) \in E, (u,v) \in p\big\} \cup \big\{(u,v) \mid (u,v) \in F, (v,u) \notin p\big\}. \tag{3.1.2}$$

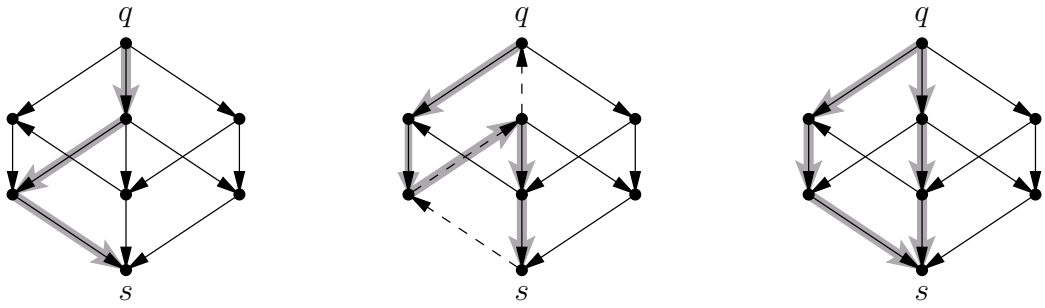
Das Ergebnis ist wieder ein gültiger Fluss, dessen Wert um 1 erhöht wurde.

Ein *Niveaunetzwerk* U^F stellt nun die Erweiterung dieser Ideen dar; es enthält alle kürzesten flussverbessernden Pfade entsprechend dem aktuellen Fluss F. U^F wird in Knoten- und Kantenebenen eingeteilt: Knotenebene m enthält alle Knoten v, die von einem kürzesten verbessernden Pfad besucht werden, und deren kürzester Rest-q-v-Pfad Länge m hat. Diese Knoten seien in Menge V_m^F. Kantenebene m enthält dann alle Kanten in E^F, die von Knoten in Ebene m zu solchen in Ebene $m+1$ gerichtet sind; sie seien in der Menge U_m^F.

$$U_m^F := \big\{(u,v) \in V_m^F \times V_{m+1}^F \mid (u,v) \in E^F\big\}$$

Die Länge des kürzesten verbessernden Pfades sei ℓ; sie entspricht der Niveaunetzwerktiefe. In jeder Phase berechnet ein Niveaunetzwerkalgorithmus U^F und ermittelt eine nicht vergrößerbare Menge B^F kantendisjunkter q-s-Pfade darin, die *Sperrfluss* heißt (siehe Abbildung 3.1.2).

Der aktuelle Fluss F wird sukzessive zu $F^{\text{neu}} := F \diamond B^F$ verbessert, wobei jeder Pfad in B^F den Flusswert um 1 erhöht. Es ist bekannt, dass F^{neu} genau dann maximal ist, wenn kein weiterer verbessernder Pfad existiert (MaxFlow-MinCut-Theorem). Dann

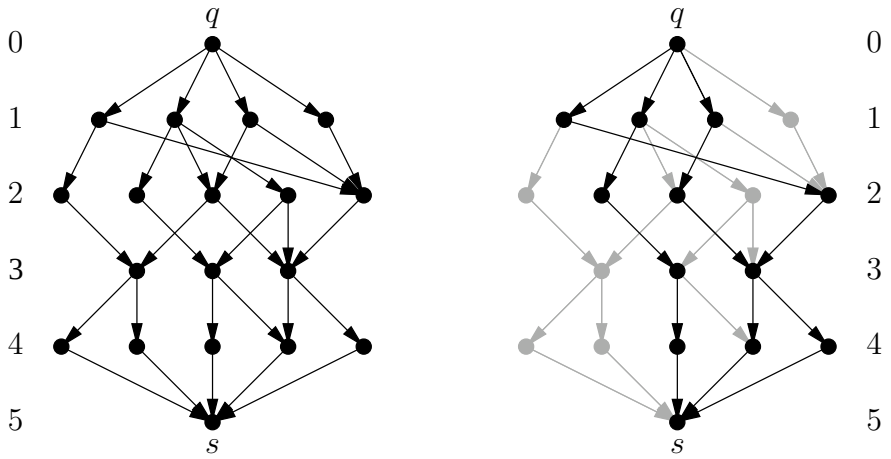


(a) Aktueller Fluss F.

(b) Flussverbessernder Pfad p.

(c) Verbesserter Fluss $F \diamond p$.

Abbildung 3.1.1: Beispielhafte Flussverbesserung. Die Untertitel der Abbildungen entsprechen dabei den grauen Objekten. Rückwärtskanten sind gestrichelt gezeichnet.



(a) U^F.

(b) B^F.

Abbildung 3.1.2: Beispielhaftes Niveaunetzwerk der Tiefe 5 und ein entsprechender Sperrfluss (schwarz).

Algorithmus 3.1.1 : Der Grobaufbau von Niveaunetzwerkalgorithmen.

Eingabe : Graph $G = (V, E)$, Knoten $q, s \in V$.
Ausgabe : Maximaler q-s-Fluss F.

```
1 F := ∅;
2 while flussverbessernder Pfad existiert do
3 |   Konstruiere Niveaunetzwerk U^F;
4 |   Berechne Sperrfluss B^F;
5 |   F := F ⋄ B^F;
6 return F;
```

stoppt der Algorithmus; sonst startet eine neue Phase. Algorithmus 3.1.1 zeigt den Grobaufbau von Niveaunetzwerkalgorithmen.

Es kann gezeigt werden, dass durch diese Vorgehensweise höchstens $|V| - 1$ Phasen benötigt werden, da ihre Niveaunetzwerktiefe bekanntermaßen streng monoton wächst. Klassische Algorithmen auf Adjazenzlistendarstellungen können U^F mithilfe einer Breitensuche aufbauen, während B^F dann mit einem Tiefensuchenansatz berechnet werden kann. Im Falle von 0-1-Graphen ist beides in Zeit $\mathcal{O}(|E|)$ möglich, sodass als Gesamtlaufzeit $\mathcal{O}(|V| \cdot |E|) = \mathcal{O}(|V|^3)$ folgt.

Zwei Eigenschaften der Eingabegraphen können den HS-Algorithmus zu $\Omega(|V| \log |V|)$ OBDD-Operationen zwingen:

1. Es existieren Flussprobleme, die $\Theta(|V|)$ Phasen erfordern.

2. Niveaunetzwerke können Tiefe $\Theta(|V|)$ haben und werden im HS-Algorithmus ebenenweise konstruiert.

Unabhängig von den auftretenden OBDD-Größen können diese Eigenschaften polylogarithmische Laufzeit vereiteln. Niveaunetzwerkansätze können eine lineare Anzahl an Phasen nicht verhindern; der IQ-Algorithmus vermeidet jedoch ebenenweises Vorgehen und ist somit nicht linear von der Niveaunetzwerktiefe abhängig.

3.2 Prioritätsfunktionen

Bevor wir uns der Beschreibung des IQ-Algorithmus widmen, stellen wir Funktionen vor, die sowohl im HS-Algorithmus als auch im IQ-Algorithmus zur heuristischen Berechnung von Abbildungen zwischen Objekten wie Knoten oder Kanten verwendet werden. Durch zwei solche Abbildungen können dann bipartite Matchings durch wenige funktionale Operationen erhalten werden.

Sei $\prec_x$ eine vollständige Ordnung auf $M := \{0, \ldots, N-1\}^k$ für $x \in \mathbb{B}^{kn}$, $n := \lceil \log N \rceil$ und konstantes k (siehe Abschnitt 2.5). Seien $x, y \in \mathbb{B}^{kn}$.

Definition 3.2.1. *Eine* Prioritätsfunktion $\Pi_\prec(x, y, z)$ *ist eine implizite Darstellung der Ordnung* $\prec_x$ *mit*

$$\Pi_\prec(x, y, z) = 1 :\Leftrightarrow |y| \prec_x |z|.$$

Die Interpretation im Rahmen der Algorithmen ist, dass ein Objekt x (z. B. ein Knoten oder eine Kante) das Objekt z „auswählt“, das minimal bez. $\prec_x$ ist. Dabei ist gewünscht, dass verschiedene Objekte x verschiedene Objekte z aus einer Teilmenge $P_x \subseteq M^k$ gültiger Objekte auswählen, die Funktion Π jedoch nicht zufällig ist, sondern kleine OBDD-Darstellung hat. Eine entsprechende Abbildung ist dann

$$D(x, z) := P(x, z) \wedge \overline{(\exists y)\ \big(P(x, y) \wedge \Pi_\prec(x, y, z)\big)} \tag{3.2.1}$$

für $P(x, z) := (z \in P_x)$. $D(x, z)$ ist also die implizite Teilmenge der gültigen Kandidaten (x, z) mit $P(x, z) = 1$, für die z minimal bez. $\Pi_\prec$ ist. Dieser Ausdruck entspricht wegen der Quantifizierung über y einer Anzahl von $\mathcal{O}(n)$ funktionaler Operationen.

Wir betrachten zwei Prioritätsfunktionen, die in Hachtel und Somenzi (1997) vorgestellt werden:

Definition 3.2.2 (Datum-Proximity-Funktion Π_{dp}).

$$\Pi_{\mathrm{dp}}(x, y, z) := (|y| < |z|)$$

Die Datum-Proximity-Funktion macht also keinen Gebrauch von der möglichen Abhängigkeit von x.

Definition 3.2.3 (Relative-Proximity-Funktion Π_{rp}).

$$\Pi_{\mathrm{rp}}(x, y, z) := |y \oplus x| < |z \oplus x|$$

Dabei bezeichnet $\oplus$ das bitweise exklusive Oder. Da diese Funktion von x abhängig ist und daher eher im Sinne der sie verwendenden impliziten Algorithmen zu streuenden Auswahlen für verschiedene Objekte x führt, erwarten wir von ihr ein typischerweise besseres Verhalten. Die Analyse auf Gittergraphen wird jedoch von der Einfachheit der Datum-Proximity-Funktion profitieren.

Anmerkung 3.2.4. *Die Argumente* $x, y, z \in \mathbb{B}^{kn}$ *bestehen jeweils aus* k *Subvektoren und es ist*

$$\Pi_{\mathrm{dp}}(x, y, z) = \bigvee_{j=1}^{k} \left(\big(|y^{(j)}| < |z^{(j)}|\big) \wedge \bigwedge_{j'=1}^{j-1} \big(|y^{(j')}| = |z^{(j')}|\big) \right).$$

Analog zu Anmerkung 2.5.7 folgt eine $\pi^{\mathrm{id}}_{k,n}$*-OBDD-Größe von* $(k^2 n)^{\mathcal{O}(k^2)}$*, bei genauerer Betrachtung jedoch ebenso* $\mathcal{O}(2^k kn)$*. Aus späteren Resultaten wird wieder direkt eine Verbesserung auf* $2^{\mathcal{O}(k^2 \log k)} n$ *folgen, mit der wir zufrieden sein werden, da* k *typischerweise konstant ist.*

Die gleichen Schranken gelten auch für Π_{rp}*, da das exklusive Oder mit* x *die Breite der einzelnen verknüpften OBDDs offensichtlich nicht wesentlich ändert.*

3.3 Flussmaximierung durch iteratives Quadrieren

Wir wenden uns schließlich dem impliziten IQ-Algorithmus zu. Genau wie der Algorithmus von Hachtel und Somenzi (1997) erhält er die Graphinstanz $G = (V, E)$ in Form ihrer charakteristischen Funktion $\chi_G \in B_{2n}$ mit $n := \log |V| + \mathcal{O}(1)$ und gibt den maximalen Fluss F in Form von $\chi_F \in B_{2n}$ aus. Wir verwenden die in Abschnitt 2.5 eingeführte Notation. Die Korrektheit des Verfahrens wird aus seiner detaillierten Beschreibung folgen. Insbesondere beeinflussen isolierte Knoten nicht die Korrektheit und wir können auf eine Berücksichtigung der Funktion χ_V verzichten. Algorithmus 3.3.1 stellt den Grobaufbau des IQ-Algorithmus dar.

Algorithmus 3.3.1 : Der Grobaufbau des IQ-Algorithmus.

Eingabe : Impliziter Graph χ_G, Knoten $q, s \in V$.
Ausgabe : Impliziter maximaler q-s-Fluss χ_F.

```
1  χ_F(x,y) := 0;
2  U(x,y) := berechneU(χ_G(x,y), q, s, χ_F(x,y));
3  while U(x,y) ≠ 0 do // Phase
4      B(x,y) := 0;
5      repeat // Sweep
6          B*(x,y) := multipfad(U(x,y));
7          if B*(x,y) = 0 then
8              B*(x,y) := einzelpfad(U(x,y));
9          B(x,y) := B(x,y) ∨ B*(x,y);
10         U(x,y) := U(x,y) ∧ ¬B*(x,y);
11     until B(x,y) ist nicht vergrößerbar;
12     χ_F(x,y) := (χ_G(x,y) ∧ B(x,y)) ∨ (χ_F(x,y) ∧ ¬B(y,x));
13     U(x,y) := berechneU(χ_G(x,y), q, s, χ_F(x,y));
14 return χ_F;
```

Wie jeder Niveaunetzwerkalgorithmus enthält er eine Hauptschleife, die solange Phasen durchführt, wie es flussverbessernde Pfade gibt (Zeilen 3–13). In jeder Phase wird ein Niveaunetzwerk U durch das Modul berechneU berechnet (Zeilen 2 und 13), das in Abschnitt 3.4 beschrieben wird. Es gibt die Funktion $U(x, y)$ der Kanten von U zurück bzw. die Nullfunktion, wenn kein flussverbessernder Pfad existiert. Im ersteren Fall wird dann der Sperrfluss B konstruiert. Wie auch im Algorithmus von Hachtel und Somenzi (1997) geschieht dies in sogenannten Sweeps (Zeilen 5–11), die jeweils in einer nicht leeren Menge B^* verbessernder Pfade in U resultieren. Dies wird durch die Kombination zweier Verfahren erreicht:

1. Die *Einzelpfadkonstruktion* findet genau einen Pfad pro Aufruf. Wir beschreiben sie in Abschnitt 3.5.

2. Die *Multipfadkonstruktion* findet durch ein heuristisches Vorgehen hoffentlich möglichst viele Pfade, während sie möglicherweise auch keinen einzigen findet. Wir beschreiben sie in Abschnitt 3.6.

Zunächst wird in jedem Sweep versucht, die Multipfadkonstruktion erfolgreich anzuwenden (Zeile 6). Resultiert dies in einem leeren Ergebnis, wird die Einzelpfadkonstruktion angewandt (Zeile 8). So liefert jeder Sweep mindestens einen Pfad. B^* wird zum aktuellen Sperrflusskandidaten B hinzugefügt (Zeile 9) und zudem aus U entfernt (Zeile 10).

Wenn s noch von q aus über die in U verbliebenen Kanten erreichbar ist, dann ist B noch kein Sperrfluss und ein neuer Sweep wird gestartet. Ansonsten ist B nicht vergrößerbar und die Sweep-Schleife wird verlassen. Um dies in Zeile 11 zu entscheiden, lösen wir das Problem ***q*-Erreichbarkeit** in U mit dem in Abschnitt 2.5 vorgestellten Iterative-Quadrieren-Verfahren (Algorithmus 2.5.2) und werten das Ergebnis $\chi_R(x)$ für $x := \langle s \rangle_2$ aus. Dies kostet uns also $\mathcal{O}(n^2)$ funktionale Operationen.

Um zu verhindern, dass folgende Multipfadkonstruktionen in unnötige Sackgassen laufen, scheint es eine sinnvolle Erweiterung zu sein, die Kanten aus U zu entfernen, die nach Zeile 10 kein Teil eines q-s-Pfades mehr sind. Dies kann im Wesentlichen ebenfalls durch eine Anwendung von Algorithmus 2.5.2 erreicht werden. Zugunsten einer kompakteren Beschreibung findet sich dies jedoch nicht in Algorithmus 3.3.1. Auch auf die spätere Analyse auf Gittergraphen wird dies keinen Einfluss haben.

Ein Schritt verbleibt noch in der aktuellen Phase: die Verbesserung von F durch B in Zeile 12 gemäß Gleichung (3.1.2). Der verbesserte Fluss besteht dann aus zwei Arten von Kanten:

1. Kanten $(u, v) \in B \cap E$, die Vorwärtskanten (u, v) in B waren.

2. Kanten (u, v), die bereits im noch nicht verbesserten Fluss vorhanden waren und deren umgekehrte Versionen (v, u) keine Rückwärtskanten in B waren. Ansonsten wären sie bei der Flussverbesserung aus F entfernt worden.

Das folgende Theorem über die Anzahl ausgeführter funktionaler Operationen wird am Ende von Abschnitt 3.6 bewiesen und interpretiert, nachdem alle Module beschrieben wurden. Sei $\mathcal{S}$ die Anzahl durchgeführter Sweeps.

Theorem 3.3.1. *Der IQ-Algorithmus berechnet einen maximalen Fluss F durch $\mathcal{O}(n^2\mathcal{S}) = \mathcal{O}(n^2|F|) = \mathcal{O}(n^2|V|)$ funktionale Operationen.*

3.4 Niveaunetzwerkkonstruktion

Vor jeder neuen Phase müssen wir ein Niveaunetzwerk U aller kürzesten flussverbessernden Pfade gemäß des aktuellen Flusses F berechnen. Dazu wird berechneU die Eingabe $\chi_G(x, y)$, q, s und $\chi_F(x, y)$ übergeben. Die Rückgabe ist die implizite

Kantenmenge $U(x,y)$ bzw. die Nullfunktion, wenn kein weiterer verbessernder Pfad existiert.

Im Gegensatz zum HS-Algorithmus bauen wir U nicht ebenenweise auf, sondern komponentenweise. Dazu berechnen wir eine Partition $C_0(x,y),\ldots,C_{r-1}(x,y)$, sodass die *Partitionskomponente* $C_i(x,y)$ die Kanten von 2^{γ_i} aufeinanderfolgenden Niveaunetzwerkebenen umfasst. Die Exponenten γ_i wachsen dabei streng monoton in i.

Die Grenzebenen aufeinanderfolgender Partitionskomponenten sind in U benachbart: C_{i-1} und C_i teilen sich also eine gemeinsame Knotenebene, die wir *Trennungsschicht* Z_i nennen. Alle kürzesten q-Z_i-Pfade haben offenbar Länge $\sum_{j=0}^{i-1} 2^{\gamma_j}$ und U hat Tiefe $\ell = \sum_{j=0}^{r-1} 2^{\gamma_j}$. Außerdem definieren wir $Z_r := \{s\}$.

3.4.1 Vorverarbeitung

Zunächst berechnen wir die Kanten $EF(x,y)$ des Restgraphen G^F gemäß Gleichung (3.1.1):

$$EF(x,y) := \big(\chi_G(x,y) \wedge \overline{\chi_F(x,y)}\big) \vee \chi_F(y,x).$$

Dann bereiten wir einige Hilfsfunktionen mit jeweiligem Index γ vor, bevor wir die Partitionskomponenten und Trennungsschichten behandeln können. Sie stellen verschiedene Arten von Pfaden im Restgraphen G^F dar, deren Längen wir iterativ verdoppeln werden. Durch die maximale Pfadlänge von $|V|-1$ reichen dann $\gamma = \mathcal{O}(\log|V|) = \mathcal{O}(n)$ Iterationen.

Zuerst betrachten wir die Funktion $PFAD^{\leq}_{\gamma}(x,y)$, die vergleichbar mit $EF(x,y)$ in Algorithmus 2.5.2 Knotenpaare $(v_{|x|}, v_{|y|})$ mit $v_{|x|}$-$v_{|y|}$-Pfaden mit Längenbeschränkung 2^{γ} repräsentiert.

$$\begin{aligned} PFAD^{\leq}_{0}(x,y) &:= (x=y) \vee EF(x,y), && (3.4.1)\\ PFAD^{\leq}_{\gamma+1}(x,y) &:= (\exists z)\ \big(PFAD^{\leq}_{\gamma}(x,z) \wedge PFAD^{\leq}_{\gamma}(z,y)\big). && (3.4.2)\end{aligned}$$

Wir gehen also genau wie in Algorithmus 2.5.2 vor. Dementsprechend definieren wir $PFAD^{=}_{\gamma}(x,y)$ für $v_{|x|}$-$v_{|y|}$-Pfade der exakten Länge 2^{γ}. Lediglich die Initialisierung muss geändert werden, die nun ausschließlich Pfade der Länge $1 = 2^0$ erlaubt:

$$\begin{aligned} PFAD^{=}_{0}(x,y) &:= EF(x,y),\\ PFAD^{=}_{\gamma+1}(x,y) &:= (\exists z)\ \big(PFAD^{=}_{\gamma}(x,z) \wedge PFAD^{=}_{\gamma}(z,y)\big).\end{aligned}$$

Weiterhin benötigen wir $PFAD^{<}_{\gamma}(x,y)$ für $v_{|x|}$-$v_{|y|}$-Pfade, die kürzer als 2^{γ} sind. Für $k=0$ ist nur Pfadlänge 0 erlaubt $(x=y)$. Im iterativen Schritt gilt $PFAD^{<}_{\gamma}(x,y) = 1$ genau dann, wenn für einen Zwischenknoten $v_{|z|}$ ein $v_{|x|}$-$v_{|z|}$-Pfad der Länge höchstens 2^{γ} und ein $v_{|z|}$-$v_{|y|}$-Pfad kürzer als 2^{γ} existiert:

$$\begin{aligned} PFAD^{<}_{0}(x,y) &:= (x=y),\\ PFAD^{<}_{\gamma+1}(x,y) &:= (\exists z)\ \big(PFAD^{\leq}_{\gamma}(x,z) \wedge PFAD^{<}_{\gamma}(z,y)\big).\end{aligned}$$

Schließlich sei $KPFAD_\gamma(x,y)$ die charakteristische Funktion der Knotenpaare $(v_{|x|}, v_{|y|})$, deren kürzester Rest-$v_{|x|}$-$v_{|y|}$-Pfad (falls existent) exakt Länge 2^γ besitzt. Für $k = 0$ entspricht dies wieder den Restgraphkanten E^F. Sonst gilt $KPFAD_\gamma(x,y) = 1$ genau dann, wenn es einen Rest-$v_{|x|}$-$v_{|y|}$-Pfad nicht länger als 2^γ gibt, jedoch keinen, der kürzer ist:

$$\begin{aligned} KPFAD_0(x,y) &:= EF(x,y), \\ KPFAD_\gamma(x,y) &:= PFAD^{=}_\gamma(x,y) \wedge \overline{PFAD^{<}_\gamma(x,y)}. \end{aligned}$$

Nun kennen wir zwar die Knoten entsprechender kürzester verbessernder Pfade, jedoch noch nicht die Kanten auf diesen Pfaden, die letztlich das Niveaunetzwerk U ausmachen. Dazu sei $EF_\gamma(\nu,w,x,y) = 1$ genau dann, wenn $(v_{|x|}, v_{|y|})$ eine Kante auf einem kürzesten Rest-$v_{|\nu|}$-$v_{|w|}$-Pfad der exakten Länge 2^γ ist.

Eine Kante $(v_{|x|}, v_{|y|}) \in E^F$ kann als Pfad der Länge 1 betrachtet werden und es gilt $EF_0(\nu,w,x,y) = 1$ für $\nu = x$ und $w = y$. $EF_{\gamma+1}(\nu,w,x,y) = 1$ gilt genau dann, wenn der Rest-$v_{|\nu|}$-$v_{|w|}$-Pfad in Teilpfade $(v_{|\nu|}, \ldots, v_{|u|})$ und $(v_{|u|}, \ldots, v_{|w|})$ der jeweiligen Länge γ zerteilt werden kann, sodass $(v_{|x|}, v_{|y|})$ in einem der beiden liegt:

$$\begin{aligned} EF_0(\nu,w,x,y) &:= EF(x,y) \wedge (\nu = x) \wedge (w = y), \\ EF_{\gamma+1}(\nu,w,x,y) &:= (\exists u) \Big(\big(EF_\gamma(\nu,u,x,y) \wedge KPFAD_\gamma(u,w) \big) \\ &\qquad \vee \big(KPFAD_\gamma(\nu,u) \wedge EF_\gamma(u,w,x,y) \big) \Big). \end{aligned}$$

Die Funktionen $PFAD^{\leq}_\gamma(x,y)$, $PFAD^{=}_\gamma(x,y)$, $PFAD^{<}_\gamma(x,y)$, $KPFAD_\gamma(x,y)$ und $EF_0(\nu,w,x,y)$ werden nun iterativ mit steigendem γ berechnet, bis für ein γ^* schließlich eine der folgenden Eigenschaften zutrifft:

- $PFAD^{\leq}_{\gamma^*}(\langle q \rangle_2, \langle s \rangle_2) = 1$: Dann ist $\gamma^* = \lceil \log \ell \rceil$ und $\ell \in]2^{\gamma^*-1}, 2^{\gamma^*}]$. Wir fahren mit der Partitionierung von U durch ein Binäre-Suche-Verfahren im nächsten Unterabschnitt fort.

- $PFAD^{\leq}_{\gamma^*+1}(x,y) = PFAD^{\leq}_{\gamma^*}(x,y)$: Kein weiterer Restpfad wurde entdeckt, insbesondere kein verbessernder Pfad. $\chi_F(x,y)$ ist bereits maximal und berechneU liefert die Nullfunktion zurück.

Offenbar ist $\gamma^* \leq \lceil \log |V| \rceil = n$.

Lemma 3.4.1. *Die Berechnung der Funktionen $PFAD^{\leq}_\gamma(x,y)$, $PFAD^{=}_\gamma(x,y)$, $PFAD^{<}_\gamma(x,y)$, $KPFAD_\gamma(x,y)$ und $EF_0(\nu,w,x,y)$ geschieht durch $\mathcal{O}(n^2)$ funktionale Operationen.*

Beweis: Sowohl die Initialisierung als auch die Rekursionsgleichung aller fünf Funktionen umfassen offenbar eine konstante Anzahl von Operationen und Quantifizierungen über Knotennummern. Letztere entsprechen jeweils $\mathcal{O}(n)$ weiteren Operationen. Wegen $\gamma^* \leq n$ werden $\mathcal{O}(n)$ Funktionen berechnet. Insgesamt werden also $\mathcal{O}(n^2)$ Operationen durchgeführt. □

3.4.2 Partitionierung des Niveaunetzwerks

Wir betrachten die Netzwerktiefe ℓ vorübergehend als Binärzahl $\ell_{\gamma^*} \ldots \ell_0$ mit $\ell = \sum_{j=0}^{\gamma^*} 2^j$. Jedes der r gesetzten Bits $\ell_j = 1$ entspricht einer Partitionskomponente i mit $\gamma_i = j$. U ist also in der gleichen Weise aus Partitionskomponenten $C_i(x, y)$ zusammengesetzt, wie ℓ aus Summanden 2^{γ_i}.

Mit Hilfe einer Art binären Suche nach der Senke s werden wir jedes ℓ_j ermitteln und so herausfinden, ob eine Partitionskomponente mit 2^j Ebenen erzeugt werden muss. Das entsprechende rekursiv arbeitende Modul findeSenke (dargestellt in Algorithmus 3.4.1) wird mit impliziten Knotenmengen $R(x)$ und $Z(x) \leq R(x)$ sowie mit einem Exponenten a aufgerufen. Es gibt ein Paar $(\mathcal{Z}, \mathcal{G})$ zurück, das aus einer Funktionenliste $\mathcal{Z}$ und einer Ganzzahlenliste $\mathcal{G}$ besteht.

Die *Konkatenation* zweier geordneter Listen $\mathcal{L}^1 := (e_1^1, \ldots, e_n^1)$ und $\mathcal{L}^2 := (e_1^2, \ldots, e_n^2)$ werden wir mit $\mathcal{L}^1 : \mathcal{L}^2 := (e_1^1, \ldots, e_n^1, e_1^2, \ldots, e_n^2)$ bezeichnen. Einelementige Listen (e) werden wir einfach durch e kennzeichnen.

Algorithmus 3.4.1 : Das rekursive Modul findeSenke.

Eingabe : Implizite Knotenmengen $R(x)$ und $Z(x)$, Exponent a.
Ausgabe : Listenpaar $(\mathcal{Z}, \mathcal{G})$.

```
1 if (KPFAD_a(x, ⟨s⟩_2) ∧ Z(x)) ≠ 0 then
2 |  return ((x = ⟨s⟩_2), a) ;                                  // Fall 1
3 else if (PFAD^≤_{a−1}(x, ⟨s⟩_2) ∧ Z(x)) ≠ 0 then
4 |  return findeSenke(R(x), Z(x), a − 1) ;                    // Fall 2
5 else
6 |  Z'(x) := (∃y) (KPFAD_{a−1}(y, x) ∧ Z(y)) ∧ ¬R(x) ;         // Fall 3
7 |  R'(x) := R(x) ∨ (∃y) (PFAD^≤_{a−1}(y, x) ∧ Z(y));
8 |  (Z', G') := findeSenke(R'(x), Z'(x), a − 1);
9 |  return (Z'(x) : Z', a − 1 : G');
```

Wenn findeSenke$(R(x), Z(x), a)$ aufgerufen wird, sind im Allgemeinen bereits folgende Dinge bekannt:

- die Bits $\gamma^*, \ldots, a+1$ von ℓ, die i gesetzte Bits enthalten,
- die Trennungsschichten $Z_0(x), \ldots, Z_i(x) = Z(x)$,

- die Exponenten $\gamma_0, \ldots, \gamma_{i-1}$ und
- die Funktion $R(x)$ der von $C_0(x), \ldots, C_{i-1}(x)$ abgedeckten Knoten.

Die aktuelle Instanz von findeSenke berechnet dann rekursiv folgende Informationen:

- die Bits $a, \ldots, 0$ von ℓ,
- die Trennungsschichten $Z_{i+1}(x), \ldots, Z_r(x)$ und
- die Exponenten $\gamma_i, \ldots, \gamma_{r-1}$.

Die resultierende Liste $\mathcal{Z}$ enthält die übrigen Trennungsschichten $Z_{i+1}(x), \ldots, Z_r(x) = (x = \langle s \rangle_2)$. Analog enthält $\mathcal{G}$ die Exponenten der übrigen Partitionskomponenten $i, \ldots, r-1$. Sie entsprechen den Indizes der gesetzten Bits von ℓ.

Für $\ell' := \ell_a \ldots \ell_0$ ist $|\ell'| < 2^{a+1}$ und der noch nicht partitionierte Teil von U ist nicht tiefer als $2^{a+1} - 1$. Wir beschreiben nun die drei Fälle, die in findeSenke unterschieden werden:

1. $KPFAD_a(x, \langle s \rangle_2) \wedge (Z(x) \neq 0)$.

   Die Tiefe ℓ' des noch nicht partitionierten Teils von U beträgt genau 2^a. Eine weitere Partitionskomponente $i = r - 1$ mit $\gamma_i = a$ wird benötigt. Dieser Exponent wird zusammen mit der letzten Trennungschicht $Z_r(x) = (x = \langle s \rangle_2)$ zurückgegeben (Zeile 2).

   Aus der Sicht einer binären Suche wurde ℓ' am „rechten“ Rand des Suchraums $]0, 2^a]$ gefunden und es gelten $\ell_a = 1$ und $\ell_j = 0$ für $j < a$.

2. $PFAD^{\leq}_{a-1}(x, \langle s \rangle_2) \wedge (Z(x) \neq 0)$.

   Es gibt einen Rest-Z-s-Pfad, der nicht länger als 2^{a-1} ist. Aus der Sicht einer binären Suche fahren wir in der „linken“ Hälfte $]0, 2^{a-1}]$ des Suchraums $]0, 2^a]$ fort. Dies geschieht durch einen rekursiven Aufruf von findeSenke in Zeile 4.

   Betrachten wir die Situation als Berechnung der Bits von ℓ, so wissen wir, dass keine Komponente der Tiefe 2^a zum Erreichen von s benötigt wird. Daher ist $\ell_a = 0$ und die Rekursion beschäftigt sich mit ℓ_{a-1}.

3. Keiner der beiden obigen Fälle trifft zu, es ist $\ell' \in]2^{a-1}, 2^a[$.

   Einerseits ist die Distanz zwischen Z und s kürzer als 2^a, andererseits ist sie länger als 2^{a-1}. Daher benötigen wir eine Partitionskomponente i mit $\gamma_i = a - 1$. Die Knoten $Z_{i+1}(x) =: Z'(x)$ (siehe Zeile 6), die von C_is letzter Kantenebene erreicht werden, stellen den Startpunkt einer neuen Suche in $]2^{a-1}, 2^a[$ dar. Die Knoten, die in $C_i(x, y)$ besucht werden, werden zu $R(x)$ hinzugefügt, das Ergebnis wird $R'(x)$ genannt (siehe Zeile 7).

Wir wissen, dass $\ell_a = 0$ und $\ell_{a-1} = 1$ sind. Also enthalten die Ergebnislisten $\mathcal{Z}$ und $\mathcal{G}$ die Rückgaben der rekursiv aufgerufenen Partitionierung, der in Zeile 8 die Parameter $R'(x)$, $Z'(x)$ und $a-1$ übergeben werden, konkateniert mit $Z'(x)$ bzw. $a-1$. Die binäre Suche fährt in der „rechten“ Hälfte des Suchraums fort.

Anfangs sind lediglich $Z_0(x) = R(x) = (x = \langle q \rangle_2)$ sowie die obere Tiefenschranke 2^{γ^*} bekannt. Die binäre Suche beginnt beim Quellenknoten q, der sowohl die Trennungsschicht $Z_0(x)$ als auch die Menge der bereits erreichten Knoten darstellt. Das anfängliche Suchintervall ist $]0, 2^{\gamma^*}]$ und wir rufen findeSenke mit dem Parametertripel $(x = \langle q \rangle_2), (x = \langle q \rangle_2), \gamma^*$ auf. Ergebnis sind dann die Listen $Z_1(x), \ldots, Z_r(x)$ und $\gamma_0, \ldots, \gamma_{r-1}$. Abbildung 3.4.1 zeigt eine beispielhafte Niveaunetzwerkpartitionierung mit $\ell = 13$.

Lemma 3.4.2. *Der Aufruf* findeSenke$((x = \langle q \rangle_2), (x = \langle q \rangle_2), \gamma^*)$ *verursacht* $\mathcal{O}(n^2)$ *funktionale Operationen.*

Beweis: Neben den Rekursionsaufrufen führt findeSenke $\mathcal{O}(n)$ Operationen aus. In jedem rekursiven Aufruf verringert sich der Parameter a um 1. Für $a = 0$ wird keine weitere Rekursion betreten. Daher wird findeSenke nicht öfter als $\gamma^* + 1 \leq n + 1$-mal aufgerufen und es ergibt sich die Gesamtaussage. □

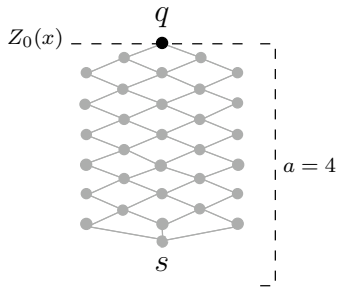
3.4.3 Berechnung der Niveaunetzwerkkanten

Wir müssen schließlich noch die impliziten Kantenmengen $C_0(x, y), \ldots, C_{r-1}(x, y)$ ermitteln. Wir erinnern uns daran, dass $C_i(x, y)$ aus allen kürzesten Restpfaden zwischen Z_i und Z_{i+1} besteht. Eine Kante $(v_{|x|}, v_{|y|})$ gehört genau dann zu einem solchen Pfad, wenn sie Teil eines $v_{|\nu|}$-$v_{|w|}$-Pfades mit $v_{|\nu|} \in Z_i$ und $v_{|w|} \in Z_{i+1}$ ist, was folgendermaßen ausgedrückt werden kann:

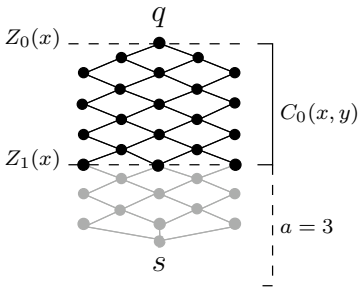
$$C_i(x, y) := (\exists \nu, w) \left(EF_{\gamma_i}(\nu, w, x, y) \wedge Z_i(\nu) \wedge Z_{i+1}(w) \right).$$

$C(x, y) := \bigvee_{i=0}^{r-1} C_i(x, y)$ enthält dann alle Niveaunetzwerkkanten, kann jedoch auch weitere Kanten enthalten. Z_i enthält alle Knoten v, deren kürzester Rest-s-v-Pfad Länge $\sum_{j=0}^{i-1} 2^{\gamma_j}$ besitzt. Insbesondere kann $Z_r(x)$ also weitere Knoten neben s enthalten, die von q aus über ℓ Kanten erreichbar sind. Daher kann $C(x, y)$ Kanten enthalten, die nicht Teil eines kürzesten flussverbessernden Pfades sind. Um diese Kanten zu entfernen, ermitteln wir die Menge $R(x)$ aller Knoten in $C(x, y)$, die auf einem q-s-Pfad liegen. Dies gilt für einen Knoten v genau dann, wenn im umgekehrten Niveaunetzwerk $C^{\mathrm{R}}(x, y) := C(y, x)$ ein v-q-Pfad existiert. Wir ermitteln $R(x)$ also durch Lösung eines ***q*-Erreichbarkeitsproblems** wie in Algorithmus 2.5.2 und filtern die Kanten $C(x, y)$ zu $U(x, y)$, den tatsächlichen Niveaunetzwerkkanten:

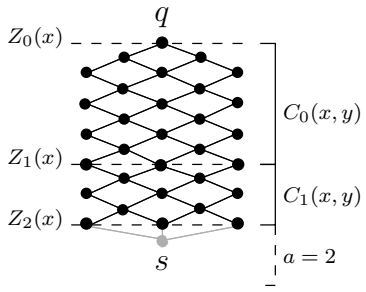
$$U(x, y) := C(x, y) \wedge R(x).$$



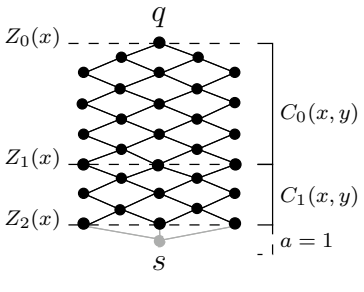
(a) Anfangssituation von findeSenke($(x = \langle q \rangle_2), (x = \langle q \rangle_2), 4$).



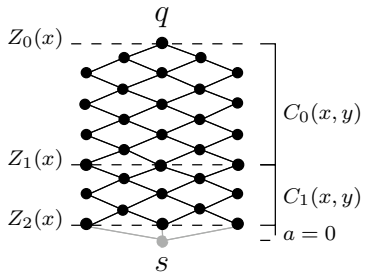
(b) Fall 3: Wegen $\ell \in]2^3, 2^4[$ wird $C_0(x, y)$ mit $\gamma_0 = 3$ erzeugt. Die Suche fährt von $Z'(x) = Z_1(x)$ aus mit $a = 3$ fort.



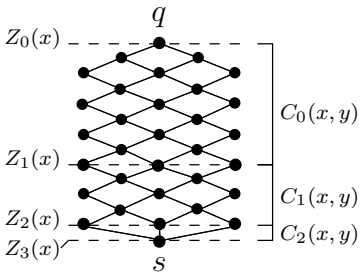
(c) Fall 3: Wegen $\ell \in]2^3 + 2^2, 2^4[$ wird $C_1(x, y)$ mit $\gamma_1 = 2$ erzeugt. Die Suche fährt von $Z'(x) = Z_2(x)$ aus mit $a = 2$ fort.



(d) Fall 2: Das Suchintervall wird auf $]2^3 + 2^2, 2^3 + 2^2 + 2^1]$ verkleinert.



(e) Fall 2: Das Suchintervall wird auf $]2^3 + 2^2, 2^3 + 2^2 + 2^0]$ verkleinert.



(f) Fall 1: s wurde in Ebene $2^3 + 2^2 + 2^0 = 13$ gefunden.

Abbildung 3.4.1: Beispielhafte Anwendung von findeSenke. Schwarze Knoten/Kanten wurden bereits betrachtet; graue Objekte sind noch unbekannt. Gestrichelte Linien kennzeichnen Trennungsschichten. Durchgehende Klammern umfassen Partitionskomponenten, gestrichelte Klammern umfassen Suchintervalle.

Nun haben wir alle Informationen, die wir benötigen, um einen Sperrfluss $B(x,y)$ in $U(x,y)$ zu berechnen. Im Gegensatz zum HS-Algorithmus wird dies ohne eine bekannte Zuordnung jedes einzelnen Knotens und jeder einzelnen Kante zu ihren jeweiligen Ebenen geschehen müssen.

Lemma 3.4.3. *Die Berechnung von $U(x,y)$ verursacht $\mathcal{O}(n^2)$ funktionale Operationen.*

Beweis: Jede der $r \leq n$ Funktionen $C_i(x,y)$, $i \in \{0,\ldots,r-1\}$, wird durch $\mathcal{O}(n)$ Operationen berechnet. Die Disjunktion aller $C_i(x,y)$ entspricht $r-1$ Operationen. Algorithmus 2.5.2 verursacht $\mathcal{O}(n^2)$ Operationen. Es folgt die Gesamtaussage. □

Aus den Lemmata 3.4.1, 3.4.2 und 3.4.3 folgt nun:

Korollar 3.4.4. *Der Aufruf* berechneU$\big(\chi_G(x,y), q, s, \chi_F(x,y)\big)$ *verursacht insgesamt $\mathcal{O}(n^2)$ funktionale Operationen.*

3.5 Einzelpfadkonstruktion

Das Modul einzelPfad konstruiert genau einen q-s-Pfad p der Länge ℓ in $U(x,y)$, implizit repräsentiert durch $B^*(x,y)$.

Die Tiefe ℓ muss keine Zweierpotenz sein. Um iteratives Quadrieren anwenden zu können, muss ℓ in eine Folge $\ell_0^1 := \ell, \ell_0^2 := \ell - 1, \ldots, \ell_t^1 := 1, \ell_t^2 := 0$ zerlegt werden. Jedes ℓ_m^a mit $m \in \{0,\ldots,t\}$ und $a \in \{1,2\}$ wird als Pfadlänge aufgefasst. Pfade der Längen ℓ_m^1 und ℓ_m^2 sollen aus Teilpfaden der Längen ℓ_{m+1}^1 und/oder ℓ_{m+1}^2 zusammengesetzt werden können. Dies wird durch die Unterteilungsregeln $\ell_{m+1}^1 := \lceil \ell_m^1/2 \rceil$ und $\ell_{m+1}^2 := \lfloor \ell_m^2/2 \rfloor = \ell_{m+1}^1 - 1$ erreicht. Für $e,o \in \{1,2\}$, sodass ℓ_m^e gerade und ℓ_m^o ungerade ist, gilt dann $\ell_m^e = 2\ell_{m+1}^e$ und $\ell_m^o = \ell_{m+1}^e + \ell_{m+1}^o$. Wir stoppen die Unterteilung für t mit $\ell_t^1 = 1$ und $\ell_t^2 = 0$. Aus $\ell_{m+1}^1, \ell_{m+1}^2 \leq (\ell_m^1 + 1)/2$ folgt $t = \mathcal{O}(\log \ell) = \mathcal{O}(n)$.

Zur Vorbereitung der Einzelpfadkonstruktion berechnen wir die Funktionen $P_{\ell_m^a}(x,y,z)$ für $m := t-1,\ldots,0$ und $a \in \{1,2\}$. Sie repräsentieren Pfade $(v_{|x|},\ldots,v_{|y|},\ldots,v_{|z|})$ der Länge ℓ_m^a, sodass die Unterteilung in $(v_{|x|},\ldots,v_{|y|})$ und $(v_{|y|},\ldots,v_{|z|})$ der Unterteilung von ℓ_m^a entspricht. Die Konstruktion wird durch Funktionen $P^*_{\ell_m^a}(x,z)$ unterstützt, die Niveaunetzwerkpfade $(v_{|x|},\ldots,v_{|z|})$ der Länge ℓ_m^a darstellen.

Seien wieder $e,o \in \{1,2\}$ so gewählt, dass ℓ_m^e gerade und ℓ_m^o ungerade ist. Wir iterieren die Berechnung für $m := t-1,\ldots,0$, $a \in \{1,2\}$ basierend auf der Initialisierung $\ell_t^1 = 1$ und $\ell_t^2 = 0$:

$$\begin{aligned} P^*_{\ell_t^1}(x,z) &:= U(x,z), \\ P^*_{\ell_t^2}(x,z) &:= (x = z), \\ P_{\ell_m^e}(x,y,z) &:= P^*_{\ell_{m+1}^e}(x,y) \wedge P^*_{\ell_{m+1}^e}(y,z), \end{aligned}$$

$$\begin{aligned} P_{\ell_m^o}(x,y,z) &:= P^*_{\ell_{m+1}^e}(x,y) \wedge P^*_{\ell_{m+1}^o}(y,z), \\ P^*_{\ell_m^a}(x,z) &:= (\exists y)\ P_{\ell_m^a}(x,y,z). \end{aligned}$$

Für Pfade $(v_{|x|}, \ldots, v_{|y|}, \ldots, v_{|z|})$ gerader Länge ℓ_m^e liegt der Knoten $v_{|y|}$ genau in der Mitte; für Pfade ungerader Länge ℓ_m^o hat der Teilpfad $(v_{|x|}, \ldots, v_{|y|})$ gerade Länge ℓ_{m+1}^e, während $(v_{|y|}, \ldots, v_{|z|})$ ungerade Länge ℓ_{m+1}^o hat. Wegen $\ell_0^1 = \ell$ repräsentieren wir das Start-End-Knotenpaar (q,s) von p durch $D_{\ell_0^1}(x,z) := (x = q) \wedge (z = s)$. Die Funktion $D_{\ell_0^2}(x,z)$ wird nicht für die folgende Aufteilung von p benötigt und daher als Nullfunktion definiert.

Die Pfadkonstruktion selbst arbeitet nun „top-down“ durch die Konkretisierung langer angedachter Pfade durch kürzere. Durch die Wahl eines Zwischenknotens $v_{|y|}$, der auf einem q-s-Pfad liegt, teilen wir p in zwei Teile $(q, \ldots, v_{|y|})$ und $(v_{|y|}, \ldots, s)$ auf, die durch $D_{\ell_1^1}(x,z)$ und/oder $D_{\ell_1^2}(x,z)$ dargestellt werden. Diese Aufteilung iterieren wir für die resultierenden Pfade, bis die Kanten D_1 von p bekannt sind. Jede Iteration führt den Unterteilungsschritt parallel für alle Teilpfade gleicher Länge aus.

Im Allgemeinen kennen wir während der Konstruktion von p die Start- und Endknoten von Teilpfaden der Längen ℓ_m^1 und ℓ_m^2, implizit repräsentiert durch $D_{\ell_m^1}(x,z)$ und $D_{\ell_m^2}(x,z)$. Wir erhalten $Q_{\ell_m^a}$, indem wir alle Pfade $(v_{|x|}, \ldots, v_{|y|}, \ldots, v_{|z|})$ aus $P_{\ell_m^a}$ für $a \in \{1,2\}$ entfernen, deren Start-End-Knotenpaare nicht in $D_{\ell_m^a}$ enthalten sind:

$$Q_{\ell_m^a}(x,y,z) := P_{\ell_m^a}(x,y,z) \wedge D_{\ell_m^a}(x,z).$$

$Q_{\ell_m^a}(x,y,z)$ stellt nun alle *möglichen* Teilpfade der Länge ℓ_m^a dar, die die bereits durch $D_{\ell_m^a}(x,z)$ fixierten Knoten besuchen. Ein Start-End-Knotenpaar $(v_{|x|}, v_{|z|})$ kann in mehreren Tripeln $(v_{|x|}, v_{|y|}, v_{|z|}) \in Q_{\ell_m^a}$ auftauchen, die jeweils einen möglichen Teilpfad von p darstellen. Durch die Wahl eines Zwischenknotens $v_{|y|}$ wird einer dieser Teilpfade fixierter Bestandteil von p. Wir berechnen dazu eine implizite Teilmenge $T_{\ell_m^a}(x,y,z) \leq Q_{\ell_m^a}(x,y,z)$ der potenziellen Pfade der Länge ℓ_m^a, sodass jedes Start-End-Knotenpaar $(v_{|x|}, v_{|z|})$ nur noch in einem Tripel $(v_{|x|}, v_{|y|}, v_{|z|}) \in T_{\ell_m^a}$ auftaucht, dessen entscheidender Zwischenknoten $v_{|y|}$ durch eine Prioritätsfunktion $\Pi_\prec$ fixiert wird:

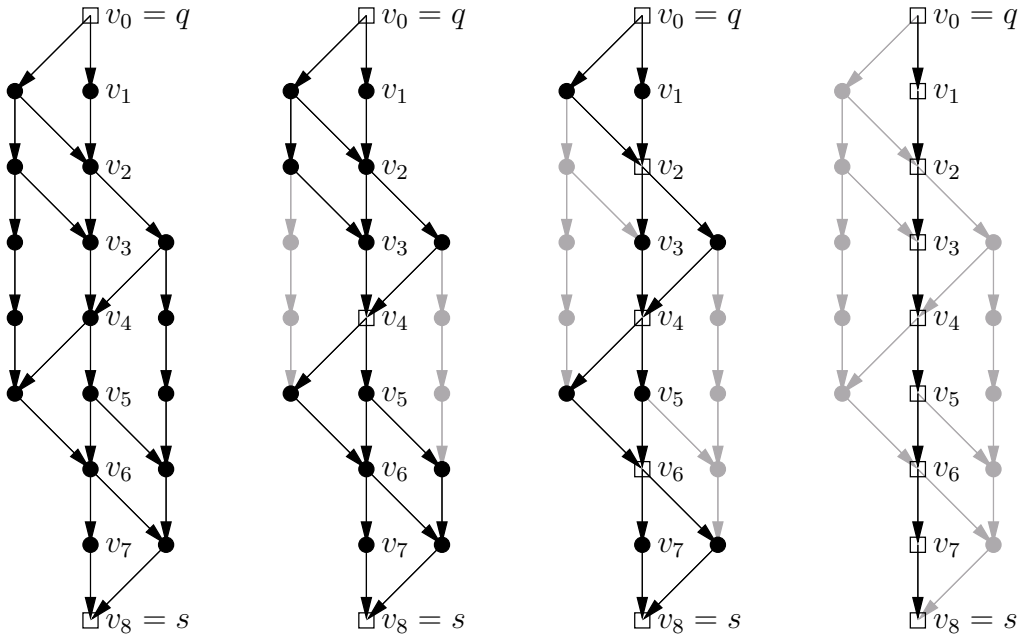
$$T_{\ell_m^a}(x,y,z) := Q_{\ell_m^a}(x,y,z) \wedge \overline{(\exists y')\ \big(Q_{\ell_m^a}(x,y',z) \wedge \Pi_\prec(x,y',y)\big)}.$$

Die Wahl von $\Pi_\prec$ lassen wir in dieser Algorithmusbeschreibung offen und kommentieren sie später. Diese Auswahl funktioniert nach dem gleichen Prinzip wie in Gleichung (3.2.1). Bevor die nächste Iteration für die Konkretisierung kürzerer Pfade begonnen werden kann, müssen die von $T_{\ell_m^a}(x,y,z)$ dargestellten Tripel (x,y,z) noch in zwei Paare (x,y) und (y,z) aufgeteilt werden, um danach in Form von $D_{\ell_{m+1}^1}(x,z)$ und $D_{\ell_{m+1}^2}(x,z)$ zur Verfügung zu stehen. Sei wieder ℓ_m^e gerade und ℓ_m^o ungerade:

$$\begin{aligned} D_{\ell_{m+1}^e}(x,z) &:= (\exists y)\ \big(T_{\ell_m^e}(x,z,y) \vee T_{\ell_m^e}(y,x,z) \vee T_{\ell_m^o}(x,z,y)\big), \\ D_{\ell_{m+1}^o}(x,z) &:= (\exists y)\ T_{\ell_m^o}(y,x,z). \end{aligned}$$

Die Teilpfade $(v_{|x|}, \ldots, v_{|z|})$ der Länge ℓ^e_{m+1} (repräsentiert durch $D_{\ell^e_{m+1}}(x,z)$) tragen als „obere“ Teile zu Pfaden $(v_{|x|}, \ldots, v_{|z|}, \ldots, v_{|y|})$ sowohl gerader Länge ℓ^e_m (entsprechend $T_{\ell^e_m}(x,z,y)$) als auch ungerader Länge ℓ^o_{m+1} (entsprechend $T_{\ell^o_m}(x,z,y)$) bei. Als „untere“ Teilpfade tragen sie lediglich zu Pfaden $(v_{|y|}, \ldots, v_{|x|}, \ldots, v_{|z|})$ gerader Länge ℓ^e_m (entsprechend $T_{\ell^e_m}(y,x,z)$) bei. Die Teilpfade $(v_{|x|}, \ldots, v_{|z|})$ der Länge ℓ^o_{m+1} (repräsentiert durch $D_{\ell^o_{m+1}}(x,z)$) tragen als „untere“ Teile zu Pfaden $(v_{|y|}, \ldots, v_{|x|}, \ldots, v_{|z|})$ ungerader Länge ℓ^o_m (entsprechend $T_{\ell^o_m}(y,x,z)$) bei.

Wegen $\ell^1_t = 1$ stellt $D_{\ell^1_t}(x,z)$ die Kanten von p dar und entspricht dem Konstruktionsergebnis $B^*(x,y) := D_{\ell^1_t}(x,y)$. Abbildung 3.5.1 zeigt eine beispielhafte Einzelpfadkonstruktion in einem Niveaunetzwerk der Tiefe $\ell = 8$.



(a) D_8. (b) D_4. (c) D_2. (d) D_1.

Abbildung 3.5.1: Beispielhafte Einzelpfadkonstruktion für $p = (s = v_0, \ldots, v_8 = t)$. Bereits fixierte Pfadknoten, die in einem $D_{\ell^a_m}$ enthalten sind, wurden rechteckig dargestellt. D_8 enthält das Start-End-Knotenpaar (v_0, v_8). Alle Knoten/Kanten können noch Teil von p werden. D_4 enthält die Paare (v_0, v_4) und (v_4, v_8); der Mittelknoten v_4 wurde ausgewählt. D_2 enthält (v_0, v_2), (v_2, v_4), (v_4, v_6) und (v_6, v_8); die Mittelknoten v_2 und v_6 wurden ausgewählt. Nach der letzten Pfadaufteilung D_1 sind alle Knoten von p selektiert und die Konstruktion ist beendet.

Lemma 3.5.1. *Eine Einzelpfadkonstruktion verursacht $\mathcal{O}(n^2)$ funktionale Operationen.*

Beweis: Die Funktionen $P_{\ell_m^a}(x,y,z)$, $P^*_{\ell_m^a}(x,z)$, $Q_{\ell_m^a}(x,y,z)$, $T_{\ell_m^a}(x,y,z)$ und $D_{\ell_m^a}(x,z)$ werden für $m \in \{0,\ldots,t\}$ und $a \in \{1,2\}$ berechnet. Die Berechnung jeder einzelnen Funktion wird durch jeweils $\mathcal{O}(n)$ Quantifizierungen dominiert. Die Aussage folgt aus $t = \mathcal{O}(\log \ell) = \mathcal{O}(n)$. □

3.6 Multipfadkonstruktion

Das Modul multipfad konstruiert eine implizite Menge $B^*(x,y)$ kantendisjunkter q-s-Pfade in $U(x,y)$. Es setzt längere Pfade der Länge ℓ_m^a aus kürzeren der Länge ℓ_{m+1}^1 und/oder ℓ_{m+1}^2 mit $m := t-1,\ldots,0$ und $a \in \{1,2\}$ zusammen. Dies wird für alle $\ell - \ell_m^a + 1$ Ebenenfolgen der momentan betrachteten Pfadlänge ℓ_m^a „gleichzeitig“, jedoch unabhängig durch wenige funktionale Operationen durchgeführt.

Eine für die Korrektheit des Moduls entscheidende Invariante ist, dass alle konstruierten Pfade einer Länge ℓ_m^a mit gleichen Start- und Endknotenebenen kantendisjunkt sind. Wir werden diese Invariante bei der folgenden Beschreibung also im Auge behalten.

Im allgemeinen Fall haben wir bereits Pfade der Länge ℓ_{m+1}^a für $a \in \{1,2\}$ konstruiert. Die Paare $\big((v_{|\nu|}, v_{|w|}), (v_{|y|}, v_{|z|})\big)$ ihrer Start- und Endkanten werden durch $D_{\ell_{m+1}^a}(\nu,w,y,z)$ dargestellt. $S_{\ell_{m+1}^a}(u,\nu,w,x,y,z)$ wird dann die Kanten $(v_{|u|}, v_{|\nu|})$ darstellen, die auf diesen Pfaden $(v_{|w|}, v_{|x|}, \ldots, v_{|y|}, v_{|z|})$ liegen. Anfangs betrachten wir alle Kanten von $U(x,y)$ als Pfade der Länge $1 = \ell_t^1 = \ell_{t-1}^2$, die offensichtlich kantendisjunkt sind:

$$D_1(\nu,w,y,z) := U(\nu,w) \wedge (\nu = y) \wedge (w = z), \tag{3.6.1}$$

$$S_1(u,\nu,w,x,y,z) := D_1(w,x,y,z) \wedge (u = w) \wedge (\nu = x). \tag{3.6.2}$$

Pfade der Länge $\ell_q^2 = 0$ werden hier nicht gebraucht, und so bleiben D_0 und S_0 undefiniert.

Wir benutzen $D_{\ell_{m+1}^a}$, um potenzielle Pfadanfänge $(v_{|\nu|}, v_{|w|}, \ldots, v_{|x|})$ (dargestellt durch $P^{\mathrm{S}}_{\ell_m^a}(\nu,w,x)$) und Pfadenden $(v_{|x|}, \ldots, v_{|y|}, v_{|z|})$ (dargestellt durch $P^{\mathrm{E}}_{\ell_m^a}(x,y,z)$) von Pfaden der Länge ℓ_m^a mit $a \in \{1,2\}$ zu berechnen. Bei geraden Längen ℓ_m^e erlauben wir sowohl Pfadanfänge als auch Pfadenden der Länge ℓ_{m+1}^e:

$$\begin{aligned} P^{\mathrm{S}}_{\ell_m^e}(\nu,w,x) &:= (\exists y)\ D_{\ell_{m+1}^e}(\nu,w,y,x), \\ P^{\mathrm{E}}_{\ell_m^e}(x,y,z) &:= (\exists w)\ D_{\ell_{m+1}^e}(x,w,y,z). \end{aligned}$$

Für ungerade Längen ℓ_m^o erlauben wir Pfadanfänge mit ℓ_{m+1}^e Kanten und Pfadenden mit ℓ_{m+1}^o Kanten:

$$\begin{aligned} P^{\mathrm{S}}_{\ell_m^o}(\nu,w,x) &:= (\exists y)\ D_{\ell_{m+1}^e}(\nu,w,y,x), \\ P^{\mathrm{E}}_{\ell_m^o}(x,y,z) &:= (\exists w)\ D_{\ell_{m+1}^o}(x,w,y,z). \end{aligned}$$

Durch Disjunktion und Quantifizierung erhalten wir Kandidaten für Start-End-Kantenpaare von Pfaden der Länge ℓ_m^a:

$$\begin{aligned} P_{\ell_m^a}(\nu, w, x, y, z) &:= P^{\mathrm{S}}_{\ell_m^a}(\nu, w, x) \wedge P^{\mathrm{E}}_{\ell_m^a}(x, y, z), \\ P^*_{\ell_m^a}(\nu, w, y, z) &:= (\exists x)\ P_{\ell_m^a}(\nu, w, x, y, z). \end{aligned}$$

Um die implizite Menge $D_{\ell_m^a}(\nu, w, y, z)$ der Pfade der Länge ℓ_m^a zu ermitteln, weisen wir Startkanten $(v_{|\nu|}, v_{|w|})$ und Endkanten $(v_{|y|}, v_{|z|})$, für die ein Pfad $(v_{|\nu|}, v_{|w|}, \ldots, v_{|y|}, v_{|z|})$ existiert, eindeutig einander zu. Wir möchten also Kanten andere Kanten wählen lassen und verwenden dazu eine Prioritätsfunktion $\Pi_\prec(u, \nu, w, x, y, z)$, deren geordnete Objekte nun Kanten $(v_{|u|}, v_{|\nu|})$, $(v_{|w|}, v_{|x|})$ und $(v_{|y|}, v_{|z|})$ sind:

$$\Pi_\prec(u, \nu, w, x, y, z) \Leftrightarrow (w, x) \prec_{(u,\nu)} (y, z).$$

Auch hier lassen wir die Wahl von $\Pi_\prec$ offen und kommentieren sie später. In einem ersten Schritt lassen wir Startkanten potenzielle Endkanten wählen und bezeichnen die so erhaltene implizite Teilmenge von $P^*_{\ell_m^a}(\nu, w, y, z)$ mit $Q_{\ell_m^a}(\nu, w, y, z)$:

$$\begin{aligned} Q_{\ell_m^a}(\nu, w, y, z) := P^*_{\ell_m^a}(\nu, w, y, z) \\ \wedge \overline{(\exists y', z')\ \big(P^*_{\ell_m^a}(\nu, w, y', z') \wedge \Pi_\prec(\nu, w, y', z', y, z)\big)}. \end{aligned} \quad (3.6.3)$$

Im zweiten Schritt wählt jede Endkante, die von mindestens einer Startkante gewählt wurde, eine von diesen aus und vervollständigt so die eindeutige Zuweisung $D_{\ell_m^a}(\nu, w, y, z)$:

$$\begin{aligned} D_{\ell_m^a}(\nu, w, y, z) := Q_{\ell_m^a}(\nu, w, y, z) \\ \wedge \overline{(\exists \nu', w')\ \big(Q_{\ell_m^a}(\nu', w', y, z) \wedge \Pi_\prec(y, z, \nu', w', \nu, w)\big)}. \end{aligned} \quad (3.6.4)$$

Die konstruierte Pfadmenge muss nicht maximal sein und die Zuweisung kann iteriert werden, bis keine weiteren Pfade der Länge ℓ_m^a mehr gefunden werden. In den folgenden Laufzeitbetrachtungen gehen wir jedoch von einer konstanten Iterationsanzahl aus. Wir betrachten die Korrektheit: Nach Induktionsvoraussetzung gilt die Invariante für die Pfadlängen ℓ_{m+1}^1 und ℓ_{m+1}^2. Die gegenseitige Auswahl von Pfaden der Längen ℓ_{m+1}^1 und ℓ_{m+1}^2 in Gleichung (3.6.3) und Gleichung (3.6.4) erzeugt mit $D_{\ell_m^1}$ und $D_{\ell_m^2}$ bipartite Matchings verbundener Pfade. Die diesen eindeutigen Zuordnungen entsprechenden Pfade mit gleicher Startknotenebene sind also wieder kantendisjunkt und die Invariante gilt auch für die nächstgrößeren Pfadlängen.

Schließlich müssen wir die Kantenmenge $S_{\ell_m^a}(u, \nu, w, x, y, z)$ der soeben konstruierten Pfade ermitteln. Eine Kante $(v_{|u|}, v_{|\nu|})$ liegt genau dann auf einem Pfad $(v_{|w|}, v_{|x|}, \ldots, v_{|y|}, v_{|z|})$ der Länge ℓ_m^a, wenn solch ein Pfad existiert (ausgedrückt durch $D_{\ell_m^a}(w, x, y, z)$, $a \in \{e, o\}$) und die Kante auf einem der beiden Pfadteile liegt, aus denen er aufgebaut wurde. Wieder hängen die Längen der beiden Teilpfade von der Parität von ℓ_m^a ab, für gerades ℓ_m^e und ungerades ℓ_m^o.

$$S_{\ell_m^e}(u,\nu,w,x,y,z) := D_{\ell_m^e}(w,x,y,z) \\ \wedge (\exists w', x') \left(S_{\ell_{m+1}^e}(u,\nu,w,x,w',x') \vee S_{\ell_{m+1}^e}(u,\nu,w',x',y,z)\right)$$

$$S_{\ell_m^o}(u,\nu,w,x,y,z) := D_{\ell_m^o}(w,x,y,z) \\ \wedge (\exists w', x') \left(S_{\ell_{m+1}^e}(u,\nu,w,x,w',x') \vee S_{\ell_{m+1}^o}(u,\nu,w',x',y,z)\right)$$

Wenn $S_{\ell_0^1}(u,\nu,w,x,y,z)$ für $\ell_0^1 = \ell$ bekannt ist, erhalten wir die Pfadkanten $B^*(x,y)$ wieder durch Quantifizierung über potenzielle Start- und Endkanten:

$$B^*(x,y) := (\exists u,\nu,w,z)\ S_{\ell_0^1}(x,y,u,\nu,w,z).$$

Weil alle Teilpfade einer bestimmten Länge unabhängig voneinander konstruiert werden, kann dieses Vorgehen in eine Sackgasse führen. Dann wendet der IQ-Algorithmus eine Einzelpfadkonstruktion an (Zeile 8). Abbildung 3.6.1 zeigt eine beispielhafte erfolglose Multipfadkonstruktion in einem Niveaunetzwerk der Tiefe $\ell = 8$. Andererseits zeigt Abbildung 3.6.2 das einfachste Beispiel, bei dem der gesamte Sperrfluss erfolgreich mit einem Aufruf von multipfad konstruiert wird.

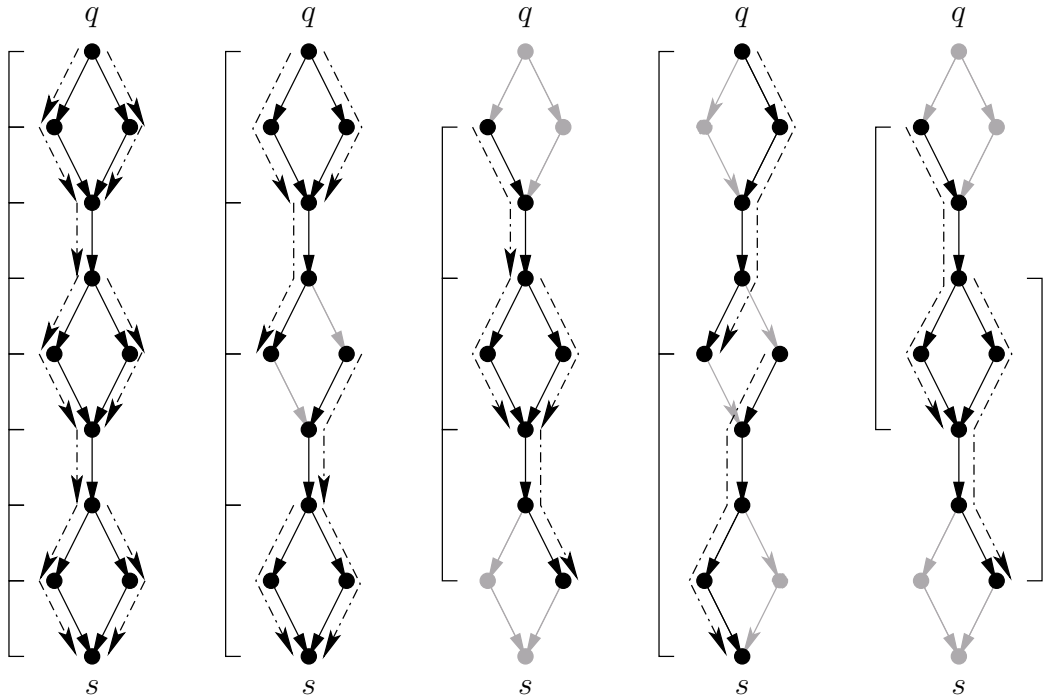
Lemma 3.6.1. *Eine Multipfadkonstruktion verursacht $\mathcal{O}(n^2)$ funktionale Operationen.*

Beweis: Gemäß der Annahme einer konstanten Anzahl an Selektionsiterationen für jede Pfadlänge ℓ_m^a mit $m \in \{0,\ldots,t\}$ und $a \in \{1,2\}$ kann analog zu Lemma 3.5.1 argumentiert werden. □

Korollar 3.4.4, Lemma 3.5.1 und Lemma 3.6.1 werden nun benutzt, um Theorem 3.3.1 zu beweisen.

Beweis von Theorem 3.3.1: Wir können o. B. d. A. annehmen, dass nach jedem Sweep ein neues Niveaunetzwerk berechnet wird. Gemäß Korollar 3.4.4 verursacht jeder dieser $\mathcal{O}(\mathcal{S})$ Aufrufe von berechneU $\mathcal{O}(n^2)$ Operationen. Gemäß Lemma 3.5.1 und Lemma 3.6.1 verursachen die Einzelpfad- und Multipfadkonstruktion jeweils ebenfalls nur $\mathcal{O}(n^2)$ Operationen. Die Erreichbarkeitsanalyse in Zeile 11 kostet weitere $\mathcal{O}(n^2)$ Operationen. Die übrigen Schritte in den Zeilen 9, 10 und 12 verursachen einen weiteren konstanten Summanden pro Phase bzw. Sweep. Insgesamt werden für jeden Sweep also höchstens $\mathcal{O}(n^2)$ funktionale Operationen verursacht und der Flusswert andererseits mindestens um den Wert 1 erhöht.

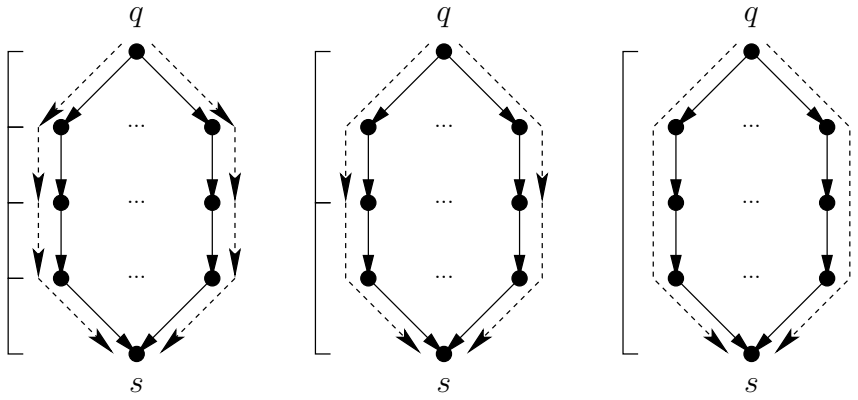
Da der Flusswert in jedem Sweep um mindestens 1 erhöht wird und q höchstens Grad $|V|-1$ hat, ist weiterhin $\mathcal{S} \leq |F| \leq |V|-1$. □



(a) D_1. (b) D'_2. (c) D''_2. (d) D'_4. (e) D''_4.

Abbildung 3.6.1: Beispielhafte erfolglose Multipfadkonstruktion. Klammern umfassen betrachtete Pfadbereiche. Gestrichelte Pfeile symbolisieren konstruierte Pfade. Graue Knoten/Kanten werden in der jeweiligen Abbildung von keinem Pfad benutzt. D_1 enthält lediglich die Kanten von U. D_2 besteht aus den Pfaden D'_2 und D''_2. D_4 besteht aus den Pfaden D'_4 und D''_4. Lediglich die Pfade in Abbildung (d) decken q und s ab und könnten einen Pfad der Länge 8 bilden. Dies ist jedoch offensichtlich nicht möglich.

Wir haben bewusst offen gelassen, welche Prioritätsfunktionen zu verwenden sind. Bei der Einzelpfadkonstruktion wird stets ein Pfad produziert und verschiedene Prioritätsfunktionen können evtl. den Gesamtverlauf des Algorithmus positiv oder negativ beeinflussen. Bei der Multipfadkonstruktion jedoch sollen möglichst viele Pfade gleichzeitig konstruiert werden. Dazu soll die Auswahl in Gleichung (3.6.3) möglichst streuend geschehen – würden viele Startkanten die gleiche Endkante wählen, entsteht letztlich für die Startkantenmenge höchstens ein Pfad. Intuitiv ist daher eine Prioritätsfunktion von Vorteil, die essenziell vom Parameter x ihrer Ordnung $\prec_x$ abhängt, wie es für Π_{rp} der Fall ist. Hier kann auch auf den jeweiligen Anwendungsfall des Algorithmus und die erwartete Struktur der Eingaben heuristisch eingegangen werden.



(a) D_1. (b) D_2. (c) D_4.

Abbildung 3.6.2: Beispielhafte erfolgreiche Multipfadkonstruktion. Das Niveaunetzwerk besteht aus parallelen einfachen q-s-Pfaden. Gestrichelte Pfeile symbolisieren konstruierte Pfade. Im Gegensatz zu Abbildung 3.6.1 sind hier nicht alle konstruierten Pfade abgebildet, sondern nur solche, die in den umklammerten Bereichen verlaufen – nur sie tragen letztlich zu $B^* = U$ bei.

Es bleibt die Frage, ob es Verfahren gibt, die stets mit $\text{poly}(n)$ funktionalen Operationen auskommen. Wir werden jedoch in Abschnitt 7.1 sehen, dass dies die Existenz von Algorithmen mit paralleler polylogarithmischer Laufzeit für die **0-1-Flussmaximierung** implizieren würde, deren Entwicklung bisher jedoch noch nicht gelang. Somit scheint hier eine bekannte Schwierigkeit zu liegen.

Schließlich haben wir den IQ-Agorithmus unabhängig von einer Datenstruktur für boolesche Funktionen beschrieben, gehen jedoch im Allgemeinen und insbesondere im nächsten Abschnitt von der Verwendung von OBDDs aus. Bevor wir nun zu Techniken für die Analyse OBDD-basierter impliziter Algorithmen auf speziellen Instanzen kommen, betrachten wir zunächst die besten Zeit- und Platzschranken, die wir aus unseren bisherigen OBDD-Kenntnissen für allgemeine Instanzen erhalten können. Natürlich erhoffen wir uns beim Einsatz dieser Heuristik ein wesentlich besseres Verhalten.

Die maximale Anzahl von Argumenten mit jeweils $n = \log(|V|) + \mathcal{O}(1)$ Variablen, auf denen Funktionen während des IQ-Algorithmus definiert sind, ist 8. Theorem 2.3.9, Theorem 2.3.17 (Punkt 9) und Theorem 3.3.1 implizieren zusammen:

Korollar 3.6.2. *Die Laufzeit des IQ-Algorithmus ist stets durch $\mathcal{O}(|V|^{17})$ beschränkt, sein Platz ist durch $\mathcal{O}(|V|^{16}/\log^2|V|)$ beschränkt.*

Diese Schranken werden von den binären Synthesen und Quantifizierungen dominiert, die dabei als quadratisch in der Worst-case-OBDD-Größe angenommen werden.

3.7 OBDD-Operationen und Vollständige-OBDD-Breite

Bevor wir zur Analyse des IQ-Algorithmus auf Gittergraphen kommen, stellen wir zunächst die Zutaten der Analysetechnik für OBDD-basierte Algorithmen vor, die eine zentrale Rolle in dieser Dissertation spielen wird. Bei der Analyse einer Folge von OBDD-Operationen, aus denen implizite Algorithmen im Wesentlichen bestehen, stößt man auf zwei Kernprobleme:

- Mit Theorem 2.3.17 (Punkt 9) können wir die OBDD-Größe des Resultats von k binären Synthesen lediglich durch $\mathcal{O}(\text{size}^{k-1}(\pi\text{-OBDD}[f]))$ beschränken, wenn f das größte minimale OBDD unter der teilnehmenden Funktionen besitzt.
- Eine Quantifizierung über ein Knotenargument $x \in \mathbb{B}^n$ verursacht $\Omega(n) = \Omega(\log |V|)$ binäre Synthesen, für die wir momentan keine Exponenzierung der OBDD-Größe ausschließen können.

Der Ausweg aus diesem Dilemma ist nun, die Breite w des hypothetischen vollständigen OBDDs einer Funktion $f \in B_n$ als Festparameter im Sinne des Konzepts der parametrisierten Komplexität zu betrachten (siehe Abschnitt 2.2). Es wird sich zeigen, dass wir die Exponenzierung auf diesen Festparameter beschränken können und damit für kleine Vollständige-OBDD-Breite polynomiell in n bleiben. Wir kürzen diesen Parameter als „Voll-OBDD-Breite“ ab.

Seien $f_1, f_2 \in B_n$ definiert auf der Variablenmenge $X := \{x_0, \ldots, x_{n-1}\}$; f_1 habe Voll-OBDD-Breite w_1 und f_2 habe Voll-OBDD-Breite w_2 für die Variablenordnung $\pi \in \Sigma_X$.

Lemma 3.7.1. *Das Ergebnis der binären Synthese* π-OBDD$[f_1 \otimes f_2]$ *für* $\otimes \in B_2$ *hat höchstens Voll-OBDD-Breite* $w_1 w_2$ *und wird von Algorithmus 2.3.1 in Zeit* $\mathcal{O}(nw_1w_2 \log(nw_1w_2))$ *auf Platz* $\mathcal{O}(nw_1w_2)$ *berechnet.*

Beweis: Sei o. B. d. A. $\pi = \text{id}$. Wir beweisen die Aussage des Lemmas zunächst für die Berechnung von id-OBDD$_c[f_1 \otimes f_2]$ aus id-OBDD$_c[f_1]$ und id-OBDD$_c[f_2]$, wobei die Unique-Table nun auch innere Knoten mit gleichen Nachfolgern erzeugt. Sei G_c^* der entsprechende vom Knoten (q_{f_1}, q_{f_2}) aus erreichbare Teilgraph des vollständigen Produktgraphen. Betrachte die OBDD-Knoten v_1 in id-OBDD$_c[f_1]$ und v_2 in id-OBDD$_c[f_2]$, die den Subfunktionen g_1 bzw. g_2 entsprechen. Weil beide OBDDs vollständig sind, nehmen g_1 und g_2 genau dann an einer rekursiven Berechnung $g_1 \otimes g_2$ teil, wenn v_1 und v_2 mit der gleichen Variable x_i markiert sind. Also haben G_c^*, id-OBDD$_c[f_1 \otimes f_2]$ und id-OBDD$[f_1 \otimes f_2]$ höchstens Breite w_1w_2.

Im Allgemeinen kann der erreichbare Teilgraph G^* des Produktgraphen von id-OBDD$[f_1]$ und id-OBDD$[f_2]$ Größe $\Omega(\text{size(id-OBDD}[f_1]) \cdot \text{size(id-OBDD}[f_2]))$ haben, da rekursive Aufrufe von Algorithmus 2.3.1 Knoten verschiedener Variablenmar-

kierungen kombinieren können (siehe Theorem 3.3.7 in Wegener (2000)). Wir werden dieses kombinatorische Wachstum auf die festen Voll-OBDD-Breiten begrenzen:

Betrachte OBDD-Knoten v_1 in id-OBDD$[f_1]$ und v_2 in id-OBDD$[f_2]$, die den Subfunktionen g_1 bzw. g_2 entsprechen. Sei v_1 mit x_i markiert und v_2 mit x_j für $i \leq j$. Wenn G^* einen Produktknoten (v_1, v_2) enthält, dann enthält G_c^* einen Knoten (v_1, v_2'), der ebenfalls $g_1 \otimes g_2$ darstellt. Knoten v_2' stellt g_2 dar und ist mit x_i markiert. Weil jedes Paar (g_1, g_2) von Subfunktionen von f_1 und f_2 höchstens einmal in G^* repräsentiert ist, gibt es eine injektive Abbildung von der Knotenmenge von G^* in die Knotenmenge von G_c^*. Somit kann G^* nicht größer sein als G_c^* und Algorithmus 2.3.1 wird auch im nicht vollständigen Fall rekursiv höchstens nw_1w_2-mal instanziiert. Zeit- und Platzschranke folgen gemäß Theorem 2.3.17 (Punkt 9). □

Lemma 3.7.2. *Sei $X' \subseteq X$. Das Ergebnis der Quantifizierung π-OBDD$[(\mathcal{Q}X')f_1]$ für $\mathcal{Q} \in \{\exists, \forall\}$ hat höchstens Voll-OBDD-Breite 2^{w_1} und wird durch das im Beweis von Theorem 2.3.17 (Punkt 10) genannte Verfahren in Zeit $\mathcal{O}(|X'|n2^{2w_1}\log(n2^{2w_1}))$ auf Platz $\mathcal{O}(n2^{2w_1})$ berechnet.*

Beweis: Seien $X' = \{x_{i_1}, \ldots, x_{i_r}\}$ und o. B. d. A. $\pi = \mathrm{id}$. $\mathcal{V}_i$ bezeichne die Menge der OBDD-Knoten in id-OBDD$_c[f_1]$, die mit x_i markiert sind. Wir konstruieren ein nicht zwingenderweise minimales OBDD $\mathcal{G}^*$, das f_1 darstellt: Es besteht aus der Knotenmenge $\mathcal{V}^* := \bigcup_{0 \leq i < n} \mathcal{P}(\mathcal{V}_i)$, wobei $v \in \mathcal{P}(\mathcal{V}_i)$ mit x_i markiert ist. Außerdem definieren wir den a-Nachfolger $v(a)$ eines Knotens für $a \in \mathbb{B}$ durch $v(a) := \bigcup_{w \in v}\{w(a)\}$. Der Quellenknoten von $\mathcal{G}^*$ ist $\{q_{f_1}\}$ für den Quellenknoten q_{f_1} von id-OBDD$[f_1]$.

Sei nun $\mathcal{G}_j^*$ das OBDD des Zwischenergebnisses $(\mathcal{Q}x_{i_{j-1}} \ldots x_{i_1})f_1 =: g_{j-1}$. Wir führen die Quantifizierung über x_{i_j} durch Ersetzung aller Kanten (u, v) durch $(u, v(0) \cup v(1))$ für alle Knoten $v \in \mathcal{P}(\mathcal{V}_{i_j})$ durch. Wenn bei der Auswertung von $\mathcal{G}_j^*$ der Knoten $v(0) \cup v(1)$ besucht wird, so repräsentiert dieser alle Knoten im ursprünglichen OBDD π-OBDD$_c[f_1]$ für alle möglichen Zuweisungen der quantifizierten Variablen $x_{i_1}, \ldots, x_{i_j}$.

Bei der Konstruktion von $\mathcal{G}^*$ und $\mathcal{G}_1^*, \ldots, \mathcal{G}_r^*$ erzeugen wir außerdem Terminalteilmengen von $\mathbb{B}$. Bei einer Folge von Existenzquantoren werden dann $\{1\}$ und $\{0, 1\}$ durch das Terminal 1 ersetzt und alle anderen durch 0 – es genügt, wenn eine mögliche Variablenzuweisung zu einem 1-Terminal führt. Bei einer Folge von Allquantoren wird folglich nur die Menge $\{1\}$ durch Terminal 1 ersetzt – alle Zuweisungen müssen zu einem 1-Terminal führen.

Wir haben ein vollständiges OBDD $\mathcal{G}_r^*$ für $(\mathcal{Q}X')f_1$ der Breite 2^{w_1} konstruiert. Die Schranke gilt natürlich auch für das minimale vollständige OBDD. Dazu haben wir $r = |X'|$ Quantifizierungen $(\mathcal{Q}x_{i_j})$ durchgeführt, die jeweils eine binäre Synthese $(g_{j-1} \mid x_{i_j} := 0) \otimes (g_{j-1} \mid x_{i_j} := 1)$ mit sich bringen. Da auch g_{j-1} höchstens Voll-OBDD-Breite 2^{w_1} hat, impliziert Lemma 3.7.1, dass jede der r binären Synthesen Zeit $\mathcal{O}(n2^{2w_1}\log(n2^{2w_1}))$ und Platz $\mathcal{O}(n2^{2w_1})$ benötigt. □

Für Umordnungen von k Argumenten hat sich bereits in Theorem 2.5.6 gezeigt, dass die OBDD-Größe dabei höchstens um den Faktor 3^k wächst. Im Hinblick auf die neu gewonnenen Resultate ist es jedoch nun auch wichtig, die Voll-OBDD-Breite im Auge zu behalten.

Sei also $f_3 \in B_{kn}$ auf k Argumenten $x^{(1)}, \ldots, x^{(k)} \in \mathbb{B}^n$ definiert; f_3 habe Voll-OBDD-Breite w_3 für die Variablenordnung $\pi^\tau_{k,n}$ mit $\tau \in \Sigma_n$. Sei $\rho \in \Sigma_k$.

Lemma 3.7.3. *Das Ergebnis der Argumentumordnung $\pi^\tau_{k,n}$-OBDD$[\mathcal{R}_\rho(f_3)]$ hat höchstens Voll-OBDD-Breite $3^k w_3$ und wird vom im Beweis von Theorem 2.5.6 angegebenen Verfahren in Zeit $\mathcal{O}(k^2 3^k w_3 n)$ auf Platz $\mathcal{O}(3^k w_3 kn)$ berechnet.*

Beweis: Analog zum Beweis von Theorem 2.5.6 kann man durch Betrachtung der beiden möglichen und in Abbildung 3.7.1 dargestellten Situationen bei einem benachbarten Variablentausch einsehen, dass sich bei der Berechnung von $\pi^\tau_{k,n}$-OBDD$[\mathcal{R}_\rho(f_3)]$ auch die Breite von $\pi^\tau_{k,n}$-OBDD$_c[f_3]$ für jede Sequenzenfolge $S_{0,j}, \ldots, S_{n-1,j}$ mit $1 \leq j \leq k$ höchstens verdreifacht. Aufgrund von $\pi^\tau_{k,n}$-OBDD$_c[f_3] \leq w_3 kn$ folgt die Aussage. □

Nachdem wir alle kritischen OBDD-Operationen betrachtet haben, die OBDDs vergrößern können, formulieren wir die Ergebnisse dieses Abschnitts in einem Theorem, das Grundlage aller oberen Laufzeitschranken dieser Dissertation sein wird.

Theorem 3.7.4. *Sei $F := \{f_1, \ldots, f_m\} \subseteq B_{kn}$ eine Menge von Funktionen, die auf den Argumenten $x^{(1)}, \ldots, x^{(k)} \in \mathbb{B}^n$ für konstantes k definiert sind. Sei w die maximale Voll-OBDD-Breite von F bez. der Variablenordnung $\pi^\tau_{k,n}$ für ein $\tau \in \Sigma_n$. Wir betrachten eine Folge $S = (\sigma_1, \ldots, \sigma_r)$ von $r = \mathcal{O}(1)$*

- *funktionalen Operationen gemäß Definition 2.3.16, Punkte 1–10,*
- *Argumentumordnungen gemäß Definition 2.5.5 und*
- *Quantifizierungsblöcken $(\mathcal{Q}X')$ für $\mathcal{Q} \in \{\exists, \forall\}$ und $X' \subseteq (x^{(1)} \cup \cdots \cup x^{(k)})$*

angewendet auf Funktionen aus F oder Ergebnisfunktionen, die vom bisherigen Präfix von S generiert wurden.

Alle von S erzeugten Funktionen haben Voll-OBDD-Breite von höchstens $\alpha(w)$ bez. $\pi^\tau_{k,n}$ für eine geeignete Funktion $\alpha\colon \mathbb{N} \to \mathbb{N}$. S kann als OBDD-basierter FPT-Algorithmus auf $\pi^\tau_{k,n}$-OBDD$[F]$ mit Parameter w, Laufzeit $\beta(w) \cdot n^2 \log n$ und Platz $\beta(w) \cdot n$ für ein geeignetes $\beta\colon \mathbb{N} \to \mathbb{N}$ implementiert werden.

Beweis: Wir führen eine vollständige Induktion über die Präfixlängen $\ell := 0, \ldots, r$ von S durch. Sei F_ℓ die Vereinigung von F und aller vom Präfix $(\sigma_1, \ldots, \sigma_\ell)$ erzeugten Funktionen. Für $\ell = 0$ gilt die Aussage wegen $F_0 = F$ offensichtlich. Für $\ell > 0$ ist die Aussage laut Induktionsvoraussetzung für $(\sigma_1, \ldots, \sigma_{\ell-1})$ und $F_{\ell-1}$ bereits erfüllt.

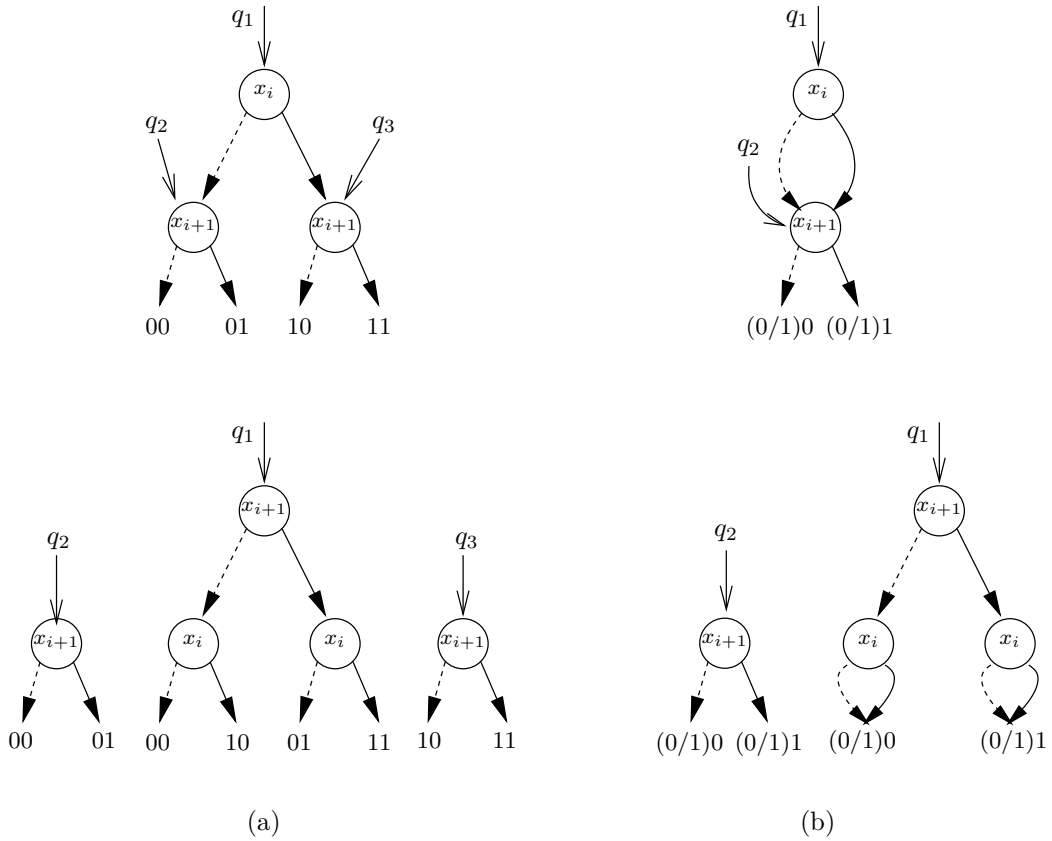


Abbildung 3.7.1: Die beiden möglichen Situationen bei einem benachbarten Variablentausch (oben vorher, unten nachher). Der ursprüngliche x_i-Knoten wird durch einen x_{i+1}-Knoten ersetzt und insgesamt entstehen pro ehemaligen x_i-Knoten höchstens zwei neue. Dabei seien q_1, q_2 und q_3 Funktionszeiger.

Es gibt also geeignete Funktionen $\alpha_{\ell-1}$ und $\beta_{\ell-1}$ und $F_{\ell-1}$ hat maximale Voll-OBDD-Breite $\alpha_{\ell-1}(w)$ bez. $\pi^{\tau}_{k,n}$.

Wir betrachten das Ergebnis g_ℓ von σ_ℓ. Unter den Operationen 1–10 von Definition 2.3.16 können lediglich binäre Synthesen und Quantifizierungen die Voll-OBDD-Breite vergrößern. Für diese Fälle sowie für Quantifizierungsblöcke und Argumentumordnungen zeigt Einsetzen von $\alpha_{\ell-1}(w)$ in die Breitenschranken w_1, w_2 und w_3 in Lemma 3.7.1, Lemma 3.7.2 bzw. Lemma 3.7.3, dass g_ℓ höchstens Voll-OBDD-Breite $2^{c\cdot\alpha_{\ell-1}(w)} =: \alpha_\ell$ bez. $\pi^{\tau}_{k,n}$ für ein geeignetes konstantes c hat.

Wir betrachten Laufzeit und Platz eines Quantifizierungsblocks, der die anderen zu betrachtenden Operationen dominiert: Wegen $|X'| \leq kn = \mathcal{O}(n)$ folgen hier die Zeitschranke $2^{d\cdot\alpha_{\ell-1}(w)} \cdot n^2 \log n$ und die Platzschranke $2^{d\cdot\alpha_{\ell-1}(w)} \cdot n$ für ein geeignetes

konstantes d. Zusammen mit den Ressourcen von $(\sigma_1, \ldots, \sigma_{\ell-1})$ ergeben sich also Zeit $\beta_\ell(w) \cdot n^2 \log n$ und Platz $\beta_\ell(w) \cdot n$ für $\beta_\ell = \big(\beta_{\ell-1}(w) + 2^{d \cdot \alpha_{\ell-1}(w)}\big)$.

Wir bemerken, dass wegen $r = \mathcal{O}(1)$ auch $\alpha_r = \alpha$ und $\beta_r = \beta$ unabhängig von n sind. □

Informell ausgedrückt wachsen Voll-OBDD-Breite, Laufzeit und Platz während einer konstanten Anzahl von OBDD-Operationen nur polynomiell in der Anzahl n der Variablen pro Argument, jedoch möglicherweise exponentiell in der maximalen Voll-OBDD-Breite w. Dabei haben wir uns auf eine konstante Argumentanzahl k geeinigt und ausschließlich auf Umordnungen ganzer Argumente statt einzelner Variablenpaare beschränkt, da dies gerade die interessanten Fälle im Rahmen impliziter Algorithmen umfasst.

Theorem 2.4.7, Beobachtung 2.4.9 und Lemma 3.7.1 implizieren nun:

Korollar 3.7.5. *Funktionen $f \in \mathbb{V}_{k,n}^W$ haben Voll-OBDD-Breite* $\mathrm{poly}(kW)$ *bez. $\pi_{k,n}^{\mathrm{id}}$. Entsprechende OBDDs können in Zeit* $\mathrm{poly}(kW) \cdot n \log n$ *erzeugt werden.*

Die in Anmerkung 2.5.7 erstmals betrachtete Funktion

$$\chi_V(x) := \bigwedge_{j=1}^{k} \big(|x^{(j)}| < t_j\big)$$

kann gemäß Beobachtung 2.4.9 durch $\mathcal{O}(k)$ binäre Synthesen und Negationen von multivariaten Thresholdfunktionen mit konstantem maximalem absolutem Gewicht erzeugt werden. Aus Theorem 2.4.7 und Lemma 3.7.1 folgt nun:

Korollar 3.7.6. *Die Funktion $\chi_V(x)$ hat Voll-OBDD-Breite $2^{\mathcal{O}(k \log k)}$ bez. $\pi_{k,n}^{\mathrm{id}}$. Das entsprechende OBDD kann in Zeit $2^{\mathcal{O}(k \log k)} \cdot n \log n$ erzeugt werden.*

Die Voll-OBDD-Breite der aktualisierten Kantenrelation $\chi_G(x, y) \wedge \chi_V(x) \wedge \chi_V(y)$ gewinnt für $k = \mathcal{O}(1)$ offenbar nur einen konstanten Faktor. Dies gilt insbesondere für den einfachen Spezialfall $\chi_V(x) = \bigwedge_{j=1}^{k} \bigwedge_{i=n}^{n+m-1} (x_i^{(j)} = 0)$ bei der Hinzufügung von insgesamt km Variablen.

Die Prioritätsfunktion Π_{dp} kann gemäß Beobachtung 2.4.9 und Anmerkung 3.2.4 durch $\mathcal{O}(k^2)$ binäre Synthesen und Negationen von multivariaten Thresholdfunktionen mit konstantem maximalem absolutem Gewicht erzeugt werden. Aus Theorem 2.4.7 und Lemma 3.7.1 folgt nun:

Korollar 3.7.7. *Die Prioritätsfunktion Π_{dp} hat Voll-OBDD-Breite $2^{\mathcal{O}(k^2 \log k)}$ bez. $\pi_{k,n}^{\mathrm{id}}$. Das entsprechende OBDD kann in Zeit $2^{\mathcal{O}(k^2 \log k)} \cdot n \log n$ erzeugt werden.*

Die Technik von Woelfel (2003) basiert zwar auch auf beschränkten Voll-OBDD-Breiten, ist jedoch zum einen auf einen speziellen Binäre-Synthese-Algorithmus angewiesen, der im Rahmen einer allgemeinen OBDD-basierten Heuristik nicht angewendet würde. Zudem erlaubt er nur stark eingeschränkte Quantifizierungsblöcke über Argumente.

3.8 Analyse auf Gittergraphen

Nun wenden wir die Ergebnisse des vorigen Abschnitts zur Analyse des IQ-Algorithmus auf quadratischen zweidimensionalen Gittergraphen an. In Gebieten, in denen theoretische Analysen spezielle Methoden erfordern, findet die Anwendung entsprechender neuer Techniken oft zunächst für einfache Beispielinstanzen statt. Jedoch stellen solche einfachen Instanzen einen Startpunkt dar, um exemplarisch die Möglichkeiten einer Analysetechnik aufzuzeigen. Bei der Analyse randomisierter Suchheuristiken ist dies z. B. die Maximierung der Einsen in einer binären Zeichenkette (Droste u. a., 2002). Die Lösung ist offensichtlich und man wird diese Funktion nicht tatsächlich mit Rechnerhilfe optimieren wollen.

Woelfel (2003) hat einen OBDD-basierten Algorithmus zum **Topologischen Sortieren** auf zweidimensionalen Gittergraphen analysiert und so eine neue Analysetechnik exemplarisch angewendet, obwohl das entsprechende Lösungs-OBDD auch effizienter erzeugt werden kann, wenn bekannt ist, dass ein Gittergraph vorliegt. Wir werden diese Analysetechnik verbessern und für den IQ-Algorithmus auf Gittergraphen anwenden. Auch hier wäre die Lösung direkt effizienter konstruierbar und es handelt sich um eine exemplarische Analyse. Im Gegensatz zur Technik von Woelfel sind wir nicht auf einen speziellen Binäre-Synthese-Algorithmus angewiesen, der bereits von kleiner Voll-OBDD-Breite ausgeht, sondern können die Resultate auch für den in der Praxis üblichen Algorithmus 2.3.1 zeigen.

3.8.1 Gittergraphen

Wir definieren zunächst formal $N \times N$-Gittergraphen, die wir in diesem Abschnitt verkürzt als G_N bezeichnen:

Definition 3.8.1. *Der* $N \times N$-Gittergraph $G_N := (V_N, E_N)$ *für* $N \in \mathbb{N}$ *besteht aus der Knotenmenge* $V_N := \big\{(i,j) \mid 0 \leq i,j < N\big\}$ *und der Kantenmenge*

$$E_N := \Big\{\big((i,j),\,(k,\ell)\big) \in V_N \times V_N \mid k - i + \ell - j = 1,\, i \leq k,\, j \leq \ell\Big\}.$$

Der Graph kann also als $N \times N$-Knotenmatrix betrachtet werden, dessen Kanten zwischen vertikal oder horizontal benachbarten Knoten in Richtung des jeweils höheren Index verlaufen. Wir interessieren uns dabei für einen maximalen Fluss von $q := (0,0)$ nach $s := (N-1, N-1)$, der offensichtlich Wert 2 besitzt. Die Analyse kann leicht auf nicht quadratische Gittergraphen verallgemeinert werden, in denen die horizontale und vertikale Distanz von q und s identische Zweierpotenzen sind. Abbildung 3.8.1 zeigt G_5.

In der Analyse gehen wir von $N = 2^\kappa + 1$ für $\kappa \in \mathbb{N}^+$ aus, um Niveaunetzwerke der Tiefe $2^{\kappa+1}$ zu erhalten. G_N besteht also aus $2^{2\kappa} + 2^{\kappa+1} + 1$ Knoten, die wir mit $2\kappa + 2$ booleschen Variablen codieren. Obwohl $2\kappa + 1 = \lceil \log |V_N| \rceil$ Variablen ausreichen würden, verwenden wir eine gerade Anzahl, um Zeilen- und Spaltenindex eines

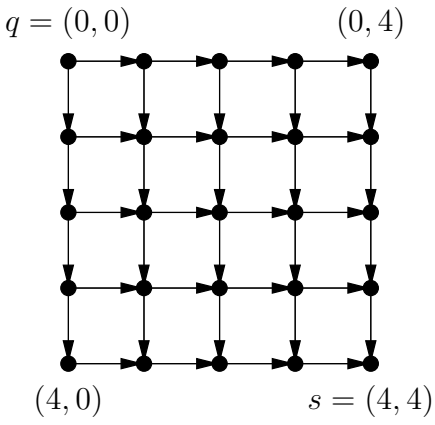


Abbildung 3.8.1: Der 5×5-Gittergraph.

Knotens durch die gleiche Variablenanzahl zu repräsentieren. Knoten (i,j) erhält die Knotennummer $i2^{\kappa+1} + j$.

3.8.2 Variablenordnung

Entsprechend unserer Einführung OBDD-basierter impliziter Algorithmen in Abschnitt 2.5 gehen wir bei einer OBDD-Realisierung des IQ-Algorithmus von einer gemischten Variablenordnung $\pi^{\tau}_{k,n'}$ aus, wobei jedes Knotenargument durch $n' := 2n = 2\kappa + 2$ Variablen codiert ist. Die charakteristische Funktion χ_{G_N} ist zwar nur auf $2n'$ Variablen definiert, der Algorithmus benötigt aber offensichtlich eine konstante Anzahl $k-2$ von Hilfsvariablenvektoren. Aus Sicht desjenigen, der die lokale Variablenordnung τ der Variablen eines Knotenarguments $x \in \mathbb{B}^{n'}$ für die Eingabe festlegt, macht es jedoch zudem Sinn, die Variablenvektoren $row(x), col(x) \in \mathbb{B}^n$ der Zeilen- und Spaltenindizes ebenfalls zu mischen und diese dann im Hinblick auf kompakte OBDDs für multivariate Thresholdfunktionen natürlich zu ordnen. Die resultierende Ordnung

$$\Big(row(x^{(1)})_0, \ldots, row(x^{(k)})_0, col(x^{(1)})_0, \ldots, col(x^{(k)})_0, row(x^{(1)})_1, \ldots\ldots, col(x^{(k)})_{n-1}\Big)$$

entspricht aber offensichtlich $\pi^{\mathrm{id}}_{2k,n}$. So gehen wir im Sinne einer vereinfachten Notation von dieser Variablenordnung aus.

3.8.3 Analyse

Ein maximaler q-s-Fluss in G_N hat Wert 2. Es reicht eine Phase, deren Niveaunetzwerk $U(x,y) = \chi_{G_N}(x,y)$ alle Kanten als Vorwärtskanten enthält. Die Anzahl der Sweeps

ist somit durch 2 beschränkt und es werden $\mathcal{O}(\log^2 |V|) = \mathcal{O}(n^2) = \mathcal{O}(\kappa^2)$ OBDD-Operationen durchgeführt. Der HS-Algorithmus benötigt hingegen offensichtlich durch seine sequenzielle Berechnung und Speicherung der Niveaunetzwerkebenen Zeit und Platz $\Omega(2^{\kappa+1})$.

Wir betrachten zunächst die Voll-OBDD-Breite der Eingabe selbst:

Lemma 3.8.2. *Sei $n := \kappa+1$. $\chi_{G_{2^\kappa+1}}$ hat Voll-OBDD-Breite $\mathcal{O}(1)$ bez. $\pi^{\mathrm{id}}_{4,n}$ und somit $\pi^{\mathrm{id}}_{4,n}$-OBDD-Größe $\mathcal{O}(n)$.*

Beweis: $\chi_{G_{2^\kappa+1}}(x,y)$ lässt sich durch

$$\begin{aligned}\chi_{G_{2^\kappa+1}}(x,y) = (&|row(x)|, |col(x)|, |row(y)|, |col(y)| < 2^\kappa + 1)\\ &\wedge (|row(y)| - |row(x)| + |col(y)| - |col(x)| = 1)\\ &\wedge (|row(y)| - |row(x)| \geq 0) \wedge (|col(y)| - |col(x)| \geq 0) \quad (3.8.1)\end{aligned}$$

darstellen und somit durch eine konstante Anzahl binärer Synthesen und Negationen von Funktionen aus $\mathbb{V}^{\mathcal{O}(1)}_{4,n}$ erzeugen. Aus Lemma 3.7.1 und Korollar 3.7.5 folgt Voll-OBDD-Breite $\mathcal{O}(1)$. □

Wir gehen zudem davon aus, dass nach jeder Phase geprüft wird, ob der aktuelle Wert von $\chi_F(x,y)$ bereits dem Grad des Quellenknotens q entspricht – dann muss kein weiteres Niveaunetzwerk berechnet werden. Diese Erweiterung des IQ-Algorithmus ist effizient realisierbar: Wir ersetzen Argument x in $\chi_F(x,y)$ und $\chi_G(x,y)$ durch $\langle q\rangle_2$ (siehe Theorem 2.3.17, Punkt 8) und vergleichen die jeweilige Anzahl erfüllender Belegungen (siehe Theorem 2.3.17, Punkt 3). Ebenso nehmen wir an, dass die Erreichbarkeitsanalyse in Zeile 11 nur dann ausgeführt wird, wenn q im nach den bisherigen Sweeps verbleibenden Niveaunetzwerk U noch ausgehende Kanten hat (entsprechend dem Test $U(\langle q\rangle_2, y) = 0?$). Diese Tests haben also lineare Laufzeit in den $\pi^{\mathrm{id}}_{2k,n}$-OBDD-Größen von $\chi_F(x,y)$, $\chi_G(x,y)$ und $U(x,y)\wedge\overline{B^*(x,y)}$. Allerdings kann die Analyse auch ohne diese Annahmen mit etwas mehr Aufwand durchgeführt werden.

Wir konzentrieren uns also im Folgenden auf die erste Phase. Der Beweis des folgenden Haupttheorems wird erst am Ende von Abschnitt 3.8.5 abgeschlossen.

Theorem 3.8.3. *Sei $n := \kappa + 1$. Der IQ-Algorithmus berechnet einen maximalen q-s-Fluss $\pi^{\mathrm{id}}_{4,n}$-OBDD$[\chi_F]$ auf $\pi^{\mathrm{id}}_{4,n}$-OBDD$[\chi_{G_{2^\kappa+1}}]$ in Zeit $\mathcal{O}(n^3 \log n)$ auf Platz $\mathcal{O}(n^2)$.*

Das Flussproblem wird vom IQ-Algorithmus also in polynomieller Zeit in der Eingabe-OBDD-Größe bzw. in polylogarithmischer Zeit bez. $|V_{2^\kappa+1}|$ gelöst. Zunächst zeigen wir für eine Auswahl der während des IQ-Algorithmus auftretenden Funktionen, dass sie durch eine konstante Anzahl binärer Synthesen und Negationen von Funktionen in $\mathbb{V}^{\mathcal{O}(1)}_{k,n}$ (und somit auch in $\mathbb{T}^{\mathcal{O}(1)}_{k,n}$) gewonnen werden kann, wobei k nun die konstante Gesamtanzahl an Knotenargumenten sei, die wir während des gesamten IQ-Algorithmus benötigen. Aus Lemma 3.7.1 und Korollar 3.7.5 folgt dann Voll-OBDD-Breite $\mathcal{O}(1)$ bez. $\pi^{\mathrm{id}}_{2k,n}$ für all diese betrachteten Funktionen.

3.8.4 Niveaunetzwerkkonstruktion

Wir beginnen mit einigen grundlegenden Eigenschaften des Niveaunetzwerks U. Der maximale q-s-Flusswert ist 2. Jeder Pfad von q nach s ist ein kürzester flussverbessernder Pfad der Länge $2^{\kappa+1}$, der ausschließlich aus Vorwärtskanten besteht. Unabhängig von der Struktur eines ausgewählten q-s-Pfades verbleibt ein zweiter in U. Somit besteht jeder Sperrfluss in U aus zwei Pfaden und es genügt stets eine Phase.

Fakt 3.8.4. *Es ist* $U(x,y) = \chi_{G_{2^\kappa+1}}(x,y)$.

Fakt 3.8.5. *Es gibt genau dann einen* $v_{|x|}$-$v_{|y|}$*-Pfad der Länge* ℓ *in* $G_{2^\kappa+1}$*, wenn die Differenzen der Zeilen- und Spaltenindizes der beiden Knoten nicht negativ sind und ihre Summe* ℓ *ergibt, formal:*

$$(|row(y)| - |row(x)| \geq 0) \wedge (|col(y)| - |col(x)| \geq 0) \\ \wedge (|row(y)| - |row(x)| + |col(y)| - |col(x)| = \ell)\,.$$

Fakt 3.8.6. *Jeder* q-$v_{|x|}$*-Pfad hat Länge* $|row(x)| + |col(x)|$ *und Knoten* $v_{|x|}$ *liegt in Knotenebene* $V_{|row(x)|+|col(x)|}$.

Vor der ersten Phase ist $\chi_F(x,y) = 0$ und $EF(x,y) = \chi_{G_{2^\kappa+1}}(x,y)$. Wir betrachten die Konstruktion von U nun getrennt nach Vorverarbeitung, Partitionierung und Kantenberechnung.

Vorverarbeitung

Im Vorverarbeitungsschritt werden die Funktionen $PFAD^{\leq}_{\gamma}(x,y)$, $PFAD^{=}_{\gamma}(x,y)$, $PFAD^{<}_{\gamma}(x,y)$, $KPFAD_{\gamma}(x,y)$ und $EF_{\gamma}(\nu,w,x,y)$ für $\gamma := 0,\ldots,\gamma^*$ und ein $\gamma^* \leq \lceil \log |V| \rceil$ berechnet. Konkret ist offenbar $\gamma^* = \kappa$. Fakt 3.8.4 und Fakt 3.8.5 implizieren

$$PFAD^{\leq}_{\gamma}(x,y) = (|row(y)| - |row(x)| \geq 0) \wedge (|col(y)| - |col(x)| \geq 0) \\ \wedge (|row(y)| - |row(x)| + |col(y)| - |col(x)| \leq 2^\gamma)$$

für $\gamma := 0,\ldots,\gamma^*$.

Dementsprechend gelten

$$PFAD^{=}_{\gamma}(x,y) = (|row(y)| - |row(x)| \geq 0) \wedge (|col(y)| - |col(x)| \geq 0) \\ \wedge (|row(y)| - |row(x)| + |col(y)| - |col(x)| = 2^\gamma)$$

und

$$PFAD^{<}_{\gamma}(x,y) = (|row(y)| - |row(x)| \geq 0) \wedge (|col(y)| - |col(x)| \geq 0)$$

$$\wedge\,(|row(y)| - |row(x)| + |col(y)| - |col(x)| < 2^{\gamma})$$

für $\gamma := 0, \ldots, \gamma^*$.

Da alle Pfade in $G_{2^\kappa+1}$ kürzeste flussverbessernde Pfade in der ersten Phase sind, ist offenbar $KPFAD_\gamma = PFAD_\gamma^=$. Wir betrachten schließlich $EF_\gamma(\nu, w, x, y)$. Eine Kante $(v_{|x|}, v_{|y|})$ liegt genau dann auf einem Rest-$v_{|\nu|}$-$v_{|w|}$-Pfad der Länge 2^γ, wenn $(v_{|x|}, v_{|y|}) \in E_{2^\kappa+1}$, $PFAD_\gamma^=(\nu, w) = 1$ und $v_{|x|}$ und $v_{|y|}$ im von $v_{|\nu|}$ und $v_{|w|}$ aufgespannten Rechteck liegen:

$$\begin{aligned} EF_\gamma(\nu, w, x, y) = {} & \chi_{G_{2^\kappa+1}}(x, y) \wedge PFAD_\gamma^=(\nu, w) \\ & \wedge\,(|row(x)| \geq |row(\nu)|) \wedge (|row(x)| < |row(w)|) \\ & \wedge\,(|col(x)| \geq |col(\nu)|) \wedge (|col(x)| < |col(w)|) \end{aligned}$$

für $\gamma := 0, \ldots, \gamma^*$.

Partitionierung des Niveaunetzwerks

Wir betrachten alle im Modul findeSenke auftauchenden Funktionen (siehe Algorithmus 3.4.1). Die Vergleiche $(x = \langle q \rangle_2)$ und $(x = \langle s \rangle_2)$ haben offenbar konstante Voll-OBDD-Breite bez. $\pi^{\mathrm{id}}_{2k,n}$. Zu betrachten bleiben die in allen Rekursionsinstanzen von findeSenke erzeugten Funktionen $Z'(x)$ und $R'(x)$, da diese wiederum den Aufrufparametern weiterer Rekursionen entsprechen.

Wurde vor dem Aufruf der aktuellen Instanz von findeSenke i-mal in Fall 3 verzweigt, so wurden bisher bereits die Knotenebenen $0, \ldots, t$ mit $t := \sum_{j=0}^{i-1} 2^{\gamma_j}$ behandelt. Das aktuelle Argument $Z(x)$ entspricht also der Knotenebene t. Durch die Berechnung von $Z'(x)$ schreitet findeSenke 2^{a-1} weitere Ebenen voran und $Z'(x)$ repräsentiert die Knoten der Ebene $t + 2^{a-1} =: t'$. Gemäß Fakt 3.8.6 gilt

$$Z'(x) = (|row(x)| + |col(x)| = t').$$

Analog dazu repräsentiert $R(x)$ alle Knoten der Ebenen $0, \ldots, t$ von U, zu denen bei der Berechnung von $R'(x)$ die nächsten 2^{a-1} Knotenebenen hinzugefügt werden. Demnach ist

$$R'(x) = (|row(x)| + |col(x)| \leq t').$$

Berechnung der Niveaunetzwerkkanten

Schließlich müssen wir die Funktionen $C_i(x, y)$ für $i := 0, \ldots, r-1$ und $C(x, y)$ betrachten, sowie die Funktionen, die im Rahmen der Erreichbarkeitsanalyse auf $C^{\mathrm{R}}(x, y)$ auftreten.

$C_i(x, y)$ stellt die Kanten der Partitionskomponente i dar, die aus den Kantenebenen $\sum_{j=0}^{i-1} 2^{\gamma_j} =: \eta, \ldots, \eta + 2^{\gamma_i} =: \eta'$ von U besteht. Dies entspricht gemäß Fakt 3.8.4 und Fakt 3.8.6 dem Ausdruck

$$C_i(x, y) = \chi_{G_{2^\kappa+1}}(x, y) \wedge (\eta \leq |row(x)| + |col(y)| < \eta')\,. \tag{3.8.2}$$

$C(x,y)$ wird dann als Disjunktion aller $C_i(x,y)$ berechnet. Wir betrachten das i-te Zwischenergebnis $C^{(i)}(x,y) := \bigvee_{j=0}^{i} C_i(x,y)$. Entsprechend unserer obigen Überlegungen repräsentiert $C^{(i)}(x,y)$ die Kantenebenen $0,\ldots,\eta'$ und wir können es durch leichte Änderung von Gleichung (3.8.2) ausdrücken:

$$C^{(i)}(x,y) = \chi_{G_{2^{\kappa}+1}}(x,y) \wedge (|row(x)| + |col(y)| < \eta') \,. \tag{3.8.3}$$

Nun zur Erreichbarkeitsanalyse auf $C^{\mathrm{R}}(x,y)$: In der i-ten Iteration der Repeat-Schleife in Algorithmus 2.5.2 stellt $A(x,y)$ alle Knotenpaare $(v_{|x|}, v_{|y|})$ mit $v_{|x|}$-$v_{|y|}$-Pfaden dar, die nicht länger als 2^i sind. Also ist dann $A(x,y) = PFAD_i^{\leq}(y,x)$, wobei letztere Funktion bereits betrachtet wurde.

Wir betrachten die Folge F aller booleschen Funktionen, die im Modul berechneU (welches ja lediglich einmal aufgerufen wird) von

- funktionalen Operationen gemäß Definition 2.3.16, Punkte 1–9,
- Argumentumordnungen gemäß Definition 2.5.5 und
- Quantifizierungsblöcken $(\exists X')$ für eine Variablenteilmenge X'

erzeugt werden. Diese Operationen werden auf die Eingabe $\chi_{G_{2^{\kappa}+1}}$, Hilfsfunktionen in $\mathbb{V}_{2k,n}^{\mathcal{O}(1)}$ mit Voll-OBDD-Breite $\mathcal{O}(1)$ (nämlich Gleichheitstest wie z. B. in Gleichung (3.4.1), siehe Korollar 3.7.5) sowie auf Zwischenergebnisse angewandt. Sei $H \subset F$ die Menge der booleschen Funktionen, für die wir in diesem Abschnitt bereits konstante Voll-OBDD-Breite nachgewiesen haben.

Beobachtung 3.8.7. *Jede Teilsequenz $F' = (f_1,\ldots,f_d)$ von F mit $f_1, f_d \in H$ und $f_i \notin H$ für $i \in \{2,\ldots,d-1\}$ hat Länge $\mathcal{O}(1)$.*

3.8.5 Sperrflussberechnung

Wir betrachten die Sperrflussberechnung und nutzen erstmalig, dass die Niveaunetzwerktiefe $\ell = 2^{\kappa+1}$ eine Zweierpotenz ist. Zunächst wird eine Multipfadkonstruktion gestartet. Wir gehen davon aus, dass eine intelligente Implementation des IQ-Algorithmus lediglich Pfadlängen $\ell_m^1 = 2^{\kappa-m+1}$ für $m := 0,\ldots,\kappa+1$ berücksichtigt, da die Längen $\ell_m^2 = \ell_m^1 - 1$ für Zweierpotenzen ℓ nicht benötigt werden. Wir werden sehen, dass ein Aufruf von multipfad genügt, um beide flussverbessernden Pfade zu erzeugen. Dabei beweisen wir, dass die Funktionen $D_{\ell_m^1}$ und $S_{\ell_m^1}$ ebenfalls durch eine konstante Anzahl binärer Synthesen und Negationen von Funktionen aus $\mathbb{V}_{2k,n}^{\mathcal{O}(1)}$ erzeugt werden können. So folgt wieder aus Lemma 3.7.1 und Korollar 3.7.5 Voll-OBDD-Breite $\mathcal{O}(1)$ bez. $\pi_{2k,n}^{\mathrm{id}}$. Für D_1 und S_1 gilt dies offensichtlich (siehe Gleichung (3.6.1) und Gleichung (3.6.2)).

Wir gehen von der Verwendung der Prioritätsfunktion Π_{dp} aus, die kleine Zeilen-/Spaltenindizes bevorzugt. Im Allgemeinen scheint sie aufgrund ihrer Unabhängigkeit vom ersten Argument x nicht gut für den Einsatz im Rahmen der Multipfadkonstruktion zu sein. In unserem Fall hilft ihre einfache Struktur jedoch bei der Analyse. Man kann von Implementationen ausgehen, die die Pfadkonstruktionen für verschiedene Prioritätsfunktionen parallel ausführen und schließlich das bessere Resultat übernehmen. So stellen unsere folgenden Ergebnisse für Π_{dp} eine obere Schranke dar. Wir erinnern uns:

$$\Pi_{\mathrm{dp}}(x, y, z) = 1 :\Leftrightarrow (|y| < |z|).$$

In unserem Fall spielt das Argument x also keine Rolle, während y und z jeweils konkatenierte binäre Nummern zweier Knoten sind. Jede Knotennummer ist wiederum ein Zeilen-Spalten-Index-Paar:

$$\Pi_{\mathrm{dp}}(\nu, w, y, z) = 1 :\Leftrightarrow (|\nu w| < |yz|)$$

für $\nu, w, y, z \in \mathbb{B}^{2n}$.

Wir betrachten nun die sieben Fälle, die bei der ersten Kantenselektion zur Berechnung von D_2 auftauchen:

1. Abbildung 3.8.2(a): Weil $v_{|x|}$ einen kleineren Zeilenindex als $v_{|y|}$ hat, wählen beide Kanten $(v_{|u|}, v_{|w|})$ und $(v_{|\nu|}, v_{|w|})$ die Kante $(v_{|w|}, v_{|x|})$. Da $v_{|u|}$ kleineren Zeilenindex als $v_{|\nu|}$ hat, wählt $(v_{|w|}, v_{|x|})$ anschließend $(v_{|u|}, v_{|w|})$ und D_2 wird um $(v_{|u|}, v_{|w|}, v_{|w|}, v_{|x|})$ erweitert.

2. Abbildung 3.8.2(b): Beide Kanten $(v_{|u|}, v_{|w|})$ und $(v_{|\nu|}, v_{|w|})$ können nur $(v_{|w|}, v_{|x|})$ wählen. Wie in Fall 1 gewinnt $(v_{|u|}, v_{|w|})$ die Selektion und D_2 wird um $(v_{|u|}, v_{|w|}, v_{|w|}, v_{|x|})$ erweitert.

3. Abbildung 3.8.2(c): Der einzig mögliche Pfad ist $(v_{|u|}, v_{|w|}, v_{|w|}, v_{|x|})$.

4. Abbildung 3.8.2(d): Wie in Fall 1 wird $(v_{|w|}, v_{|x|})$ von $(v_{|u|}, v_{|w|})$ bevorzugt. Letzterer hat keinen Konkurrenten und D_2 wird um $(v_{|u|}, v_{|w|}, v_{|w|}, v_{|x|})$ erweitert.

5. Abbildung 3.8.2(e): Wie in Fall 1 wird $(v_{|w|}, v_{|x|})$ von $(v_{|\nu|}, v_{|w|})$ bevorzugt. Letzterer hat keinen Konkurrenten und D_2 wird um $(v_{|\nu|}, v_{|w|}, v_{|w|}, v_{|x|})$ erweitert.

6. Abbildung 3.8.2(f): Der einzig mögliche Pfad ist $(v_{|\nu|}, v_{|w|}, v_{|w|}, v_{|x|})$.

7. Abbildung 3.8.2(g): Beide Kanten $(v_{|u|}, v_{|w|})$ und $(v_{|\nu|}, v_{|w|})$ können nur $(v_{|w|}, v_{|y|})$ wählen. Wie in Fall 1 und Fall 2 gewinnt $(v_{|u|}, v_{|w|})$ die Selektion und D_2 wird um $(v_{|u|}, v_{|w|}, v_{|w|}, v_{|y|})$ erweitert.

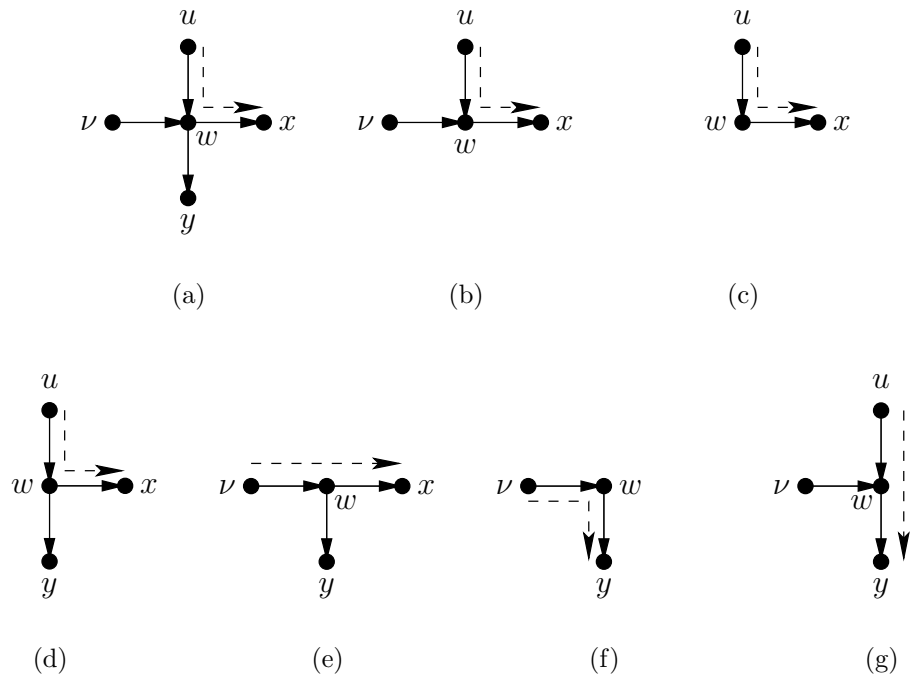


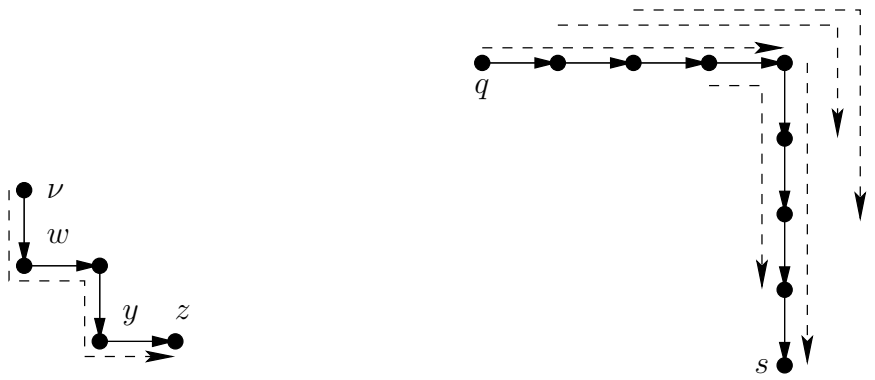
Abbildung 3.8.2: Die sieben Fälle, die bei der Kantenselektion von D_2 auftreten können. Gestrichelte Pfeile symbolisieren konstruierte Pfade.

Wir haben alle Pfade in D_2 betrachtet und teilen sie in zwei Mengen: Wir bezeichnen die Pfade der Fälle 1 bis 4 mit D_2^1 und die der Fälle 5 bis 7 mit D_2^2. Dementsprechend kann man per vollständiger Induktion einsehen, dass spätere Pfade der Länge 2^m mit $m > 1$ in Mengen $D_{2^m}^1$ und $D_{2^m}^2$ eingeteilt werden können, sodass $D_{2^m}^a$ ausschließlich Pfade enthält, die aus Teilpfaden aus $D_{2^{m-1}}^a$ zusammengesetzt wurden (für $a \in \{1, 2\}$).

Pfade $(v_{|\nu|}, v_{|w|}, v_{|y|}, v_{|z|})$ in $D_{2^m}^1$ werden für jedes Paar einer vertikalen Kante $(v_{|\nu|}, v_{|w|})$ und einer horizontalen Kante $(v_{|y|}, v_{|z|})$ erzeugt, sodass $v_{|\nu|}$ und $v_{|z|}$ auf derselben Diagonalen liegen und Distanz 2^m haben (siehe Abbildung 3.8.3(a)). Also kann $D_{2^m}^1$ folgendermaßen beschrieben werden:

$$\begin{aligned} D_{2^m}^1(\nu, w, y, z) = {} & \chi_{G_{2^\kappa+1}}(\nu, w) \wedge \chi_{G_{2^\kappa+1}}(y, z) \\ & \wedge \big[(col(\nu) = col(w)) \wedge (row(y) = row(z))\big] \\ & \wedge (row(z) - row(\nu) = 2^{m-1}) \wedge (col(z) - col(\nu) = 2^{m-1}). \end{aligned}$$

Als Nächstes betrachten wir die Funktionen $S_{2^m}^1(u, \nu, w, x, y, z)$, die die Kanten (u, ν) von Pfaden $(v_{|w|}, v_{|x|}, \ldots, v_{|y|}, v_{|z|})$ der Länge 2^m darstellen. Im folgenden Ausdruck wird geprüft, ob $(v_{|u|}, v_{|\nu|})$ eine Kante ist und ob ein entsprechender Pfad von $(v_{|w|}, v_{|x|})$



(a) Struktur der Pfade in D_4^1.

(b) Alle Pfade in D_4^2.

Abbildung 3.8.3: Die verschiedenen Pfadstrukturen in D_4^1 und D_4^2. Gestrichelte Pfeile symbolisieren konstruierte Pfade.

nach $(v_{|y|}, v_{|z|})$ in $D_{2^m}^1$ existiert. Außerdem muss $(v_{|u|}, v_{|\nu|})$ im Rechteck zwischen $v_{|w|}$ und $v_{|z|}$ liegen, um Teil dieses Pfades zu sein. Schließlich muss der Endknoten $v_{|\nu|}$ einer horizontalen Kante $(v_{|u|}, v_{|\nu|})$ auf der gleichen Diagonalen wie $v_{|w|}$ liegen, während der Startknoten $v_{|u|}$ einer vertikalen Kante $(v_{|u|}, v_{|\nu|})$ auf der gleichen Diagonalen wie $v_{|w|}$ liegen muss.

$$\begin{aligned} S_{2^m}^1(u,\nu,w,x,y,z) = \chi_{G_{2^\kappa+1}}(u,\nu) \wedge D_{2^m}^1(w,x,y,z) \\ \wedge (row(w) \leq row(u)) \wedge (col(w) \leq col(u)) \\ \wedge (row(\nu) \leq row(z)) \wedge (col(\nu) \leq col(z)) \\ \wedge \Big[\big((row(u) = row(\nu)) \wedge (row(\nu) - row(w) = col(\nu) - col(w))\big) \\ \vee \big((col(u) = col(\nu)) \wedge (row(u) - row(w) = col(u) - col(w))\big)\Big] \end{aligned}$$

Wir betrachten nun $D_{2^m}^2$ (siehe Abbildung 3.8.3(b)). Diese Funktion repräsentiert einfach alle Pfade der Länge 2^m, die auf dem oberen rechten Rand des Gitters verlaufen. Dies lässt sich folgendermaßen beschreiben:

$$\begin{aligned} D_{2^m}^2(\nu,w,y,z) = \chi_{G_{2^\kappa+1}}(\nu,w) \wedge \chi_{G_{2^\kappa+1}}(y,z) \\ \wedge \big[(row(w) = 0) \vee (col(\nu) = 2^\kappa)\big] \wedge \big[(row(z) = 0) \vee (col(y) = 2^\kappa)\big] \\ \wedge (row(z) - row(\nu) \geq 0) \wedge (col(z) - col(\nu) \geq 0) \\ \wedge (row(z) - row(\nu) + col(z) - col(\nu) = 2^m). \end{aligned}$$

Konsequenterweise betrachten wir nun die Funktion $S_{2^m}^2(u,\nu,w,x,y,z)$. Der entsprechende Ausdruck prüft, ob (u,ν) eine Kante ist und ob ein Pfad von (w,x) nach

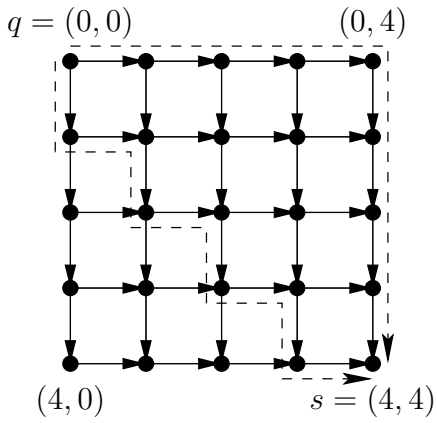


Abbildung 3.8.4: Der vom IQ-Algorithmus konstruierte maximale Fluss im 5×5-Gittergraphen. Gestrichelte Pfeile symbolisieren Flusskanten.

(y, z) der Länge 2^m konstruiert wurde. Außerdem muss (u, ν) auf dem rechten oberen Rand zwischen w und z liegen.

$$\begin{aligned} S^2_{2^m}(u,\nu,w,x,y,z) = \chi_{G_{2^{\kappa+1}}}(u,\nu) \wedge D^2_{2^m}(w,x,y,z) \\ \wedge \big((row(\nu) = 0) \vee (col(u) = 2^\kappa)\big) \\ \wedge (row(w) \leq row(u)) \wedge (col(w) \leq col(u)) \\ \wedge (row(\nu) \leq row(z)) \wedge (col(\nu) \leq col(z)) \end{aligned}$$

Schließlich stellt $S^1_{2^{\kappa+1}}$ einen flussverbessernden Pfad entsprechend Abbildung 3.8.3(a) dar, während $S^2_{2^{\kappa+1}}$ alle Kanten des oberen rechten Randes des Gitters enthält. Abbildung 3.8.4 zeigt den entsprechenden maximalen Fluss in G_5. Schließlich bemerken wir $D_{2^m} = D^1_{2^m} \vee D^2_{2^m}$ und $S_{2^m} = S^1_{2^m} \vee S^2_{2^m}$.

Wir betrachten wieder die Folge F aller booleschen Funktionen, die im Modul multipfad (welches lediglich einmal aufgerufen wird) von

- funktionalen Operationen gemäß Definition 2.3.16, Punkte 1–9,
- Argumentumordnungen gemäß Definition 2.5.5 und
- Quantifizierungsblöcken $(\exists X')$ für eine Variablenteilmenge X'

erzeugt werden. Diese Operationen werden auf die Eingabe $\chi_{G_{2^{\kappa+1}}}$, Hilfsfunktionen in $\mathbb{V}^{\mathcal{O}(1)}_{2k,n}$ mit Voll-OBDD-Breite $\mathcal{O}(1)$ (nämlich Gleichheitstest wie z. B. in Gleichung (3.6.1), siehe Korollar 3.7.5), Prioritätsfunktionen mit Voll-OBDD-Breite $\mathcal{O}(1)$ (siehe Korollar 3.7.7) sowie auf Zwischenergebnisse angewandt. Sei $H \subset F$ die Menge der booleschen Funktionen, für die wir in diesem Abschnitt bereits konstante Voll-OBDD-Breite nachgewiesen haben.

Beobachtung 3.8.8. *Jede Teilsequenz $F' = (f_1, \ldots, f_d)$ von F mit $f_1, f_d \in H$ und $f_i \notin H$ für $i \in \{2, \ldots, d-1\}$ hat Länge $\mathcal{O}(1)$.*

Nun können wir Theorem 3.8.3 beweisen:

Beweis von Theorem 3.8.3: Wir betrachten im Folgenden alle Voll-OBDD-Breiten bez. $\pi^{\mathrm{id}}_{2k,n}$, ohne es jeweils explizit zu erwähnen. Laut Lemma 3.8.2 hat $\chi_{G_{2^\kappa+1}}$ konstante Voll-OBDD-Breite. Dies gilt aufgrund von Lemma 3.7.1 und Korollar 3.7.5 ebenso für alle Funktionen, für die wir in Abschnitt 3.8.4 und Abschnitt 3.8.5 gezeigt haben, dass sie durch eine konstante Anzahl binärer Synthesen und Negationen von Funktionen aus $\mathbb{V}^{\mathcal{O}(1)}_{2k,n}$ erzeugt werden können. Laut Beobachtung 3.8.7 und Beobachtung 3.8.8 hat in den Aufrufen von berechneU und multipfad jede Folge S von OBDD-Operationen, Argumentumordnungen und Quantifizierungsblöcken, für deren Ergebnisfunktionen noch keine konstante Voll-OBDD-Breite nachgewiesen wurde, lediglich konstante Länge. Also kann Theorem 3.7.4 angewandt werden, und jede Folge S hat Laufzeit $\mathcal{O}(n^2 \log n)$ auf Platz $\mathcal{O}(n)$.

Laut Theorem 3.3.1 gibt es nur $\mathcal{O}(n^2)$ solche Teilfolgen. Jedoch haben wir dabei einen Faktor n für die Quantifizierungen über Argumente mit jeweils $\Theta(n)$ Variablen doppelt gezählt. Es gibt also tatsächlich nur $\mathcal{O}(n)$ solche Folgen S.

Der Algorithmus erzeugt zudem $\pi^{\mathrm{id}}_{2k,n}$-OBDDs für Hilfsfunktionen. Dies sind Gleichheitstests wie z. B. in Gleichung (3.4.1) sowie Prioritätsfunktionen Π_{dp}, die nach Korollar 3.7.5 und Korollar 3.7.7 jeweils in Zeit $\mathcal{O}(n)$ aufgebaut werden können.

Schließlich betrachten wir die Operationen außerhalb von berechneU und multipfad: Dies sind die Zeilen 9, 10 und 12 in Algorithmus 3.3.1. Hier tauchen jedoch nur Nullfunktionen (bisheriges $B(x, y)$ und bisheriges $\chi_F(x, y)$) auf sowie solche, für die die konstante Voll-OBDD-Breite bereits gezeigt wurde ($B^*(x, y)$, bisheriges $U(x, y)$, $\chi_G(x, y)$ und $B(x, y)$). Hinzu kommen die Tests, ob ein weiterer Sweep bzw. eine weitere Phase benötigt werden. Sie benötigen nach unseren Überlegungen in Abschnitt 3.8.3 lineare Zeit in den $\pi^{\mathrm{id}}_{2k,n}$-OBDD-Größen von $\chi_F(x, y)$, $\chi_G(x, y)$ und $U(x, y) \wedge \overline{B^*(x, y)}$, deren konstante Voll-OBDD-Breite nun bekannt ist. Es folgt Gesamtlaufzeit $\mathcal{O}(n^3 \log n)$ und Platz $\mathcal{O}(n^2)$. □

3.8.6 Experimentelle Ergebnisse

Sowohl der HS-Algorithmus als auch der IQ-Algorithmus wurden in C++ implementiert, um die praktische Relevanz der theoretischen Ergebnisse dieses Abschnitts zu überprüfen. Dabei wurden beide Algorithmen auf $G_{2^\kappa+1}$ für $\kappa := 0, \ldots, 16$ auf einem PC mit Pentium-4-Prozessor (2 GHz) und 512 MB Hauptspeicher angewandt. Die Implementation verwendet das OBDD-Paket CUDD von F. Somenzi (erhältlich unter `http://vlsi.colorado.edu/`) und macht Gebrauch von komplementierten Kanten.

Bereits ab $\kappa > 16$ reichte für den HS-Algorithmus der Speicher nicht mehr aus, während der IQ-Algorithmus auch noch für $\kappa := 17, 18, 19$ angewandt wurde und die

Speichergrenze noch nicht erreichte. Abbildung 3.8.5 zeigt die experimentellen Ergebnisse im Sinne von Zeit- und Platzbedarf. Die Laufzeit wurde in Prozessorzeitsekunden gemessen, der Platz in der maximalen Anzahl an OBDD-Knoten im SBDD, die während der Ausführung erreicht wurde. Wir sehen, dass der HS-Algorithmus ab $\kappa = 13$ im Platzbedarf vom IQ-Algorithmus geschlagen wird, während dies für die Laufzeit ab $\kappa = 16$ passiert.

So ist die asymptotische exponentielle Überlegenheit des IQ-Algorithmus auch im Experiment zu beobachten. Dies bestätigt die Hoffnung, dass die möglicherweise mehrfach exponentielle Funktion β in Theorem 3.7.4 eine starke Überschätzung des Faktors in der $\mathcal{O}$-Notation darstellt.

3.8.7 Mehrdimensionale Gittergraphen

Verschiedene Verallgemeinerungen der bisher betrachteten zweidimensionalen Gittergraphen sind denkbar. Wir möchten vier davon kurz ansprechen.

1. Gittergraphen, in denen Quelle und Senke nicht auf der gleichen Diagonalen liegen, deren horizontale und vertikale Abstände also verschieden sein dürfen. Hier besteht die Vermutung, dass ebenfalls konstante Voll-OBDD-Breite für alle auftretenden Funktionen nachgewiesen werden kann, wenn die Niveaunetzwerktiefe eine Zweierpotenz ist.

2. $N \times N$-Gittergraphen, wenn $N - 1$ keine Zweierpotenz ist. Dies zerstört zunächst einmal die einfache Struktur der konstruierten Pfade und es besteht die Vermutung, dass es so zu exponentiell größeren Laufzeiten kommen kann.

3. Leicht irreguläre Gittergraphen, in denen wenige Kanten entfernt und/oder hinzugefügt wurden. Es besteht die Vermutung, dass bereits kleine Änderungen zu einer exponentiell größeren Laufzeit führen können. Interessant wäre eine Klassifizierung der Typen von Änderungen, für die der IQ-Algorithmus effizient bleibt.

4. Mehrdimensionale $(2^\kappa + 1) \times \cdots \times (2^\kappa + 1)$-Gittergraphen. Die Analyse kann auf diesen Fall verallgemeinert werden, wenn auch die Anzahl der Dimensionen eine Zweierpotenz ist. Ansonsten wird vermutet, dass es zu exponentiell größeren Laufzeiten kommen kann.

Natürlich sind auch Kombinationen denkbar. Die Verallgemeinerungen 1–3 wurden noch nicht näher untersucht. Im Fall 1 kann jedoch vermutlich ähnlich vorgegangen werden wie in diesem Abschnitt, und so würden keine wesentlichen neuen Techniken beigesteuert werden. Dementsprechend werden wir auch den Beweis für Fall 4 hier nur kurz skizzieren, da er lediglich eine technisch aufwendigere Version des Beweises für zwei Dimensionen darstellt.

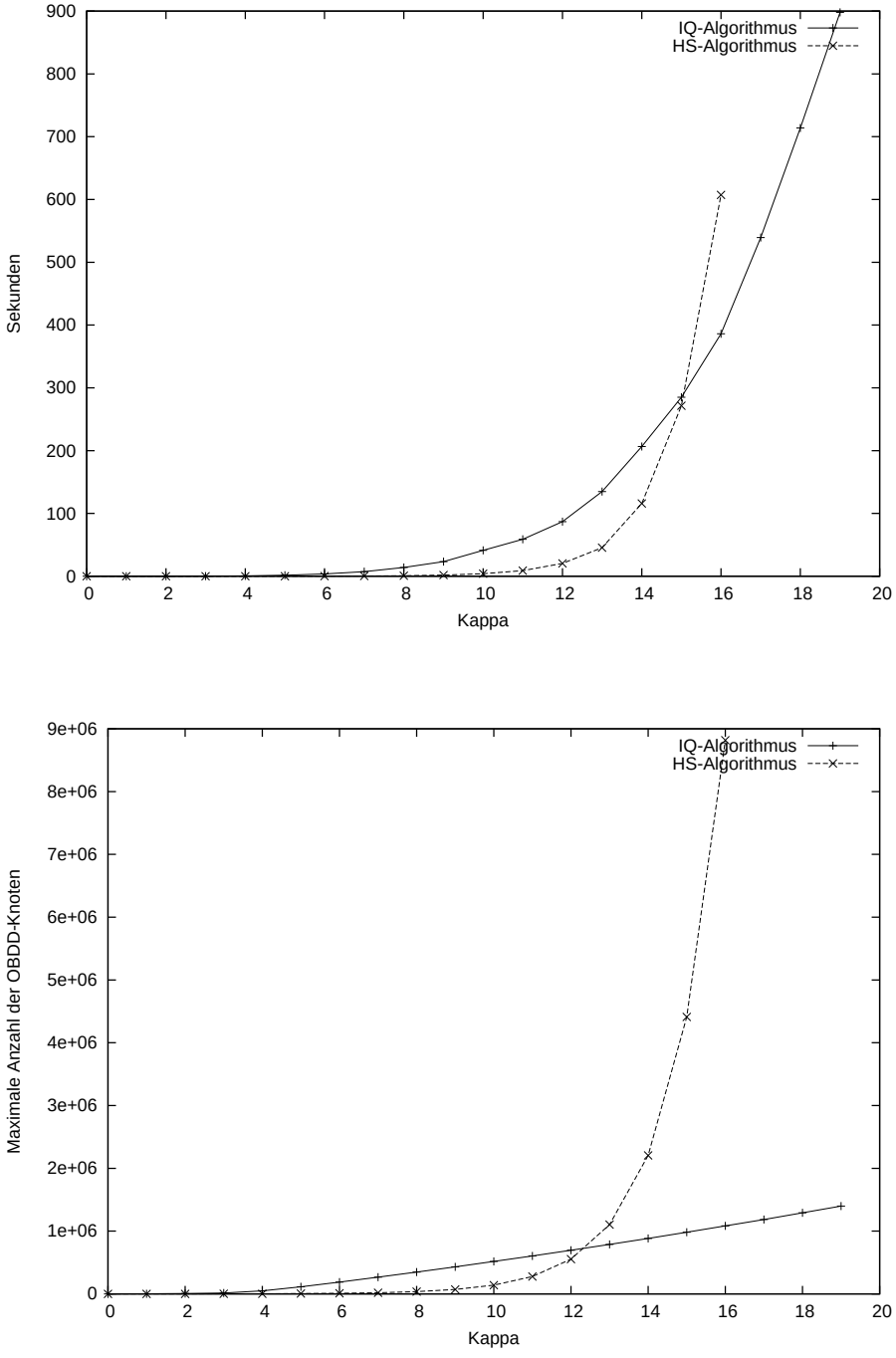


Abbildung 3.8.5: Experimentelle Ergebnisse auf $(2^\kappa + 1) \times (2^\kappa + 1)$-Gittergraphen.

Sei also nun $G_N^d := (V_N^d, E_N^d)$ mit $V_N^d := \big\{(i_1, \ldots, i_d) \mid 0 \leq i_1, \ldots, i_d < N\big\}$ und der Kantenmenge

$$E_N^d := \Big\{\big((i_1, \ldots, i_d), (j_1, \ldots, j_d)\big) \in V_N^d \times V_N^d \mid \sum_{k=1}^{d}(j_k - i_k) = 1, \forall k \in \{1, \ldots, d\} \colon i_k \leq j_k\Big\}.$$

Der Graph kann also als d-dimensionale Knotenmatrix mit Ausdehnung N betrachtet werden, dessen Kanten zwischen Knoten mit Hammingabstand 1 in Richtung des jeweils lexikografisch höherwertigen Knotens verlaufen. In unserer bisherigen Darstellung von $G_{2^\kappa+1}^2$ entsprach der Zeilenindex i_1 und der Spaltenindex i_2. Wir interessieren uns nun analog zum zweidimensionalen Fall für einen maximalen Fluss von $q := (0, \ldots, 0)$ nach $s := (N-1, \ldots, N-1)$, der offensichtlich Wert d besitzt:

Theorem 3.8.9. *Seien $\kappa, \lambda \in \mathbb{N}^+$, $n := \kappa+1$ und $d = 2^\lambda = \mathcal{O}(1)$. Der IQ-Algorithmus berechnet einen maximalen q-s-Fluss $\pi_{2d,n}^{\mathrm{id}}$-OBDD$[\chi_F]$ auf $\pi_{2d,n}^{\mathrm{id}}$-OBDD$[\chi_{G_{2^\kappa+1}^d}]$ in Zeit $\mathcal{O}(n^3 \log n)$ auf Platz $\mathcal{O}(n^2)$.*

(Wieder kann das Ergebnis leicht auf Gittergraphen verallgemeinert werden, deren Dimensionen verschiedene Ausdehnungen haben, wenn die Distanzen von q und s bez. aller Dimensionen identische Zweierpotenzen sind.)

Offenbar sind auch in G_N^d alle q-s-Pfade kürzeste flussverbessernde Pfade der Länge $d(N-1) = 2^{\kappa+\lambda}$. Das gesamte Gitter macht auch das erste Niveaunetzwerk aus. Ein Knoten $(i_1, \ldots, i_d)$ liegt in Knotenebene $i_1 + \cdots + i_d$. Wir fassen dabei d als konstant auf. Die Verallgemeinerung des Beweises von Theorem 3.8.3 für die Analyse der Niveaunetzwerkkonstruktion in berechneU ist somit trivial. Insbesondere kann Gleichung (3.8.1) leicht an den d-dimensionalen Fall angepasst werden.

Interessanter ist hingegen die Struktur der Pfade, die in den einzelnen Sweeps konstruiert werden. Aus den folgenden verbalen Beschreibungen können die Leserin und der Leser leicht die Existenz eines entsprechenden Ausdrucks konstanter Länge auf $\mathbb{V}_{dk,n}^{\mathcal{O}(1)}$-Funktionen folgern, auf deren explizite Angabe wir verzichten wollen. So soll hier wenigstens eine Intuition für die d-dimensionale Situation vermittelt werden. Analog zu unseren Überlegungen in Abschnitt 3.8.1 und Abschnitt 3.8.2 codieren wir Knoten $(i_1, \ldots, i_d)$ durch $d(\kappa+1)$ boolesche Variablen und verwenden die Variablenordnung $\pi_{dk,n}^{\mathrm{id}}$ für konstante Anzahl k von Knotenargumenten, die die jeweilige Implementation des IQ-Algorithmus insgesamt benötigt. Wir bemerken, dass der HS-Algorithmus hier Laufzeit $\Omega(dN)$ hätte.

Zunächst erinnern wir uns an die in Abschnitt 3.8.5 diskutierten Fälle 5–7 bei der Multipfadkonstruktion. Sie führten zu einem Pfad, der am rechten oberen Rand des Gitters entlang lief. Im d-dimensionalen Falle entsprechen ihnen die d „Ränder“ von $G_{2^\kappa+1}^d$, die aus den Kanten

$$\big((0, \ldots, 0, i_j, 2^\kappa, \ldots, 2^\kappa), (0, \ldots, 0, i_j+1, 2^\kappa, \ldots, 2^\kappa)\big)$$

für $j := 1, \ldots, d$ und $i_j := 0, \ldots, 2^\kappa - 1$ bestehen. Da die Pfadlänge $d(N-1)$ wieder eine Zweierpotenz ist, fügen sich all diese Kanten wieder zu einem q-s-Pfad in $D_{2^{\kappa+\lambda}}$ und $S_{2^{\kappa+\lambda}}$ zusammen. Wir nennen diesen Pfad fortan *Primärpfad*.

Nun erinnern wir uns an die in Abschnitt 3.8.5 diskutierten Fälle 1–4 bei der Multipfadkonstruktion. Auf den d-dimensionalen Fall übertragen entstehen hier analog zu Abbildung 3.8.3(a) Winkel bei der Paarung von entlang Dimension 1 verlaufenden Kanten mit entlang Dimension d verlaufenden Kanten. Es paaren sich also Kanten

$$\big((i_1, i_2, \ldots, i_{d-1}, i_d),\ (i_1 + 1, i_2, \ldots, i_{d-1}, i_d)\big)$$

mit Kanten

$$\big((i_1 + 1, i_2, \ldots, i_{d-1}, i_d),\ (i_1 + 1, i_2, \ldots, i_{d-1}, i_d + 1)\big)$$

für $i_1, i_d := 0, \ldots, 2^\kappa - 1$ und $i_2, \ldots, i_{d-1} := 0, \ldots, 2^\kappa$. Wie im zweidimensionalen Fall setzen sich diese Kantenwinkel zu Pfaden der Länge $2^{\kappa+1}$ zusammen, die jeweils von $(0, i_2, \ldots, i_{d-1}, 0)$ zu $(2^\kappa, i_2, \ldots, i_{d-1}, 2^\kappa)$ für $i_2, \ldots, i_{d-1} := 0, \ldots, 2^\kappa$ verlaufen.

Wir betrachten den Pfad mit $i_2, \ldots, i_{d-1} = 0$. Er endet in $(2^\kappa, 0, \ldots, 0, 2^\kappa)$. Dort setzt er sich wie der Primärpfad über die Ränder der verbleibenden $d-2$ Dimensionen fort und erreicht so $(2^\kappa, \ldots, 2^\kappa)$. Seine restlichen Kanten sind also

$$\big((2^\kappa, 0, \ldots, 0, i_j, 2^\kappa, \ldots, 2^\kappa),\ (2^\kappa, 0, \ldots, 0, i_j + 1, 2^\kappa, \ldots, 2^\kappa)\big)$$

für $j := 2, \ldots, d-1$ und $i_j := 0, \ldots, 2^\kappa - 1$. Wir nennen diesen Pfad *Sekundärpfad*.

Schließlich betrachten wir die $2(d-2)$ zweidimensionalen Seitenflächen der d-dimensionalen Matrix, die von den Dimensionen $j \in \{2, \ldots, d-1\}$ und d aufgespannt werden. Für ein j betrachten wir diejenige der beiden Seitenflächen, deren restliche Indizes $i_{j'}$ für $j' \notin \{j, d\}$ Null sind. Ihre Knoten werden also von keinen Kanten entlang Dimension 1 erreicht. Dort entstehen daher Zuordnungen von Kanten

$$\big((0, \ldots, 0, i_j, 0, \ldots, 0, i_d),\ (0, \ldots, 0, i_j + 1, 0, \ldots, 0, i_d)\big)$$

zu Kanten

$$\big((0, \ldots, 0, i_j + 1, 0, \ldots, 0, i_d),\ (0, \ldots, 0, i_j + 1, 0, \ldots, 0, i_d + 1)\big)$$

für $i_j, i_d := 0, \ldots, 2^\kappa - 1$. Diese Winkel setzen sich zu Pfaden der Länge $2^{\kappa+1}$ zusammen, die jeweils von $(0, \ldots, 0)$ zu $(\underbrace{0, \ldots, 0}_{j-1}, 2^\kappa, 0, \ldots, 0, 2^\kappa)$ verlaufen. Von dort setzen sie sich über die Ränder der Dimensionen $d-1$ bis $j+1$ zu $(\underbrace{0, \ldots, 0}_{j-1}, 2^\kappa, \ldots, 2^\kappa)$ fort. Die Fortsetzung über die Ränder der Dimensionen $j-1$ bis 1 ist jedoch erfolglos, da dieser Restpfad sich bereits mit dem entsprechenden Anfang des Primärpfades paart. In Sweep 1 werden also zwei q-s-Pfade konstruiert.

Nun werden Primär- und Sekundärpfad entfernt und eine weitere Multipfadkonstruktion auf dem verbleibenden Niveaunetzwerk gestartet. Wieder entstehen Winkel auf den oben genannten $d-2$ Seitenflächen der Matrix sowie entsprechende Pfade von $(0,\ldots,0)$ zu $(\underbrace{0,\ldots,0}_{j-1},2^\kappa,0,\ldots,0,2^\kappa)$, die wieder über die Dimensionsränder $d-1$ bis $j+1$ zu $(\underbrace{0,\ldots,0}_{j-1},2^\kappa,\ldots,2^\kappa)$ fortgesetzt werden. Nun paaren sich diese Pfadteile jedoch mit dem Pfad, der zu $(0,\ldots,0,1,0,\underbrace{2^\kappa,\ldots,2^\kappa}_{d-j+1})$ und über den Rand von Dimension $j-1$ nach $(0,\ldots,0,1,\underbrace{2^\kappa,\ldots,2^\kappa}_{d-j+2})$ läuft. Von dort geht es dann zu $(0,\ldots,0,1,1,\underbrace{2^\kappa,\ldots,2^\kappa}_{d-j+2})$ und über den Rand der nächsten Dimension weiter, bis schließlich $(1,2^\kappa-1,2^\kappa,\ldots,2^\kappa)$ erreicht wird (der Knoten $(1,2^\kappa,\ldots,2^\kappa)$ wäre eine Sackgasse). Von dort wird über den Rand der Dimension 1 nach $(2^\kappa,2^\kappa-1,2^\kappa,\ldots,2^\kappa)$ und schließlich nach $(2^\kappa,\ldots,2^\kappa)$ gelaufen.

Es gibt ein Problem dabei: Der Teilpfad von $(0,\ldots,0,1,0,\underbrace{2^\kappa,\ldots,2^\kappa}_{d-j+1})$ nach $(0,\ldots,0,1,\underbrace{2^\kappa,\ldots,2^\kappa}_{d-j+2})$ wird so von $d-j$ Pfaden ausgewählt. Die Prioritätsfunktion bewirkt aber, dass in diesem Sweep nur ein Pfad konstruiert wird, nämlich der, der auf der von den Dimensionen $d-1$ und d aufgespannten Seitenfläche beginnt. Die Minimalität seiner Startkante lässt ihn den Zuschlag erhalten. Dementsprechend kann man einsehen, dass in Sweep k der auf dem Dimensionenpaar $(d-k,d)$ beginnende Pfad erfolgreich konstruiert wird. Jedoch wird Dimension $j-1$ nun von $(0,\ldots,0,k-1,0,\underbrace{2^\kappa,\ldots,2^\kappa}_{d-j+1})$ aus abgelaufen, da die Ausweichknoten $(0,\ldots,0,a,0,\underbrace{2^\kappa,\ldots,2^\kappa}_{d-j+1})$ mit $a<k-1$ in vorigen Sweeps verwandt wurden.

Es werden also $d-1$ Sweeps ausgeführt. Da bereits diese Beschreibung der Grobstruktur der konstruierten Pfade recht aufwendig war, beenden wir hier die Beweisskizze.

3.8.8 Zeitexpandierte Graphen

Schließlich sprechen wir eine weitere Anwendung der in diesem Abschnitt vorgestellten und angewandten Analysetechnik im Zusammenhang mit maximalen Flüssen an. In Sawitzki (2004d) ist dokumentiert, wie implizite Flussmaximierungsalgorithmen zur Berechnung von zeitabhängigen Flüssen verwendet werden können. Es handelt sich um ein erweitertes erstmals von Ford und Fulkerson (1958) eingeführtes Flussproblem, bei der jeder Graphkante e neben der Flusskapazität $\phi(e)$ zudem eine *Reisedauer* $\psi(e)$

zugeordnet wird, die festlegt, wie lange eine Flusseinheit benötigt, um durch e zu fließen. Nun ist es auch erlaubt, dass Flusseinheiten vorübergehend in einem Knoten verweilen, bevor sie weitergeschickt werden. Schließlich soll die Anzahl Flusseinheiten maximiert werden, die innerhalb eines festen Zeithorizonts T von der Quelle zur Senke geschickt werden. Im Falle ganzzahliger Funktionen ϕ und ψ müssen lediglich diskrete Zeitpunkte betrachtet werden.

Neben einer in Ford und Fulkerson (1958) vorgestellten Reduktion auf die Berechnung kostenminimaler Flüsse haben Lösungsansätze an Bedeutung gewonnen, die mit *zeitexpandierten Graphen* G^* arbeiten. Letzterer enthält eine Kopie des ursprünglichen Eingabegraphen G für jeden diskreten Zeitpunkt $0, \ldots, T$. Jede Kante (u, v) aus G taucht in G^* zwischen allen Knotenkopien u_t und $v_{t'}$ mit $t' - t = \psi(u, v)$ auf. Ein maximaler Fluss in G^* in dem von uns bisher diskutierten Sinne entspricht dann einem maximalen zeitabhängigen Fluss in G (siehe z. B. Fleischer und Skutella (2002)). Damit hat G^* dann aber leider pseudopolynomielle Größe.

In Sawitzki (2004d) wird daher gezeigt, wie die charakteristische Funktion von G^* als boolescher Ausdruck über χ_G und multivariaten Thresholdfunktionen dargestellt werden kann. Analog zur Analyse dieses Abschnitts folgt so eine kompakte OBDD-Darstellung von χ_{G^*}, wenn die Voll-OBDD-Breite von χ_G klein ist. Für strukturierte Graphen G mit kleiner OBDD-Darstellung ist also auch G^* kompakt darstellbar. Damit ist zwar eine erste Voraussetzung erfüllt, entscheidend für die Effizienz dieses Ansatzes ist aber natürlich wieder die Gesamtlaufzeit eines impliziten Flussmaximierungsverfahrens auf χ_{G^*}, die noch nicht für konkrete Instanzen betrachtet wurde.

4 Graphen grundlegender Funktionen

Wir haben uns im letzten Kapitel mit Techniken beschäftigt, mit denen effizientes Laufzeitverhalten impliziter Algorithmen auf eingeschränkten Eingabeinstanzen nachgewiesen werden kann. Diese werden auch in späteren Kapiteln von Bedeutung sein, teilweise zur Konstruktion effizienter Reduktionen beim Härtenachweis von Problemen. In diesem Kapitel stellen wir Hilfsmittel für Beweise von konkreten unteren Laufzeit- und Platzschranken vor.

4.1 Motivation und Definition

Bereits Feigenbaum u. a. (1998) haben nachgewiesen, dass das im klassischen algorithmischen Szenario einfache Problem ***q*-*s*-Pfad-Existenz** PSPACE-hart auf OBDD-repräsentierten Eingabegraphen ist. In dieser Dissertation stellen wir jedoch noch differenziertere Fragen und möchten unter anderem konkrete untere Schranken für die OBDD-Größe von Funktionen beweisen, die im Laufe eines Algorithmus sogar auf stark eingeschränkten Instanzen auftreten können. Ein nahe liegender Ansatz ist dabei, pathologische Instanzen so zu konstruieren, dass der Algorithmus Funktionen erzeugt, die Subfunktionen mit bekanntermaßen großen OBDDs haben. Wir codieren also eine für OBDDs schwierige Funktion in die Eingabe hinein.

Diese schwierige Funktion sollte also exponentielle OBDD-Größe bez. aller Variablenordnungen haben, und zudem eine einfache Struktur mit Bezug zum betrachteten Graphproblem haben. Die Definitionen entsprechender Kandidaten wie die in Abschnitt 2.4 kennengelernten Funktionen MUL, ISA und HWB scheinen jedoch nicht immer für eine komfortable Codierung in Probleminstanzen geeignet zu sein. Wir betrachten ein Beispiel, das in Abschnitt 7.3.4 diskutiert werden wird: Die Berechnung eines maximalen q-s-Flusses in einem OBDD-repräsentierten Graph $G = (V, E, \phi)$ mit Kantenkapazitätsfunktion $\phi\colon E \to \mathbb{N}$.

Eingabe ist also Gs charakteristische Funktion $\chi_G(x, y, a)$, die Tripel (x, y, a) mit $(v_{|x|}, v_{|y|}) \in E$ und $\phi(v_{|x|}, v_{|y|}) = |a|$ repräsentiert. Analog wird der maximale Fluss $F\colon E \to \mathbb{N}$ durch $\chi_F(x, y, a)$ dargestellt. In Abschnitt 7.3.4 wird eine Eingabeinstanz G^* mit konstanter Voll-OBDD-Breite konstruiert, deren maximaler Fluss inhärent das Problem der Multiplikation enthält: Durch die Parallelschaltung von i Pfaden der Kapazität j können insgesamt $i \cdot j$ Flusseinheiten durch diesen Teil von G^* transportiert werden. Es lässt sich also eine Subfunktion f finden, die binär codierte Zahlen i, j und ℓ genau dann auf 1 abbildet, wenn $i \cdot j = \ell$ gilt. Exponentielle untere Schranken sind jedoch nur für einzelne Ergebnisbits der Multiplikation wie $MUL_{n,n-1}$ bekannt

(siehe Theorem 2.4.17) und für die gemeinsame Darstellung aller Ergebnisbitfunktionen in einem SBDD. In Abschnitt 7.3.5 werden wir zwar auch eine Anwendung so einer einzelnen Funktion $MUL_{n,2n-1}$ kennenlernen. Öfter jedoch befinden wir uns in einer ähnlichen Situation wie im obigen Beispiel der Flussmaximierung und wir können einen impliziten Graphalgorithmus zur Darstellung einer Subfunktion zwingen, die sowohl auf den Eingabebits als auch auf den Ausgabebits eines Funktionenvektors wie $(MUL_{n,0}, \ldots, MUL_{n,2n-1})$ definiert ist. Dann brauchen wir untere Schranken für die Konjunktion aller Ergebnistests der Form $MUL_{n,i}(x, y) = z_i$. Wir formalisieren dieses Prinzip:

Definition 4.1.1. *Sei $f := (f_0, \ldots, f_{m-1})$ ein Vektor von m booleschen Funktionen $f_i \in B_n$ für $n, m \in \mathbb{N}$, $0 \leq i \leq m-1$. Der* Graph *von f, genannt f-GRAPH $\in B_{n+m}$, ist definiert durch*

$$f\text{-}GRAPH(x_0, \ldots, x_{n-1}, y_0, \ldots, y_{m-1}) := \bigwedge_{i=0}^{m-1} [f_i(x) = y_i] .$$

Man könnte vermuten, dass untere Schranken für die OBDD-Größe einzelner Funktionen f_i direkt entsprechende Schranken für f-*GRAPH* implizieren – so würde sich etwa das Resultat für das mittlere Bit $MUL_{n,n-1}$ der Multiplikation direkt auf den Graphen der Multiplikation übertragen. Wir werden in diesem Kapitel aber ein Gegenbeispiel zu dieser Vermutung angeben. Dennoch gelingt es im weiteren Verlauf dieses Kapitels, die bekannten exponentiellen Schranken für die OBDD-Größe von MUL, ISA und HWB auf ihre entsprechenden Graphen zu übertragen. MUL besitzt dabei schon den Funktionenvektor $(MUL_{n,0}, \ldots, MUL_{n,2n-1})$, für ISA und HWB müssen wir solche erst durch Verallgemeinerung der Grundfunktionen definieren.

4.2 Das Gegenbeispiel Full Storage Access (FSA)

Die Definition der Gegenbeispielfunktion wird auf Verallgemeinerungen der Funktionen DSA und ISA basieren:

Definition 4.2.1 (Shifted Direct Storage Access (SDSA)). *Seien $n = 2^m$, $m \in \mathbb{N}^+$ und $i \in \mathbb{N}$. Die* Shifted-Direct-Storage-Access-Funktion *$SDSA_{n,i} \in B_{n+m}$ ist auf den $n + m$ Variablen $x \in \mathbb{B}^n$ und $y \in \mathbb{B}^m$ definiert und berechnet $x_{\alpha(y)}$ für $\alpha(y) := (|y| + i) \bmod n$.*

Es ist also $SDSA_{n,0} = DSA_n$.

Definition 4.2.2 (Shifted Indirect Storage Access (SISA)). *Seien $n = 2^m$, $m \in \mathbb{N}^+$ und $i \in \mathbb{N}$. Die* Shifted-Indirect-Storage-Access-Funktion *$SISA_{n,i} \in B_{n+m}$ ist auf den $n + m$ Variablen $x \in \mathbb{B}^n$ und $y \in \mathbb{B}^m$ definiert und berechnet $SDSA_{n,i}(x, SDSA_{n,0}(x, y), \ldots, SDSA_{n,m-1}(x, y))$.*

Wir geben also das Bit $x_{(\alpha(x,y)+i) \bmod n}$ an der indirekten Adresse $\alpha(x,y) := |x_{|y| \bmod n} \dots x_{(|y|+m-1) \bmod n}|$ aus, und es ist $SISA_{n,0} = ISA_n$.

Wir kombinieren SDSA und SISA nun zu einer Gegenbeispielfunktion:

Definition 4.2.3 (Full Storage Access (FSA)). *Seien $n = 2^m$, $m \in \mathbb{N}^+$ und $i \in \mathbb{N}$. Der* Full-Storage-Access-Funktionenvektor *$FSA_n = (FSA_{n,0}, \dots, FSA_{n,2m-1})$ ist definiert durch*

$$FSA_{n,i}(x,y) := \begin{cases} SDSA_{n,i}(x,y) & , 0 \le i \le m-1 \\ SISA_{n,i-m}(x,y) & , m \le i \le 2m-1 \end{cases}$$

für $n+m$ Variablen $x \in \mathbb{B}^n$ und $y \in \mathbb{B}^m$.

Beobachtung 4.2.4. *Wenn also alle Funktionen des Vektors FSA_n gemeinsam in einem SBDD dargestellt werden, wird dessen Größe offenbar von der Funktion $FSA_{n,m} = SISA_{n,0} = ISA_n$ dominiert. ISA_n hat gemäß Theorem 2.4.13 für jede Variablenordnung exponentielle OBDD-Größe.*

Theorem 4.2.5. *Seien $n = 2^m$, $m \in \mathbb{N}^+$, $x \in \mathbb{B}^n$, $y \in \mathbb{B}^m$ und $z \in \mathbb{B}^{2m}$. Für*

$$FSA\text{-}GRAPH(x,y,z) = \bigwedge_{i=0}^{2m} [FSA_{n,i}(x,y) = z_i]$$

und die Variablenordnung $\pi = (y_0, \dots, y_{m-1}, z_0, \dots, z_{2m-1}, x_0, \dots, x_{n-1})$ ist die π-OBDD-Größe von FSA_n-GRAPH $\mathcal{O}(n^3 \cdot \log n)$.

Beweis: Wir geben ein entsprechendes π-OBDD P an. P besteht zunächst aus dem vollständigen binären Baum auf den Argumenten y und z der Größe $\mathcal{O}(n^3)$ Knoten. An jedem seiner n^3 Blätter sind alle y- und z-Variablen also schon fixiert. Es verbleibt zu prüfen, ob $z_i = SDSA_{n,i}(x,y)$ für alle $i := 0, \dots, m-1$ und ob $z_j = SISA_{n,j-m}(x,y)$ für alle $j := m, \dots, 2m-1$. Ein Blatt b_{y^*,z^*}, das der Belegung $y := y^*$ und $z := z^*$ entspricht, repräsentiert also eine Subfunktion, die für genau eine Belegung einer höchstens $2m$-elementigen Teilmenge der x-Variablen wahr ist. Somit kann sie durch eine Kette von höchstens $2 \log n$ x-Variablen realisiert werden, die an b_{y^*,z^*} gehängt wird. Es ergibt sich die Gesamtgröße $\mathcal{O}(n^3 \cdot \log n)$. □

Wir haben somit bewiesen, dass die Konjunktion aller Funktionen eines Vektors eine wesentlich kleinere OBDD-Größe haben kann als das SBDD aller Einzelfunktionen. Wir beweisen nun, dass dies jedoch nicht der Fall für MUL, ISA und HWB ist.

4.3 Multiplikation (MUL)

Die Multiplikation natürlicher Zahlen ist eine der wichtigsten und als schwierigsten bekannten Funktionen im Bereich der Schaltkreisverifikation. Alle BDD-Varianten, die

effiziente Verifikationsalgorithmen ermöglichen, haben exponentielle Größe für diese Funktion (siehe z. B. Abschnitte 12.1 und 12.2 in Wegener (2000)). Lediglich für Multiplikationsschaltkreise spezieller Struktur sind beweisbar effiziente Verfahren bekannt (siehe Burch (1991) und Keim u. a. (2003)).

Andererseits ist die Multiplikation einfach strukturiert und hat intuitiven Bezug zu einigen wichtigen Graphproblemen, was wir uns später zu Nutzen machen werden. So interessieren wir uns also für die OBDD-Größe der Funktion MUL_n-$GRAPH$ für

$$MUL_n = (MUL_{n,0}, \ldots, MUL_{n,2n-1}) .$$

Dieses Problem ist bisher offen gewesen. Aus einer polynomiellen oberen Schranke würde jedoch leicht die Existenz eines polynomiellen Faktorisierungsalgorithmus folgen (siehe Theorem 1.4.3 in Wegener (2000)).

Jukna (1995) stellt eine exponentielle untere Schranke für die Größe von nichtdeterministischen Read-k-times-BPs für einen auf $\sqrt{n}$ Ergebnisbits eingeschränkten Graphen der Multiplikationsfunktion MUL_n vor. Trotz des Titels der Arbeit handelt es sich nicht um den Graphen der Multiplikation, so wie wir ihn definiert haben. Insbesondere impliziert dieses Resultat nicht die folgenden Hauptresultate dieses Abschnitts:

Theorem 4.3.1. *Jedes π-stereotype BP für MUL_n-$GRAPH$, dessen Variablenfolge π jede Variable höchstens k-mal enthält, hat mindestens Größe $2^{n/\left((4k-1)2^{8k}\right)-1}$.*

Ein OBDD ist ein stereotypes BP mit $k = 1$, und so folgt direkt:

Korollar 4.3.2. *Ein OBDD für MUL_n-$GRAPH$ hat mindestens Größe $2^{n/768-1}$.*

Die Read-once-Projektion von MUL auf SQU in Theorem 4.6.2 in Wegener (2000) ist momentan der einzige Weg, um exponentielle OBDD-Größe für SQU zu beweisen. Sie kann ebenfalls dazu verwandt werden, um das Ergebnis von Theorem 4.3.1 auf einen eingeschränkten Graphen des Quadrierens zu übertragen.

Korollar 4.3.3. *Seien $m \in \mathbb{N}$, $n := 3m + 2$ und SQU_n^*-$GRAPH \in B_{8m+4}$ definiert durch SQU_n^*-$GRAPH(x, y) := \bigwedge_{i=2m+3}^{4m+2} \left[SQU_{n,i}(x) = y_i\right]$ für $x \in \mathbb{B}^n$ und $y \in \mathbb{B}^{2m}$. Ein OBDD für diese Funktion hat mindestens Größe $2^{m/768-1} \geq 2^{n/2304-2}$.*

Um schließlich Theorem 4.3.1 zu beweisen, gehen wir ähnlich wie Gergov (1994) vor. Dabei werden wir Grundbegriffe auf dem Gebiet der Kommunikationskomplexität verwenden, die z. B. in Hromkovič (1997) nachgeschlagen werden können. Wir führen dieses Konzept hier nur sehr knapp ein.

Sei $f\colon \mathbb{B}^n \times \mathbb{B}^m \to \mathbb{B}$.

Definition 4.3.4 (Deterministische Kommunikationskomplexität). *Zwei Personen Alice und Bob wollen f berechnen. Alice kennt nur den ersten Teil $a \in \mathbb{B}^n$ der*

Eingabe, Bob nur den zweiten Teil $b \in \mathbb{B}^m$. Sie einigen sich auf ein Kommunikationsprotokoll, bevor die Berechnung von $f(a,b)$ beginnt. Die deterministische Kommunikationskomplexität *von f ist dann die Worst-case-Anzahl von Bits, die Alice und Bob über so ein Protokoll austauschen müssen, bevor mindestens einem von beiden $f(a,b)$ bekannt ist und beide wissen, dass das Protokoll beendet ist.*

Theorem 4.3.5 (Siehe z. B. Hromkovič (1997)). *Sei M die $2^n \times 2^m$-Kommunikationsmatrix von f definiert durch $M(|a|,|b|) := f(a,b)$. Der Wert $\log(\text{rank}(M))$ ist eine untere Schranke für die deterministische Kommunikationskomplexität von f.*

Nun kommen wir zum Beweis von Theorem 4.3.1. Seien $X := \{x_0,\ldots,x_{n-1}\}$ und $Y := \{y_0,\ldots,y_{n-1}\}$ die Mengen der Faktorvariablen von MUL_n-$GRAPH$ und $Z := \{z_0,\ldots,z_{2n-1}\}$ die entsprechenden Produktvariablen. Es ist also $MUL_n\text{-}GRAPH(x,y,z) = 1 \Leftrightarrow |x|\cdot|y| = |z|$ für $x = x_{n-1}\ldots x_0$, $y = y_{n-1}\ldots y_0$ und $z = z_{2n-1}\ldots z_0$. Sei P ein π-stereotypes BP für MUL_n-$GRAPH$, dessen Variablensequenz π jede Variable höchstens k-mal enthält.

Wir zeigen nun zunächst eine untere Schranke für eine eingeschränkte Version von MUL_n-$GRAPH$. Wir betrachten eine beliebige Sequenz ρ, die jede Variable aus X, Y und Z genau $2k =: \ell$-mal enthält und die π als Teilsequenz enthält.

Definition 4.3.6. *Für zwei disjunkte Teilmengen $S,T \subseteq X$ heißt ein Intervall $(\rho_i,\ldots,\rho_j)$ in ρ* Link*, wenn $\rho_{i+1},\ldots,\rho_{j-1} \notin S \cup T$ und entweder $\rho_i \in T \wedge \rho_j \in S$ oder $\rho_i \in S \wedge \rho_j \in T$.*

Theorem 4.3.7 (Siehe Alon und Maass (1988)). *Sei $M = (m_1,\ldots,m_{n\ell})$ eine Sequenz, die jedes Element $m_i \in X$ genau ℓ-mal enthält. Wir betrachten die Partition von X in zwei disjunkte nicht leere Mengen X_1 und X_2. Dann gibt es zwei Teilmengen $S \subseteq X_1$ und $T \subseteq X_2$ mit $|S| \geq |X_1|/2^{2\ell-1}$ und $|T| \geq |X_2|/2^{2\ell-1}$, sodass die Anzahl der Links zwischen S und T nicht größer als $2\cdot\ell-1$ ist.*

Seien $X_1 := \{x_0,\ldots,x_{\lfloor n/2\rfloor}\}$ und $X_2 := \{x_{\lfloor n/2\rfloor+1},\ldots,x_{n-1}\}$. Gemäß Theorem 4.3.7 gibt es Teilmengen $S \subseteq X_1$ und $T \subseteq X_2$, sodass $|S|,|T| \geq (n/2-1)/2^{2\ell-1} > n/2^{2\ell}-1$ gilt und es nicht mehr als $2\cdot\ell-1$ Links zwischen S und T in ρ gibt.

Da in $D := \{(x_i,x_j) \mid x_i \in S,\, x_j \in T\}$ mindestens $\left(n/2^{2\ell}-1\right)^2$ Paare enthalten sind, gibt es eine Indexmenge $I \subseteq \{0,\ldots,n-1\}$ und einen Distanzparameter $d \in \{1,\ldots,n-1\}$ mit $\max I < \min I + d$, sodass $D' := \{(x_i,x_{i+d}) \mid i \in I\} \subseteq D$ gilt und D' mindestens $\left(n/2^{2\ell}-1\right)^2/n \geq n/2^{4\ell}-1$ Paare enthält.

Sei MUL_n^*-$GRAPH := \bigwedge_{i=\min I+d}^{\max I+d}[MUL_{n,i}(x,y) = z_i]$. Wir betrachten eine Subfunktion f_n von MUL_n^*-$GRAPH$, die durch folgende Variablenersetzungen definiert ist:

$$x_i := \begin{cases} 1 & \text{für } \begin{array}{l}(i = \min I) \vee (i = \min I + d)\\ \vee\,[(\min I \leq i \leq \max I) \wedge (i \notin I)]\end{array}\\[2ex] 0 & \text{für } \begin{array}{l}(i = \max I) \vee (i = \max I + d)\\ \vee\,[((i < \min I) \vee (\max I < i)) \wedge (i - d \notin I)]\end{array}\end{cases} \tag{4.3.1}$$

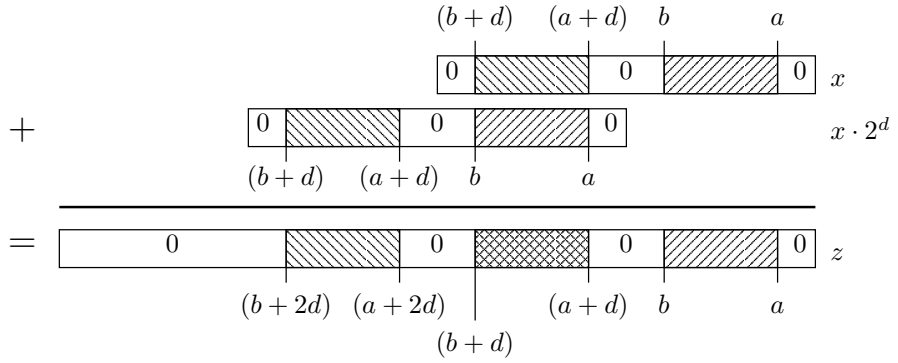


Abbildung 4.3.1: Veranschaulichung des Effekts der Variablenersetzungen in Gleichung (4.3.1), Gleichung (4.3.2) und Gleichung (4.3.3) für $a := \min I$ und $b := \max I$. *MUL_n^*-GRAPH* verifiziert lediglich die Ergebnisbits $z_{a+d}, \ldots, z_{b+d}$, die der Summe von $x_{b+d} \ldots x_{a+d}$ und $x_b \ldots x_a$ entsprechen.

$$y_j := \begin{cases} 1 & \text{für } (j = 0) \vee (j = d) \\ 0 & \text{sonst} \end{cases} \tag{4.3.2}$$

$$z_r := \begin{cases} 1 & \text{für } r = \max I + d \\ 0 & \text{sonst} \end{cases} \tag{4.3.3}$$

Lemma 4.3.8. *Ein ρ-stereotypes BP Q für f_n hat mindestens Größe $2^{n/\left((2\ell-1)\left(2^{4\ell}\right)\right)-1}$.*

Beweis: In Gleichung (4.3.1) ersetzen wir alle Variablen zwischen $\min I$ und $\max I$ durch 1, die nicht in D'-Paaren enthalten sind, sowie $x_{\min I}$ und $x_{\min I+d}$. Alle nicht in D'-Paaren enthaltenen x-Variablen außerhalb des Intervalls $\{\min I, \ldots, \max I\}$ sowie $x_{\max I}$ und $x_{\max I+d}$ werden durch 0 ersetzt. In Gleichung (4.3.2) werden die y-Variablen so belegt, dass die Multiplikation der Summation von $|x|$ und $|x| \cdot 2^d$ entspricht. Das Ergebnis wird mit $|z| = 2^{\max I+d}$ verglichen (siehe Gleichung (4.3.3)). Abbildung 4.3.1 veranschaulicht dieses Ersetzungsschema.

Die Funktion f_n hängt nun nur noch von den x-Variablen ab, die in Paaren in $D' \setminus \{(x_{\min I}, x_{\min I+d}), (x_{\max I}, x_{\max I+d})\}$ auftauchen; wir sehen, dass genau dann eine 1 berechnet wird, wenn $x_i \neq x_{i+d}$ für alle $i \in I \setminus \{\min I, \max I\} =: I'$ gilt.

Behauptung. *Die deterministische Kommunikationskomplexität von f_n bez. der Variablenpartition $\{x_i \mid i \in I'\} \cup \{x_{i+d} \mid i \in I'\}$ ist mindestens $|I'| = |I| - 2$.*

Sei M die $2^{|I'|} \times 2^{|I'|}$-Kommunikationsmatrix von f_n. Es ist genau dann $M(i, j) = 1$, wenn i und j zueinander inverse Binärdarstellungen haben. M ist also eine Permutationsmatrix mit Rang $2^{|I'|}$, und nach Theorem 4.3.5 ist $|I'|$ eine untere Schranke für die deterministische Kommunikationskomplexität von f_n.

Wegen der speziell gewählten Struktur von I' kann jedes ρ-stereotype BP Q für f_n in höchstens $2 \cdot \ell$ Teile $S_1, T_1, \ldots, S_\ell, T_\ell$ zerlegt werden, sodass S_i (bzw. T_i) ausschließlich Variablen aus S (bzw. T) enthält. Somit kann aus Q ein Kommunikationsprotokoll gewonnen werden, das nicht mehr als $(2 \cdot \ell - 1) \cdot \log \text{breite}(B)$ Bits für eine ρ-Einbettung B von Q verschickt. (Die triviale Konstruktion kann z. B. in Abschnitt 7.5 in Wegener (2000) nachgelesen werden.) Wegen der unteren Schranke von $|I'| = |I| - 2 \geq n/2^{4\ell} - 3$ für die Kommunikationskomplexität von f_n muss B mindestens Breite $2^{n/\left((2\ell-1)\left(2^{4\ell}\right)\right)-3/(2\ell-1)} \geq 2^{n/\left((2\ell-1)\left(2^{4\ell}\right)\right)-1}$ haben, was offensichtlich auch eine untere Schranke für die Größe von Q ist. □

Nun sind wir im Stande, die untere Schranke für die Größe von P zu zeigen.

Beweis von Theorem 4.3.1: Die untere Schranke in Lemma 4.3.8 gilt ebenso für P, da wir ein stereotypes BP P' für f_n aus P konstruieren können, ohne es zu vergrößern: Zunächst wenden wir die Variablenersetzungen in Gleichung (4.3.1), Gleichung (4.3.2) und Gleichung (4.3.3) an. Dann müssen wir die Abhängigkeit von den z-Variablen $z_0, \ldots, z_{\min I+d-1}, z_{\max I+d+1}, \ldots, z_{2n-1}$ beseitigen. In erfüllenden Belegungen für f_n gilt $z_i = x_i$ für $i \in I$ sowie $z_j = x_{j-d}$ für $j - 2d \in I$. Um diese Eigenschaften zu erzwingen, ersetzen wir Variablenmarkierungen z_i für $i \in I$ durch x_i sowie Markierungen z_j für $j - 2d \in I$ durch x_{j-d}. Die verbleibenden z-Variablen ersetzen wir durch 0. Das resultierende BP P' repräsentiert offenbar f_n.

Dabei haben wir nicht mehr als k Knoten für jede z-Variable hinzugefügt, und P' ist π'-stereotyp für eine Variablensequenz π', die jede Variable $2k = \ell$-mal enthält und π als Teilsequenz hat. Lemma 4.3.8 impliziert somit die untere Schranke von $2^{n/\left((2\ell-1)\left(2^{4\ell}\right)\right)-1} = 2^{n/\left((4k-1)\left(2^{8k}\right)\right)-1}$ für die Größe von sowohl P' als auch P. □

4.4 Indirect Storage Access (ISA)

In Abschnitt 4.2 haben wir die ISA-Verallgemeinerung SISA definiert. Breitbart u. a. (1995) stellen eine exponentielle untere Schranke von $2^{\lfloor n/\log n \rfloor - 1}$ für die OBDD-Größe von ISA_n vor. Sei $SISA_{n,w}\text{-}GRAPH \in B_{n+m+w}$ der Graph des Funktionenvektors $(SISA_{n,0}, \ldots, SISA_{n,w-1})$ für $n = 2^m$ und $1 \leq w \leq n$. Auch für $SISA_{n,w}\text{-}GRAPH$ kann eine superpolynomielle untere Schranke für die OBDD-Größe gezeigt werden, wenn $w = o\big(n/\log^2 n\big)$ gilt. Sei $(a)_2$ die Binärdarstellung einer Zahl $a \in \mathbb{N}$ kontextabhängiger Länge.

Theorem 4.4.1. *Ein OBDD für $SISA_{n,w}$-GRAPH hat mindestens Größe $2^{\lfloor n/(w \cdot \log n) \rfloor - 3}$.*

Beweis: Sei also $SISA_{n,w}\text{-}GRAPH(x, y, z) = 1 \Leftrightarrow \bigwedge_{i=0}^{w-1} [SISA_{n,i}(x, y) = z_i]$ für die Variablenmengen $X := \{x_0, \ldots, x_{n-1}\}$, $Y := \{y_0, \ldots, y_{m-1}\}$ und $Z := \{z_0, \ldots, z_{w-1}\}$.

Wir definieren eine Subfunktion $SISA_{n,w}$-$GRAPH^*$ von $SISA_{n,w}$-$GRAPH$, die mindestens π-OBDD-Größe $2^{n/(w\cdot\log n)-4}$ für jede Variablenordnung π hat. Wir unterteilen X in $\lfloor n/\log n\rfloor$ Blöcke der Länge $m = \log n$ plus einen zusätzlichen Block der Länge $\ell < m$. $X' \subset X$ bezeichne die Menge der ersten $\lfloor n/\log n\rfloor - 1$ x-Variablen, die von π gelesen werden. Es gibt also mindestens einen Block $B = (x_b, \ldots, x_{b+m-1})$, dessen Variablen vollständig außerhalb von X' liegen.

Wir ersetzen nun y durch $(b)_2$ und z durch $(0)_2$. Außerdem ersetzen wir die Variablen $x_{(b-w+1) \bmod n}, \ldots, x_{(b-1) \bmod n}$ und $x_{(b+m) \bmod n}, \ldots, x_{(b+m+w-2) \bmod n}$ durch 0. Schließlich führen wir eine zyklische Traversierung modulo n der x-Variablen durch, die noch nicht fixiert worden sind. Wenn wir dabei eine Variable aus X' passieren, ersetzen wir die folgenden $w-1$ x-Variablen durch 0. $X'' \subseteq X'$ bezeichne die Teilmenge der X'-Variablen, die immer noch nicht fixiert worden sind. Wir wiederholen die Traversierung, bis alle Paare zweier aufeinanderfolgender Variablen aus X'' durch mindestens $w-1$ Variablen aus $X \setminus X''$ getrennt sind. Wir sehen leicht, dass X'' mindestens Größe

$$\left\lfloor \frac{\left\lfloor \frac{n}{\log n} \right\rfloor - 1 - 2(w-1)}{w} \right\rfloor - 1 \geq \left\lfloor \frac{n}{w \cdot \log n} \right\rfloor - 3$$

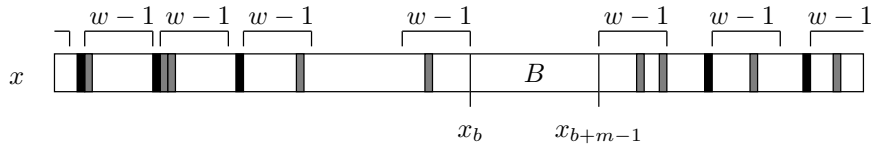
hat. Die verbliebenen Variablen aus $X \setminus (X'' \cup B)$ werden ebenso durch 0 ersetzt. Wir haben die Subfunktion $SISA_{n,w}$-$GRAPH^*$ erhalten, die lediglich von X'' und B abhängt.

Wir betrachten zwei verschiedene Belegungen a und b der Variablen in X''. Wir zeigen, dass die entsprechenden Subfunktionen $f_a := SISA_{n,w}\text{-}GRAPH^* \mid a$ bzw. $f_b := SISA_{n,w}\text{-}GRAPH^* \mid b$ verschieden sind. Nehmen wir an, dass a der Variablen $x_i \in X''$ eine 0 zuweist, während b ihr eine 1 zuweist. Die Funktionen f_a und f_b hängen lediglich von $x_b, \ldots, x_{b+m-1}$ ab.

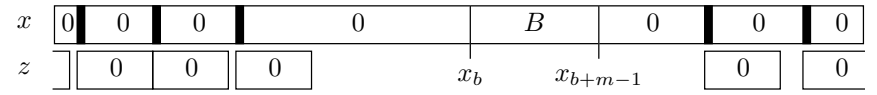
Behauptung. $f_a((i)_2) = 1 \neq 0 = f_b((i)_2)$.

Es ist also $B = (i)$, und die Variablen $x_i, \ldots, x_{i+m-1}$ werden mit 0^w verglichen. Jede Sequenz von $w-1$ Variablen, die einer Variablen aus X'' folgt, ist durch 0 ersetzt worden. Außerdem haben wir genug Platz um B gelassen. Also entscheidet x_i, ob die Subfunktion eine 1 berechnet. Die Belegung $B = (i)$ ist somit ein Zeuge für $f_a \neq f_b$. (Siehe Abbildung 4.4.1.)

Alle Variablen aus X'' werden gemäß π vor denen in Block B gelesen. Aus Theorem 2.3.6 folgt somit, dass die Anzahl $|X''| \geq 2^{\lfloor n/(w\cdot\log n)\rfloor - 3}$ der verschiedenen Subfunktionen $\left\{ f_c \mid c \in \mathbb{B}^{|X''|} \right\}$ eine untere Schranke für die π-OBDD-Größe von $SISA_{n,w}$-$GRAPH^*$ ist. □



(a) Die Eingabevariablen von $SISA_{n,w}$-$GRAPH$. Schwarze und graue Zellen symbolisieren Variablen in X''. Der Besuch einer schwarzen Zelle während der Traversierung lässt graue Zellen durch 0 ersetzen. Für jede schwarze Zelle verlieren wir höchstens $w-1$ graue. Außerdem säubern wir $w-1$ Zellen vor und hinter Block B.



(b) Die Eingabevariablen von $SISA_{n,w}$-$GRAPH^*$. Die Funktion hängt ausschließlich von schwarzen Zellen und von Block B ab. Durch Wahl einer entsprechenden Belegung von B kann das Ergebnis $z = 0^w$ an jeder schwarzen Zelle ausgerichtet werden und so als Zeuge für die Ungleichheit aller Subfunktionen $\left\{f_c \mid c \in \mathbb{B}^{|X''|}\right\}$ dienen.

Abbildung 4.4.1: Veranschaulichung von $SISA_{n,w}$-$GRAPH^*$ und den Zeugen für die Verschiedenheit aller Subfunktionen $\left\{f_c \mid c \in \mathbb{B}^{|X''|}\right\}$.

4.5 Hidden Weighted Bit (HWB)

Wir betrachten die Verallgemeinerung einer weiteren bekannten Speicherzugriffsfunktion.

Definition 4.5.1 (Shifted Hidden Weighted Bit (SHWB)). *Seien $n \in \mathbb{N}^+$ und $i \in \mathbb{N}$. Die* Shifted-Hidden-Weighted-Bit-Funktion *$SHWB_{n,i} \in B_n$ ist auf den Variablen $x \in \mathbb{B}^n$ definiert und berechnet $x_{\alpha(x)}$ für $\alpha(x) := \left(\sum_{j=1}^{n} x_j + i\right) \bmod n$ und $x_0 := 0$.*

Die Adresse $\alpha(x)$ des Ausgabebits $x_{\alpha(x)}$ ergibt sich also aus der Anzahl Einsen in x plus i, und es ist $SHWB_{n,0} = HWB_n$.

In Abschnitt 2.4 haben wir bereits eine untere Schranke von $\Omega\left(2^{n/5}\right)$ für die OBDD-Größe der HWB-Funktion kennengelernt. Sei nun $SHWB_{n,w}$-$GRAPH \in B_{n+w}$ der Graph des Vektors $(SHWB_{n,0}, \ldots, SHWB_{n,w-1})$ für $n \in \mathbb{N}^+$ und $1 \leq w \leq n$.

Theorem 4.5.2. *Ein OBDD für $SHWB_{n,w}$-$GRAPH$ mit $w \leq n - 10$ hat mindestens Größe $2^{\lfloor (n-10)/(11w) \rfloor - 1}$.*

Beweis: Sei also $SHWB_{n,w}\text{-}GRAPH(x,y) = \bigwedge_{i=0}^{w-1}[SHWB_{n,i}(x) = y_i]$ für $x \in \mathbb{B}^n$ und $y \in \mathbb{B}^w$ und sei π eine Variablenordnung der x- und y-Variablen. Wir definieren Schritt für Schritt eine Subfunktion $SHWB_{n,w}\text{-}GRAPH^*$ von $SHWB_{n,w}\text{-}GRAPH$, über die wir die untere Schranke erzielen werden. Zunächst ersetzen wir alle y-Variablen durch 0.

Die Anzahl $\alpha(x)$ der Einsen in x liegt genau dann im Fenster $\{s,\ldots,s+n-k\} =: W(k,s)$, wenn die k ersten x-Variablen bez. π mit s Einsen belegt sind. Sei $X(k,s)$ die Menge der Variablen x_i, deren Index i in $W(k,s)$ liegt und die unter den k ersten x-Variablen bez. π liegen.

Wir nehmen nun vorübergehend an, dass $n = 11m$ für $m \in \mathbb{N}^+$ gilt, und wählen $k = 7m$. Für $s = m$ gilt $W(k,s) = \{m,\ldots,5m\}$, für $s = 5m$ gilt $W(k,s) = \{5m,\ldots,9m\}$. In einem dieser beiden Fälle hat $X(k,s)$ mindestens $2m$ Elemente.

Unser Ziel ist nun, genügend viele Variablen mit Index in $W(k,s)$ durch Ersetzung durch 0 zu „räumen“, sodass der Vergleich $(x_i,\ldots,x_{i+w-1}) = 0^w$ für eine hinreichende Anzahl von Positionen i ausschließlich von der ersten Variable $x_i \in X(k,s)$ abhängt. Für jede ersetzte Variable schrumpft die Fenstergröße $|W(k,s)|$ offenbar um höchstens 1. Nach der Ersetzung von höchstens m Variablen durch 0 kann sich $\alpha(x)$ also noch im Fenster $\{m,\ldots,4m\} =: W'(k,s)$ für $s = m$ bzw. $\{5m,\ldots,8m\} =: W'(k,s)$ für $s = 5m$ befinden. Wegen $|X(k,s)| \geq 2m$ gibt es immer noch mindestens m Variablen aus $X(k,s)$ mit Index in $W'(k,s)$. Wir stellen ein ähnliches Ersetzungsschema vor wie im Beweis von Theorem 4.4.1:

Zunächst ersetzen wir $x_{s+n-k-m-w+2},\ldots,x_{s+n-k-m}$ durch 0. Dann traversieren wir $x_s,\ldots,x_{s+n-k-m-w+1}$; wird eine Variable aus $X(k,s)$ besucht, so ersetzen wir die folgenden $w-1$ Variablen durch 0. Wir wiederholen dies, bis genau $\lfloor (m-(w-1))/w \rfloor =: c$ Variablen aus $X(k,s)$ auf diese Art besucht wurden, ohne ersetzt worden zu sein. Wir bezeichnen ihre Menge mit $X'(k,s)$. Nicht mehr als $(w-1)+c\cdot(w-1) \leq m$ Variablen wurden durch 0 ersetzt. Dies beendet die Definition von $SHWB_{n,w}\text{-}GRAPH^*$.

$X''(k,s)$ bezeichne die Menge der Variablen, von denen $SHWB_{n,w}\text{-}GRAPH^*$ noch abhängt, und die unter den k ersten Positionen bez. π sind.

Behauptung. *Wir betrachten zwei Belegungen a und b der Variablen in $X''(k,s)$, sodass den k ersten x-Variablen bez. π eine Anzahl von s Einsen zugewiesen werden. Wenn a und b sich in der Belegung einer Variablen $x_i \in X'(k,s) \subseteq X''(k,s)$ unterscheiden, sind ihre entsprechenden Subfunktionen $f_a := SHWB_{n,w}\text{-}GRAPH^* \mid X''(k,s) = a$ und $f_b := SHWB_{n,w}\text{-}GRAPH^* \mid X''(k,s) = b$ verschieden.*

Entsprechend der Definition von $SHWB_{n,w}\text{-}GRAPH^*$ ist $i \in W'(k,s)$. Also existiert eine ergänzende Belegung γ für a und b, sodass x genau i Einsen enthält. Die Funktion vergleicht also das Wort $x_i \ldots x_{i+w-1}$ mit 0^w. Da a und b sich in x_i unterscheiden und $x_{i+1},\ldots,x_{i+w-1}$ durch 0 ersetzt wurden, gilt $f_a(\gamma) \neq f_b(\gamma)$.

Wir definieren nun $\binom{n}{k} := 0$ für $k < 0$ und $k > n$. Betrachte den Fall $s = m$. Wegen $|X''(k,s)| - |X'(k,s)| \geq (k-m) - c$ gibt es mindestens

$$\begin{aligned}\binom{c}{s-(k-c-m)}+\cdots+\binom{c}{s} \\ \geq \binom{\lfloor m/w-1\rfloor}{m-(7m-\lceil m/w\rceil-m)}+\cdots+\binom{\lfloor m/w-1\rfloor}{m} \\ = \binom{\lfloor m/w-1\rfloor}{0}+\cdots+\binom{\lfloor m/w-1\rfloor}{\lfloor m/w-1\rfloor}=2^{\lfloor m/w-1\rfloor}\end{aligned}$$

verschiedene Belegungen der Variablen in $X'(k,s)$, sodass die k ersten x-Variablen bez. π genau s Einsen enthalten. Für den Fall $s=5m$ ist diese Anzahl ebenso mindestens

$$\begin{aligned}\binom{\lfloor m/w-1\rfloor}{5m-(7m-\lceil m/w\rceil-m)}+\cdots+\binom{\lfloor m/w-1\rfloor}{5m} \\ = \binom{\lfloor m/w-1\rfloor}{0}+\cdots+\binom{\lfloor m/w-1\rfloor}{\lfloor m/w-1\rfloor}=2^{\lfloor m/w-1\rfloor}.\end{aligned}$$

Durch Fixierung der k ersten x-Variablen bez. π ergeben sich also mindestens $2^{\lfloor m/w\rfloor-1}=2^{\lfloor n/(11w)\rfloor-1}$ verschiedene Subfunktionen von $SHWB_{n,w}$-$GRAPH$, und die Aussage des Theorems folgt aus Theorem 2.3.6. Falls n kein Vielfaches von 11 ist, so gilt doch eine untere Schranke von $2^{\lfloor(n-10)/(11w)\rfloor-1}$, da leicht ersichtlich $SHWB_{n,w}$-$GRAPH$ Subfunktion von $SHWB_{n',w}$-$GRAPH$ mit $n\leq n'$ ist. Hier wird schließlich die Einschränkung $w\leq n-10$ notwendig. □

Die OBDD-Größe von $SHWB_{n,w}$-$GRAPH$ ist also superpolynomiell in n für $w=o(n/\log n)$.

Wir haben drei bekannte exponentielle Schranken auf die entsprechenden Graphen der Funktionen übertragen. Im Rahmen dieser Dissertation werden wir jedoch lediglich das Resultat für den Graphen der Multiplikation anwenden.

5 Kürzeste Wege

Wir wollen uns in diesem Kapitel zwei Kürzeste-Wege-Problemen widmen: **Single-Source-Shortest-Paths** und **All-Pairs-Shortest-Paths**. Zunächst stellen wir einen spezialisierten impliziten Iterative-Quadrieren-Ansatz für die auf positive Kantenlängen eingeschränkte Problemvariante $\mathbf{APSP}^+$ vor und zeigen, dass es sich um einen Festparameteralgorithmus handelt. Parameter ist dabei das Maximum der Voll-OBDD-Breiten von Eingabe und Ausgabe bez. natürlicher gemischter Variablenordnungen.

Um dieses positive Ergebnis zu bekräftigen, zeigen wir im zweiten Teil des Kapitels, dass die Klasse der Graphen, deren oben genannter Parameter konstant ist, gegen wichtige Graphoperationen abgeschlossen ist. Somit ist der vorgestellte $\mathbf{APSP}^+$-Algorithmus auch auf solchen Graphen effizient, die dementsprechend konstruiert werden können.

Um die Vermutung der Notwendigkeit einer Einschränkung auf positive Kantenlängen für die Existenz eines entsprechenden FPT-Algorithmus zu untermauern, zeigen wir im dritten Teil des Kapitels, dass nahe liegende, kanonische Iterative-Quadrieren-Ansätze, die auch mit Nullkanten umgehen können, trotz konstanter Voll-OBDD-Breite von Eingabe und Ausgabe exponentiell große Zwischen-OBDDs erzeugen müssen.

Im vierten Teil des Kapitels werden wir uns schließlich vorübergehend von unserem Ziel polylogarithmischer Laufzeit auf stark strukturierten Instanzen abwenden und implizite Versionen zweier populärer Kürzeste-Wege-Algorithmen vorstellen. Sie machen keinen Gebrauch von iterativem Quadrieren und führen $\Omega(|V|)$ funktionale Operationen durch. So ist zwar von vorherein die Hoffnung auf polylogarithmische Laufzeit vereitelt, jedoch besteht die Annahme, dass so für einen größeren Anteil von Eingaben kleinere OBDDs entstehen als bei spezialisierten impliziten Verfahren. In Experimenten auf verschiedenen zufälligen Graphen und speziellen strukturierten Graphen zeigt sich tatsächlich ein gutes durchschnittliches Verhalten, das teilweise wesentlich effizienter ist, als es einfache obere Schranken erwarten lassen.

Wir überlegen uns zunächst eine sinnvolle Übertragung der Problemdefinitionen ins implizite Szenario. Die Eingabe $G = (V, E, \phi)$ ist, wie in Abschnitt 2.5 eingeführt, als charakteristische Funktion $\chi_G \in B_{3n}$ gegeben. Bei **SSSP** kommt der Startknoten $q \in V$ hinzu, von dem aus alle anderen Knoten auf kürzesten Wegen erreicht werden sollen. Die Ausgabefunktion $\Delta\colon V \to \mathbb{N}$ für **SSSP** bzw. $\Delta\colon E \to \mathbb{N}$ für **APSP** setzen wir kanonisch in Funktionen $\chi_\Delta(x, a) = 1 :\Leftrightarrow \Delta(v_{|x|}) = |a|$ bzw. $\chi_\Delta(x, y, a) = 1 :\Leftrightarrow$

$\Delta(v_{|x|}, v_{|y|}) = |a|$ um. Später überlegen wir dann, wie auch die Kanten eines konkreten Pfades ermittelt werden können.

Da wir für alle Argumente der charakteristischen Funktionen die gleiche Anzahl boolescher Variablen verwenden wollen, mit der auch alle kürzesten Pfadlängen binär dargestellt werden sollen, wählen wir $n := \lceil \log(|V| \cdot B) \rceil$ für $B - 1 = \max_{e \in E}\{\phi(e)\}$, da $|V| \cdot B$ offenbar eine obere Schranke für Pfadlängen in G ist, die wir mit den Variablen eines Argumentvektors binär darstellen wollen. Isolierte zusätzliche Knoten beeinflussen auch hier nicht die Korrektheit der Algorithmen – wir können also auf eine Berücksichtigung der Funktion $\chi_V = 1$ verzichten.

5.1 Ein APSP-Festparameteralgorithmus

Wir möchten also iteratives Quadrieren einsetzen, um eine Operationsanzahl von poly(n) zu erreichen. Analog zu den bisher kennengelernten Iterative-Quadrieren-Ansätzen liegt es nahe, die maximale Anzahl der Pfadkanten iterativ zu verdoppeln, und man erhält den nahe liegenden Algorithmus 5.1.1.

Algorithmus 5.1.1 : Ein impliziter Iterative-Quadrieren-Ansatz für **APSP**.

Eingabe : Impliziter Graph $\chi_G(x, y, a)$.
Ausgabe : Implizite Distanzfunktion $\chi_\Delta(x, y, a)$.

1 $\chi_\Delta(x, y, a) := [(x = y) \wedge (|a| = 0)] \vee \chi_G(x, y, a)$;
2 **repeat**
3 $\quad \chi_{\Delta'}(x, y, a) := \chi_\Delta(x, y, a)$;
4 $\quad S(x, y, a) := (\exists z, b, c)\ [\chi_{\Delta'}(x, z, b) \wedge \chi_{\Delta'}(z, y, c) \wedge (|a| = |b| + |c|)]$;
5 $\quad \chi_\Delta(x, y, a) := S(x, y, a) \wedge \overline{(\exists b)\ [S(x, y, b) \wedge (b < a)]}$;
6 **until** $\chi_\Delta(x, y, a) = \chi_{\Delta'}(x, y, a)$;
7 **return** $\chi_\Delta(x, y, a)$;

Die Korrektheit ist offensichtlich, es werden $\mathcal{O}(n \cdot \log |V|) = \mathcal{O}(n^2)$ funktionale Operationen ausgeführt. Wieder bestätigt sich, dass sich grundlegende Probleme oft durch einfache implizite Iterative-Quadrieren-Ansätze mit poly(n) funktionalen Operationen lösen lassen, wenn Hilfsargumente wie z, b und c verwandt werden dürfen. Auch hier könnten wir wie in Abschnitt 3.8 die Gesamtlaufzeit auf speziellen Instanzen wie z. B. Gittern mit geeignet regelmäßigen Kantenlängen analysieren.

Stattdessen stellen wir den alternativen Algorithmus 5.1.2 vor, für den wir ein allgemeineres Laufzeitresultat erzielen können. Er löst jedoch nur das eingeschränkte Problem $\mathbf{APSP}^+$, und für die Eingabe $G = (V, E, \phi)$ ist nun $\phi\colon E \to \mathbb{N}^+$.

Wir diskutieren zunächst die Korrektheit und die Anzahl funktionaler Operationen. Dabei hilft uns das folgende Lemma, das in Abbildung 5.1.1 veranschaulicht wird.

Algorithmus 5.1.2 : Ein impliziter Iterative-Quadrieren-Ansatz für $\mathbf{APSP}^+$.

Eingabe : Impliziter Graph $\chi_G(x, y, a)$.
Ausgabe : Implizite Distanzfunktion $\chi_\Delta(x, y, a)$.

1 $R(x, y, a) := [(|a| = 0) \wedge (x = y)] \vee [(|a| = 1) \wedge \chi_G(x, y, a)]$;
2 **for** $i := 2$ **to** $\lceil \log(|V| \cdot B) \rceil$ **do**
3 $\quad S(x, y, a) := R(x, y, a)$
4 $\quad\quad \vee (\exists w, z, b, c, d) \left[R(x, w, b) \wedge \chi_G(w, z, c) \wedge R(z, y, d) \wedge (|a| = |b| + |c| + |d|)\right]$;
5 $\quad R(x, y, a) := S(x, y, a) \wedge (|a| < 2^i) \wedge \overline{(\exists b)\ [S(x, y, b) \wedge (b < a)]}$;
6 $\chi_\Delta(x, y, a) := R(x, y, a)$;
7 **return** $\chi_\Delta(x, y, a)$;

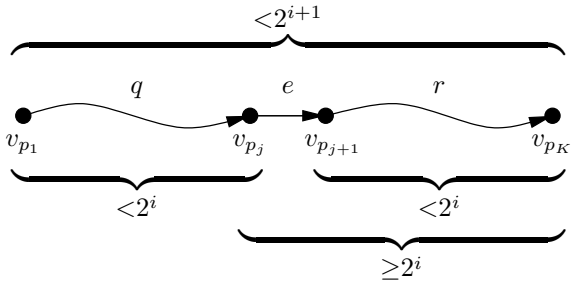


Abbildung 5.1.1: Veranschaulichung von Lemma 5.1.1.

Lemma 5.1.1. *Sei $p = (v_{p_1}, \ldots, v_{p_K})$ ein Pfad in G mit $K \geq 2$ und $||p|| < 2^{i+1}$ für $||p|| := \sum_{j=1}^{K-1} \phi(v_{p_j}, v_{p_{j+1}})$ und $i \in \mathbb{N}$. Dann gibt es eine Kante $(v_{p_j}, v_{p_{j+1}}) =: e$, sodass für $(v_{p_1}, \ldots, v_{p_j}) =: q$ und $(v_{p_{j+1}}, \ldots, v_{p_K}) =: r$ die Schranken $||q||, ||r|| < 2^i$ gelten.*

Beweis: Wir wählen die Kante $e = (v_{p_j}, v_{p_{j+1}})$ mit dem kleinsten Index j, sodass $||r|| < 2^i$. Für $j = 1$ folgt $||q|| = 0$. Für $j > 1$ garantiert die Wahl von j sowie $\phi(e) > 0$, dass $||(v_{p_j}, \ldots, v_{p_K})|| \geq 2^i$. Wegen $||p|| < 2^{i+1}$ folgt somit auch $||q|| < 2^i$. □

Theorem 5.1.2. *Algorithmus 5.1.2 berechnet korrekt $\chi_\Delta(x, y, a)$ und führt $\mathcal{O}(n^2)$ funktionale Operationen aus.*

Beweis: Da alle Kanten positive Längen haben, stellt $R(x, y, a)$ vor dem ersten Schleifendurchlauf offenbar alle kürzesten Pfade der Längen 0 und 1 dar (siehe Zeile 1).

Behauptung. *Nach dem j-ten Schleifendurchlauf gilt*

$$R(x, y, a) = \chi_\Delta(x, y, a) \wedge (|a| < 2^{j+1}).$$

Wir beweisen zunächst die Behauptung per vollständiger Induktion. Für $j = 0$ gilt die Aussage nach obiger Feststellung. Nach Induktionsvoraussetzung sind also nach dem $(j-1)$-ten Schleifendurchlauf alle Distanzen $|a|$ kürzester $v_{|x|}$-$v_{|y|}$-Pfade mit $|a| < 2^j$ bekannt und durch $R(x, y, a)$ repräsentiert. Nach Lemma 5.1.1 kann ein kürzester $v_{|x|}$-$v_{|y|}$-Pfad p mit $||p|| \in \{2^j, \ldots, 2^{j+1} - 1\}$ in zwei Teilpfade q und r mit $||q||, ||r|| < 2^j$ und eine trennende Zwischenkante e eingeteilt werden. Die Teilpfade q und r sind ebenfalls kürzeste Pfade und werden nach Induktionsvoraussetzung seit Beendigung von Schleifendurchlauf $j-1$ bereits durch $R(x, y, a)$ abgedeckt. Eine Betrachtung der Zeilen 3 und 4 ergibt, dass das Tripel $(x, y, (||p||)_2)$ durch die Funktion S abgedeckt wird. Zeile 5 sorgt dafür, dass Pfade p mit $||p|| \geq 2^{j+1}$ aus S entfernt werden, da nach Zeile 2 $i = j + 1$ gilt. Außerdem werden hier die Längen nicht minimaler $v_{|x|}$-$v_{|y|}$-Pfade entfernt. $R(x, y, a)$ repräsentiert nun alle kürzesten Pfade mit Länge in $\{0, \ldots, 2^{j+1} - 1\}$.

Nach Abbruch der Schleife ist also

$$R(x, y, a) = \chi_\Delta(x, y, a) \wedge (|a| < 2^{\lceil \log(|V| \cdot B) \rceil}) = \chi_\Delta(x, y, a),$$

und der Algorithmus berechnet korrekt die Ausgabefunktion $\chi_\Delta(x, y, a)$.

Es gibt $\mathcal{O}(\log(|V| \cdot B)) = \mathcal{O}(n)$ Schleifendurchläufe. In jedem Durchlauf und in Zeile 1 werden jeweils $\mathcal{O}(n)$ funktionale Operationen ausgeführt, da jede Quantifizierung über ein Argument n einzelnen Quantifizierungen entspricht. Es ergibt sich eine Gesamtanzahl von $\mathcal{O}(n^2)$ Operationen. □

Wir verdoppeln also in Algorithmus 5.1.2 nicht die Anzahl der Pfadkanten, sondern die tatsächlichen Pfadlängen bez. ϕ. Im Beweis von Theorem 5.1.2 haben wir bereits eine zentrale Eigenschaft bewiesen: Eine Funktion in jedem Schleifendurchlauf lässt sich durch Konjunktion der Ausgabe und einer negierten multivariaten Thresholdfunktion darstellen. Die Leserin und der Leser sollten sich nun an unsere in Abschnitt 3.7 eingeführte Technik erinnern und einsehen, dass es sich bei Algorithmus 5.1.2 um einen FPT-Algorithmus handelt, dessen Parameter das Maximum der Voll-OBDD-Breiten von Ein- und Ausgabe für natürliche gemischte Variablenordnungen ist. Wir werden dies nun natürlich auch formal beweisen.

Theorem 5.1.3. *Algorithmus 5.1.2 berechnet $\pi^{\mathrm{id}}_{3,n}$-OBDD$[\chi_\Delta]$ in Zeit $\alpha(w) \cdot n^3 \log n$ und auf Platz $\alpha(w) \cdot n$. Dabei sei w das Maximum der Voll-OBDD-Breiten von χ_G und χ_Δ bez. $\pi^{\mathrm{id}}_{3,n}$ und α eine geeignete Funktion $\mathbb{N} \to \mathbb{N}$.*

Beweis: Wir betrachten im Folgenden alle Voll-OBDD-Breiten bez. $\pi^{\mathrm{id}}_{k,n}$ für die Anzahl $k = \mathcal{O}(1)$ der insgesamt benötigten Knotennummern-/Längenargumente, ohne es jeweils explizit zu erwähnen. Nach unseren Überlegungen im Beweis von Theorem 5.1.2 gilt $R(x, y, a) = \chi_\Delta(x, y, a) \wedge (|a| < 2^{j+1})$ nach jedem j-tem Schleifendurchlauf. Der Vergleich von $|a|$ mit 2^{j+1} ist eine negierte multivariate Thresholdfunktion aus $\mathbb{T}^{\mathcal{O}(1)}_{k,n}$

und hat demnach konstante Voll-OBDD-Breite. Laut Theorem 3.7.4 hat $R(x, y, a)$ also Voll-OBDD-Breite $\beta(w)$ für ein geeignetes $\beta\colon \mathbb{N} \to \mathbb{N}$ und ist unabhängig von n.

Wir stellen fest, dass zwischen jeweils zwei Funktionen $R(x, y, a)$, für die diese Breiteneigenschaft gilt, stets nur eine Sequenz S von $\mathcal{O}(1)$ funktionalen Operationen und Quantifizierungsblöcken gemäß Theorem 3.7.4 durchgeführt wird. Diese werden angewandt auf χ_G, Hilfsfunktionen in $\mathbb{V}^{\mathrm{id}}_{k,n}$ mit Voll-OBDD-Breite $\mathcal{O}(1)$ sowie auf Zwischenergebnisse. Es existiert also wieder ein $\gamma\colon \mathbb{N} \to \mathbb{N}$, sodass die Ausführung jeder Sequenz S in Zeit $\gamma(\beta(w)) \cdot n^2 \log n$ und auf Platz $\gamma(\beta(w)) \cdot n$ geschieht. Da es $\mathcal{O}(\log(|V| \cdot B)) = \mathcal{O}(n)$ solche Sequenzen gibt, bis χ_Δ ausgegeben wird, ist die Gesamtlaufzeit $\alpha(w) \cdot n^3 \log n$ mit $\alpha := \gamma \circ \beta$. Da stets nur konstant viele Funktionen gespeichert werden, beträgt ebenfalls der Gesamtplatz $\alpha(w) \cdot n$.

Schließlich bemerken wir, dass die Erzeugung der Hilfsfunktionen in $\mathbb{V}^{\mathrm{id}}_{k,n}$ (z. B. $|a| < 2^i$) natürlich in linearer Zeit $\mathcal{O}(n)$ geschehen kann (siehe Korollar 3.7.5). □

Anmerkung 5.1.4. *Wir betrachten Algorithmus 5.1.2 genauer: Für einen komfortablen Beweis von Theorem 5.1.3 mithilfe von Theorem 3.7.4 sind wir von der Variablenordnung $\pi^{\mathrm{id}}_{k,n}$ ausgegangen. Die natürliche Ordnung* id *war dabei für die OBDD-Breite der multivariaten Thresholdfunktionen relevant. Wir können diese Annahme jedoch etwas lockern: Multivariate Thresholdfunktionen sind in Algorithmus 5.1.2 nur auf Längenargumenten definiert. Knotenargumente werden lediglich auf Gleichheit getestet. Theorem 5.1.3 gilt also auch dann, wenn die Knoten codierenden Variablen eine andere lokale Ordnung verwenden als* id *– diese muss jedoch wieder für alle Knotenargumente gleich sein. Die Implementation muss dann sicherstellen, dass unterschiedliche Variablenvektoren für Knotennummern und Längenwerte verwandt werden.*

5.1.1 Berechnung konkreter Pfadkanten

Wir interessieren uns in dieser Dissertation im Zusammenhang mit Kürzeste-Wege-Problemen hauptsächlich für die Berechnung der Längen solcher Wege. Dies liegt zum einen daran, dass wir bereits für diese Variante Härteresultate in Kapitel 7 kennenlernen werden. Zum anderen ist es noch nicht gelungen, einen Festparameteralgorithmus wie im vorherigen Abschnitt auch für die anschließende Berechnung konkreter Pfade zu entwickeln.

Algorithmus 5.1.2 kann jedoch leicht so erweitert werden, dass alle Kanten kürzester Wege ausgegeben werden, ohne dass die Festparametereigenschaften dadurch beeinträchtigt werden. Sei dazu $P(w, x, y, z) = 1$ genau dann, wenn $(v_{|w|}, v_{|x|})$ auf einem kürzesten $v_{|y|}$-$v_{|z|}$-Pfad liegt. Wir können diese Funktionen folgendermaßen berechnen:

$$P(w, x, y, z) := (\exists a, b, c, d)\ [(|a| = |b| + |c| + |d|) \wedge \chi_\Delta(y, z, a) \\ \wedge \chi_\Delta(y, w, b) \wedge \chi_G(w, x, c) \wedge \chi_\Delta(x, z, d)]\,.$$

Mit den gleichen Argumenten wie im Beweis von Theorem 5.1.3 folgt nun:

Korollar 5.1.5. *Durch einen entsprechend erweiterten Algorithmus 5.1.2 kann auch* $\pi^{\mathrm{id}}_{3,n}$-OBDD$[P]$ *in Zeit* $\beta(\alpha(w))\cdot n^3 \log n$ *und auf Platz* $\alpha(w)\cdot n$ *für das Maximum* w *der Voll-OBDD-Breiten von* χ_G *und* χ_Δ *bez.* $\pi^{\mathrm{id}}_{3,n}$ *und ein geeignetes* $\alpha\colon \mathbb{N} \to \mathbb{N}$ *berechnet werden.*

Um nun eine Funktion $P^*(w,x,y,z)$ zu berechnen, die für jedes Knotenpaar $(v_{|y|}, v_{|z|})$ eine Teilmenge der von $P(w,x,y,z)$ dargestellten Kanten repräsentiert, die genau einem kürzesten $v_{|y|}$-$v_{|z|}$-Pfad entsprechen, kann eine Niveaunetzwerkkonstruktion wie in Abschnitt 3.4 mit anschließender Einzelpfadkonstruktion wie in Abschnitt 3.5 verwandt werden. Eingabe wäre aber dann nicht die charakteristische Funktion $\chi_G(w,x)$ *eines* Graphen, sondern die Funktion $\chi_G(w,x,y,z) := P(w,x,y,z)$, die der Graphenmenge $\{G_{y,z}\}_{y,z}$ aller Kürzeste-Wege-Teilgraphen für $v_{|y|}, v_{|z|} \in V$ entspricht. Für einen derartigen Ansatz existiert jedoch noch kein Analyseresultat.

5.2 Graphoperationen und Vollständige-OBDD-Breite

Die Einschränkung auf Graphen, deren charakteristische Funktion konstante Voll-OBDD-Breite besitzt, scheint sehr restriktiv zu sein. Selbst bei konstanter OBDD-Breite kann die Voll-OBDD-Breite bereits in der Variablenanzahl wachsen. Wir werden in Kapitel 7 sehen, dass selbst für Graphen mit dieser starken Einschränkung viele grundlegende OBDD-Suchprobleme schwierig sind. Diese Beweise gründen jedoch gerade darauf, dass einige wichtige und nicht triviale Graphinstanzen nur konstante Voll-OBDD-Breite haben. Die Negativresultate dieser Dissertation zeigen also, dass auch derartig eingeschränkte OBDDs noch eine hohe Ausdruckskraft haben. In Sawitzki (2004e) wird zudem für einfache Basisgraphen wie Sterne und Räder sowie für eingeschränkte Threshold-Graphen konstante Voll-OBDD-Breite nachgewiesen.

Aber auch wenn die Eingabeinstanz konstante Voll-OBDD-Breite besitzt, sagt dies noch nichts über die Ausgabefunktion aus. Wir werden sehen, dass diese exponentiell größere OBDDs erfordern kann als die Eingabefunktion. Dennoch wollen wir hier ein Argument für die Nützlichkeit von Algorithmus 5.1.2 vorstellen: die Abgeschlossenheit konstanter Voll-OBDD-Breite von χ_G *und* χ_Δ gegen einige wichtige Graphoperationen. Allgemeiner ausgedrückt zeigen wir, dass das Wachstum der OBDD-Größe bei Anwendung dieser Graphoperationen wieder auf die Parameter der Voll-OBDD-Breiten von χ_G und χ_Δ beschränkt werden kann. Algorithmus 5.1.2 ist somit auch dann auf einem Graphen G effizient im Sinne von Theorem 5.1.3, wenn G durch konstant viele solche Graphoperationen aus einfacheren Graphen $G_1, \ldots, G_r$ komponiert werden kann, für die der Algorithmus ebenfalls effizient ist.

Seien nun G_1, G_2 und G_3 drei schlaufenfreie gewichtete gerichtete Graphen.

Definition 5.2.1 (Graphoperationen).

(a) G_3 *heißt* Cojoin *von* G_1 *und* G_2*, wenn* $V_3 = V_1 \cup V_2$*,* $E_3 = E_1 \cup E_2$ *und* $\phi_3(e) = \phi_i(e)$ *für* $e \in E_i$ *gelten.*

(b) G_3 *heißt* L-Join *von* G_1 *und* G_2 *für* $L \in \mathbb{N}$*, wenn* $V_3 = V_1 \cup V_2$*,* $E_3 = E_1 \cup E_2 \cup (V_1 \times V_2)$ *und* $\phi_3(e) = \phi_i(e)$ *für* $e \in E_i$ *bzw.* $\phi_3(e) = L$ *für* $e \in V_1 \times V_2$ *gelten.*

(c) G_3 *heißt* Knotenersetzung *von* G_1 *in* G_2*, wenn* $V_3 = V_1 \times V_2$ *und*

$$E_3 = \Big\{ \big((u,u'),\ (v,v')\big) \mid \big((u,v) \in E_1 \wedge (u' = v')\big) \vee (u',v') \in E_2 \Big\}$$

gelten sowie $\phi_3\big((u,u'),\ (v,v')\big) = \phi_1(u,v)$ *für* $u' = v'$ *bzw.* $\phi_3\big((u,u'),\ (v,v')\big) = \phi_2(u',v')$ *für* $(u',v') \in E_2$*.*

(d) G_3 *heißt* Produkt *von* G_1 *und* G_2*, wenn* $V_3 = V_1 \times V_2$ *und*

$$E_3 = \Big\{ \big((u,u'),\ (v,v')\big) \mid \big((u,v) \in E_1 \wedge (u' = v')\big) \vee \big((u = v) \wedge (u',v') \in E_2\big) \Big\}$$

gelten sowie $\phi_3\big((u,u'),\ (v,v')\big) = \phi_1(u,v)$ *für* $u' = v'$ *bzw.* $\phi_3\big((u,u'),\ (v,v')\big) = \phi_2(u',v')$ *für* $u = v$*.*

Ein Cojoin ist also eine disjunkte Vereinigung, während bei einem L-Join ein vollständiger bipartiter von G_1 nach G_2 gerichteter Graph mit Kantenlängen L zwischen den Knoten von G_1 und G_2 hinzugefügt wird. Diese beiden Operationen spielen z. B. eine grundlegende Rolle für konstruktiv definierte Graphklassen wie die der Cographen, die durch sukzessive Anwendung von Join- und Cojoin-Operationen auf anfangs isolierte Knoten erhalten werden. Bei der Knotenersetzung von G_1 in G_2 wird jeder Knoten in G_2 durch eine Kopie des Graphen G_1 ersetzt, und bisherige Kanten in G_2 werden durch vollständige gerichtete bipartite Graphen zwischen den entsprechenden Kopien von G_1 ersetzt. Beim Produkt schließlich handelt es sich um das kanonisch definierte kartesische Produkt beider Graphen. Schließlich können entsprechende Resultate auch leicht für viele Variationen dieser vier Operationen gezeigt werden. Derartige Graphoperationen, die das kartesische Produkt der Knotenmengen erzeugen, spielen z. B. immer dann eine Rolle, wenn der Modellierung realer Objekte durch Graphknoten ein Aspekt (bzw. eine Dimension) hinzugefügt wird. So haben wir in Abschnitt 3.8.8 den Knoten eines Graphen eine Zeitkomponente hinzugefügt.

Wir betrachten die Vereinigungsoperationen (a) und (b) nun getrennt von den Produktoperationen (c) und (d).

5.2.1 Vereinigungsoperationen

Wir betrachten den Fall $V_3 = V_1 \cup V_2$. Wir nehmen o. B. d. A. an, dass von vornherein alle betrachteten charakteristischen Funktionen in B_{3n} sind und

$$n = \Theta(\log(|V_3| \cdot B_3)) = \Theta(\log(\max\{|V_1|, |V_2|\} \cdot \max\{B_1, B_2, L\}))$$

wieder hinreichend groß definiert ist, um die maximale Länge kürzester Pfade in G_1, G_2 und G_3 codieren zu können. Die charakteristischen Funktionen seien durch $\pi^{\mathrm{id}}_{3,n}$-OBDDs dargestellt. Dabei sei x_{i^*} ein ausgezeichnetes Bit in jedem Knotenargument $x \in \mathbb{B}^n$, das angibt, welchem der beiden ursprünglichen Graphen G_1 oder G_2 der Knoten $v_{|x|}$ entspringt. Die Funktionen χ_{G_1}, χ_{G_2}, χ_{Δ_1} und χ_{Δ_2} seien sinnvollerweise von x_{i^*} unabhängig.

Theorem 5.2.2. *Sei G_3 der Cojoin von G_1 und G_2. Wenn die Funktionen χ_{G_1}, χ_{G_2}, χ_{Δ_1} und χ_{Δ_2} jeweils höchstens Voll-OBDD-Breite w bez. einer Variablenordnung π haben, dann haben χ_{G_3} und χ_{Δ_3} jeweils höchstens Voll-OBDD-Breite $\mathcal{O}(w^2)$ bez. π.*

Beweis: Kürzeste Pfade werden durch die disjunkte Vereinigung nicht tangiert. Wir geben entsprechende boolesche Ausdrücke für χ_{G_3} und χ_{Δ_3} an:

$$\chi_{G_3}(x,y,a) = \big[(v_{|x|}, v_{|y|} \in V_1) \wedge \chi_{G_1}(x,y,a)\big] \vee \big[(v_{|x|}, v_{|y|} \in V_2) \wedge \chi_{G_2}(x,y,a)\big],$$

$$\chi_{\Delta_3}(x,y,a) = \big[(v_{|x|}, v_{|y|} \in V_1) \wedge \chi_{\Delta_1}(x,y,a)\big] \vee \big[(v_{|x|}, v_{|y|} \in V_2) \wedge \chi_{\Delta_2}(x,y,a)\big].$$

Tests der Form $v_{|x|} \in V_1$ sind durch Überprüfung des Bits x_{i^*} zu realisieren, die wiederum offenbar Voll-OBDD-Breite $\mathcal{O}(1)$ haben. Laut Lemma 3.7.1 entsprechen χ_{G_3} und χ_{Δ_3} also jeweils dem Ergebnis einer binären Synthese zweier Funktionen der Voll-OBDD-Breite $\mathcal{O}(w)$, das demnach Voll-OBDD-Breite $\mathcal{O}(w^2)$ besitzt. □

Theorem 5.2.3. *Sei G_3 der L-Join von G_1 und G_2. Wenn die Funktionen χ_{G_1}, χ_{G_2}, χ_{Δ_1} und χ_{Δ_2} jeweils höchstens Voll-OBDD-Breite w bez. einer Variablenordnung π haben, dann haben χ_{G_3} und χ_{Δ_3} jeweils höchstens Voll-OBDD-Breite $\mathcal{O}(w^2)$ bez. π.*

Beweis: Kürzeste Pfade in G_1 und G_2 werden durch den L-Join nicht tangiert, kürzeste Pfade von G_1 nach G_2 sind genau die neuen Kanten der Länge L. Wir geben entsprechende boolesche Ausdrücke für χ_{G_3} und χ_{Δ_3} an:

$$\begin{aligned}\chi_{G_3}(x,y,a) = {} & \big[(v_{|x|}, v_{|y|} \in V_1) \wedge \chi_{G_1}(x,y,a)\big] \\ & \vee \big[(v_{|x|}, v_{|y|} \in V_2) \wedge \chi_{G_2}(x,y,a)\big] \\ & \vee \big[(v_{|x|} \in V_1) \wedge (v_{|y|} \in V_2) \wedge (|a| = L)\big], \quad (5.2.1)\end{aligned}$$

$$\chi_{\Delta_3}(x,y,a) = \big[(v_{|x|}, v_{|y|} \in V_1) \wedge \chi_{\Delta_1}(x,y,a)\big]$$

$$\vee \big[(v_{|x|}, v_{|y|} \in V_2) \wedge \chi_{\Delta_2}(x, y, a)\big]$$
$$\vee \big[(v_{|x|} \in V_1) \wedge (v_{|y|} \in V_2) \wedge (|a| = L)\big]. \quad (5.2.2)$$

Wir argumentieren analog zu Theorem 5.2.2. Für jede der beiden Funktionen kommt nun eine binäre Synthese mit einem jeweiligen Teilausdruck für den Fall $(v_{|x|} \in V_1) \wedge (v_{|y|} \in V_2)$ hinzu. Diese Teilausdrücke in den dritten Zeilen der Gleichungen (5.2.1) und (5.2.2) sind offenbar aus Funktionen konstanter Voll-OBDD-Breite komponiert, und so ergibt sich ebenfalls Gesamt-Voll-OBDD-Breite $\mathcal{O}(w^2)$. □

5.2.2 Produktoperationen

Wir betrachten den Fall $V_3 = V_1 \times V_2$. Wir nehmen diesmal o. B. d. A. an, dass die charakteristischen Funktionen für G_1 und G_2 in B_{3kn} sind, mit $k = \mathcal{O}(1)$, und

$$n = \Theta(\log(|V_3| \cdot B_3)) = \Theta(\log(|V_1| \cdot |V_2| \cdot \max\{B_1, B_2\}))$$

wieder hinreichend groß definiert ist, um die maximale Länge kürzester Pfade in G_1, G_2 und G_3 codieren zu können. Die charakteristischen Funktionen χ_{G_1}, χ_{G_2}, χ_{Δ_1} und χ_{Δ_2} seien durch $\pi^{\mathrm{id}}_{3k,n}$-OBDDs dargestellt. Die Ergebnisfunktionen χ_{G_3} und χ_{Δ_3} definieren wir als in B_{6kn} mit Variablenordnung $\pi^{\mathrm{id}}_{6k,n}$, weil wir ein Knotenargument $x \in \mathbb{B}^{2kn}$ nun ähnlich wie bei der Darstellung von Gittergraphen (siehe Abschnitt 3.8.2) aus zwei gemischt gelesenen Komponenten $q(x)$ und $r(x)$ in $\mathbb{B}^{kn}$ bestehen lassen. Schließlich sollen im Sinne unserer formalen Vereinbarungen Knotennummern und Längenwerte durch die gleiche Anzahl von jeweils $2kn$ Variablen codiert werden, wobei aufgrund unserer anfänglichen Wahl von n nur eines der Längenwertteilargumente benötigt wird. So codiert ein gültiges x den Knoten $(v_{|q(x)|}, v_{|r(x)|}) \in V_1 \times V_2$.

Theorem 5.2.4. *Sei G_3 die Knotenersetzung von G_1 in G_2. Wenn die Funktionen χ_{G_1}, χ_{G_2}, χ_{Δ_1} und χ_{Δ_2} jeweils höchstens Voll-OBDD-Breite w bez. $\pi^{\mathrm{id}}_{3k,n}$ für $k = \mathcal{O}(1)$ haben, dann haben χ_{G_3} und χ_{Δ_3} jeweils höchstens Voll-OBDD-Breite $\alpha(w)$ bez. $\pi^{\mathrm{id}}_{6k,n}$ für ein geeignetes $\alpha\colon \mathbb{N} \to \mathbb{N}$.*

Beweis: Wir geben einen entsprechenden booleschen Ausdruck für χ_{G_3} an:

$$\chi_{G_3}(x, y, a) = \big[\chi_{G_1}(q(x), q(y), a) \wedge (r(x) = r(y))\big] \vee \chi_{G_2}(r(x), r(y), a).$$

Wir betrachten einen kürzesten Pfad zwischen Knoten $u := (u', u'') \in V_3$ und $v := (v', v'') \in V_3$. Befinden sich u und v in unterschiedlichen Kopien von G_1, so ist offenbar $\Delta_3(u, v) = \Delta_2(u'', v'')$ und $u'' \neq v''$ – welche G_1-Knoten auf diesem Weg besucht werden, ist für seine Länge irrelevant. Anderenfalls befinden sich u und v in der gleichen Kopie $u'' = v''$ von G_1. Ein kürzester Pfad p kann in dieser Kopie bleiben, woraus $\Delta_3(u, v) = \Delta_1(u', v')$ folgt. Alternativ darf p einen „Umweg“ durch andere Kopien von G_1 in G_2 beschreiten. Solch ein Umweg entspricht einem Kreis $(u'', \ldots, t, \ldots, u'')$

mit $t \neq u'' = v''$, für den wieder irrelevant ist, welche G_1-Knoten besucht werden. Wir überführen diese Beobachtungen in einen booleschen Ausdruck für χ_{Δ_3}:

$$D(r(x), a) := (\exists z, b, c)\big[(r(x) \neq z) \wedge \chi_{\Delta_2}(r(x), z, b) \wedge \chi_{\Delta_2}(z, r(x), c) \wedge (|a| = |b| + |c|)\big],$$

$$\begin{aligned}\chi_{\Delta_3}(x, y, a) = {} & \big[(r(x) \neq r(y)) \wedge \chi_{\Delta_2}(r(x), r(y), a)\big] \\ & \vee \Big[(r(x) = r(y)) \wedge \chi_{\Delta_1}(q(x), q(y), a) \wedge \overline{(\exists b)\big[(|b| < |a|) \wedge D(r(x), b)\big]}\Big] \\ & \vee \Big[(r(x) = r(y)) \wedge D(r(x), a) \wedge \overline{(\exists b)\big[(|b| < |a|) \wedge \chi_{\Delta_1}(q(x), q(y), b)\big]}\Big].\end{aligned}$$

Diesmal haben wir auch Hilfsvariablen und Existenzquantoren eingesetzt. Wir betrachten die Voll-OBDD-Breiten: Die Ausdrücke für χ_{G_3} und χ_{Δ_3} entsprechen einer konstanten Anzahl an OBDD-Operationen, Argumentumordnungen und Quantifizierungsblöcken, die auf auf χ_{G_1}, χ_{G_2}, χ_{Δ_1} und χ_{Δ_2} sowie auf Funktionen in $\mathbb{V}^{\mathcal{O}(1)}_{\mathcal{O}(k),n}$ und auf Zwischenergebnisse angewandt werden. Die Breitenbeschränkung folgt direkt aus Theorem 3.7.4 und Korollar 3.7.5. □

Theorem 5.2.5. *Sei G_3 das Produkt von G_1 und G_2. Wenn die Funktionen χ_{G_1}, χ_{G_2}, χ_{Δ_1} und χ_{Δ_2} jeweils höchstens Voll-OBDD-Breite w bez. $\pi^{\mathrm{id}}_{3k,n}$ für $k = \mathcal{O}(1)$ haben, dann haben χ_{G_3} und χ_{Δ_3} jeweils höchstens Voll-OBDD-Breite $\alpha(w)$ bez. $\pi^{\mathrm{id}}_{6k,n}$ für ein geeignetes $\alpha\colon \mathbb{N} \to \mathbb{N}$.*

Beweis: Ein kürzester Pfad in G_3 ist aus kürzesten Pfaden in G_1 und G_2 zusammengesetzt. In welcher Reihenfolge Pfadkanten bez. G_1 und G_2 beschritten werden, ist für die Pfadlänge irrelevant – stets ändert sich nur jeweils eine Komponente eines Knotentupels (v, v'). Ein kürzester (u, u')-(v, v')-Pfad hat also Länge $\Delta_1(u, v) + \Delta_2(u', v')$. Wir geben boolesche Ausdrücke für χ_{G_3} und χ_{Δ_3} an:

$$\begin{aligned}\chi_{G_3}(x, y, a) = {} & \big[\chi_{G_1}(q(x), q(y), a) \wedge (r(x) = r(y))\big] \\ & \vee \big[(q(x) = q(y)) \wedge \chi_{G_2}(r(x), r(y), a)\big],\end{aligned}$$

$$\chi_{\Delta_3}(x, y, a) \;=\; (\exists b, c)\big[(|a| \;=\; |b| + |c|) \wedge \chi_{\Delta_1}(q(x), q(y), b) \wedge \chi_{\Delta_2}(r(x), r(y), c)\big].$$

Die Breitenbeschränkung folgt nun mit den gleichen Argumenten wie im Beweis von Theorem 5.2.4. □

Anmerkung 5.2.6. *Obwohl die Voll-OBDD-Breiten der Funktionen χ_{G_3} und χ_{Δ_3} hier bez. der Variablenordnung $\pi^{\mathrm{id}}_{6k,n}$ betrachtet wurden (entgegen der formalen Voraussetzung von Theorem 5.1.3, dass alle drei Argumente x, y und a jeweils bez.* id $\neq \pi^{\mathrm{id}}_{2k,n}$ *gelesen werden), gelten die oberen Schranken des Theorems aufgrund von Anmerkung 5.1.4 dennoch. Entscheidend ist wieder nur, dass alle Knotennummern- und Längenwertargumente jeweils mit gleicher Ordnung gelesen werden und dass das eine relevante der $2k$ formal eingeführten Längenwertteilargumente bez.* id *gelesen wird.*

Seien nun $G_1, \ldots, G_r$ gewichtete gerichtete schlaufenfreie Graphen mit entsprechenden **APSP**-Lösungen $\Delta_1, \ldots, \Delta_r$. Sei weiterhin $G = (V, E, \phi)$ das Ergebnis einer konstanten Anzahl von Graphoperationen auf $G_1, \ldots, G_r$ gemäß Definition 5.2.1 mit **APSP**-Lösung Δ und durch B beschränkten Kantenlängen.

Korollar 5.2.7. *Haben $\chi_{G_1}, \ldots, \chi_{G_r}$ und $\chi_{\Delta_1}, \ldots, \chi_{\Delta_r}$ maximale Voll-OBDD-Breite w bez. $\pi^{\mathrm{id}}_{3k,n}$ mit $k = \mathcal{O}(1)$, so haben χ_G und χ_Δ höchstens Voll-OBDD-Breite $\beta(w)$ bez. $\pi^{\mathrm{id}}_{3k',n'}$ für $k' = \mathcal{O}(k)$, $k'n' = \Theta(\log(|V| \cdot B))$ und geeignetes $\beta\colon \mathbb{N} \to \mathbb{N}$.*

Wir sind in diesem Abschnitt vereinfachend davon ausgegangen, dass keine Knotenfunktion χ_V benötigt wird. Tatsächlich enthalten die ursprünglichen Graphen, auf die die Graphoperationen angewandt werden, möglicherweise isolierte Knoten, die aufgrund der Knotencodierung entstanden sind. Diese sollen natürlich unverbunden bleiben bzw. wieder isolierten Knoten in $V_1 \times V_2$ entsprechen. Dies kann jedoch leicht erreicht werden, indem in sämtlichen booleschen Ausdrücken dieses Abschnitts Tests der Art $v_{|x|} \in V_i$ und Vorkommen von $\chi_{G_i}(x, y, a)$ für $i \in \{1, 2\}$ mit $\chi_{V_i}(x)$ bzw. $\chi_{V_i}(x) \wedge \chi_{V_i}(y)$ konjunktiv verknüpft werden. Wenn χ_{V_1} und χ_{V_2} ebenfalls höchstens Voll-OBDD-Breite w haben, so gelten die Theoreme dieses Abschnitts sowie Korollar 5.2.7 nach dieser Ergänzung offenbar immer noch. Bei den Vereinigungsoperationen erhöht sich der Grad des Polynoms, das die Voll-OBDD-Breite beschränkt, auf 4.

Schließlich haben auch

$$\chi_{V_3}(x) := \big[(v_{|x|} \in V_1) \wedge \chi_{V_1}(x)\big] \vee \big[(v_{|x|} \in V_2) \wedge \chi_{V_2}(x)\big]$$

für Vereinigungsoperationen bzw.

$$\chi_{V_3}(x) := \chi_{V_1}(q(x)) \wedge \chi_{V_2}(r(x))$$

für Produktoperationen leicht ersichtlich Voll-OBDD-Breite $\mathcal{O}(w^2)$ bez. ihrer jeweiligen Variablenordnung.

5.3 Eine exponentielle untere Schranke für einfache Iterative-Quadrieren-Ansätze

Algorithmus 5.1.2 ist an die Bedingung gebunden, dass der Eingabegraph keine Kanten der Länge 0 enthält. Wir beschäftigen uns nun mit der Frage, ob auch ohne diese Einschränkung die gleichen Festparametereigenschaften erreicht werden können. Uns gelingt leider nur eine eingeschränkte Antwort: Die Algorithmus 5.1.1 entsprechenden einfachen und einzigen bisher bekannten Iterative-Quadrieren-Ansätze für **APSP** sind keine FPT-Algorithmen. Solche Ansätze verdoppeln iterativ die Anzahl der erlaubten Pfadkanten, berechnen also eine Funktionenfolge $S^{(0)}, \ldots, S^{(r)}$ mit $r \leq \lceil \log |V| \rceil$, sodass $S^{(i)}(x, y, a) = 1$ genau dann gilt, wenn alle kürzesten $v_{|x|}$-$v_{|y|}$-Pfade, die aus nicht mehr als 2^i Kanten bestehen, Länge $|a|$ haben.

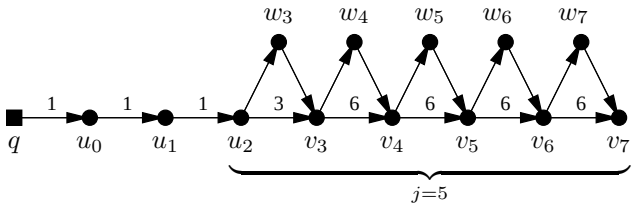


Abbildung 5.3.1: Der Teilgraph $G_3^{(6,5)}$. Zu w-Knoten inzidente Kanten haben Länge 1.

Dies ist die einzige Annahme, die wir verwenden werden, um für eine Instanz G_n mit konstanter Eingabe- und Ausgabe-Voll-OBDD-Breite eine exponentielle untere Schranke in n für das Zwischen-OBDD π-OBDD$[S_n^{(n)}]$ zu zeigen, die für jedes π gilt.

Wir definieren die pathologische Instanz: $G_n := (V_n, E_n)$ ist die Vereinigung der Teilgraphen $G_n^{(i,j)}$ mit $2^{n-1}+1 \leq i, j \leq 2^n - 1$, die lediglich einen besonderen Knoten q gemeinsam haben. Jedes $G_n^{(i,j)}$ ist ein Pfad $(q, u_0, \ldots, u_{2^n-j-1}, v_{2^n-j}, \ldots, v_{2^n-1}) =: p_{i,j}$ mit den Kantenlängen $\phi_n(\bullet, u_\bullet) := 1$, $\phi_n(v_\bullet, \bullet) := i$ und $\phi_n(u_{2^n-j-1}, v_{2^n-j}) := i+j-2^n$. Außerdem gibt es Knoten $w_{2^n-j}, \ldots, w_{2^n-1}$, die durch sogenannte *Kurzschlusskanten* (u_{2^n-j-1}, w_{2^n-j}), (w_{2^n-j}, v_{2^n-j}), $(v_\ell, w_{\ell+1})$, und $(w_{\ell+1}, v_{\ell+1})$ der Länge 1 für $2^n - j \leq \ell \leq 2^n - 2$ angebunden sind. Kurzschlusskanten überbrücken also alle Kanten, deren Länge größer als 1 ist. Abbildung 5.3.1 zeigt den Teilgraphen $G_3^{(6,5)}$.

Wir werden nun zum einen zeigen, dass Eingabe G_n und Ausgabe Δ_n konstante Voll-OBDD-Breite bez. einer natürlichen gemischten Variablenordnung haben und Algorithmus 5.1.2 somit Laufzeit $\mathcal{O}(n^3 \log n)$ hat. Andererseits werden wir zeigen, dass die Zwischenfunktion $S_n^{(n)}$ für kürzeste Pfade mit höchstens 2^n Kanten, die die in diesem Abschnitt diskutierten Iterative-Quadrieren-Ansätze auf G_n berechnen, exponentielle OBDD-Größe hat.

5.3.1 Implizite Darstellung von G_n

Wir stellen eine Knotencodierung für G_n vor und beweisen durch Angabe entsprechender boolescher Ausdrücke konstanter Länge, dass χ_{G_n} und χ_{Δ_n} konstante Voll-OBDD-Breite haben.

Knotencodierung

Zunächst betrachten wir die Knotennummern- und Längenwertcodierung: Gemäß unserer Vereinbarung in Abschnitt 2.5 sollen alle Argumente charakteristischer Funktionen aus der gleichen Anzahl von Teilargumenten der wiederum gleichen Länge bestehen. Die maximale Kantenlänge in G_n ist $2^n - 1$. Kein Pfad in G_n besteht aus mehr als 2^{n+1} Kanten, und so ist $2^{2n+1} - 1$ eine obere Schranke für die Länge eines jeden gerichteten Pfades. Wir werden Längenwerte daher durch jeweils $2n + 1$ Variablen

codieren, die Anmerkung 5.1.4 entsprechend gemäß der Ordnung id gelesen werden müssen. Knotenargumente müssen wir jedoch in drei gemischt gelesene Teilargumente aufteilen, und so ergibt sich die Gesamtanzahl von $3(2n+1)$ Variablen pro Knoten- und Längenwertargument. Nur eines der drei formal eingeführten Längenwertteilargumente wird daher relevant sein.

Wir überschätzen $|V_n|$ also durch $2^{3(2n+1)}$ und bezeichnen die Knoten diesmal mit $s_0, \ldots, s_{2^{3(2n+1)}-1}$. Jeder Knoten $s \in V_n$, der zum Teilgraphen $G_n^{(i,j)}$ gehört, erhält eine binäre Knotennummer x, die aus vier Komponenten besteht:

1. der binären Darstellung $(i)_2 \in \mathbb{B}^{2n+1}$ von i,
2. der binären Darstellung $(j)_2 \in \mathbb{B}^{2n+1}$ von j,
3. einem Bit c, wobei $c = 0$ kennzeichnet, dass s ein u- oder v-Knoten in $G_n^{(i,j)}$ ist, während $c = 1$ für w-Knoten gesetzt ist, sowie
4. dem Index $b \in \mathbb{B}^{2n+1}$ von s auf $p_{i,j}$ bzw. dem entsprechenden w-Teil.

Bit c integrieren wir in das Indexargument b. So codiert z. B. $(i)_2\,(j)_2\,0^{2n+1}$ den Knoten u_0 in $G_n^{(i,j)}$. Die unbenutzten isolierten Knoten stören unsere weiteren Betrachtungen nicht. Schließlich definieren wir, dass q die Knotennummer 0 hat.

Insgesamt ist die charakteristische Funktion χ_{G_n} mit

$$\chi_{G_n}(x,y,a) = 1 :\Leftrightarrow \big[(s_{|x|}, s_{|y|}) \in E_n\big] \wedge [\phi_n(|x|,|y|) = |a|]$$

also auf $9(2n+1)$ booleschen Variablen definiert, die durch $\pi^{\mathrm{id}}_{9,2n+1}$ geordnet seien. Wir referenzieren mit $i(x)$, $j(x)$, $c(x)$ und $b(x)$ die jeweiligen Komponenten einer binären Knotennummer $x \in \mathbb{B}^{3(2n+1)}$. Einem impliziten Algorithmus wird jedoch pro Knotennummer und Längenwert nur ein Argument mit $3(2n+1)$ Variablen übergeben. Die gemischte Ordnung der Teilargumente verbergen wir analog zu Abschnitt 3.8.2 in der lokalen Ordnung $\tau = \pi^{\mathrm{id}}_{3,2n+1}$ der dem Algorithmus präsentierten Variablenordnung $\pi^{\tau}_{3,3(2n+1)}$.

Die Eingabe χ_{G_n}

Lemma 5.3.1. *Die Funktion χ_{G_n} hat konstante Voll-OBDD-Breite bez. $\pi^{\mathrm{id}}_{9,2n+1}$.*

Beweis: Wir geben einen entsprechenden booleschen Ausdruck für χ_{G_n} an, der aus später zu definierenden Hilfsfunktionen $H_n^{(1)}, \ldots, H_n^{(6)}$ besteht, die jeweils verschiedene Typen von Kanten von G_n darstellen:

$$\chi_{G_n}(x,y,a) := \bigvee_{\ell=1}^{6} H_n^{(\ell)}(x,y,a).$$

$H_n^{(1)}$ stellt die zu q inzidenten Kanten dar:

$$H_n^{(1)}(x,y,a) := (|x| = 0) \wedge (|a| = 1) \\ \wedge \left(2^{n-1} < |i(y)|, |j(y)| < 2^n\right) \wedge (c(y) = |b(y)| = 0).$$

$H_n^{(2)}$ stellt Kanten $(u_\ell, u_{\ell+1})$ für $0 \leq \ell \leq 2^n - j - 2$ dar:

$$H_n^{(2)}(x,y,a) := \left(2^{n-1} < |i(x)|, |j(x)| < 2^n\right) \wedge (|a| = 1) \\ \wedge (i(x) = i(y)) \wedge (j(x) = j(y)) \wedge (c(x) = c(y) = 0) \\ \wedge (|b(y)| = |b(x)| + 1 \leq 2^n - j - 1)\,.$$

$H_n^{(3)}$ stellt Kanten (u_{2^n-j-1}, v_{2^n-j}) dar:

$$H_n^{(3)}(x,y,a) := \left(2^{n-1} < |i(x)|, |j(x)| < 2^n\right) \wedge (|a| = |i(x)| + |j(x)| - 2^n) \\ \wedge (i(x) = i(y)) \wedge (j(x) = j(y)) \wedge (c(x) = c(y) = 0) \\ \wedge (|b(x)| = 2^n - |j(x)| - 1) \wedge (|b(y)| = 2^n - |j(x)|)\,.$$

$H_n^{(4)}$ stellt Kanten $(v_\ell, v_{\ell+1})$ mit $2^n - j \leq \ell \leq 2^n - 2$ dar:

$$H_n^{(4)}(x,y,a) := \left(2^{n-1} < |i|, |j| < 2^n\right) \wedge (|a| = |i(x)|) \\ \wedge (i(x) = i(y)) \wedge (j(x) = j(y)) \wedge (c(x) = c(y) = 0) \wedge (|b(y)| = |b(x)| + 1) \\ \wedge (2^n - |j(x)| \leq |b(x)| \leq 2^n - 2)\,.$$

$H_n^{(5)}$ stellt Kurzschlusskanten dar, die in w-Knoten enden:

$$H_n^{(5)}(x,y,a) := \left(2^{n-1} < |i|, |j| < 2^n\right) \wedge (|a| = 1) \\ \wedge (i(x) = i(y)) \wedge (j(x) = j(y)) \wedge (c(x) = 0) \wedge (c(y) = 1) \wedge (|b(y)| = |b(x)| + 1) \\ \wedge (2^n - |j(x)| - 1 \leq |b(x)| \leq 2^n - 2)\,.$$

$H_n^{(6)}$ stellt Kurzschlusskanten dar, die in v-Knoten enden:

$$H_n^{(6)}(x,y,a) := \left(2^{n-1} < |i(x)|, |j(y)| < 2^n\right) \wedge (|a| = 1) \\ \wedge (i(x) = i(y)) \wedge (j(x) = j(y)) \wedge (c(x) = 1) \wedge (c(y) = 0) \wedge (|b(x)| = |b(y)|) \\ \wedge (2^n - |j(x)| \leq |b(x)| \leq 2^n - 1)\,.$$

Offenbar stellt χ_{G_n} den Graphen G_n korrekt dar. Wir haben die charakteristische Funktion durch einen Ausdruck konstanter Länge gemäß Theorem 3.7.4 dargestellt. Seine grundlegenden Bestandteile sind multivariate Vergleichsfunktionen mit konstantem maximalem absolutem Gewicht, die nach Korollar 3.7.5 konstante Voll-OBDD-Breite bez. $\pi^{\mathrm{id}}_{9,2n+1}$ haben. Insgesamt folgt also auch für χ_{G_n} konstante Voll-OBDD-Breite bez. $\pi^{\mathrm{id}}_{9,2n+1}$. □

Die Ausgabe χ_{Δ_n}

Kürzeste Pfade in $G_n^{(i,j)}$ benutzen Kurzschlusskanten, um den Besuch teurer Kanten der Länge i oder $i+j-2^n>1$ zu vermeiden, und einfaches Ausrechnen ergibt:

$$\begin{aligned}
\Delta_n(t_\ell, t_{\ell'}) &= \max\{(2^n-j-1)-\ell, 0\} - \max\{(2^n-j-1)-\ell', 0\} \\
&\quad +2\cdot\Big[\max\{\ell'-(2^n-j-1), 0\} - \max\{\ell-(2^n-j-1), 0\}\Big] \\
&\leq 2\cdot(\ell'-\ell), \\
\Delta_n(t_\ell, w_{\ell'}) &= \Delta_n(t_\ell, t_{\ell'-1})+1, \\
\Delta_n(w_\ell, t_{\ell'}) &= \Delta_n(t_\ell, t_{\ell'})+1, \\
\Delta_n(w_\ell, w_{\ell'}) &= \Delta_n(t_\ell, t_{\ell'-1})+2
\end{aligned}$$

sowie $\Delta_n(q,s)=\Delta_n(u_0,s)+1$ für $t\in\{u,v\}$, $0\leq\ell\leq\ell'\leq 2^n-1$, und $s\in V_n^{(i,j)}\setminus\{q\}$.

Wir betrachten eine Subfunktion $\chi_{\Delta'_n}\in B_{9(2n+1)}$ der Ausgabefunktion χ_{Δ_n} mit

$$\chi_{\Delta'_n}(x,y,a)=1 :\Leftrightarrow (s_{|x|}, s_{|y|}\notin\{q,w_\ell\}_\ell)\wedge\Delta_n\left(s_{|x|}, s_{|y|}\right)=|a|.$$

Die Fälle für q- und w-Knoten sind laut obiger Formeln trivial, tragen leicht ersichtlich nur einen konstanten Faktor zur OBDD-Breite bei und werden daher hier nicht formal erfasst.

Lemma 5.3.2. *Die Funktion $\chi_{\Delta'_n}$ hat konstante Voll-OBDD-Breite bez. $\pi^{\mathrm{id}}_{9,2n+1}$.*

Beweis: Wir geben einen entsprechenden booleschen Ausdruck für $\chi_{\Delta'_n}$ an, der wieder aus später zu definierenden Hilfsfunktionen $P_n^{(1)}, P^{(2)}, P_n^{(3)}$ besteht, die jeweils verschiedene Typen von Pfaden in G_n darstellen.

$$\chi_{\Delta'_n}(x,y,a) := \bigvee_{\ell=1}^{3} P_n^{(\ell)}(x,y,a).$$

$P_n^{(1)}$ deckt den Fall ab, dass sowohl Start- als auch Endknoten u-Knoten sind:

$$\begin{aligned}
P_n^{(1)}(x,y,a) &:= \left(2^{n-1}<|i|,|j|<2^n\right)\wedge(i(x)=i(y))\wedge(j(x)=j(y)) \\
&\quad\wedge(c(x)=c(y)=0)\wedge(|b(x)|<|b(y)|\leq 2^n-|j(x)|-1)\wedge(|a|=|b(y)|-|b(x)|).
\end{aligned}$$

$P_n^{(2)}$ deckt den Fall an, dass sowohl Start- als auch Endknoten v-Knoten sind:

$$\begin{aligned}
&P_n^{(2)}(x,y,a) := \left(2^{n-1}<|i|,|j|<2^n\right)\wedge(i(x)=i(y))\wedge(j(x)=j(y)) \\
&\wedge(c(x)=c(y)=0)\wedge(2^n-|j(x)|\leq|b(x)|<|b(y)|<2^n)\wedge\big[|a|=2\cdot(|b(y)|-|b(x)|)\big].
\end{aligned}$$

$P_n^{(3)}$ deckt den Fall ab, dass der Startknoten ein u-Knoten und der Endknoten ein v-Knoten ist:

$$P_n^{(3)}(x,y,a) := \big(2^{n-1} < |i|, |j| < 2^n\big) \wedge (i(x) = i(y)) \wedge (j(x) = j(y))$$
$$\wedge\, (c(x) = c(y) = 0) \wedge (|b(x)| \leq 2^n - |j(x)| - 1) \wedge (2^n - |j(x)| \leq |b(y)| < 2^n)$$
$$\wedge\, \big[|a| = (2^n - |j(x)| - 1 - |b(x)|) + 2 \cdot (|b(y)| - 2^n + |j(x)| + 1)\big].$$

Wir argumentieren nun analog zum Beweis von Lemma 5.3.1. □

Anmerkung 5.3.3. *Sowohl Eingabe χ_{G_n} als auch Ausgabe χ_{Δ_n} haben also konstante Voll-OBDD-Breite bez. $\pi^{\mathrm{id}}_{9,2n+1}$. Entgegen unserer abgeschwächten Voraussetzung in Anmerkung 5.1.4 liest $\pi^{\tau}_{3,3(2n+1)}$ die Längenargumente nicht mit aufsteigender Bitsignifikanz, sondern wiederum bez. einer lokalen gemischten Ordnung $\pi^{\mathrm{id}}_{3,2n+1}$. Zwei der drei Längenargumente der Länge $2n+1$ haben wir jedoch nur aus formalen Gründen eingeführt. Wie schon in Anmerkung 5.2.6 bemerkt würde man bei einer tatsächlichen Implementation erlauben, dass Knotennummern und Längenwerte durch verschiedene Anzahlen von Variablen codiert werden, und für Längenwerte nur durch gemäß* id *gelesene Argumente verwenden. So stellen wir fest, dass Algorithmus 5.1.2 auf $\pi^{\mathrm{id}}_{9,2n+1}$-OBDD$[\chi_{G_n}]$ Laufzeit $\mathcal{O}(n^3 \log n)$ auf Platz $\mathcal{O}(n)$ hat.*

5.3.2 Kürzeste Pfade mit höchstens 2^n Kanten

Das Zwischenergebnis $S_n^{(n)}$, das Algorithmus 5.1.1 auf G_n im n-ten Schleifendurchlauf berechnet, repräsentiert nicht den kürzesten q-v_{2^n-1}-Pfad in $G_n^{(i,j)}$ (der Kurzschlusskanten benutzt), weil dieser aus mehr als 2^n Kanten besteht. Somit muss $S_n^{(n)}$ den direkten Pfad $p_{i,j}$ der Länge $(2^n - j) + (i + j - 2^n) + (j-1)i = i \cdot j$ darstellen. Wir werden zeigen, dass $S_n^{(n)}$ somit inhärent den Graphen der Multiplikation enthält und die OBDD-Größe exponentiell groß in der Anzahl $9(2n+1) = \Theta(\log(|V_n| \cdot \max_e\{\phi(e)\}))$ boolescher Variablen ist. So folgt schließlich, dass Algorithmus 5.1.1 nicht die gleichen vorteilhaften Festparametereigenschaften hat wie Algorithmus 5.1.2, und wir werden erstmalig Theorem 4.3.1 verwenden.

Theorem 5.3.4. *Die Funktion $S_n^{(n)}$ hat exponentielle π-OBDD-Größe in n für jede Variablenordnung π auf $9(2n+1)$ Variablen.*

Beweis: Wir nehmen o. B. d. A. $n = 3m + 1$ für $m \in \mathbb{N}$ an. Sei nun P ein π-OBDD für $S_n^{(n)}$ für eine Variablenordnung π auf $9(2n+1)$ Variablen. Wir konstruieren aus P ein π'-stereotypes BP P' der Größe $\mathcal{O}(m \cdot \mathrm{size}(P))$ für MUL_m-$GRAPH$, dessen Variablensequenz π' jede Variable höchstens zweimal enthält. Aus Theorem 4.3.1 folgt so die exponentielle untere Schranke für $S_n^{(n)}$.

Um P' zu konstruieren, ersetzen wir zuerst x durch die binäre Nummer $(0)_2$ des Startknotens q. Weiterhin ersetzen wir $c(y)$ durch 0 und $b(y)$ durch $(2^n - 1)_2$. Der Endknoten $s_{|y|}$ ist also v_{2^n-1} auf $p_{i,j}$. Schließlich ersetzen wir $i(y)_{3m}$ und $j(y)_{2m}$ durch 1 und alle anderen Variablen $i(y)_\ell, j(y)_\ell$ mit $\ell \geq m$ durch 0. Wir bezeichnen die erhaltene Funktion mit f_n, die noch nicht ersetzten m i-Variablen mit $x \in \mathbb{B}^m$ sowie die übrigen m j-Variablen mit $y \in \mathbb{B}^m$.

Da von $S_n^{(n)}$ repräsentierte Pfade höchstens 2^n Kanten enthalten, dürfen keine Kurzschlusskanten auf dargestellten q-v_{2^n-1}-Pfaden in $G_n^{(i,j)}$ benutzt werden. Somit gilt $f_n(x,y,a)=1$ genau dann, wenn

$$|a| = i \cdot j = \big(|x| + 2^{3m}\big) \cdot \big(|y| + 2^{2m}\big) = |x| \cdot |y| + |x| \cdot 2^{2m} + |y| \cdot 2^{3m} + 2^{5m}$$

und $|x|, |y| > 0$ gelten. Dann entsprechen die $2m$ Variablen $a_{2m-1}, \ldots, a_0$ dem Produkt $(|x| \cdot |y|) =: r$. Außerdem entsprechen $a_{3m-1}, \ldots, a_{2m}$ dem Faktor x und $a_{4m-1}, \ldots, a_{3m}$ dem Faktor y. Abgesehen von $a_{5m} = 1$ sind alle anderen a-Variablen 0.

Wir zwingen nun a-Bits, die nicht zu r gehören, dazu, in dieses Schema für erfüllende Belegungen zu passen: Wir ersetzen a_{5m} durch 1 und ändern die Variablenmarkierungen a_{2m+k} in x_k bzw. a_{3m+k} in y_k um für $0 \leq k \leq m-1$. Alle anderen nicht zu r gehörenden a-Variablen werden durch 0 ersetzt.

Die übrigen $2m$ a-Variablen bezeichnen wir nun mit $z \in \mathbb{B}^{2m}$. Wir haben eine eingeschränkte Version MUL_m^*-$GRAPH$ von MUL_m-$GRAPH$ erhalten mit

$$MUL_m^*\text{-}GRAPH(x,y,z) = 1 \Leftrightarrow (|x| \cdot |y| = |z|) \wedge (|x|, |y| > 0),$$

die einfach durch

$$MUL_m^*\text{-}GRAPH(x,y,z) \vee \Big[\big((|x| = 0) \vee (|y| = 0)\big) \wedge (|z| = 0)\Big] \tag{5.3.1}$$

zu MUL_m-$GRAPH$ erweitert werden kann.

Sei P' das resultierende BP. Nach der Änderung der Variablenmarkierungen wird jede Variable höchstens zweimal gelesen. Konstante Variablenersetzungen erhöhen die BP-Größe nicht. Vergleiche von Argumenten mit 0 haben konstante Voll-OBDD-Breite für jede Variablenordnung. Also verursacht die Disjunktion mit MUL_m^*-$GRAPH(x,y,z)$ in Gleichung (5.3.1) höchstens ein Größenwachstum um einen Faktor von $\mathcal{O}(m)$, und P' hat Größe $\mathcal{O}(m \cdot \text{size}(P))$. □

Beobachtung 5.3.5. *Algorithmus 5.1.1 kann so modifiziert werden, dass die Herstellung der Pfadlängenminimalität in Zeile 5 erst nach den Schleifendurchläufen geschieht. Dies ändert jedoch nichts an den Ergebnissen dieses Abschnitts, da es nur einen q-v_{2^n-1}-Pfad in jedem $G_n^{(i,j)}$ gibt, der nicht mehr als 2^n Kanten enthält.*

5.4 Klassische SSSP-Ansätze

Bisher haben wir in diesem Kapitel Algorithmen betrachtet, die durch iteratives Quadrieren imstande waren, Kürzeste-Wege-Probleme durch nur wenige funktionale Operationen zu lösen. Aus der Praxis wissen wir, dass solche Ansätze typischerweise zu sehr großen OBDDs führen (siehe z. B. Bloem u. a. (2000) und Hojati u. a. (1993)). Wir konnten nur unter speziellen Einschränkungen effizientes Laufzeitverhalten nachweisen.

Daher fragen wir uns nun, wie sich wohl implizite Äquivalente zu klassischen Algorithmen für das **SSSP**-Problem im Durchschnitt verhalten. Wir betrachten dabei die Ansätze von Dijkstra und Bellman-Ford. Da sie ohne iteratives Quadrieren funktionieren, benötigen sie $\Omega(|V|)$ funktionale Operationen. Dennoch erhoffen wir uns von ihnen, dass sie auf zufälligen Graphen im Durchschnitt kompaktere OBDDs erzeugen, die hoffentlich zu relativ effizientem, wenn auch nicht polylogarithmischem Verhalten führen. Letzteres wäre in Bezug auf die Graphgröße auch gar nicht möglich, da OBDDs zufälliger Graphen bereits erwartete Größe $\Omega(|V|)$ besitzen (siehe Beobachtung 2.5.2).

Wir werden also sowohl einen Dijkstra-inspirierten als auch einen Bellman-Ford-inspirierten impliziten Algorithmus vorstellen und auf verschiedene Arten von (größtenteils randomisiert erzeugten) Graphen anwenden. Aus diesen experimentellen Ergebnissen möchten wir dann Vermutungen über das erwartete asymptotische Verhalten bez. der Eingabe-OBDD-Größe ableiten und bestätigen. Obwohl bereits implizite Algorithmen für Kürzeste-Wege-Probleme vorgestellt wurden (siehe z. B. Bahar u. a. (1994)), so basieren diese jedoch auf ADDs statt auf OBDDs. Es handelt sich in diesem Abschnitt also um die ersten OBDD-basierten impliziten Umsetzungen der Algorithmen von Dijkstra und Bellman-Ford und um die ersten experimentellen Untersuchungen, die Laufzeit und Platzbedarf auch in Relation zur Eingabe-OBDD-Größe betrachten.

5.4.1 Ein impliziter Dijkstra-Algorithmus

Dijkstras Algorithmus verwaltet eine Menge $A \subseteq V$ der Knoten v, für die ein kürzester q-v-Pfad bereits bekannt ist (siehe Dijkstra (1959)). Anfangs sind $A = \{q\}$, $\Delta(q) = 0$ und $\Delta(v) = \infty$ für alle Knoten $v \neq q$. In jeder Iteration des Algorithmus fügen wir einen Knoten zu A hinzu. Sei u der letzte Knoten, der zu A hinzugefügt wurde. Für jede Kante (u, v) wird nun geprüft, ob $\Delta(u) + \phi(u, v) < \Delta(v)$ gilt. Wenn ja, dann aktualisieren wir $\Delta(v)$ zu $\Delta(u) + \phi(u, v)$. Nach dieser Relaxation fügen wir den Knoten $v^{\min} \in V \setminus A$ mit minimalem Wert $\Delta(v^{\min})$ in A ein. Wenn jedoch nach der Relaxation $\{v \in V \setminus A \mid \Delta(v) \neq \infty\} = \emptyset$ gilt, ist die Berechnung von Δ abgeschlossen und der Algorithmus terminiert. Werden zur Knotenverwaltung Prioritätsheaps mit Zugriffszeit $\mathcal{O}(\log |V|)$ verwandt, so hat dieser Algorithmus Laufzeit $\mathcal{O}((|V| + |E|) \cdot \log |V|)$.

Wir wandeln diesen Ansatz nun in einen impliziten Algorithmus um, der mit entsprechenden Funktionen $A(x)$ und $\chi_\Delta(x, a)$ arbeitet. Beide werden anfangs initialisiert:

$$\begin{aligned} A(x) &:= (x = \langle q \rangle_2), \\ \chi_\Delta(x, a) &:= (x = \langle q \rangle_2) \wedge (|a| = 0). \end{aligned}$$

Es seien im Folgenden $x^{\min} := \langle v^{\min} \rangle_2$ und $a^{\min} := (\Delta(v^{\min}))_2$ für den jeweils zuletzt zu A hinzugefügten Knoten $v^{\min}$. Anfangs sei $v^{\min} := q$.

Nun müssen alle $v^{\min}$ verlassenden Kanten relaxiert werden. Wir führen dazu drei Hilfsfunktionen ein, die wir zur Aktualisierung von $\chi_\Delta(x,a)$ benutzen werden: $RELAX(x,a)$ stellt Paare (x,a) dar, sodass $(v^{\min}, v_{|x|}) \in E$ und $\Delta(v^{\min}) + \phi(v^{\min}, v_{|x|}) = |a|$ gelten. $D_1(x,a)$ und $D_2(x,a)$ repräsentieren zwei Fälle für Knoten $v_{|x|} \notin A$:

1. $D_1(x,a) = 1$ gilt genau dann, wenn $|a|$ der relaxierte Distanzwert für $v_{|x|}$ ist, der nicht größer ist als die aktuelle Distanz $\Delta(v_{|x|})$.
2. $D_2(x,a) = 1$ gilt genau dann, wenn $|a|$ der aktuelle Distanzwert $\Delta(v_{|x|})$ ist, der nicht größer ist als die relaxierte Distanz von $v_{|x|}$.

Fall 1 entspricht also einer Aktualisierung von $\Delta(v_{|x|})$, während Fall 2 seiner Beibehaltung entspricht. Schließlich lassen wir $\chi_\Delta(x,a)$ unverändert für Knoten $v_{|x|} \in A$, wohingegen für Knoten $v_{|x|} \notin A$ mit $(v^{\min}, v_{|x|}) \in E$ mindestens einer der beiden Fälle 1 oder 2 zutrifft. Dies führt direkt zu folgender Formulierung:

$$\begin{aligned}
RELAX(x,a) &:= (\exists b)\ \big[\chi_G(x^{\min}, x, b) \wedge (|a| = |a^{\min}| + |b|\big], \\
D_1(x,a) &:= RELAX(x,a) \wedge \overline{(\exists b)\ \big[\chi_\Delta(x,b) \wedge (|b| < |a|)\big]}, \\
D_2(x,a) &:= \chi_\Delta(x,a) \wedge \overline{(\exists b)\ \big[RELAX(x,b) \wedge (|b| < |a|)\big]}, \\
\chi_\Delta(x,a) &:= \big[A(x) \wedge \chi_\Delta(x,a)\big] \vee \Big[\overline{A(x)} \wedge [D_1(x,a) \vee D_2(x,a)]\Big].
\end{aligned}$$

Dann wählen wir den neuen minimalen Knoten $v^{\min}$ – gibt es mehrere solche, entscheiden wir uns für den mit der kleinsten Knotennummer. Dazu führen wir die Vergleichsfunktion *LESS* ein:

$$LESS(x,a,y,b) := (|a| < |b|) \vee \big[(a = b) \wedge (|x| < |y|)\big].$$

Aus Theorem 3.7.4 und Korollar 3.7.5 folgt direkt, dass *LESS* konstante Voll-OBDD-Breite bez. $\pi^{\mathrm{id}}_{k,n}$ für die konstante Anzahl an Argumentvektoren k hat und das entsprechende OBDD effizient erzeugt werden kann. Wir kommen zur Selektionsfunktion $SEL(x,a)$:

$$SEL(x,a) := \overline{A(x)} \wedge \chi_\Delta(x,a) \wedge \overline{(\exists y,b)\ \Big[\overline{A(y)} \wedge \chi_\Delta(y,b) \wedge LESS(y,b,x,a)\Big]}.$$

Wir folgen also dem Disjkstra-Prinzip und wählen das minimale Knotennummer-Distanz-Paar im aktuellen χ_Δ, dessen Knoten nicht in A enthalten ist. Ist *SEL* die Nullfunktion, so wurden alle von q erreichbaren Knoten in A eingefügt und wir terminieren mit Ausgabe χ_Δ. Ansonsten enthält *SEL* genau eine erfüllende Belegung für x und a, die in linearer Zeit bez. der OBDD-Größe extrahiert werden kann (siehe Theorem 2.3.17, Punkt 4). Schließlich fügen wir $x^{\min}$ in $A(x)$ ein:

$$(x^{\min}, a^{\min}) := SEL^{-1}(1),$$

$$A(x) \quad := \quad A(x) \vee (x = x^{\min}).$$

Wir verzweigen wieder zum Relaxationsschritt. Die Korrektheit dieses impliziten Algorithmus folgt aus der des Dijkstra-Algorithmus. Wir betrachten die Anzahl ausgeführter funktionaler Operationen:

Theorem 5.4.1. *Der implizite Dijkstra-Algorithmus führt bei der Berechnung von* χ_Δ $\mathcal{O}(|V| \log(|V| \cdot B))$ *funktionale Operationen durch.*

Beweis: Es werden $\mathcal{O}(|V|)$ Knoten in A eingefügt und somit auch $\mathcal{O}(|V|)$ Relaxations- und Selektionsschritte ausgeführt. In jeder dieser Phasen führt der Algorithmus eine konstante Anzahl an Quantifizierungsblöcken aus, die die Operationsanzahl mit $\mathcal{O}(n) = \mathcal{O}(\log(|V| \cdot B))$ dominieren. □

Für die Gesamtlaufzeit einer OBDD-basierten Implementation ist natürlich wieder die Größe der einzelnen OBDDs entscheidend. Wir bemerken, dass der vorgestellte Algorithmus ähnlich sequenziell wie der ursprüngliche Dijkstra-Algorithmus arbeitet. Eine Parallelisierung hat nur insofern stattgefunden, als dass einzelne zu $v^{\min}$ adjazente Knoten nicht mehr einzeln betrachtet werden. Es ist jedoch auch leicht realisierbar, durch $\mathcal{O}(n)$ funktionale Operationen eine maximale Anzahl von bez. der Relaxation voneinander unabhängigen Knoten in A einzufügen. So kann in vielen Fällen die Operationsanzahl verringert werden. Dieser Ansatz wird im Folgenden jedoch nicht näher dargestellt – in Experimenten hat sich gezeigt, dass die damit verbundenen komplizierteren booleschen Formulierungen oft größere OBDDs mit sich bringen, die durch eine geringere Operationsanzahl nur selten kompensiert werden.

Die maximale Anzahl von Argumenten mit jeweils $n = \log(|V| \cdot B) + \mathcal{O}(1)$ Variablen, auf denen Funktionen während des impliziten Dijkstra-Algorithmus definiert sind, ist 4. Theorem 2.3.9, Theorem 2.3.17 (Punkt 9) und Theorem 5.4.1 implizieren nun zusammen:

Korollar 5.4.2. *Die Laufzeit des impliziten Dijkstra-Algorithmus ist durch* $\mathcal{O}(|V|^9 \cdot B^8 / \log(|V| \cdot B))$ *beschränkt. Sein Platz ist durch* $\mathcal{O}(|V|^8 \cdot B^8 / \log^2(|V| \cdot B))$ *beschränkt.*

Diese Schranken werden von den binären Synthesen und Quantifizierungen dominiert, die dabei als quadratisch in der Worst-case-OBDD-Größe angenommen werden.

5.4.2 Ein impliziter Bellman-Ford-Algorithmus

Im Gegensatz zu Dijkstras Algorithmus wählt der Bellman-Ford-Algorithmus nicht spezielle Kanten zur Relaxation, sondern führt in seiner einfachsten Form $|V|$ Iterationen über alle Kanten in $(u, v) \in E$ aus, um die Bedingung $\Delta(u) + \phi(u, v) < \Delta(v)$ zu prüfen und gegebenenfalls $\Delta(v)$ zu aktualisieren (siehe Bellman (1958)). Somit ist

seine Laufzeit durch $\mathcal{O}(|V| \cdot |E|)$ beschränkt. Zudem kann dieser Ansatz auch mit negativen Kantenlängen umgehen, wenn die Eingabe keine negativen Kreise enthält. Die Entwicklung eines entsprechenden impliziten Algorithmus wurde durch die gute Parallelisierbarkeit dieses Ansatzes motiviert: Wenige funktionale Operationen sollen möglichst viele Kanten gleichzeitig relaxieren.

Wieder initialisieren wir χ_Δ für den Startknoten q:

$$\chi_\Delta(x, a) := (x = \langle q \rangle_2) \wedge (|a| = 0).$$

Wir verwenden wieder eine Funktion $RELAX(x, y, a)$, die die Kandidaten für eine mögliche Kantenrelaxation repräsentiert. Es sei genau dann $RELAX(x, y, a) = 1$, wenn $\Delta(v_{|x|}) + \phi(v_{|x|}, v_{|y|}) = |a|$ gilt und $|a|$ kleiner als das aktuelle $\Delta(v_{|y|})$ ist:

$$RELAX(x, y, a) := (\exists b, c)\ \big[\chi_\Delta(x, b) \wedge \chi_G(x, y, c) \wedge (|a| = |b| + |c|)\big] \\ \wedge\ \overline{(\exists b)\ \big[\chi_\Delta(y, b) \wedge (|b| \leq |a|)\big]}.$$

Wenn $RELAX$ die Nullfunktion ist, sind keine weiteren Relaxationen möglich und χ_Δ stellt die korrekte Distanzfunktion dar. Ansonsten müssen wir für jeden Knoten $v_{|y|}$ die Relaxation (x, y, a) mit minimalem $|a|$ auswählen. Dies geschieht wieder mithilfe der Vergleichfunktion $LESS$:

$$SEL(x, y, a) := RELAX(x, y, a) \wedge \overline{(\exists z, b)\ \big[RELAX(z, y, b) \wedge LESS(z, b, x, a)\big]}.$$

Schließlich berechnen wir die Funktion $U(x, a)$ der Paare, die wir in $\chi_\Delta(x, a)$ aktualisieren müssen, weil sie Teil einer selektierten Relaxation sind:

$$\begin{aligned} U(x, a) &:= (\exists y)\ SEL(y, x, a), \\ \chi_\Delta(x, a) &:= U(x, a) \vee \Big[\overline{U(x, a)} \wedge \chi_\Delta(x, a)\Big]. \end{aligned}$$

So werden aktualisierte Distanzwerte in χ_Δ übernommen und sonstige beibehalten. Die neue Iteration beginnt mit einer erneuten Berechnung von $RELAX$. Die Korrektheit des Verfahrens folgt wieder aus der Korrektheit des klassischen Bellman-Ford-Algorithmus.

Theorem 5.4.3. *Der implizite Bellman-Ford-Algorithmus führt bei der Berechnung von χ_Δ $\mathcal{O}(|V| \cdot |E| \cdot \log(|V| \cdot B))$ funktionale Operationen durch.*

Beweis: Bellman-Ford-Implementationen führen nicht mehr als $\mathcal{O}(|V| \cdot |E|)$ Relaxationen durch. In jeder Iteration des impliziten Bellman-Ford-Algorithmus wird mindestens eine Relaxation ausgewählt und übernommen. Dies verursacht eine konstante Anzahl an Quantifizierungsblöcken, die die Operationsanzahl mit $\mathcal{O}(n) = \mathcal{O}(\log(|V| \cdot B))$ dominieren. □

Tatsächlich gehen wir bei der Anwendung dieses Verfahrens natürlich davon aus, dass typischerweise wesentlich mehr als nur eine Relaxation in jeder Iteration ausgewählt werden und die Anzahl der funktionalen Operationen oft sublinear in $|V|$ ist. Dennoch sei hier erneut bemerkt, dass wir auch mit diesem Ansatz keine sublineare Operationsanzahl bez. des Graphdurchmessers D erreichen können: Das Beispiel eines einfachen Pfades macht deutlich, dass wir trotz paralleler Relaxationen stets mindestens $\Omega(D)$ Iterationen benötigen, um kürzeste Pfade von q zu allen anderen Knoten zu propagieren.

Die maximale Anzahl von Argumenten mit jeweils $n = \log(|V| \cdot B) + \mathcal{O}(1)$ Variablen, auf denen Funktionen während des impliziten Bellman-Ford-Algorithmus definiert sind, ist 5. Theorem 2.3.9, Theorem 2.3.17 (Punkt 9) und Theorem 5.4.3 implizieren nun zusammen:

Korollar 5.4.4. *Die Laufzeit des impliziten Bellman-Ford-Algorithmus ist durch* $\mathcal{O}(|V|^{11} \cdot |E| \cdot B^{10} / \log(|V| \cdot B))$ *beschränkt. Sein Platz ist durch* $\mathcal{O}(|V|^{10} \cdot B^{10} / \log^2(|V| \cdot B))$ *beschränkt.*

Diese Schranken werden von den binären Synthesen und Quantifizierungen dominiert, die dabei als quadratisch in der Worst-case-OBDD-Größe angenommen werden.

5.4.3 Berechnung konkreter Pfadkanten

Außer χ_Δ berechnen klassische **SSSP**-Algorithmen für jeden Knoten $v \in V$ einen Vorgängerknoten $\text{pred}(v) =: u$, sodass es einen kürzesten q-v-Pfad gibt, der die Kante (u, v) enthält. Wir können diese Vorgänger auch implizit berechnen. Dazu sei $P(x, y, a, b, c)$ genau dann erfüllt, wenn $\Delta(v_{|x|}) = |a|$, $\phi(v_{|x|}, v_{|y|}) = |b|$, $\Delta(v_{|y|}) = |c|$ und $|a| + |b| = |c|$ gelten:

$$\begin{aligned} P(x, y, a, b, c) &:= \chi_\Delta(x, a) \wedge \chi_G(x, y, b) \wedge \chi_\Delta(y, c) \wedge (|a| + |b| = |c|), \\ \chi_{\text{pred}}(x, y) &:= (\exists a, b, c)\ P(x, y, a, b, c). \end{aligned}$$

Durch Existenzquantifizierung über a, b und c erhalten wir so die Funktion $\chi_{\text{pred}}(x, y)$, die genau dann erfüllt ist, wenn $v_{|x|}$ ein möglicher Vorgänger von $v_{|y|}$ auf einem kürzesten q-$v_{|y|}$-Pfad in G ist.

Sind wir lediglich an einem beliebigen Vorgänger eines konkreten Knotens v interessiert, so können die Quantifizierungen über die Längenargumente vermieden werden: Wir ersetzen Argument y in P durch $\langle v \rangle_2$ und extrahieren eine erfüllende Belegung für P. Die Berechnung von χ_Δ spielt also die zentrale Rolle bei der Lösung des impliziten **SSSP**-Problems, die wir nun auch experimentell untersuchen werden.

5.4.4 Experimentelle Untersuchungen

Beide implizite Algorithmen wurden in C++ mithilfe des OBDD-Pakets CUDD von F. Somenzi implementiert. Die Implementation macht Gebrauch von komplementier-

ten Kanten. So wurden beide Algorithmen auf verschiedenen (größtenteils randomisiert erzeugten) Graphen angewandt, auf die wir im Folgenden eingehen. Die Experimente fanden auf einem PC mit Pentium-4-Prozessor (2 GHz) und 512 MB Hauptspeicher statt. Die Variablenordnung war weiterhin $\pi^{\mathrm{id}}_{k,n}$ für die jeweilige konstante Anzahl k der Argumente und die jeweilige Argumentlänge von n Variablen; so hatten Hilfsfunktionen, die durch Komposition von multivariaten Thresholdfunktionen konstanten maximalen absoluten Gewichts gewonnen werden können, konstante Voll-OBDD-Breite. Laufzeit wurde in Prozessorzeitsekunden gemessen, Speicherplatz in der maximalen Anzahl existierender OBDD-Knoten im SBDD.

Die Experimente wurden bis auf eine Ausnahme auf gerichteten Zufallsgraphen mit den Knotenanzahlen 100, 200, 300 und 400 und Kantenwahrscheinlichkeiten $Z/20$ für $Z := 1, \ldots, 20$ durchgeführt. Knoten 0 diente stets als Startknoten q. Die dokumentierten Ergebniswerte sind jeweils der Durchschnitt der Werte von 10 Experimenten pro Parameterkombination. Da diese jeweiligen 10 Werte kaum vom Durchschnitt abweichen, stellen wir in den folgenden Abbildungen nur diesen Durchschnitt dar. Wir interessieren uns wie bereits erwähnt für die tatsächliche Nutzbarkeit der Algorithmen im durchschnittlichen Anwendungsfall, und so betrachten wir die beobachteten Daten in Abhängigkeit von der Eingabe-OBDD-Größe.

Für eine jeweilige feste Knotenanzahl wurde bei allen Zufallsgraphen beobachtet, dass mit steigender Kantenwahrscheinlichkeit sowohl Eingabe-OBDD-Größe als auch Laufzeit und Speicherplatz monoton und nicht schneller als linear wachsen. Für unterschiedliche Knotenanzahlen gab es jedoch teilweise große Unterschiede. Dies motiviert die grafische Darstellung der Ergebnisse, bei der Messpunkte für Experimente mit gleicher Knotenanzahl verbunden und mit jeweils eigenem Symbol versehen sind.

Zufallsgraphen mit einheitlicher Kantenlänge

Zunächst betrachten wir den Fall einheitlicher Kantenlängen $\phi = 1$ (siehe Abbildung 5.4.1). Es fällt auf, dass der Dijkstra-Algorithmus ein wesentlich besseres Laufzeitverhalten als der Bellman-Ford-Algorithmus hat, während das Gegenteil für den Speicherplatz gilt: Bellman-Fords Laufzeit wächst linear in der Eingabe-OBDD-Größe mit gleicher Steigung für alle Knotenanzahlen. Bei Dijkstra hingegen unterscheiden sich die Steigungen für unterschiedliche Knotenanzahlen. Ebenso wächst Dijkstras maximale OBDD-Knotenanzahl linear in der Eingabe-OBDD-Größe mit einheitlicher Steigung, während dies nicht so eindeutig für Bellman-Ford zu beobachten ist.

Zufallsgraphen mit Differenz-Kantenlängen

Als Nächstes betrachten wir den Fall weniger trivialer Kantenlängen (siehe Abbildung 5.4.2). Dennoch soll eine von den Adjazenzen separate Beschreibung der Kantenlängen kompakte OBDDs haben und die Auswirkung dieser Veränderung auf das Verhalten der impliziten Algorithmen untersucht werden. Dazu wurde $\phi(v_i, v_j) =$

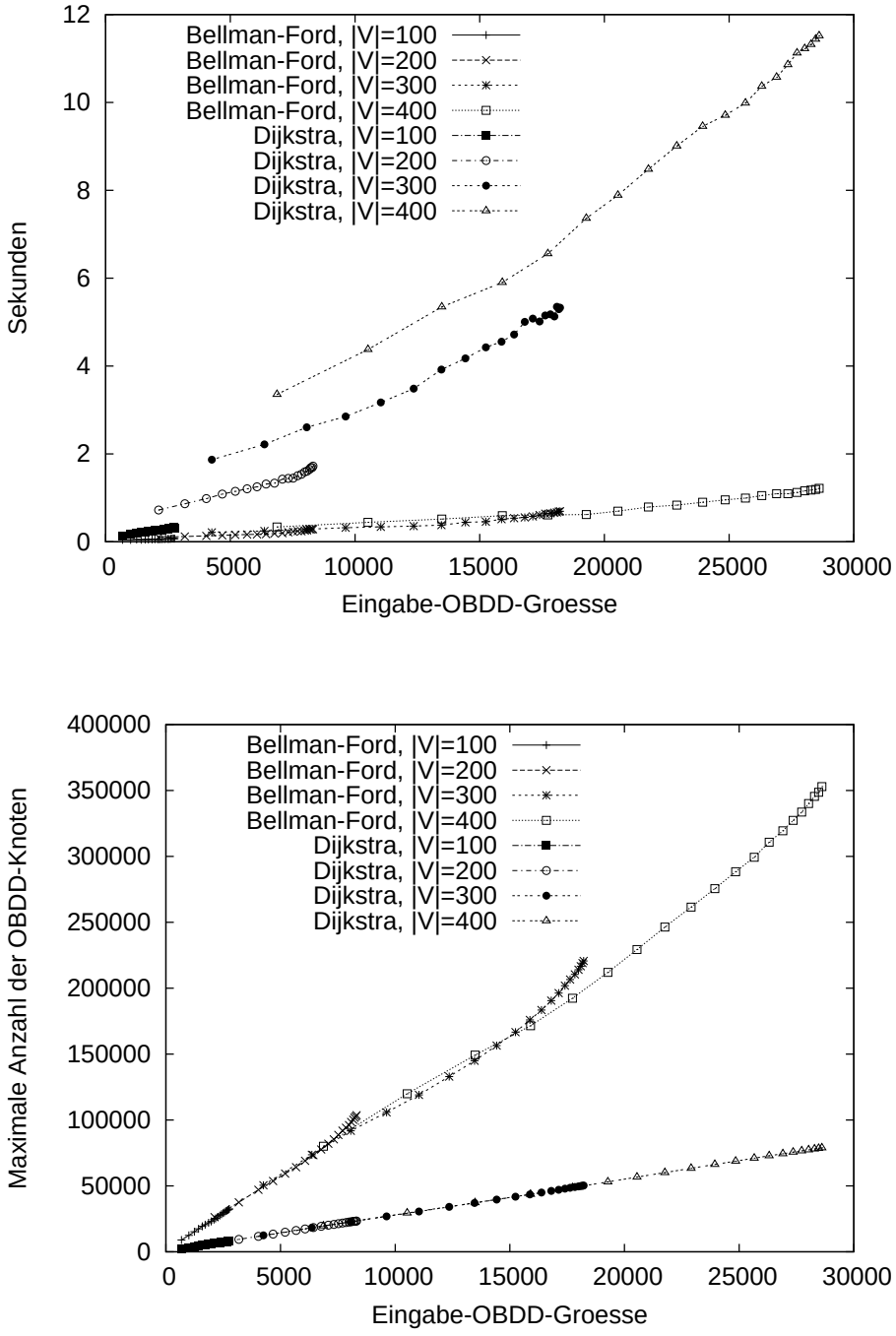


Abbildung 5.4.1: Experimentelle Ergebnisse auf Zufallsgraphen mit einheitlicher Kantenlänge 1.

$|i-j|$ mod 200 gewählt, sodass $\chi_G(x,y,a) = A(x,y) \wedge B(x,y,a)$ sich in einen zufälligen Anteil $A(x,y)$ und einen strukturierten Teil $B(x,y,a) = (|a| = |x| - |y| \bmod 200)$ zerlegen lässt. Die Eingabe-OBDD-Größe sollte also durch die nun nicht mehr einheitliche Kantenlängenfunktion nicht wesentlich vergrößert werden.

Dennoch ändert sich das Laufzeitverhalten des Bellman-Ford-Algorithmus, das nun nicht mehr linear mit gleicher Steigung für alle Knotenanzahlen ist. Und doch ist dieser Ansatz dem Dijkstra-Algorithmus überlegen. Bezüglich der maximalen OBDD-Knotenanzahl ist die Situation im Vergleich zu einheitlichen Kantenlängen asymptotisch gleich geblieben: Der Dijkstra-Algorithmus ist überlegen und hat linearen Platzbedarf in der Eingabe-OBDD-Größe.

Zufallsgraphen mit zufälligen Kantenlängen

Schließlich wurde der Fall gleichverteilt gezogener Kantenlängen in $\{0, 200\}$ betrachtet (siehe Abbildung 5.4.3). Diese weitergehende Einbeziehung des Zufalls verursacht erhöhte Eingabe-OBDD-Größen und Messwerte. Wesentlich ändert sich die Situation im Vergleich zu Differenz-Kantenlängen aber nicht: Wieder kann linearer Speicherplatz mit für alle Knotenanzahlen einheitlicher Steigung nur beim Dijkstra-Algorithmus beobachtet werden, während alle anderen Messreihen lineares Wachstum für feste Knotenanzahl haben. Jedoch zeigt Bellman-Ford hier kein überlegenes Laufzeitverhalten mehr – das parallele Vorgehen kann die höheren Zeitkosten pro OBDD-Operation nicht mehr wesentlich kompensieren.

Gittergraphen

Wenn wir polylogarithmische Laufzeitschranken auf Gittergraphen bez. $|V|$ für spezialisierte implizite Flussmaximierungs- und APSP-Algorithmen beweisen konnten, so sollten doch auch die impliziten Dijkstra- und Bellman-Ford-Algorithmen hier besonders effizient sein, sollen sie doch sogar für durchschnittliche Eingaben geeignet sein. Daher wurden zu guter Letzt Experimente auf $2^{\kappa/2} \times 2^{\kappa/2}$-Gittergraphen durchgeführt mit $\kappa \in \{2, 4, 6, 8, 10, 12, 14, 16\}$ und einheitlicher Kantenlänge 1 (siehe Abbildung 5.4.4). Aufgrund der sehr kleinen beobachteten Laufzeitwerte und ihrer somit schlechten Vergleichbarkeit wird hier lediglich die während der Algorithmen beobachtete maximale OBDD-Größe präsentiert. Wieder ist Dijkstra hier überlegen, während die Messwerte jedoch für beide Algorithmen linear in κ wachsen, wobei auch die Eingabe-OBDD-Größe hier laut Lemma 3.8.2 $\mathcal{O}(\kappa)$ ist. Die asymptotische Laufzeit enthält jedoch jeweils noch einen Faktor von $\Omega(2^{\kappa/2})$ aufgrund der allgemeinen unteren Schranke für die Anzahl der OBDD-Operationen beider Algorithmen – derartige Faktoren können wir nur mithilfe iterativen Quadrierens beseitigen.

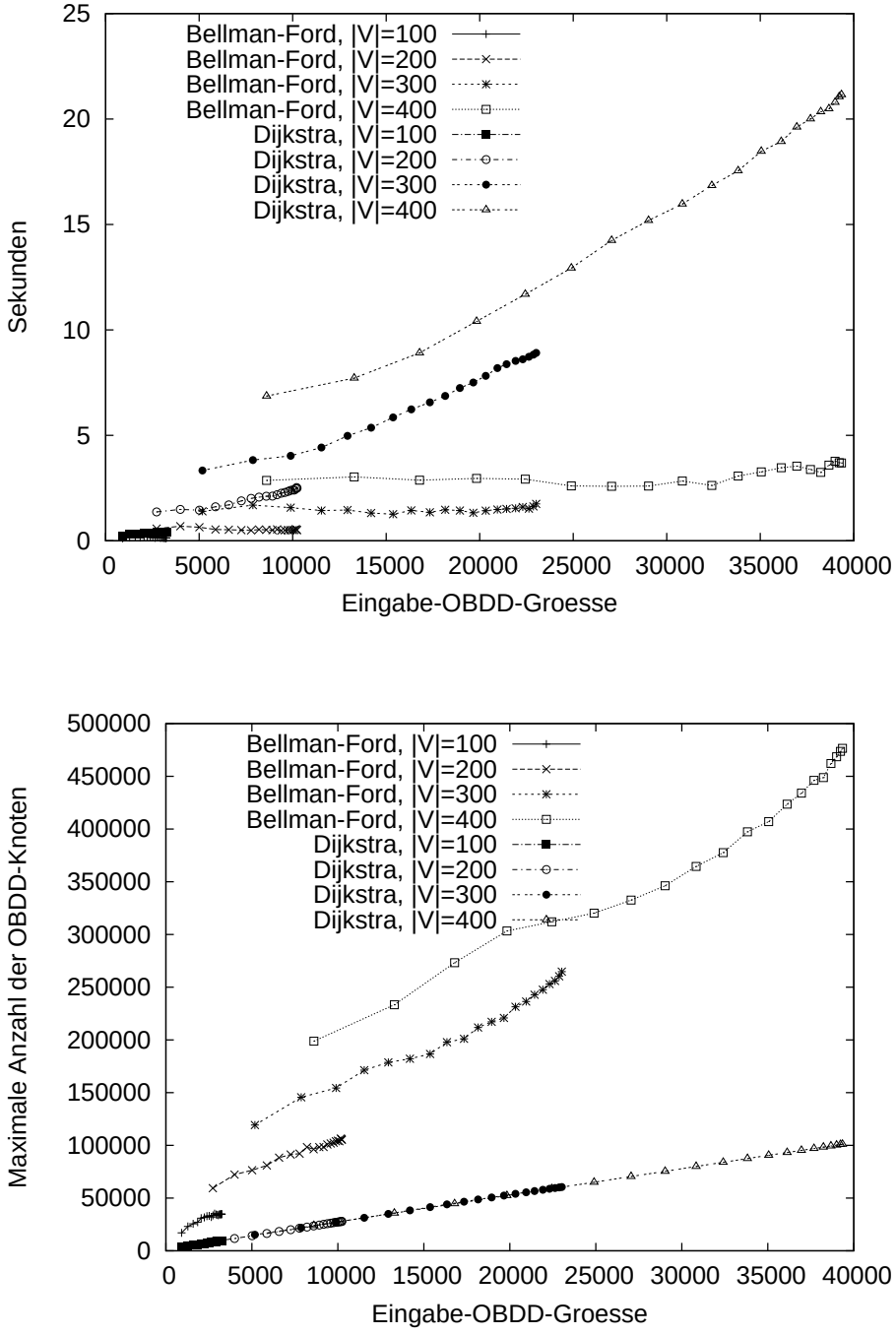
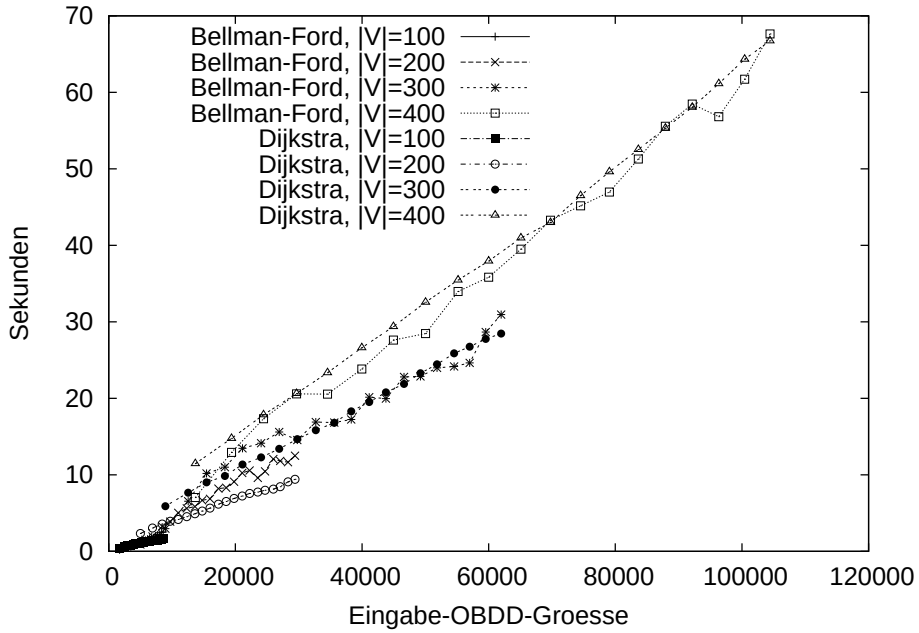


Abbildung 5.4.2: Experimentelle Ergebnisse auf Zufallsgraphen mit Differenz-Kantenlängen.



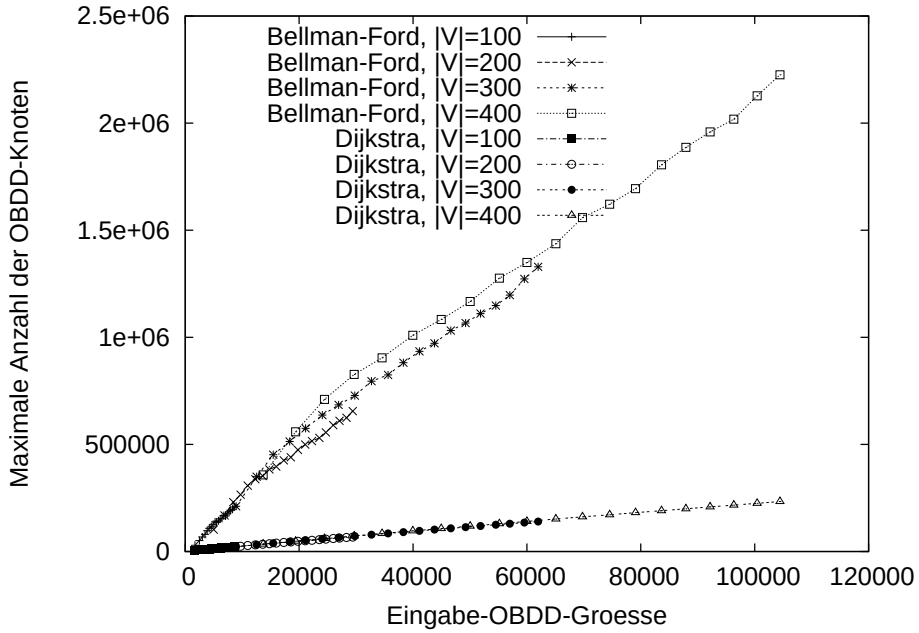


Abbildung 5.4.3: Experimentelle Ergebnisse auf Zufallsgraphen mit gleichverteilten Kantenlängen in $\{0, \ldots, 200\}$.

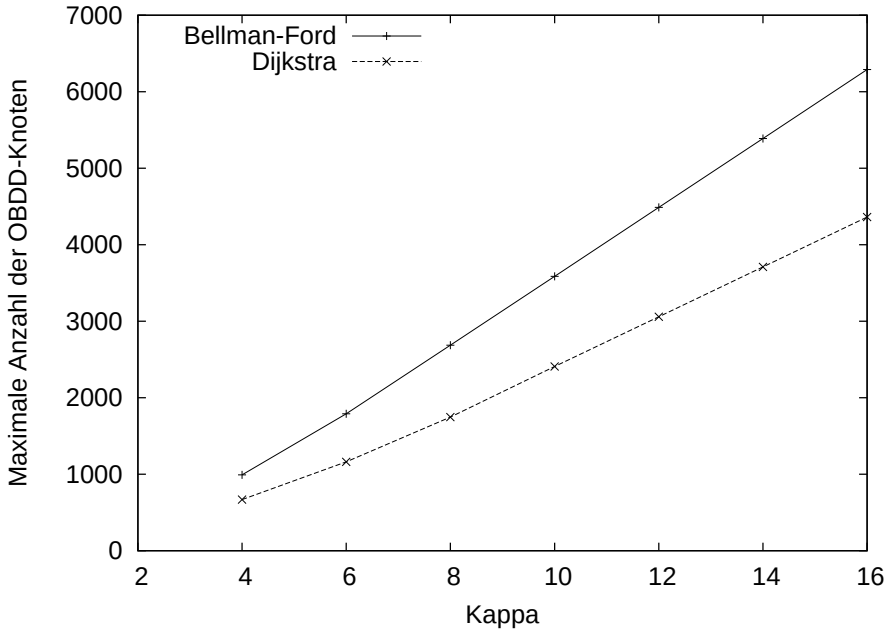


Abbildung 5.4.4: Experimentelle Ergebnisse auf $2^{\kappa/2} \times 2^{\kappa/2}$-Gittergraphen mit einheitlicher Kantenlänge 1.

5.4.5 Interpretation der experimentellen Ergebnisse

Der Dijkstra-Ansatz ist insgesamt überlegen im Platzbedarf, während der Bellman-Ford-Ansatz in allen Experimenten eine bessere Laufzeit aufweist: Eine mögliche Interpretation ist, dass das explorative Verhalten beim Dijkstra-Ansatz zu strukturierten Zwischenergebnissen und entsprechend kleinen OBDDs führt – dennoch müssen alle Knoten einzeln betrachtet werden. Der Bellman-Ford-Ansatz hingegen relaxiert Kanten parallel durch wenige OBDD-Operationen, was trotz unstrukturierteren Zwischenfunktionen und größeren OBDDs oft zu einer besseren Gesamtlaufzeit führt.

Die betrachteten Zufallsgraphen stellen Instanzen mit erwarteter OBDD-Größe $\Omega(|V|^{2-\varepsilon})$ dar (siehe Beobachtung 2.5.2). Sie sind also als durchschnittliche Eingaben zu betrachten, für die wir keine kompakten OBDDs erwarten können. Während der Bellman-Ford-Algorithmus nur im Falle einheitlicher Kantenlängen durchschnittliche lineare Laufzeit mit von $|V|$ unabhängiger Steigung aufweist, ist für den Dijkstra-Algorithmus in allen Fällen linearer Platzbedarf signifikant zu beobachten; die lineare Regressionsfunktion weicht stets weniger als 6% von den Messwerten ab. Dieser Ansatz stellt also eine algorithmische Fortsetzung der heuristischen Motivation zur Verwendung von OBDDs bei der Darstellung von großen Graphen dar: Auch die Ausführung des impliziten Dijkstra-Algorithmus vergrößert den Platzbedarf im

Durchschnitt nicht wesentlich. Insbesondere ist seine maximale OBDD-Größe somit auch wesentlich besser, als es Korollar 5.4.2 erwarten lässt:

Es werden Operationen auf OBDDs angewandt, die auf bis zu $4n$ Variablen definiert sind; laut Theorem 2.3.9 ist ihre OBDD-Größe durch $\mathcal{O}(2^{4n}/n)$ beschränkt. Für die Größe des während einer der bis zu $\mathcal{O}(|V|\cdot n)$ durchgeführten binären Synthesen erzeugten erreichbaren Produktgraphen zweier Operanden-OBDDs (siehe Theorem 2.3.17, Punkt 9) ergibt sich also die Schranke $\mathcal{O}(2^{8n}/n^2)$. Bezogen auf die Eingabe-OBDD-Größe $S = \mathcal{O}(2^{2n}/n)$ erhalten wir so die einzige bekannte allgemeine Platzschranke $\mathcal{O}(S^4 \cdot n^2) = \omega(S)$.

In den Fällen, in denen kein durchschnittliches lineares Wachstum mit einheitlicher Steigung für alle Knotenanzahlen beobachtet werden kann, gelingt dies jedoch wenigstens separat für jede einzelne Knotenanzahl. Für festes $|V|$ sorgen steigende Kantenwahrscheinlichkeiten für wachsende Eingabe-OBDD-Größen, in denen Zeit und Platz beider Algorithmen also stets linear wachsen.

Beide Algorithmen haben zudem lineare maximale OBDD-Größe auf Gittergraphen, die wiederum polylogarithmische Eingabe-OBDD-Größe bez. ihrer Knotenanzahl haben. Dies ist jedoch nicht überraschend und kann leicht mit den gleichen Methoden wie in Abschnitt 3.8 nachgewiesen werden.

Mit den vorgestellten Verfahren können wir also auf keinen Fall das dieser Dissertation zugrunde gelegte Ziel der Laufzeit $\text{polylog}(|V| \cdot B)$ erreichen; Experimente weisen jedoch darauf hin, dass die bewährten Ansätze von Dijkstra und Bellman-Ford auch im impliziten Szenario zu praktikablen Algorithmen führen, wenn keine strukturellen Eingabeeigenschaften bekannt sind.

6 Stundenpläne mit beschränkten Bearbeitungszeiten

Bei der algorithmischen Entwurfsmethode des Dynamischen Programmierens werden aus Lösungen für Teilprobleme eines Problems effizient Lösungen für größere Teilprobleme zusammengesetzt. Dies führt zu insgesamt effizienten Algorithmen, wenn es nur polynomiell viele verschiedene Teilprobleme gibt. Im Kontrast dazu würden rekursiv arbeitende Algorithmen unter Umständen Probleme mehrfach lösen und somit exponentiell schlechter sein. Typischerweise werden die Werte bisheriger Teillösungen in einer Tabelle verwaltet. Zeiger auf Tabelleneinträge, die in die Berechnung eines Eintrags eingeflossen sind, ermöglichen, im Nachhinein einen Pfad/Baum von Teilproblemen zu traversieren und eine konkrete optimale Lösung zu konstruieren. Ein bekanntes Beispiel aus der Bioinformatik ist der Smith-Waterman-Algorithmus, der eine optimale Ausrichtung zweier Zeichenketten A und B bez. eines Abstandsmaßes in Zeit und Platz $\mathcal{O}(\Lambda(A) \cdot \Lambda(B))$ ermittelt (Smith und Waterman, 1981).

Graphprobleme stehen im Fokus dieser Dissertation, jedoch lässt sich auch die Lösungswerttabelle eines Dynamische-Programmieren-Verfahrens als Graph auffassen: Knoten sind dann die einzelnen Felder der Tabelle und Kanten sind die Zeiger zwischen solchen Einträgen, die in Teillösungsbeziehung stehen. Zudem verläuft die kanonische implizite Darstellung der Abbildung der Feldpositionen auf ihre Werte durch charakteristische Funktionen analog zur impliziten Darstellung einer (gewichteten) Kantenmenge. Schließlich können entsprechende Tabellen konstanter Dimension trotzdem bereits inpraktikable Größenordnung besitzen, sodass eine Ausnutzung eventueller Regelmäßigkeiten durch OBDD-basierte Heuristiken wünschenswert ist.

So verlassen wir in diesem Kapitel vorübergehend die Welt der klassischen Graphprobleme und untersuchen eine implizite Umsetzung eines bekannten Dynamische-Programmieren-Ansatzes für ein spezielles Stundenplanproblem: N unabhängige Aufgaben mit natürlichen Bearbeitungszeiten sollen m identischen parallelen Maschinen zugeordnet werden, sodass die maximale Gesamtzeit, die eine Maschine benötigt, minimiert wird. Dieser Wert wird *Makespan* genannt. Es ist bekannt, dass dieses allgemeine Stundenplanproblem stark NP-hart ist (siehe z. B. Garey und Johnson (1979)). Wir betrachten daher den Fall, dass alle Aufgaben der Eingabe nur eine konstante Anzahl k verschiedener Bearbeitungszeiten besitzen. Weiterhin interessieren wir uns zunächst nur für das Entscheidungsproblem, ob ein Stundenplan existiert, dessen Makespan nicht größer als ein Eingabewert D ist. Der Einfachheit halber nennen wir dieses eingeschränkte Problem in dieser Dissertation trotzdem Stundenplanproblem. Leung (1982) stellt einen Dynamische-Programmieren-Ansatz vor, der eine k-dimensionale

Tabelle verwendet. So enthält auch die optimale Laufzeitschranke k im Exponenten und der Algorithmus wird, obwohl theoretisch polynomiell für konstantes k, in der Praxis bereits für kleine Werte k und große Aufgabenanzahlen inpraktikabel.

Wir definieren das betrachtete Problem zunächst formal und stellen Leungs Algorithmus vor. Dann präsentieren wir eine implizite Version des Verfahrens. Dabei ist die Hoffnung – ähnlich wie in Abschnitt 5.4 –, eine praktikablere Heuristik zu erhalten, die auf durchschnittlichen Probleminstanzen die Strukturiertheit der Lösungswerttabelle ausnutzt und so lediglich recht kompakte OBDDs manipuliert, statt jedenfalls sämtliche Einträge einer großen Matrix betrachten zu müssen. Um diese Hoffnung zu bestätigen, wurden Experimente auf zufällig erzeugten Eingaben mit üblichen Verteilungen durchgeführt. Tatsächlich schlagen wir Leungs Algorithmus für große Aufgabenanzahlen.

6.1 Problemdefinition und Leungs Algorithmus

Definition 6.1.1. *Ein* Stundenplanproblem P *besteht aus* k *Bearbeitungszeiten* $t_1, \ldots, t_k \in \mathbb{N}$, *entsprechenden* Anforderungen $N_1, \ldots, N_k \in \mathbb{N}$, *einer* Maschinenanzahl $m \in \mathbb{N}$ *sowie einer* Makespan-Schranke $D \in \mathbb{N}$. *Die* Gesamtanforderung *von* P *ist dann* $N := \sum_{i=1}^{k} N_i$.

Ein Stundenplan $S\colon \{1, \ldots, k\} \times \{1, \ldots, m\} \to \mathbb{N}$ *für* P *heißt genau dann* gültig, *wenn* $\sum_{j=1}^{m} S(i,j) \geq N_i$ *für jedes* $i \in \{1, \ldots, k\}$ *und andererseits* $\sum_{i=1}^{k} t_i \cdot S(i,j) \leq D$ *für jedes* $j \in \{1, \ldots, m\}$ *gelten.*

Ein Stundenplanalgorithmus *entscheidet, ob es einen gültigen Stundenplan* S *für* P *gibt.*

Sei P also ein Stundenplanproblem. Wir nehmen zunächst vereinfachend an, dass m eine Zweierpotenz ist. Leungs Algorithmus berechnet nun eine k-dimensionale Tabelle T mit Einträgen $T(\ell, i_1, \ldots, i_{k-1})$ für $\ell := 0, \ldots, \log m$, $i_j := 0, \ldots, N_j$ und $j := 1, \ldots, k-1$. Solch ein Eintrag entspricht der maximalen Anzahl I an Aufgaben des Typs k, die auf 2^ℓ Maschinen zusammen mit jeweils i_j Aufgaben des Typs j für alle Typen $j = 1, \ldots, k-1$ bearbeitet werden kann.

Wir definieren obere Schranken $B_j := \min\{N_j, \lfloor D/t_j \rfloor\}$ für $j := 1, \ldots, k$ für die maximale Anzahl der Aufgaben des Typs j, die einer Maschine zugewiesen werden können. Betrachten wir den Fall $\ell = 0$: Wenn $0 \leq i_j \leq B_j$ für $j := 1, \ldots, k-1$ und $D \geq \sum_{j=1}^{k-1} t_j \cdot i_j$ gelten, dann ist $I = \left\lfloor \left(D - \sum_{j=1}^{k-1} t_j \cdot i_j \right) / t_k \right\rfloor$; anderenfalls definieren wir $I := -1$.

Nachdem alle $\Pi_{j=1}^{k-1} (N_j + 1)$ Einträge $T(\ell, i_1, \ldots, i_{k-1})$ für eine Maschinenanzahl 2^ℓ bekannt sind, können die Werte für $2^{\ell+1}$ Maschinen folgendermaßen berechnet werden:

$$T(\ell+1, i_1, \ldots, i_{k-1}) := \max \big\{ -1,\ T(\ell, i'_1, \ldots, i'_{k-1}) + T(\ell, i''_1, \ldots, i''_{k-1}) \\ \mid \forall j \in \{1, \ldots, k-1\}\colon i'_j, i''_j \in \{0, \ldots, N_j\},\ i_j = i'_j + i''_j,$$

$$T(\ell, i'_1, \ldots, i'_{k-1}) \neq -1 \neq T(\ell, i''_1, \ldots, i''_{k-1})\big\}. \quad (6.1.1)$$

Analog zur Sperrflusskonstruktion in Abschnitt 3.5 kann diese Methode einfach an Maschinenanzahlen m angepasst werden, die keine Zweierpotenzen sind. Entscheidend ist, dass T stets nur $\mathcal{O}(\log m)$ verschiedene Werte in seiner ersten Dimension annehmen muss. Schließlich gibt es offenbar genau dann einen gültigen Stundenplan für P, wenn $T(\log m, N_1, \ldots, N_{k-1}) \geq N_k$. Insgesamt werden $\mathcal{O}(\log m \cdot \Pi_{j=1}^{k-1}(N_j + 1)) = \mathcal{O}(\log m \cdot N^{k-1})$ Tabelleneinträge berechnet. Dabei wurde jeweils das Maximum von $\mathcal{O}(\Pi_{j=1}^{k-1}(N_j + 1)) = \mathcal{O}(N^{k-1})$ Lösungswerten berechnet, die jeweils einem Vektor $\big(i'_1, \ldots, i'_{k-1}\big)$ entsprechen. Insgesamt ergibt sich so die Laufzeit $\mathcal{O}(\log m \cdot N^{2(k-1)})$.

Der minimale Makespan kann schließlich durch eine binäre Suche ermittelt werden, die Leungs Algorithmus $\mathcal{O}(\log \max\{t_1, \ldots, t_k\})$-mal ausführt. Offenbar können wir eine konkrete Zuweisung von Aufgabenanzahlen zu Maschinen erhalten, indem wir für jeden Tabelleneintrag $T(\ell + 1, i_1, \ldots, i_{k-1})$ den jeweils optimalen Partitionsvektor $\big(i'_1, \ldots, i'_{k-1}\big)$ abspeichern. Analog können Zeiger auf die beiden Einträge $T(\ell, i'_1, \ldots, i'_{k-1})$ und $T(\ell, i'_1, \ldots, i'_{k-1})$ verwaltet werden, aus denen der aktuelle Lösungswert gewonnen wurde.

Leung betrachtet diesen Algorithmus als polynomiell, da die Eingabelänge allgemeiner Stundenplanprobleme $\Omega(N)$ ist. Im Falle eingeschränkter Bearbeitungszeiten kann eine Instanz jedoch offenbar durch nur $2k+2$ Zahlen der Länge $\log(mD)$ repräsentiert werden. Zudem kann dieser Ansatz wie bereits erwähnt in praktischen Anwendungen mit kleinem k und großem N als nicht mehr effizient gelten.

6.2 Eine implizite Version von Leungs Algorithms

Wir stellen eine implizite Version von Leungs Algorithmus vor und hoffen, dass bei einer OBDD-basierten Implementation die charakteristische Funktion der Tabelle T in typischen Anwendungsfällen wesentlich kompakter dargestellt werden kann.

Wieder nehmen wir $\log m \in \mathbb{N}$ an. Außerdem ist es sinnvoll, von $t_i \leq D$ und $N_i \leq mD$ für $i := 1, \ldots, k$ auszugehen. Dann sind $\lceil \log(mD+1) \rceil =: n$ boolesche Variablen ausreichend, um alle Zahlen darzustellen, die während des Algorithmus auftreten können.

Wir arbeiten mit $\log m$ charakteristischen Funktionen $\chi_{T,\ell} \in B_{kn}$, um die einzelnen Zeilen von T implizit darzustellen. Es ist

$$\chi_{T,\ell}\big(x^{(1)}, \ldots, x^{(k)}\big) = 1 :\Leftrightarrow T\big(\ell, \big|x^{(1)}\big|, \ldots, \big|x^{(k-1)}\big|\big) = \big|x^{(k)}\big|$$

für $\ell = 0, \ldots, \log m$ und Argumentvektoren $x^{(j)} \in \mathbb{B}^n$ für $j := 1, \ldots, k$. Der Wert $\big|x^{(j)}\big|$ entspricht dann der Aufgabenanzahl i_j im ursprünglichen Algorithmus. Für $T\big(\ell, \big|x^{(1)}\big|, \ldots, \big|x^{(k-1)}\big|\big) \geq 0$ hat die Subfunktion $\chi_{T,\ell}\big(x^{(1)}, \ldots, x^{(k-1)}\big)$ also genau eine erfüllende Belegung für die $x^{(k)}$-Variablen. Enthält T an dieser Position lediglich den symbolischen Wert -1, so ist die obige Subfunktion die Nullfunktion.

Um nun die initiale Funktion $\chi_{T,0}$ zu berechnen, drücken wir die entsprechenden Bedingungen für den Fall $\ell = 0$ durch boolesche Formeln aus. Gemäß unseren Vereinbarungen in Abschnitt 2.5 verwenden wir ausschließlich Funktionen mit kompakten $\pi^{\mathrm{id}}_{k,n}$-OBDDs als Bausteine: multivariate Thresholdfunktionen sowie multivariate Modulofunktionen (siehe Theorem 2.4.7 und Theorem 2.4.11). Zunächst stellen wir eine Funktion g für die Bedingung $\left|x^{(k)}\right| = \left\lfloor \left(D - \sum_{j=1}^{k-1} t_j \cdot \left|x^{(j)}\right|\right)/t_k \right\rfloor$ auf, die leicht ersichtlich äquivalent zu

$$D - \sum_{j=1}^{k-1} t_j \cdot \left|x^{(j)}\right| = \left|x^{(k)}\right| \cdot t_k + \left(D - \sum_{j=1}^{k-1} t_j \cdot \left|x^{(j)}\right| \right) \bmod t_k$$

ist. Dies führt zu folgender Formulierung für g, die die Hilfsvektoren $y, z \in \mathbb{B}^n$ verwendet:

$$g\big(x^{(1)}, \ldots, x^{(k)}\big) := (\exists y, z) \left[\left(|y| = D - \sum_{j=1}^{k-1} t_j \cdot \left|x^{(j)}\right| \right) \right.$$
$$\left. \wedge \left(|z| < t_k\right) \wedge \left(|y| - |z| \bmod t_k = 0\right) \wedge \left(|y| = \left|x^{(k)}\right| \cdot t_k + |z|\right) \right].$$

Insgesamt kann $\chi_{T,0}$ also folgendermaßen berechnet werden:

$$\chi_{T,0}\big(x^{(1)}, \ldots, x^{(k)}\big) := \bigwedge_{j=1}^{k-1} \left(\left|x^{(j)}\right| \le B_j\right) \wedge \left(D \ge \sum_{j=1}^{k-1} t_j \cdot \left|x^{(j)}\right| \right) \wedge g\big(x^{(1)}, \ldots, x^{(k)}\big).$$

Nun betrachten wir den Iterationsschritt in Gleichung (6.1.1), den wir implizit mithilfe von $2k$ zusätzlichen Vektoren $u^{(1)}, v^{(1)}, \ldots, u^{(k)}, v^{(k)} \in \mathbb{B}^n$ von Hilfsvariablen realisieren. Wir nehmen also an, dass $\chi_{T,\ell}$ für ein $\ell \in \{0, \ldots, \log m - 1\}$ bereits berechnet wurde, und definieren $h_{\ell+1} \in B_{kn}$:

$$h_{\ell+1}\big(x^{(1)}, \ldots, x^{(k)}\big) := \big(\exists u^{(1)}, v^{(1)}, \ldots, u^{(k)}, v^{(k)}\big)$$
$$\left[\bigwedge_{j=1}^{k} \left(\left|x^{(j)}\right| = \left|u^{(j)}\right| + \left|v^{(j)}\right|\right) \wedge \chi_{T,\ell}\big(u^{(1)}, \ldots, u^{(k)}\big) \wedge \chi_{T,\ell}\big(v^{(1)}, \ldots, v^{(k)}\big) \right].$$

Somit repräsentiert $h_{\ell+1}$ gültige Lastvektoren $\left(\left|x^{(1)}\right|, \ldots, \left|x^{(k)}\right|\right)$, die als Summe zweier Lastvektoren $\left(\left|u^{(1)}\right|, \ldots, \left|u^{(k)}\right|\right)$ und $\left(\left|v^{(1)}\right|, \ldots, \left|v^{(k)}\right|\right)$ dargestellt werden können, für die gilt: Komponente $\left|u^{(k)}\right|$ (bzw. $\left|v^{(k)}\right|$) ist die maximale Anzahl von Typ-k-Aufgaben, die gemeinsam mit $\left|u^{(j)}\right|$ (bzw. $\left|v^{(j)}\right|$) Typ-j-Aufgaben auf 2^ℓ Maschinen mit Makespan D ausgeführt werden können (für alle $j := 1, \ldots, k$).

Schließlich müssen wir die Maximalität von $\left|x^{(k)}\right|$ garantieren:

$$\chi_{T,\ell+1}\left(x^{(1)},\ldots,x^{(k)}\right) := h_{\ell+1}\left(x^{(1)},\ldots,x^{(k)}\right) \wedge \overline{(\exists y)\left[(|y| > |x^{(k)}|) \wedge h_{\ell+1}(x^{(1)},\ldots,x^{(k-1)},y)\right]}.$$

Für jede Belegung der Argumente $x^{(1)},\ldots,x^{(k-1)}$ existiert also nur noch eine erfüllende Belegung der $x^{(k)}$-Variablen, die maximal entsprechend unserer Definition der Tabelle T ist: Es gibt keine Anzahl $|y|$ von Typ-k-Aufgaben größer als $\left|x^{(k)}\right|$, die gemeinsam mit den anderen $\left|x^{(j)}\right|$ Typ-j-Aufgaben (für alle $j := 1,\ldots,k-1$) auf $2^{\ell+1}$ mit Makespan D ausgeführt werden können.

Haben wir auf diese Weise $\chi_{T,\log m}$ berechnet, ersetzen wir die Argumente $x^{(1)},\ldots,x^{(k-1)}$ durch $(N_1)_2,\ldots,(N_{k-1})_2$. Die einzige erfüllende Belegung der $x^{(k)}$-Variablen der so erhaltenen Funktion entspricht nun dem erwünschten Ergebnis $T(\log m, N_1,\ldots,N_{k-1})$. Ein Vergleich dieses Werts mit N_k gibt uns schließlich die Antwort auf unser Entscheidungsproblem. Die Korrektheit des Verfahrens ergibt sich aus der Korrektheit von Leungs Algorithmus. Wir haben das Stundenplanproblem P mithilfe eines impliziten Algorithmus gelöst. Eine Anpassung an allgemeine Maschinenanzahlen m, die nicht zwingend Zweierpotenzen sind, kann auf gleiche Weise wie für den ursprünglichen Leung-Algorithmus erfolgen und ist unabhängig von der konkreten Darstellung von T.

Theorem 6.2.1. *Der implizite Stundenplanalgorithmus löst ein Stundenplanproblem P mit Bearbeitungszeiten $t_1,\ldots,t_k$, Anforderungen $N_1,\ldots,N_k$, Maschinenanzahl m und Makespan-Schranke D durch Ausführung von $\mathcal{O}(k\log m\log(mD))$ funktionalen Operationen auf booleschen Funktionen, die jeweils auf $(3k+2)n$ Variablen mit $n := \lceil\log(mD+1)\rceil$ definiert sind.*

Beweis: Keine während des Algorithmus auftretende Funktion ist auf mehr als $(3k+2)n$ Variablen definiert. Es werden $\log m+1$ Funktionen $\chi_{T,\ell}$ mit $\ell := 0,\ldots,\log m$ berechnet. Daran ist jeweils eine nur konstante Anzahl an Quantifizierungsblöcken beteiligt, die die Gesamtoperationsanzahl von $\mathcal{O}(kn) = \mathcal{O}(k\log(mD))$ pro Funktion dominiert. □

Anmerkung 6.2.2. *Wir betrachten die vom Algorithmus verwendeten Bausteinfunktionen: Sie stammen aus den Klassen $\mathbb{V}^t_{\mathcal{O}(k),n}$ und $\mathbb{M}^t_{\mathcal{O}(k),n}$ für $t := \max\{t_1,\ldots,t_k\}$. Da unser Algorithmenentwurf auf die Kompaktheit der OBDDs wenigstens dieser Bausteine gesetzt hat, wählen wir die Variablenordnung $\pi^{\mathrm{id}}_{r,n}$ für die maximale Argumentanzahl $r := 3k+2$. Somit können wir die Voll-OBDD-Breite dieser Bausteine nur durch* $\mathrm{poly}(tk)$ *beschränken. Die Länge der Zahl t geht hier demnach (im Gegensatz zum ursprünglichen Leung-Algorithmus) möglicherweise exponentiell in die Laufzeit ein. Wir müssen also damit rechnen, dass der implizite Ansatz bei OBDD-basierter Implementation den ursprünglichen Leung-Algorithmus nur/vor allem für große Gesamtanforderungen N* und *nicht zu großes t schlagen kann.*

Die obere Schranke von $(2 + o(1))\, 2^n/n$ für die OBDD-Größe einer Funktion $f \in B_n$ in Theorem 2.3.9 sowie Theorem 2.3.17 (Punkt 9) implizieren nun einfache obere Schranken:

Korollar 6.2.3. *Die maximale während des impliziten Leung-Algorithmus auftretende OBDD-Größe ist $\mathcal{O}((mD)^{3k+2}/\log(mD))$. Jede der $\mathcal{O}(k \log m \log(mD))$ OBDD-Operationen verursacht daher Laufzeit $\mathcal{O}((mD)^{6k+4}/\log^2(mD))$. Letzteres ist auch eine Schranke für den Gesamtplatz.*

So hoffen wir bei Verwendung des impliziten Algorithmus natürlich, dass er sich tatsächlich wesentlich effizienter verhält.

6.3 Lösung des Optimierungsproblems

Die vorgestellte implizite Version des Algorithmus von Leung löst lediglich die Entscheidungsvariante des Stundenplanproblems. Wir sprechen hier kurz eine Modifikation an, mit der auch ein konkreter Stundenplan berechnet werden kann, dessen Makespan nicht größer als D ist (falls existent).

Dazu ändern wir die Definition der Funktion h_ℓ, sodass nicht nur die Existenz einer entsprechenden Partition des Lastvektors $\left(\left|x^{(1)}\right|, \ldots, \left|x^{(k)}\right|\right)$ festgestellt wird, sondern diese ebenfalls repräsentiert wird:

$$h_{\ell+1}\big(x^{(1)}, \ldots, x^{(k)}, u^{(1)}, \ldots, u^{(k)}, v^{(1)}, \ldots, v^{(k)}\big) := \bigwedge_{j=1}^{k} \left(\left|x^{(j)}\right| = \left|u^{(j)}\right| + \left|v^{(j)}\right|\right) \\ \wedge \chi_{T,\ell}\big(u^{(1)}, \ldots, u^{(k)}\big) \wedge \chi_{T,\ell}\big(v^{(1)}, \ldots, v^{(k)}\big).$$

Entsprechend verändern wir die Formel zur iterativen Berechnung von $\chi_{T,\ell+1}$:

$$\chi_{T,\ell+1}\big(x^{(1)}, \ldots, x^{(k)}, u^{(1)}, \ldots, u^{(k)}, v^{(1)}, \ldots, v^{(k)}\big) \\ := h_{\ell+1}\big(x^{(1)}, \ldots, x^{(k)}, u^{(1)}, \ldots, u^{(k)}, v^{(1)}, \ldots, v^{(k)}\big) \\ \wedge \overline{(\exists y)\,[(|y| > |x^{(k)}|) \wedge h_{\ell+1}(x^{(1)}, \ldots, x^{(k-1)}, y, u^{(1)}, \ldots, u^{(k)}, v^{(1)}, \ldots, v^{(k)})]}.$$

Ist schließlich $\chi_{T,\log m}$ bekannt, so ersetzen wir wieder die Argumente $x^{(1)}, \ldots, x^{(k-1)}$ durch $(N_1)_2, \ldots, (N_{k-1})_2$. Das Ergebnis hat nun ebenfalls nur eine erfüllende Belegung $x^{(k,*)}, u^{(1,*)}, \ldots, u^{(k,*)}, v^{(1,*)}, \ldots, v^{(k,*)}$ der Variablen $x^{(k)}, u^{(1)}, \ldots, u^{(k)}, v^{(1)}, \ldots, v^{(k)}$. Ist $\left|x^{(k,*)}\right| \geq N_k$, so existiert eine gültige Lösung unseres Stundenplanproblems. Um diese zu erhalten, betrachten wir die Subfunktionen

$$\chi_{T,\log m-1} \mid x^{(1)} := u^{(1,*)}, \ldots, x^{(k,*)} := u^{(k,*)}$$

und

$$\chi_{T,\log m-1} \mid x^{(1)} := v^{(1,*)}, \ldots, x^{(k,*)} := v^{(k,*)}.$$

Sie haben wiederum jeweils nur eine erfüllende Belegung der noch freien u- und v-Variablen, die wir wieder extrahieren. Die so erhaltenen nun vier Lastvektoren setzen wir wieder in die x-Variablen von $\chi_{T,\log m-2}$ ein. Dieses Verfahren wiederholen wir, bis für $\chi_{T,0}$ schließlich m Lastvektoren bekannt sind, die der Belegung der m Maschinen mit den Aufgabentypen 1 bis k entsprechen.

Wie eingangs erwähnt, kann jedes $\chi_{T,\ell}$ auch als charakteristische Funktion eines Graphen mit Knotenmenge $\{0,\ldots,mD\}^k$ aufgefasst werden, wobei die x-Variablen einen Startknoten und die u- und v-Variablen jeweils einen Zielknoten der Kanten darstellen, deren Rückverfolgung von $\chi_{T,m}$ zu $\chi_{T,0}$ schließlich den konkreten gültigen Stundenplan ergibt.

Anmerkung 6.3.1. *Variablenersetzungen durch Konstanten sowie die Extrahierung einer erfüllenden Belegung einer Funktion sind effiziente OBDD-Operationen mit linearer Laufzeit. Offenbar können wir eine konkrete Lösung in polynomieller Zeit bez. der Summe der OBDD-Größen von $\chi_{T,0},\ldots,\chi_{T,\log m}$ berechnen. Während bei der Modifikation des Algorithmus Quantifizierungen bei der Berechnung von $h_{\ell+1}$ wegfallen, sind die Funktionen h_ℓ und $\chi_{T,\ell}$ für alle $\ell := 0,\ldots,\log m$ nun auf $2kn$ weiteren Variablen definiert. Die maximale Gesamtanzahl $(3k+2)n$ verwandter Variablen ändert sich dadurch jedoch nicht und Korollar 6.2.3 gilt auch hier noch.*

6.4 Experimentelle Untersuchungen

Der implizite Stundenplanalgorithmus wurde in C++ mithilfe des OBDD-Pakets CUDD von F. Somenzi implementiert. Die Implementation macht Gebrauch von komplementierten Kanten. Anfangs beginnt sie mit der Variablenordnung $\pi^{\mathrm{id}}_{r,n}$ für die maximale Anzahl $r = \mathcal{O}(k)$ der verwandten Argumente. Insbesondere werden für diese Variablenordnung die OBDDs aller Bausteinfunktionen erzeugt. Im späteren Verlauf des Algorithmus wird nach Quantifizierungsblöcken eine Heuristik zur Optimierung der Variablenordnung aufgerufen, da gerade nach solchen Operationen ein hohes Optimierungspotenzial beobachtet wurde. Die verwandte Heuristik war die sogenannte Fensterpermutationsmethode (Window Permutation Approach) von Ishiura u. a. (1991), da sie in Stichproben die besten Ergebnisse erzielte. Sie bildet in jeder Iteration alle Permutationen dreier aufeinanderfolgender Variablen der aktuellen lokalen Ordnung τ (anfangs id) und wählt diejenige aus, die zur minimalen SBDD-Größe führt. Dabei bleibt die Gesamtordnung r-gemischt. Dies wird iteriert, bis auf diese Weise keine Verbesserung mehr möglich ist.

6.4.1 Generierung der Probleminstanzen

Die Implementation wurde auf zufällig erzeugte Stundenplanprobleme angewandt. Ein solches Problem P hat eine Gesamtlast $L := \sum_{j=1}^{k} t_j \cdot N_j$ mit Erwartungswert $\mathrm{E}[L] = M := (mD/1{,}2)$ Die tatsächliche maximale Kapazität der m Maschi-

nen bei Makespan D ist also 20% größer als die erwartete Gesamtlast. Dies wird erreicht, indem zunächst gleichverteilte Anteile $F_1, \ldots, F_k$ von M gezogen werden, sodass $\sum_{j=1}^{k} F_j = M$ für jeden Aufgabentyp $j := 1, \ldots, k$ gilt. Dann wird die Anzahl N_j von Typ-j-Aufgaben gleichverteilt mit fest gewähltem Erwartungswert $\mathrm{E}[N_j]$ gezogen. Schließlich werden die Bearbeitungszeiten t_j in verschiedenen Experimentserien gemäß Gleichverteilung, Exponentialverteilung und Erlangverteilung mit Formparameter 2 mit jeweiligem Erwartungswert $\mathrm{E}[t_j] := F_j/N_j$ gezogen. Dies sind übliche Verteilungen bei der Generierung von synthetischen Stundenplanproblemen (siehe z. B. Albers und Schröder (2002)). Wir rechnen nach, dass für dieses Verfahren $\mathrm{E}[L] = M$ gilt:

$$\mathrm{E}[L] = \mathrm{E}\left[\sum_{j=1}^{k} t_j \cdot N_j\right] = \sum_{j=1}^{k} \mathrm{E}[t_j \cdot N_j] = \sum_{j=1}^{k} \mathrm{E}[t_j] \cdot N_j = \sum_{j=1}^{k} F_j = M.$$

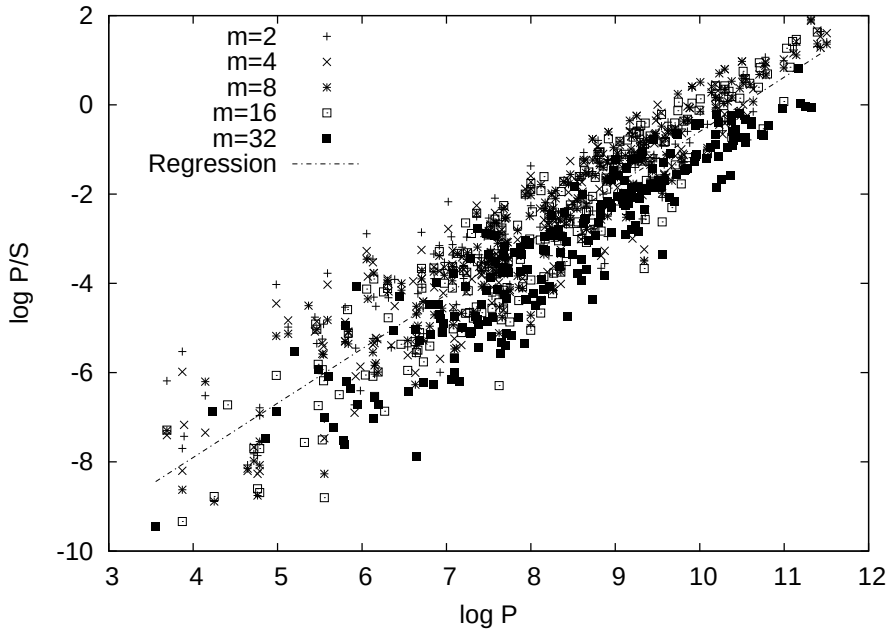
Dieses randomisierte Verfahren hat also die Parameter m, D und $\mathrm{E}[N_j]$ für $j = 1, \ldots, k$. Die erwartete verfügbare Gesamtkapazität M wird also zunächst gleichverteilt unter den Aufgabentypen aufgeteilt. Zudem wird durch Wahl von $\mathrm{E}[t_j] := F_j/N_j$ im Erwartungsfall jeder Anteil F_j unter N_j Typ-j-Aufgaben aufgeteilt. Die gezogenen Werte t_j und N_j werden gleichverteilt zu Ganzzahlen gerundet. Es werden zudem nur solche Probleme akzeptiert, die die Bedingungen $t_j \leq D$, $N_j \leq mD$ und $t_j \neq t_{j'}$ für $1 \leq j < j' \leq k$ erfüllen, ansonsten wird das Problem verworfen.

Die Experimente bestanden aus drei Serien mit $mD := 800, 1600, 3200$ und $k := 3$. Für jedes Produkt mD wurden zudem verschiedene Maschinenanzahlen 2, 4, 8, 16 und 32 gewählt sowie zehn verschiedene erwartete Aufgabenanzahlen $\mathrm{E}[N_j]$, die gleichmäßig zwischen $(mD)/6$ und $(mD)/3$ angesiedelt waren ($\mathrm{E}[N_1] = \cdots = \mathrm{E}[N_k]$). Für jeden solchen Parametersatz und jede der drei Verteilungen für die Bearbeitungszeiten wurden jeweils 20 Instanzen generiert.

6.4.2 Experimentelle Ergebnisse

Die Experimente fanden auf einem PC mit Pentium-4-Prozessor (3 GHz) und 1 GB Hauptspeicher statt. Laufzeit wurde in Prozessorzeitsekunden gemessen, der Platz in der maximalen Anzahl an OBDD-Knoten im SBDD, die während der Ausführung erreicht wurde.

Um den ursprünglichen Algorithmus von Leung mit seiner impliziten Version vergleichen zu können, betrachten wir die Ressourcen des Ersteren genauer: Da wir ausschließlich Zweierpotenzen als Maschinenanzahlen verwandt haben, kann jede Subtabelle $T(\ell, \ldots)$ aus dem Speicher entfernt werden, nachdem $T(\ell + 1, \ldots)$ berechnet wurde. Eine effiziente Implementation wählt zudem $N_k = \max\{N_1, \ldots, N_k\}$, um die Größe von T zu minimieren. Es wird also Platz $\Omega(\Pi_{j=1}^{k-1} N_j)$ und Zeit $\Omega(\Pi_{j=1}^{k-1} N_j^2)$ benötigt. Wir verwenden diese unteren Schranken, um die beiden Algorithmen asymptotisch zu vergleichen.



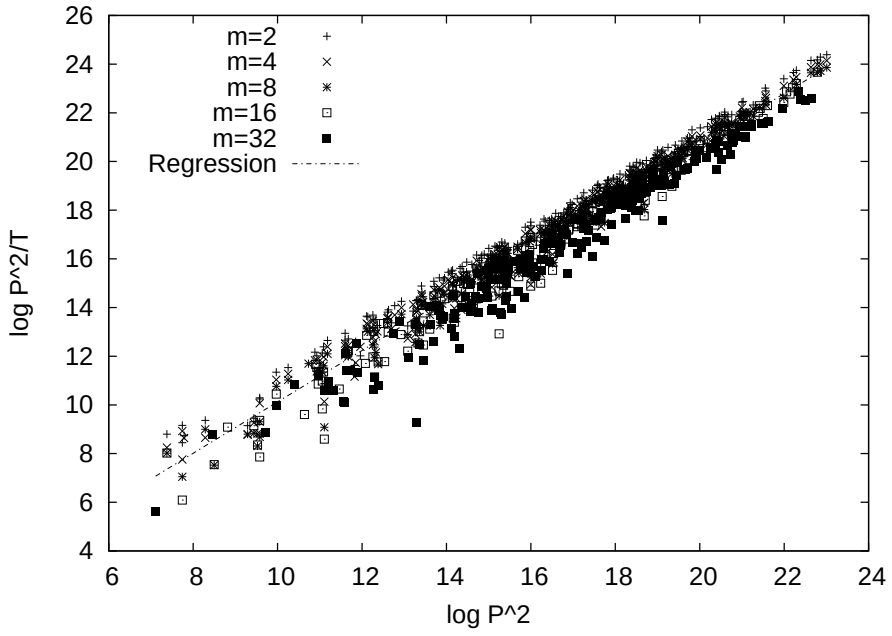
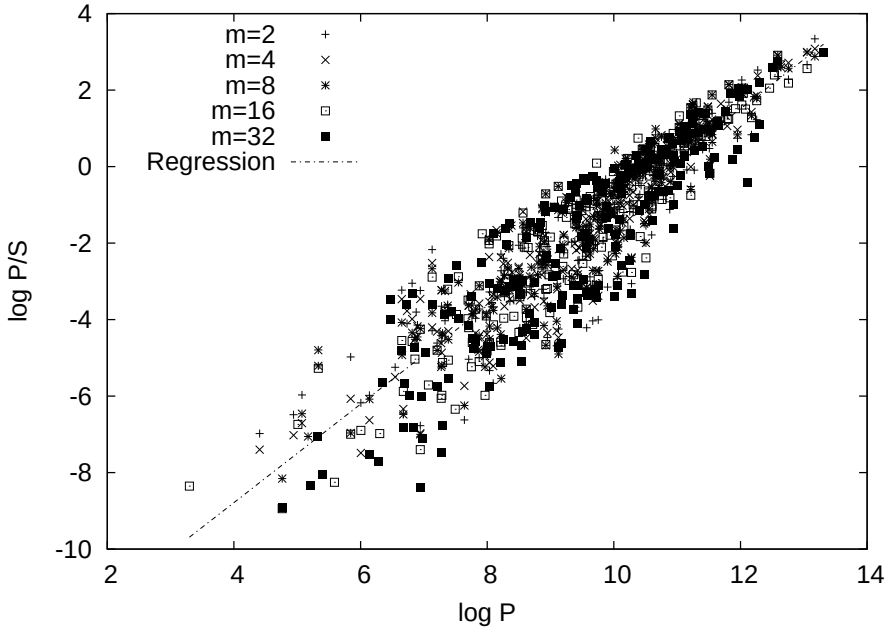


Abbildung 6.4.1: Experimentelle Ergebnisse des impliziten Leung-Algorithmus auf Instanzen mit exponentiell verteilten Bearbeitungszeiten und $mD = 800$. P bezeichnet $\Pi_{j=1}^{k-1} N_j$, S bezeichnet den beobachteten Speicherplatz, T bezeichnet die beobachtete Laufzeit.



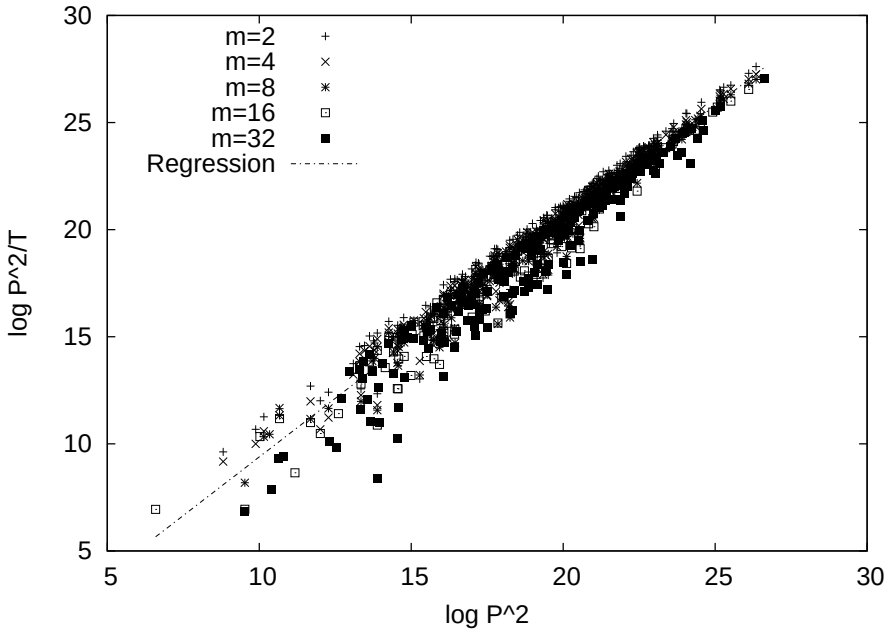
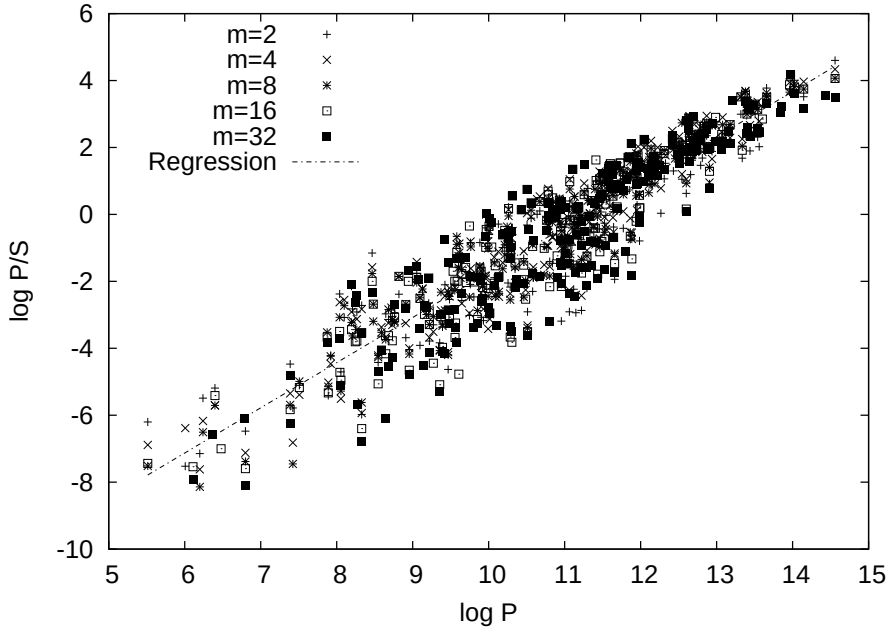


Abbildung 6.4.2: Experimentelle Ergebnisse des impliziten Leung-Algorithmus auf Instanzen mit exponentiell verteilten Bearbeitungszeiten für $mD = 1600$. P bezeichnet $\Pi_{j=1}^{k-1} N_j$, S bezeichnet den beobachteten Speicherplatz, T bezeichnet die beobachtete Laufzeit.



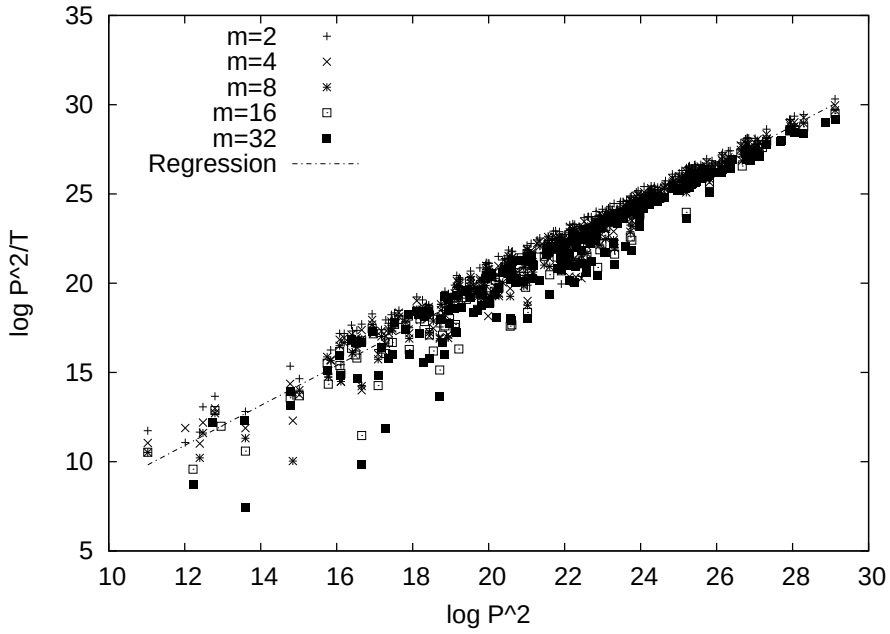


Abbildung 6.4.3: Experimentelle Ergebnisse des impliziten Leung-Algorithmus auf Instanzen mit exponentiell verteilten Bearbeitungszeiten für $mD = 3200$. P bezeichnet $\Pi_{j=1}^{k-1} N_j$, S bezeichnet den beobachteten Speicherplatz, T bezeichnet die beobachtete Laufzeit.

Vert. / mD	800	1600	3200
Gleichvert.	1,28270 / -13,21930	1,32832 / -14,21380	1,36477 / -15,22760
Exp.-Vert.	1,21732 / -12,77160	1,28701 / -13,93220	1,34987 / -15,22830
Erlangvert.	1,26583 / -13,11690	1,37385 / -14,63660	1,44698 / -16,14440

(a) Platz.

Vert. / mD	800	1600	3200
Gleichvert.	1,06093 / -0,47254	1,08898 / -1,35841	1,12353 / -2,51579
Exp.-Vert.	1,05560 / -0,43123	1,09418 / -1,55419	1,11825 / -2,50760
Erlangvert.	1,06234 / -0,52043	1,11002 / -1,82228	1,15972 / -3,31591

(b) Zeit.

Tabelle 6.4.1: Regressionsergebnisse a_1 und b_1 für P/S bzw. a_2 und b_2 für P^2/T.

Wir betrachten die experimentellen Ergebnisse für exponentiell verteilte Bearbeitungszeiten: Seien P und T der jeweils beobachtete Speicherplatz bzw. die jeweils beobachtete Laufzeit des impliziten Algorithmus. Abbildung 6.4.1, Abbildung 6.4.2 und Abbildung 6.4.3 zeigen die Logarithmen der Verhältnisse P/S und P^2/T für $P := \Pi_{j=1}^{k-1} N_j$ in Abhängigkeit von den Logarithmen der jeweiligen unteren Schranken P bzw. P^2. Informell gesprochen bilden wir also auf der y-Achse mit logarithmischer Skala den Faktor ab, um den der implizite Algorithmus der klassischen Version überlegen ist. Dies geschieht wiederum in Abhängigkeit von einer wachsenden unteren Schranke für die explizite Version. Die so dargestellten Funktionen scheinen linear zu wachsen, und so wurde ein Least-Squares-Verfahren angewandt, um entsprechende Parameter a_1 und b_1 für $\log(P/S) = a_1 \cdot \log P + b_1$ bzw. a_2 und b_2 für $\log(P^2/T) = a_2 \cdot \log P^2 + b_2$ zu schätzen.

Tabelle 6.4.1 zeigt diese Regressionsergebnisse für alle drei Werte von mD sowie für alle drei Verteilungen der Bearbeitungszeiten. Dabei ist der asymptotische Standardfehler der Steigungen a_1 und a_2 in keinem Fall größer als 1,4%, und unsere Hypothese linearen Wachstums ist signifikant bestätigt. Insgesamt sind keine wesentlichen Unterschiede zwischen den drei Verteilungen zu erkennen, lediglich die Erlangverteilung scheint insgesamt etwas größere absolute Werte a und $|b|$ zu verursachen.

6.4.3 Interpretation der experimentellen Ergebnisse

Die beobachteten linearen Abhängigkeiten implizieren $P/S = P^{a_1} \cdot 2^{b_1} \Leftrightarrow S = P^{1-a_1} \cdot 2^{-b_1}$ bzw. $T = P^{2(1-a_2)} \cdot 2^{-b_2}$. Wegen $1 < a_1, a_2 < 1{,}5$ folgt für den Platz $2^{-b_1}/\sqrt[c_1]{P}$ mit $c_1 = 1/(a_1 - 1) > 2$ sowie $2^{-b_2}/\sqrt[c_2]{P}$ für die Laufzeit mit $c_2 = 1/(2a_2 - 2) > 1$. Für

feste Werte mD und k sinkt die OBDD-Größe also sogar proportional zu $\sqrt[c_1]{P}$, und die a- und b-Parameter scheinen lediglich von der Gesamtkapazität mD sowie k abzuhängen, während sie unabhängig von m und $\mathrm{E}[N_j]$ sind. Wir bemerken, dass für hohes P die erwarteten Bearbeitungszeiten t_j klein sein müssen, da die erwartete Gesamtlast mit mD gleich bleibt. Für kleine Werte t_j erwarten wir entsprechend Anmerkung 6.2.2 kompaktere OBDDs für die verwandten Bausteinfunktionen. Unabhängig von der Größe von $\max_j\{t_j\}$ muss der ursprüngliche Leung-Algorithmus jedoch für großes P eine große Tabelle T generieren. Dies ist eine mögliche Erklärung für die antiproportionale Beziehung von $\log P$ zu S und T. Sinnvolle Instanzen können aber nicht beliebig kleine Bearbeitungszeiten haben, und so sind P bzw. $1/S$ durch die Wahl von mD und k natürliche Grenzen gesetzt. Wir dokumentieren zwar lediglich Ergebnisse für drei verschiedene Bearbeitungszeiten, entsprechende Resultate wurden jedoch auch für andere Werte k beobachtet.

Die a Werte wachsen in mD, während die b-Werte in mD sinken. Der anfängliche „Vorsprung“ der klassischen Version für kleine Werte P nimmt also mit mD zu, gleichzeitig nimmt aber mit größerem mD auch die „Aufholgeschwindigkeit“ a der impliziten Version zu. Aus unseren Experimenten lässt sich also ein wesentlich besseres asymptotisches Verhalten der OBDD-basierten impliziten Version des Leung-Algorithmus bez. des wachsenden Produkts P der Einzelanforderungen ablesen.

Betrachten wir die Größen der Werte $\log(P/S)$ und $\log(P^2/T)$, so nehmen sie für steigendes mD auch höhere Maximalwerte an. Bereits in unseren Experimenten wurde beispielweise für $mD = 3200$ ein Faktor $P/S \geq 2^6$ beobachtet. Bei derartigen Größenordnungen spielt die Diskrepanz zwischen der Speichermessung in OBDD-Knoten und der in tatsächlichen Bytes also keine Rolle mehr – mithilfe von OBDDs konnte die Tabelle T wesentlich kompakter dargestellt werden. Wir vermuten daher zu guter Letzt, dass für jedes k Werte mD und P existieren, ab denen der implizite Algorithmus den tatsächlichen Ressourcen einer jeweiligen Implementation des ursprünglichen Leung-Algorithmus überlegen ist.

7 Komplexitätsresultate

Bisher wurde die Dissertation vom Entwurf konkreter Algorithmen und ihrer theoretischen und experimentellen Untersuchung dominiert. Dabei haben wir sowohl Obereals auch Untere-Schranken-Techniken kennengelernt. In diesem Kapitel beschäftigen wir uns schließlich mit der Frage, was implizite und insbesondere OBDD-basierte Algorithmen überhaupt im Falle einiger ausgewählter grundlegender Probleme leisten können. Zuerst betrachten wir das primäre Entwurfsziel einer kleinen Anzahl funktionaler Operationen und sehen, dass selbst dieses für in $|V|$ polylogarithmische Gesamtlaufzeit nicht hinreichende, jedoch notwendige Teilziel nicht erreicht werden kann, wenn das zu lösende Problem P-vollständig ist (unter der üblichen Annahme $\mathrm{P} \neq \mathrm{NC}$).

Anschließend stellen wir uns die Frage, für welche Probleme überhaupt Festparameteralgorithmen bez. einer festen Voll-OBDD-Breite von Eingabe bzw. Eingabe und Ausgabe existieren können. Für die meisten der betrachteten Probleme können wir tatsächlich zeigen, dass dies nicht der Fall ist.

Im dritten Teil des Kapitels erweitern wir dieses Modell um die Betrachtung des Speicherplatzes und stellen eine Entwurfsmethode für Algorithmen vor, die nur polynomiellen Platz benötigen, wenn Eingabe- und Ausgabe-OBDD-Größe polynomiell verknüpft sind. Dann wenden wir uns wieder Negativresultaten zu: Während sich die bisherigen Ergebnisse zur Nichtexistenz von Festparameteralgorithmen auf die Laufzeit unter der Annahme $\mathrm{P} \neq \mathrm{PSPACE}$ bezogen haben, stellen wir nun konkrete exponentielle untere Schranken für den Speicherplatz OBDD-basierter Algorithmen auf Eingabeinstanzen konstanter Voll-OBDD-Breite für die drei betrachteten Zusammenhangs-Zerlegungs-Probleme sowie für **SSSP$^+$**, **APSP$^+$**, **Flussmaximierung**, **Topologisches Sortieren**, ***q*-Erreichbarkeit** und **Transitiver Abschluss** vor. Die letzten beiden Resultate werden lediglich für eine eingeschränkte Problemvariante gezeigt, beinhalten jedoch die erste konkrete exponentielle untere Schranke für die OBDD-Größe des höchstwertigen Bits der Multiplikation bez. einer speziellen Variablenordnung.

7.1 P-vollständige Probleme und die Anzahl funktionaler Operationen

Im uneingeschränkten Szenario impliziter Algorithmen, in dem im Unterschied zu beispielweise Gentilini und Policriti (2003) und Gentilini u. a. (2003) auch Zusatzvariablen erlaubt sind, können viele Graphprobleme von impliziten Algorithmen mit polylog($|V|$)

funktionalen Operationen gelöst werden. Dies kann leicht für die meisten der in Abschnitt 2.1.1 eingeführten Graphprobleme gezeigt werden. (Wir verzichten auf die Angabe konkreter Algorithmen, da sie alle auf einer einfachen Anwendung der Iterative-Quadrieren-Technik beruhen.) Offen ist dies jedoch noch für die Flussmaximierungs- und Matchingprobleme sowie für den Planaritätstest.

Insbesondere konnten wir in Kapitel 3 für Algorithmus 3.3.1 nicht garantieren, stets nur polylog($|V|$) Operationen auszuführen. Wir fragen uns also, ob manche Probleme in P möglicherweise keine solchen Algorithmen besitzen. Die Beobachtung, dass eine funktionale Operation die Werte einer boolesche Funktion für alle Variablenbelegungen „gleichzeitig“ verarbeitet, führt zum Vergleich mit parallelen Algorithmen: Hier ist ein Ziel, Probleme auf Eingaben der Länge N in Zeit polylog(N) mithilfe von poly(N) parallel arbeitenden Prozessoren zu lösen. Eine Einführung in dieses Gebiet geben etwa Greenlaw u. a. (1995). Wir definieren hier lediglich knapp das Modell sogenannter CREW-PRAMs:

Definition 7.1.1. *Eine* CREW-PRAM (Concurrent Read Exclusive Write Parallel Random Access Machine) *besteht aus aus einer unendlichen Folge $P_0, P_1, \ldots$ von Prozessoren mit identischer Programmierung und einer unendlichen Folge gemeinsamer ganzzahliger Speicherzellen $R := R_0, R_1, \ldots$. Jeder Prozessor P_i hat einen eigenen lokalen unendlichen Speicher $R^{(i)} := R_0^{(i)}, R_1^{(i)}, \ldots$, kennt seinen Index i, besitzt Instruktionen zum direkten und indirekten Zugriff auf seinen lokalen sowie den gemeinsamen Speicher und entspricht in seinen übrigen Fähigkeiten einer klassischen Registermaschine (RAM).*

Die Eingabe $I \in \mathbb{B}^N$ liegt anfangs im gemeinsamen Speicher R. Prozessor P_0 nutzt eine spezielle Instruktion, mit der er weitere Prozessoren aktivieren kann (nur er nutzt sie, da er seinen Index 0 kennt). Eine Speicherzelle R_j darf jederzeit von beliebig vielen Prozessoren gleichzeitig gelesen werden, die Programmierung muss jedoch garantieren, dass in jedem Rechenschritt höchstens ein Prozessor schreibend auf so eine gemeinsame Zelle zugreift.

Schließlich wird die Ausgabe O wieder in die ersten Register von R platziert und die Berechnung beendet. Als parallele Laufzeit *gilt schließlich die maximale Laufzeit eines Prozessors bis zur vollständigen Ausgabe von O.*

So ist NC die Klasse der Entscheidungsprobleme, die von CREW-PRAMs in paralleler Zeit polylog(N) gelöst werden können und dabei nur poly(N) Prozessoren aktivieren. Ähnlich wie die Vermutung NP $\neq P$ in der NP-Vollständigkeitstheorie wird hier NC $\neq$ P vermutet. Die Probleme in P, auf die alle Probleme in P mithilfe paralleler NC-Algorithmen reduziert werden können, heißen dementsprechend P-vollständig, und für kein solches Problem L konnte bisher $L \in$ NC gezeigt werden. Für Suchprobleme gibt es die entsprechende Klasse FP und den FP-Vollständigkeitsbegriff. Prominente P- bzw. FP-vollständige Probleme sind z. B. das Circuit-Value-Problem und die Lineare Programmierung. Aus dem Gebiet der Graphprobleme seien hier die

Flussmaximierung, die Berechnung der lexikografisch ersten nicht vergrößerbaren unabhängigen Menge sowie die Suche nach einem knoteninduzierten Teilgraphen mit minimalem Grad k genannt.

Um zu zeigen, dass implizite Algorithmen mit einer Anzahl von polylog(N) funktionalen Operationen von CREW-PRAMs mit poly(N) Prozessoren in paralleler Zeit polylog(N) simuliert werden können, benötigen wir ein ähnlich abstraktes und allgemeines Maschinenmodell für implizite Algorithmen. Es soll unabhängig von der konkreten Darstellung boolescher Funktionen arbeiten und funktionale Operationen als Basisoperationen beherrschen. Zudem möchten wir uns hier zunächst nicht speziell auf implizite Graphdarstellungen beziehen, sondern auf allgemeine implizite Darstellungen von Zeichenketten $I \in \mathbb{B}^N$, die wir anschließend zur bisherigen Definition charakteristischer Funktionen in Abschnitt 2.5 in Beziehung setzen werden.

Definition 7.1.2. *Sei $I \in \mathbb{B}^N$. Die* charakteristische Funktion $\chi_I \in B_n$ *von I mit $n := \lceil \log N \rceil$ ist durch*

$$\chi_I(x) = 1 :\Leftrightarrow I_{|x|} = 1$$

für $x \in \mathbb{B}^n$ definiert.

Schließlich führen wir ein neues Maschinenmodell für implizite Algorithmen ein, das unseren Anforderungen in Abschnitt 2.5 entspricht:

Definition 7.1.3. *Eine* Implizite Registermaschine (IRAM) $\mathcal{M}$ *entspricht einer RAM ohne Eingabe- und Ausgabebänder. Zusätzlich zu den ganzzahligen Registern $R = R_0, R_1, \ldots$ besitzt sie* implizite Register $S = S_0, S_1, \ldots$, *die boolesche Funktionen enthalten (anfangs mit der Nullfunktion initialisiert). Die Eingabe $I \in \mathbb{B}^N$ wird $\mathcal{M}$ in Form von χ_I in S_0 präsentiert. Schließlich gibt $\mathcal{M}$ ihre Ausgabe $O \in \mathbb{B}^*$ als χ_O in S_0 aus.*

Neben den üblichen RAM-Operationen für ganzzahlige Register bietet eine IRAM $\mathcal{M}$ die folgenden Operationen auf impliziten Registern S_i und S_j bzw. den darin enthaltenen Funktionen an, deren Ergebnisse jeweils in ein ausgewähltes R- bzw. S-Ergebnisregister abgelegt werden:

- *Ermittle die Anzahl n der Variablen, auf der alle booleschen Funktionen in impliziten Registern zurzeit definiert sind (anfangs $\lceil \log N \rceil$).*
- *Erhöhe die Variablenanzahl n um $\Delta n \in \mathbb{N}$.*
- *Setze $S_i := S_j$.*
- *Werte S_i gemäß einer Variablenbelegung $a \in \mathbb{B}^n$ aus.*
- *Berechne $\overline{S_i}$.*
- *Berechne $S_i \otimes S_j$ für einen binären Infix-Operator $\otimes \in B_2$.*

- *Ersetze eine Variable x_k in S_i durch eine Konstante $c \in \mathbb{B}$.*
- *Vertausche zwei Variablen x_k, x_ℓ in S_i, berechne also die Funktion*

$$S'(x_0, \ldots, x_k, \ldots, x_\ell, \ldots, x_n) := S_i(x_0, \ldots, x_{k-1}, x_\ell, x_{k+1}, \ldots, x_{\ell-1}, x_k, x_{\ell+1}, \ldots, x_{n-1}).$$

- *Entscheide, ob $S_i = S_j$ gilt.*
- *Berechne die Anzahl $|S_i^{-1}(1)|$ erfüllender Variablenbelegungen.*
- *Gib alle erfüllenden Belegungen $S_i^{-1}(1)$ in R-Registern aus.*
- *Entscheide, ob S_i essenziell von einer Variablen x_k abhängt.*
- *Belege ein ausgewähltes implizites Register mit einer Funktion $f \in B_n$, die in R-Registern entsprechend einer Standardcodierung vorliegt (z. B. als Polynom, Schaltkreis oder OBDD). Die Codierung muss Auswertungsoperationen in linearer Zeit bez. ihrer Darstellungsgröße erlauben.*

Jede Operation kostet so viele Zeiteinheiten, wie sie Register liest oder beschreibt.

Wir stellen fest, dass für jede in Definition 2.3.16 eingeführte Operation mit Ausnahme der Quantifizierung eine entsprechende IRAM-Operation existiert. Quantifizierungen wiederum werden durch jeweils $\mathcal{O}(1)$ Negationen und binäre Synthesen realisiert. Die letzte Operation in obiger Definition erlaubt zudem, einfache Bausteinfunktionen wie multivariate Threshold- oder Modulofunktionen in einem impliziten Register zu initialisieren.

Beobachtung 7.1.4. *Ein gemäß Abschnitt 2.5 sinnvoll definierter impliziter Algorithmus führt neben seinen funktionalen Operationen keine wesentlichen Berechnungen durch. Die Anzahl der vom ihm ausgeführten funktionalen Operationen ist also polynomiell mit der Laufzeit der ihm entsprechenden IRAM verknüpft.*

Analog zur Klasse FP definieren wir P-Ops:

Definition 7.1.5. P-Ops *ist die Klasse der Suchprobleme $\Pi\colon \mathbb{B}^* \to \mathcal{P}(\mathbb{B}^*)$, für die IRAMs auf χ_I mit $I \in \mathbb{B}^N$ eine Ausgabe χ_O mit $O \in \Pi(I)$ in Zeit* $\operatorname{polylog}(N)$ *unter Verwendung von $\mathcal{O}(\log N)$ Variablen berechnen können.*

Theorem 7.1.6. *Eine IRAM $\mathcal{M}$ mit Laufzeit $t_{\mathcal{M}}(N)$, die höchstens $k \log N$ boolesche Variablen auf implizit repräsentierten Eingaben $I \in \mathbb{B}^N$ verwendet, kann durch eine CREW-PRAM $\mathcal{M}'$ in paralleler Zeit $\mathcal{O}(t_{\mathcal{M}}(N) \cdot \max\{t_{\mathcal{M}}(N), \log N\})$ mit $\mathcal{O}(N^k)$ Prozessoren auf I simuliert werden.*

Beweis: Jede Belegung $a \in \mathbb{B}^n$ der $n \leq k \log N$ booleschen Variablen, auf denen eine der in den höchsten $t_{\mathcal{M}}(N)$ benutzten impliziten Registern von $\mathcal{M}$ vorhandene Funktion maximal definiert sein kann, wird von ihrem eigenen Prozessor $P_{|a|}$ verwaltet. Er speichert in seinen lokalen Registern $R^{(|a|)}$ die Werte $S_i(a)$ für alle impliziten Register S_i, die auf die bereits zugegriffen wurde. Es werden also stets $2^n = \mathcal{O}(N^k)$ Prozessoren verwandt. Anfangs liest jeder Prozessor $P_{|a|}$ das $|a|$-te Bit der Eingabe I aus Register $R_{|a|}$ und setzt $S_0(a)$ dementsprechend. Übliche RAM-Instruktionen von $\mathcal{M}$ werden nur von P_0 auf R ausgeführt. Jede implizite Operation, die $\mathcal{M}$ ausführen würde, wird jeweils in Zeit $\mathcal{O}(\max\{t_{\mathcal{M}}(N), \log N\})$ simuliert (detailliert im nächsten Abschnitt beschrieben). Schließlich enthält S_0 die charakteristische Funktion der Ausgabe χ_O und jeder Prozessor $P_{|a|}$ schreibt $S_0(a)$ in $R_{|a|}$.

Wir betrachten also die parallele Simulation aller funktionalen IRAM-Operationen genauer unter der Annahme, dass S_0 als Ergebnisregister dient.

Erhöhung der Variablenanzahl n. Um n um Δn zu erhöhen, schreiben wir zunächst alle Funktionswerte in R-Register. Hat $\mathcal{M}$ momentan die impliziten Register $S_0, \ldots, S_{L-1}$ aktiviert, so schreibt Prozessor $P_{|a|}$ alle bei sich lokal gespeicherten Werte $S_0(a), \ldots, S_{L-1}(a)$ in die R-Register $\mu + |a| \cdot L, \ldots, \mu + |a| \cdot L + L - 1$, wobei μ ein sinnvoll gewählter Wert ist, um den aktuellen Arbeitszustand von R zu schützen. P_0 aktiviert weitere $2^{n+\Delta n} - 2^n$ Prozessoren. Wenn ein Prozessor bisher für die Belegung $a \in \mathbb{B}^n$ verantwortlich war, so verwaltet er nun $0^{\Delta n}a$. Ein jeder der neuen Prozessoren $P_{|bc|}$ mit $|b| \in \mathbb{B}^{\Delta n}$ und $|c| \in \mathbb{B}^n$ kopiert schließlich die R-Register $\mu + |c| \cdot L, \ldots, \mu + |c| \cdot L + L - 1$ in seinen lokalen Speicher – die Operation ist abgeschlossen.

Kopie von S_j in S_i. Jeder Prozessor $P_{|a|}$ für $a \in \mathbb{B}^n$ setzt $S_i(a)$ auf $S_j(a)$.

Auswertung von S_i. Um S_i für eine Belegung $a \in \mathbb{B}^n$ auszuwerten, schreibt Prozessor $P_{|a|}$ den Wert $S_i(a)$ in Register R_0.

Negation von S_i. Jeder Prozessor $P_{|a|}$ für $a \in \mathbb{B}^n$ setzt $S_0(a)$ auf $\overline{S_i(a)}$.

Berechnung von $S_i \otimes S_j$. Jeder Prozessor $P_{|a|}$ für $a \in \mathbb{B}^n$ setzt $S_0(a)$ auf $S_i(a) \otimes S_j(a)$.

Ersetzung einer Variablen x_k durch eine Konstante c für S_i. Analog zur Erhöhung der Variablenanzahl schreibt jeder Prozessor $P_{|a|}$ für $a \in \mathbb{B}^n$ seinen Funktionswert $S_i(a)$ in Position $|a|$ innerhalb eines freien Intervalls von R-Registern. Dann setzt jeder $P_{|a|}$ in seinem lokalen Speicher $S_0(a)$ auf den Wert $S_i(b)$ mit $b := a_0, \ldots, a_{k-1}, c, a_{k+1}, \ldots, a_{n-1}$, den er aus dem entsprechenden R-Register lesen kann.

Vertauschung zweier Variablen x_k und x_ℓ für S_i. Analog zur Erhöhung der Variablenanzahl schreibt jeder Prozessor $P_{|a|}$ für $a \in \mathbb{B}^n$ seinen Funktionswert $S_i(a)$

in Position $|a|$ innerhalb eines freien Intervalls von R-Registern. Wir nehmen o. B. d. A. $k < \ell$ an: Jedes $P_{|a|}$ setzt in seinem lokalen Speicher $S_i(a)$ auf $S_i(b)$ für $b := a_0, \ldots, a_{k-1}, a_\ell, a_{k+1}, \ldots, a_{\ell-1}, a_k, a_{\ell+1}, \ldots, a_{n-1}$.

Gleichheitstest $S_i = S_j$. Ähnlich wie bei der Erhöhung der Variablenanzahl schreibt jeder Prozessor $P_{|a|}$ für $a \in \mathbb{B}^n$ das Vergleichsergebnis $(S_i(a) = S_j(a)) =: C(a) \in \mathbb{B}$ in Position $|a|$ innerhalb eines freien Intervalls von R-Registern. Nun muss effizient geprüft werden, ob es ein negatives Vergleichsergebnis gibt. Wir können dabei o. B. d. A. annehmen, dass zwei Werte $C(b)$ und $C(b+1)$ mit $b \in \mathbb{B}^n$ und $b_0 = 0$ stets in benachbarte R-Register abgelegt werden. Jedem solchen Paar wird sein eigener Prozessor $P_{|b|}$ zugewiesen, der $(C(b) \wedge C(b+1))$ in ein R-Register mit Index $\mu + \lfloor |b|/2 \rfloor$ legt. Der Summand μ sorgt wieder dafür, dass die Vergleichsergebnisse in einen noch freien Speicherbereich der Länge 2^{n-1} abgelegt werden. Dieses Verfahren wird nun auf die soeben produzierten 2^{n-1} Ergebnisse angewandt und iterativ wiederholt, bis schließlich ein Gesamtergebnis verbleibt. Offenbar reichen $\mathcal{O}(N \log N)$ R-Register sowie parallele Zeit $\mathcal{O}(\log N)$.

Berechnung von $|S_i^{-1}(1)|$. Analog zur Erhöhung der Variablenanzahl schreibt jeder Prozessor $P_{|a|}$ für $a \in \mathbb{B}^n$ seinen Funktionswert $S_i(a)$ in Position $|a|$ innerhalb eines freien Intervalls von R-Registern. Wir können dabei o. B. d. A. annehmen, dass zwei Werte $S_i(b)$ und $S_i(b+1)$ mit $b \in \mathbb{B}^n$ und $b_0 = 0$ stets in benachbarte R-Register abgelegt werden. Jedem solchen Paar wird ein eigener Prozessor $P_{|b|}$ zugewiesen, der $S_i(b) + S_i(b+1)$ in ein R-Register mit Index $\mu + \lfloor |b|/2 \rfloor$ legt. Der Summand μ sorgt wieder dafür, dass die Vergleichsergebnisse in einen noch freien Speicherbereich der Länge 2^{n-1} abgelegt werden. Dieses Verfahren wird nun auf die soeben produzierten 2^{n-1} Ergebnisse angewandt und iterativ wiederholt, bis schließlich ein Gesamtergebnis verbleibt. Offenbar reichen $\mathcal{O}(N \log N)$ R-Register sowie parallele Zeit $\mathcal{O}(\log N)$.

Ausgabe von $S_i^{-1}(1)$ in R. Analog zur Erhöhung der Variablenanzahl schreibt jeder Prozessor $P_{|a|}$ für $a \in \mathbb{B}^n$ seinen Funktionswert $S_i(a)$ in Position $|a|$ innerhalb eines freien Intervalls von R-Registern. Wir nehmen an, dass die Folge $S_i^{-1}(1)$ der $|S_i^{-1}(1)|$ erfüllenden Variablenbelegungen der jeweiligen Länge n in lexikografischer Reihenfolge in die R-Register ab einer Position μ ausgegeben werden soll. Sei also b die j_b-te erfüllende Belegung, so werden ihre n Bits von Prozessor $P_{|b|}$ in die R-Register $\mu + (j_b - 1) \cdot n, \ldots, \mu + j_b \cdot n - 1$ geschrieben; insgesamt verursacht dieses reine Ausgeben parallele Zeit $\mathcal{O}(n) = \mathcal{O}(\log N)$. Vor diesem Ausgabeschritt müssen wir jedoch den Rang j_b von b in der lexikografisch geordneten Folge aller erfüllenden Belegungen ermitteln. Dies geschieht ähnlich wie beim Gleichheitstest zweier Funktionen durch $n = \mathcal{O}(\log N)$ Iterationen: Zu Beginn der k-ten Iteration sei der Rang $j_b^{(k)}$ von b bezogen auf das Teilintervall $\{(\alpha)_2, \ldots, (\beta)_2\} \in \mathbb{B}^n$ der Länge 2^{k-1} bekannt für $\alpha := \lfloor |b|/2^{k-1} \rfloor \cdot 2^{k-1}$

und $\beta := \alpha + 2^{k-1} - 1$. Für Belegungen b in der „linken“ Hälfte des nächstgröberen Intervalls der Länge 2^k (d. h. $|b| < \lfloor |b|/2^k \rfloor \cdot 2^k + 2^{k-1} =: \gamma$) gilt $j_b^{(k+1)} = j_b^{(k)}$. Liegt b in der „rechten“ Hälfte, so berechnet sein Prozessor $P_{|b|}$ den Rang $j_b^{(k+1)} := j_b^{(k)} + j_{(\gamma-1)_2}^{(k)}$. Dies ist korrekt, da $j_{(\gamma-1)_2}^{(k)}$ der Summe der Einsen im Intervall $\left\{(\gamma - 2^{k-1})_2, \ldots, (\gamma-1)_2\right\}$ entspricht. Die Zwischenergebnisse $j_b^{(k)}$ werden nach jeder Iteration wieder in sinnvoll gewählte R-Register geschrieben, um so für alle Prozessoren in der nächsten Iteration zur Verfügung zu stehen. Schließlich haben wir in Iteration n das Gesamtintervall betrachtet und kennen die Position, an der $P_{|b|}$ die Belegung b ausgeben muss. Die parallele Laufzeit ist wieder $\mathcal{O}(\log N)$ auf Platz $\mathcal{O}(N \log N)$.

Entscheidung, ob S_i essenziell von x_k abhängt. Analog zur Erhöhung der Variablenanzahl schreibt jeder Prozessor $P_{|a|}$ für $a \in \mathbb{B}^n$ seinen Funktionswert $S_i(a)$ in Position $|a|$ innerhalb eines freien Intervalls von R-Registern. S_i hängt genau dann essenziell von x_k ab, wenn nicht für alle $b := b_0, \ldots, b_{k-1}, 0, b_{k+1}, \ldots, b_{n-1}$ die Gleichheit $S_i(b) = S_i((|b|+2^k)_2)$ gilt. Analog zum Vorgehen beim Gleichheitstest zweier Funktionen kann dies in paralleler Zeit $\mathcal{O}(\log N)$ auf Platz $\mathcal{O}(N \log N)$ geschehen.

Belegung von S_0 mit einer in R vorliegenden Funktion $f \in B_n$. Ein jeder Prozessor P_a für $a \in \mathbb{B}^n$ setzt einfach $S_0(a) := f(a)$. Die in R-Registern vorliegende Codierung von f hat höchstens Länge $t_{\mathcal{M}}(N)$ und jede der parallel ausgeführten Auswertungen verursacht nach Voraussetzung Zeit $\mathcal{O}(t_{\mathcal{M}}(N))$.

□

Korollar 7.1.7. P-Ops $\subseteq$ NC.

Unter der allgemein akzeptierten Annahme P $\neq$ NC sind P- bzw. FP-vollständige Probleme also weder in NC noch in P-Ops enthalten. Bisher gelten diese Negativresultate jedoch nur für implizite Algorithmen, deren Eingabe die charakteristische Funktion χ_I einer Eingabezeichenkette $I \in \mathbb{B}^*$ ist, die die Eingabeinstanz auf eine übliche Weise codiert. Dies könnte im Falle eines Graphen die Codierung einer Gewichtsmatrix der Länge $\Theta(|V|^2 \cdot \log B)$ oder etwa die eines Adjazenzlistenvektors bei Länge $\Theta(|V| + |E| \cdot \log(|V| \cdot B))$ sein, wobei B wieder eine obere Schranke für die Gewichtswerte sei.

Beobachtung 7.1.8. *Gehen wir davon aus, dass I die zeilenweise gelesene Adjazenzmatrix eines ungewichteten Graphen $G = (V, E)$ mit $|V| = 2^n$ ist, so existiert offenbar folgender einfacher Zusammenhang zwischen χ_I und χ_G, so wie wir es in Abschnitt 2.5 definiert haben:*

$$\chi_G(x, y) = \chi_I((|x| \cdot 2^n + |y|)_2).$$

Es gilt $\Lambda(I) = 2^{2n}$ *für die Anzahl* $n = \log|V|$ *knotencodierender Variablen. Umwandlungen zwischen* χ_I *und* χ_G *können leicht ersichtlich von einer IRAM in Zeit* $\mathcal{O}(n)$ *durchgeführt werden.*

Bei der Frage nach der Existenz von NC-Algorithmen ist zudem nicht relevant, ob wir einen ungewichteten Eingabegraphen als Adjazenzmatrix oder als Adjazenzlisten darstellen, da diese Darstellungen nur polynomielle Größenunterschiede besitzen. P- bzw. FP-vollständige Probleme auf ungerichteten Graphen haben also auch keine P-Ops-Algorithmen für das in dieser Dissertation im Vordergrund stehende Eingabeformat χ_G.

Anders sieht dies leider für gewichtete Graphen aus: Die notwendige Korrespondenz $n = \Theta(\log \Lambda(I))$ für die Anzahl $n = \Theta(\log(\max\{|V|, B\}))$ codierender Variablen eines entsprechenden impliziten Algorithmus gilt nur unter der Bedingung $B = \text{poly}(\Lambda(I))$, also für polynomiell größenbeschränkte Kantengewichte. Konkret folgt also z. B. aus Korollar 7.1.7 und der FP-Vollständigkeit der **Flussmaximierung** noch nicht, dass wir dieses Problem auf der Eingabe $\chi_G \in B_{3n}$ nicht mit $\text{polylog}(|V| \cdot B)$ funktionalen Operationen lösen können. Dies wäre nur der Fall, wenn **Flussmaximierung** für polynomiell beschränkte Kantengewichte FP-vollständig wäre; dies ist jedoch noch ein offenes Problem, da die bisher bekannte Reduktion exponentiell große Kantengewichte erzeugt (siehe Anhang A.4.4 in Greenlaw u. a. (1995)).

Ebenso ist noch offen, ob die **0-1-Flussmaximierung** FP-vollständig oder in NC enthalten ist (siehe Anhang B.9.6 in Greenlaw u. a. (1995)). Ein entsprechender impliziter Algorithmus würde diese offenbar schwierige Frage der Komplexitätstheorie lösen, und somit verwundert es nicht, dass wir in Kapitel 3 für Algorithmus 3.3.1 nicht imstande waren, stets eine polylogarithmische Anzahl funktionaler Operationen zu garantieren.

7.2 Die Nichtexistenz von Festparameteralgorithmen

Wir bereits in der Einleitung erwähnt haben Feigenbaum u. a. (1998) gezeigt, dass selbst das grundlegende Problem ***q*-*s*-Pfad-Existenz** auf OBDD-repräsentierten Graphen PSPACE-hart ist. Auf der Suche nach geeigneten Eingabeeigenschaften, die in der Praxis trotzdem oft einen erfolgreichen Einsatz OBDD-basierter Methoden ermöglichen, haben wir als ersten kanonischen Schritt die Voll-OBDD-Breite betrachtet. Obwohl in dieser Dissertation polynomielle Laufzeiten impliziter Algorithmen (bez. der Eingabe-OBDD-Größe) auf speziellen Instanzen mit konstanter Voll-OBDD-Breite nachgewiesen werden, gelingt dies nur in Einzelfällen und unter starken Annahmen. So stellt sich die Frage, ob Graphprobleme auf OBDD-repräsentierten Graphen nicht trotz starker Breitenbeschränkungen schwierig bleiben.

Dieser Gedanke führt nun zu Erweiterungen des PSPACE-Härte-Ergebnisses von Feigenbaum u. a. (1998) und so zum Nachweis der Nichtexistenz entsprechender

Festparameteralgorithmen für viele grundlegende Graphprobleme unter der Annahme P $\neq$ PSPACE. Die betrachteten Parameter sind dabei die vollständigen OBDD-Breiten von Eingabe- und Ausgabe-OBDD. Dazu formalisieren wir zunächst den Begriff impliziter OBDD-basierter Probleme und führen eine Klasse von Problemen mit entsprechenden FPT-Algorithmen ein.

Definition 7.2.1 (OBDD-Suchprobleme). *Sei* $\Pi\colon \mathbb{B}^* \to \mathbb{B}^*$ *ein Suchproblem, dessen Eingabe* I *unter anderem aus einer Menge* $S(I)$ *von OBDDs auf* n *Variablen besteht. Es gelte zudem o. B. d. A.* $\Lambda(I) = \Theta(\max\{n, \sum_{P\in S(I)} \text{size}(P)\})$. *Entsprechend gelte für jede gültige Ausgabe* $O \in \Pi(I)$ *und ihre jeweilige OBDD-Menge* $S(O)$ $\Lambda(O) = \Theta(\max\{n, \sum_{P\in S(O)} \text{size}(P)\})$. Π *heißt im Folgenden* OBDD-Suchproblem. *Für* $\Pi(I) \in \mathbb{B}$ *heißt* Π *auch* OBDD-Entscheidungsproblem.

Die Gesamtgröße von Eingabe und Ausgabe soll also im Wesentlichen der der jeweiligen OBDD-Menge entsprechen. Stets sollen jedoch Zusatzinformationen der Größe $\mathcal{O}(n)$ erlaubt und Algorithmen eine Laufzeit von $\mathcal{O}(n)$ gestattet sein. Die in Abschnitt 2.5 eingeführte Sichtweise auf implizite Graphprobleme und -algorithmen war an einer konsistenten und möglichst kanonischen Übertragung von Problemen und Algorithmen vom klassischen algorithmischen Szenario in das impliziter Darstellungen ausgerichtet. Diese Entwurfsrichtlinien für implizite Algorithmen sollten vernünftige Annahmen über Variablenordnung, binäre Codierung und erlaubte Operationen umfassen. Unser neues Modell der OBDD-Probleme soll hingegen unabhängig von Graphproblemen eine Basis für die sinnvolle und möglichst allgemeine Übertragung des Festparameterbegriffs in die Welt OBDD-basierter impliziter Algorithmen sein. Im Gegensatz zu unserer ersten Berührung mit diesem Gebiet in Abschnitt 5.1 sollen die FPT-Eigenschaften hier nun für alle Variablenordnungen gelten; so reicht jeweils eine Gegenbeispielvariablenordnung, um Negativresultate zu erzielen.

Definition 7.2.2 (I-FPT-T). I-FPT-T *ist die Klasse der OBDD-Suchprobleme* Π, *für die ein Algorithmus* A *mit folgenden Eigenschaften existiert:* A *berechnet ein* $O \in \Pi(I)$ *für jede* Π*-Eingabe* I *in Zeit und auf Platz* $\alpha(w) \cdot \text{poly}(\Lambda(I))$ *für ein geeignetes* $\alpha\colon \mathbb{N} \to \mathbb{N}$, *wenn* w *eine obere Schranke für die maximale vollständige Breite der OBDDs in* $S(I)$ *ist.*

Definition 7.2.3 (IO-FPT-T). IO-FPT-T *ist die Klasse der OBDD-Suchprobleme* Π, *für die ein Algorithmus* A *mit folgenden Eigenschaften existiert:* A *berechnet ein* $O \in \Pi(I)$ *für jede* Π*-Eingabe* I *in Zeit und auf Platz* $\alpha(w)\cdot\text{poly}(\Lambda(I))$ *für ein geeignetes* $\alpha\colon \mathbb{N} \to \mathbb{N}$, *wenn* w *eine obere Schranke für die maximale vollständige Breite der OBDDs in* $S(I) \cup \bigcup_{O\in\Pi(I)} S(O)$ *ist.*

„I“ bzw. „IO“ kennzeichnen dabei, ob nur die Voll-OBDD-Breiten der Eingabe oder auch die der Ausgabe in den Festparameter einfließen. „T“ wie „Time“ kennzeichnet, dass die Laufzeit (und damit auch der Platz) beschränkt sein muss. Wir werden später auch andere platzbezogene Varianten betrachten. Algorithmus 5.1.2 war zwar ein

Festparameteralgorithmus bez. der Voll-OBDD-Breite für $\pi^{\mathrm{id}}_{k,n}$, für die Zugehörigkeit zu IO-FPT-T reicht dies jedoch nicht, da wir uns im Sinne einfacherer Beweise im Folgenden nicht auf eine Variablenordnung festlegen wollen. Wir bezeichnen nun mit den in Abschnitt 2.1.1 eingeführten Problemnamen stets die impliziten OBDD-basierten Varianten.

Fakt 7.2.4. *Offenbar gilt die Beziehung* I-FPT-T $\subseteq$ IO-FPT-T.

Anmerkung 7.2.5. *Die Definition von Festparameteralgorithmen (siehe Abschnitt 2.2) sieht eigentlich den Festparameter als Teil der Eingabe vor. In Definition 7.2.3 ist w jedoch kein Teil sondern eine Eigenschaft der Eingabe. Diese formale Unstimmigkeit kann jedoch leicht aufgelöst werden, indem ein entsprechender FPT-Algorithmus eine zusätzliche Eingabe w' erhält und anfangs prüft, ob $w = w'$. Ist dies nicht der Fall, meldet er eine inkorrekt codierte Eingabe und bricht ab. Auf diesen Kunstgriff werden wir aber im Folgenden zugunsten einer einfacheren Beschreibung verzichten.*

Wir interessieren uns zunächst also nur für die Existenz von FPT-Algorithmen, deren Festparameter die Voll-OBDD-Breite des Eingabe-OBDD bez. natürlicher gemischter Variablenordnungen ist. Dabei hilft folgender Zusammenhang (siehe z. B. Downey und Fellows (1999)):

Theorem 7.2.6. *Sei Π' ein parametrisiertes Problem mit Eingaben $(I', k') \in \Gamma^* \times \mathbb{N}$ für ein endliches Alphabet Γ. Existiert für jedes $\Pi \in$ PSPACE eine polynomiell berechenbare Turingreduktion f auf Π' und eine Konstante k^*, sodass f stets Π'-Eingaben mit $k' \leq k^*$ erzeugt, so hat Π' keine FPT-Algorithmen, es sei denn es gilt* P = PSPACE.

Verkürzt sagen wir, dass Π' bereits für konstanten Parameter PSPACE-hart ist. Können wir dies also für ein OBDD-basiertes Problem ($\Gamma = \mathbb{B}$) und den Parameter der Voll-OBDD-Breite der Eingabe nachweisen, so folgt direkt die Nichtexistenz von I-FPT-T-Algorithmen unter der Annahme P $\neq$ PSPACE.

Wir beginnen unsere Arbeit mit einer modifizierten Version des PSPACE-Härte-Beweises von Feigenbaum u. a. (1998) für zwei künstliche Probleme, deren Eingeschränktheit wir in späteren Beweisen benötigen werden.

Definition 7.2.7.

(a) ***q*-*s*-Pfad-Existenz*** *sei das Problem* ***q*-*s*-Pfad-Existenz** *auf gerichteten kreisfreien planaren Graphen $G = (V, E)$, deren umgekehrter Graph $G^{\mathrm{R}} := (V, E^{\mathrm{R}})$ mit $(u, v) \in E :\Leftrightarrow (v, u) \in E^{\mathrm{R}}$ aus genau zwei gewurzelten Bäumen besteht, und in denen jeder q-s-Pfad, falls existent, ein längster Pfad in G bez. seiner Kantenanzahl ist.*

(b) **Ungerichtete *q*-*s*-Pfad-Existenz*** *sei das Problem* **q-s-Pfad-Existenz** *auf ungerichteten kreisfreien planaren Graphen $G = (V, E)$, die aus genau zwei Bäumen bestehen, und in denen jeder q-s-Pfad, falls existent, ein längster Pfad in G bez. seiner Kantenanzahl ist.*

Wir werden in den folgenden Reduktionen stets darauf achten, dass zusätzlich entstehende isolierte Knoten die Korrektheit nicht beeinflussen. Somit können wir die formal vorhandene Funktion $\chi_V(x) = 1$ vernachlässigen.

Theorem 7.2.8. *Die Probleme* **q-s-Pfad-Existenz*** *und* **Ungerichtete q-s-Pfad-Existenz*** *auf OBDD-repräsentierten Graphen $G = (V, E)$ sind bereits für Eingaben $\pi^{\mathrm{id}}_{2,n}$-OBDD$[\chi_G]$ mit konstanter Voll-OBDD-Breite und $n = \log|V| + \mathcal{O}(1)$* PSPACE*-hart.*

Beweis: Sei $\Pi \in$ PSPACE und $\mathcal{M}$ eine Turingmaschine für Π mit Platzschranke $p(\Lambda(I)) = \mathrm{poly}(\Lambda(I))$ für Eingaben $I \in \mathbb{B}^*$, Bandalphabet Γ und Zustandsmenge Q. Eine Konfigurationscodierung C sei eine binäre Codierung des Bandinhalts der Bandpositionen $0, \ldots, p(\Lambda(I)) - 1$ für $m = \lceil \log(|\Gamma| + |Q|) \rceil$, in die zudem die Nummer $(q)_2$ des aktuellen Zustands $q \in Q$ an der aktuellen Position des Lese-/Schreibkopfes eingefügt ist. An diese $m(p(\Lambda(I)) + 1)$ Bits hängen wir einen Rechenschrittzähler, für den offenbar ebenfalls $m(p(\Lambda(I)) + 1)$ Bits reichen, da nur $2^{m(p(\Lambda(I))+1)}$ verschiedene Konfigurationscodierungen existieren. Insgesamt hat C also die Länge $2m(p(\Lambda(I)) + 1)$. Die von C codierten Buchstaben/Zustände an den Positionen $i - 1$, i und $i + 1$ für die aktuelle Kopfposition i sei $B(C) \in \{\Gamma, Q\}^4$, die codierte Rechenschrittnummer sei $Z(C)$.

Wir können o. B. d. A. annehmen, dass $\mathcal{M}$ seine Berechnung stets in einer von zwei definierten Konfigurationen C_0 für Verwerfung bzw. C_1 für Akzeptanz der Eingabe I mit $Z(C_0) = Z(C_1) = p(\Lambda(I)) - 1$ beendet. $G_N = (V_N, E_N)$ sei der zugehörige Graph der Konfigurationscodierungen für Eingaben I der Länge N, es ist also $V_N = \mathbb{B}^{(2m(p(N)+1)}$. Zwei Knoten/Konfigurationen C und C' sind sinnvollerweise genau dann adjazent, wenn ihre entsprechenden Bandinhalte, Kopfpositionen und Zustände einen korrekten Rechenschritt von $\mathcal{M}$ widerspiegeln und zudem $Z(C') = Z(C) + 1$ gilt. Wir führen jedoch zwei Sonderregeln ein: Knoten C mit $Z(C) < p(N) - 1$, die ungültige Codierungen sind und daher den Ausgrad 0 besitzen, sind zu ihrem Pendant C' mit $Z(C') = Z(C) + 1$ adjazent, welches abgesehen vom Rechenschrittzähler identisch ist. Knoten $C \notin \{C_1, C_0\}$ mit $Z(C) = p(N) - 1$ seien des Weiteren adjazent zu C_0.

Wir betrachten zunächst die Struktur von G_N: Jeder Knoten außer den Endknoten C_1 und C_0 hat den Ausgrad 1. Die beiden Endknoten haben Ausgrad 0. G_N ist kreisfrei und planar. Jeder Knoten liegt auf einem eindeutigen Pfad zu einem der beiden Endknoten. C_1 ist genau dann von einem Knoten C_I mit $Z(C_I) = 0$ erreichbar, der einer Startkonfiguration bei Eingabe I entspricht, wenn $I \in \Pi$ gilt. So ein Pfad hat

dann maximale Länge $p(N)-1$. Durch Umkehrung aller Kanten erhalten wir offenbar einen Wald aus genau zwei gewurzelten Bäumen.

Wir zeigen, dass die charakteristische Funktion χ_{G_N} konstante Voll-OBDD-Breite bez. $\pi^{\mathrm{id}}_{2,n}$ für $n = \log|V_N| = 2m(p(N)+1) = \mathcal{O}(p(N))$ hat. Dazu betrachten wir die Hilfsfunktionen f_1, f_2 und f_3. Funktion $f_1(C)$ soll prüfen, ob C eine gültige Konfigurationscodierung ist; dies ist der Fall, wenn sie genau eine Zustandscodierung enthalten und ansonsten ausschließlich gültige Buchstabencodierungen. Für einen Ausschnitt $x \in \mathbb{B}^m$ von C, der eine Position der Konfiguration codiert, kann die Zugehörigkeit zu Γ oder Q durch Vergleiche $a \leq |x| \leq b$ für entsprechende Wertebereiche $\{a,\ldots,b\}$ geschehen, die wiederum gemäß Korollar 3.7.5 jeweils konstante Voll-OBDD-Breite haben. Dies gilt dann auch für f_1, da die einzelnen Vergleiche auf paarweise disjunkten Variablenvektoren definiert sind.

Funktion $f_2(C,C')$ soll prüfen, ob C' deterministischer Nachfolger von C ist. Dies kann von einem OBDD so realisiert werden, dass beim sequenziellen Lesen der gemischten Bits von C und C' durch Besuch entsprechender OBDD-Knoten stets das Paar $(B(C),B(C')) \in \{\Gamma,Q\}^8$ der letzten vier gelesenen Konfigurationspositionen von C und C' speichert. Für jedes solches Paar wird nun unabhängig voneinander entschieden, ob es einen korrekten lokalen Übergang in diesem Fenster der Länge 4 darstellt. Die Anzahl 2^{8m} solcher Paare ist somit ebenfalls eine obere Schranke für die Voll-OBDD-Breite von f_2.

Funktion f_3 soll schließlich alle Bits von C und C' mit Ausnahme der jeweils $m(p(N)+1)$ Zählerbits auf Gleichheit prüfen. Genau wie vollständige Vergleiche zweier gemischt gelesener Argumente verursacht auch dieser eingeschränkte Vergleich nur konstante Voll-OBDD-Breite.

Die Gesamtfunktion χ_G ergibt sich dann nach obiger Definition durch

$$\begin{aligned}\chi_{G_N}(C,C') := {} & [f_1(C) \wedge f_1(C') \wedge f_2(C,C') \wedge (Z(C') = Z(C)+1)] \\ & \vee \left[\overline{f_1(C)} \wedge f_3(C,C') \wedge (Z(C') = Z(C)+1 < p(N))\right] \\ & \vee [(C_1 \neq C \neq C_0) \wedge (Z(C) = p(N)-1) \wedge (C' = C_0)]\,.\end{aligned}$$

Offenbar kann wieder Theorem 3.7.4 angewandt werden, und auch χ_G hat ein $\pi^{\mathrm{id}}_{2,n}$-OBDD konstanter Voll-OBDD-Breite, das in Zeit $\mathrm{poly}(\Lambda(I))$ berechnet werden kann. Dies gilt dementsprechend ebenso für die symmetrische Variante $\chi_{G'_N}(C,C') := \chi_{G_N}(C,C') \vee \chi_{G_N}(C',C)$. Der dargestellte Graph G' kann auch als ungerichteter kreisfreier Graph aufgefasst werden, der aus zwei Zusammenhangskomponenten besteht. Somit haben wir eine Eingabe für ***q*-*s*-Pfad-Existenz*** als auch für **Ungerichtete *q*-*s*-Pfad-Existenz*** konstruiert.

Als Startknoten q wählen wir in beiden Fällen C_I, als Endknoten s wählen wir C_1. I ist nun genau dann in Π, wenn ein q-s-Pfad in G'_N existiert. Wir haben in polynomieller Zeit Eingabeinstanzen mit konstanter Voll-OBDD-Breite konstruiert. □

Korollar 7.2.9. (Ungerichtete) *q-s*-Pfad-Existenz* $\notin$ I-FPT-T, *es sei denn* P = PSPACE.

Damit haben wir das Resultat von Feigenbaum u. a. (1998) in den Bereich der parametrisierten Komplexität ausgeweitet. Um nach dieser Master-Reduktion komfortabler vorgehen zu können, definieren wir einen entsprechenden Reduktionsbegriff.

Definition 7.2.10. *Seien* Π *und* Π' *OBDD-Probleme.* Π *heißt* implizit reduzierbar *auf* Π' *(Notation* $\Pi \prec \Pi'$*), wenn eine Turingreduktion von* Π *auf* Π' *existiert, die auf* Π*-Eingaben* I *mit Variablenordnung* $\pi^{\mathrm{id}}_{k,n}$ *und maximaler Voll-OBDD-Breite* w *die Laufzeit* $\alpha(w) \cdot \mathrm{poly}(\Lambda(I))$ *besitzt und dabei* Π'*-Orakeleingaben mit Variablenordnung* $\pi^{\mathrm{id}}_{k',n'}$ *und maximaler Voll-OBDD-Breite* $w' \leq \beta(w)$ *erzeugt. Dabei seien* $k' = \mathcal{O}(k)$, $n' = \mathcal{O}(n)$ *und* α *und* β *geeignete Funktionen* $\mathbb{N} \to \mathbb{N}$. *Die Kosten für eine Orakelbefragung seien 1.*

Implizite Reduktionen können auch als FPT-Reduktionen aufgefasst werden, wie sie in der parametrisierten Komplexität üblicherweise definiert werden: Wachstum von Problemgröße und Parameter (hier w) müssen sinnvoll beschränkt sein, und die Reduktion muss selbst ein FPT-Algorithmus bez. natürlicher gemischter Variablenordnungen sein. Da das in Theorem 7.2.8 konstruierte OBDD ebenfalls die Variablenordnung $\pi^{\mathrm{id}}_{2,n}$ hat, gelten leicht ersichtlich die folgenden Eigenschaften:

Theorem 7.2.11.

(a) **(Ungerichtete) *q-s*-Pfad-Existenz*** $\prec \Pi' \Rightarrow \Pi' \notin$ I-FPT-T, *es sei denn* P = PSPACE.

(b) Die Relation $\prec$ *ist transitiv.*

Beweis: Zu (a): Nach Theorem 7.2.8 sind die Probleme ***q-s*-Pfad-Existenz*** und **Ungerichtete *q-s*-Pfad-Existenz*** auf OBDD-repräsentierten Graphen $G = (V, E)$ für Eingaben $\pi^{\mathrm{id}}_{2,n}$-OBDD$[\chi_G] =: I$ mit konstanter Voll-OBDD-Breite w und $n = \log|V| + \mathcal{O}(1)$ Variablen PSPACE-hart. Wir nehmen an, Π' sei in I-FPT-T, habe also einen FPT-Algorithmus A mit Laufzeit $\gamma(w') \cdot \mathrm{poly}(\Lambda(I'))$ auf Eingaben I' mit maximaler Voll-OBDD-Breite w' für ein geeignetes $\gamma\colon \mathbb{N} \to \mathbb{N}$. Es existiert nun nach Voraussetzung eine implizite Reduktion f von **(Ungerichtete) *q-s*-Pfad-Existenz*** auf Π' mit Laufzeit $\alpha(w) \cdot \mathrm{poly}(\Lambda(I))$ für geeignetes $\alpha\colon \mathbb{N} \to \mathbb{N}$; f rufe das Π'-Orakel nach Definition höchstens $\alpha(w) \cdot \mathrm{poly}(\Lambda(I))$-mal mit Eingaben I' der maximalen Voll-OBDD-Breite $w' \leq \beta(w)$ bez. $\pi^{\mathrm{id}}_{k',n'}$ für $k' = \mathcal{O}(1)$, $n' = \mathcal{O}(n)$ und geeignetes $\beta\colon \mathbb{N} \to \mathbb{N}$ auf. Wir wenden f auf I an und realisieren die Orakelaufrufe durch Aufrufe von A. Reduktion f hat also die Laufzeit

$$\alpha(w) \cdot \mathrm{poly}(\Lambda(I)) + \gamma(\beta(w)) \cdot \mathrm{poly}(\Lambda(I')) \cdot \alpha(w) \cdot \mathrm{poly}(\Lambda(I)).$$

Es gelten aber $w = \mathcal{O}(1)$ und $\Lambda(I') \leq \beta(w) \cdot n' = \mathcal{O}(\Lambda(I))$, und so lösen f und A ein PSPACE-hartes Problem in Zeit $\text{poly}(\Lambda(I))$.

Zu (b): Es gelte $\Pi \prec \Pi' \prec \Pi''$ für OBDD-Suchprobleme Π, Π' und Π''. Die Variablenordnung der Π-Eingaben I sei $\pi^{\text{id}}_{k,n}$. Seien f und g entsprechende implizite Reduktionen mit entsprechend definierten $k' = \mathcal{O}(k)$, $k'' = \mathcal{O}(k')$, $n' = \mathcal{O}(n)$, $n'' = \mathcal{O}(n')$, α, α', β sowie β' gemäß Definition 7.2.10. Wir bilden die Komposition $h = f \circ g$, indem wir das Π'-Orakel in f durch g ersetzen. Auf einer Π-Eingabe I mit Voll-OBDD-Breite w bez. $\pi^{\text{id}}_{k,n}$ hat h also Laufzeit

$$\alpha(w) \cdot \text{poly}(\Lambda(I)) + \alpha'(\beta(w)) \cdot \text{poly}(\beta(w) \cdot n') \cdot \alpha(w) \cdot \text{poly}(\Lambda(I))$$

und ruft das Π''-Orakel mit Eingaben der maximalen Voll-OBDD-Breite $w'' \leq \beta'(\beta(w)) =: \beta^*(w)$ bez. $\pi^{\text{id}}_{k'',n''}$ auf.

Für geeignetes α^* und β^* hat h also wieder Laufzeit $\alpha^*(w) \cdot \text{poly}(\Lambda(I))$ und ruft das Π''-Orakel mit Eingaben der Voll-OBDD-Breite $w'' \leq \beta^*(w)$ bez. $\pi^{\text{id}}_{k'',n''}$ auf. Es sind $k'' = \mathcal{O}(k)$ und $n'' = \mathcal{O}(n)$ und h ist eine implizite Reduktion von Π auf Π''. □

Theorem 7.2.12. Ungerichtete *q*-*s*-Pfad-Existenz* $\prec$ Kreis-Existenz.

Beweis: In einem ungerichteten kreisfreien Graphen G befinden sich zwei Knoten q und s genau dann in der gleichen Zusammenhangskomponente, wenn der Graph nach Einfügung der Kante $\{q, s\}$ einen Kreis enthält. Sei $\chi_G(x, y)$ die zu transformierende Eingabefunktion mit Voll-OBDD-Breite w bez. $\pi^{\text{id}}_{2k,n}$. Die Reduktion berechnet aus der impliziten Eingabe $\chi_G(x, y)$ also $\chi_{G'}(x, y) := \chi_G(x, y) \vee [E(x, y) \wedge E(y, x)]$ für $E(x, y) := (v_{|x|} = q) \wedge (v_{|y|} = s)$. Gemäß Theorem 3.7.4 kann $\chi_{G'}$ in Zeit $\alpha(w) \cdot \text{poly}(n)$ berechnet werden und hat höchstens Voll-OBDD-Breite $\beta(w)$ bez. $\pi^{\text{id}}_{2k,n}$ für geeignete Funktionen $\alpha, \beta \colon \mathbb{N} \to \mathbb{N}$. □

Theorem 7.2.13.

$$\textbf{Ungerichtete } \textbf{\textit{q}-\textit{s}-Pfad-Existenz}^* \prec \begin{array}{l} \textbf{Zusammenhang}, \\ \textbf{Starker Zusammenhang}. \end{array}$$

Beweis: In einem ungerichteten Graphen G, der aus zwei Zusammenhangskomponenten besteht, befinden sich zwei Knoten q und s genau dann in unterschiedlichen Zusammenhangskomponenten, wenn G durch Einfügung der Kante $\{q, s\}$ zusammenhängend wird. Die charakteristische Funktion eines ungerichteten Graphen G gleicht der eines entsprechenden gerichteten symmetrischen Graphen H. Die Knoten q und s liegen also genau dann in der gleichen Zusammenhangskomponente von G, wenn sie in der gleichen starken Zusammenhangskomponente von H liegen. Die Aussage des Theorems folgt nun analog zum Beweis von Theorem 7.2.12. □

Theorem 7.2.14. Zusammenhang $\prec$ Eulerkreis-Existenz.

Beweis: Wir konstruieren aus dem in der Reduktionseingabe $\chi_G \in B_{2kn}$ implizit dargestellten Graphen $G = (V, E)$ die Funktion $\chi_{G'} \in B_{2(2k+1)n}$ eines Bildgraphen $G' = (V', E')$ mit

$$V' := V \cup \{u_{ij} \mid 0 \leq i < j < |V|\} \cup \{a_i, b_i \mid 0 \leq i < |V|\}.$$

E' enthält E und die Kanten $\{v_i, a_i\}$, $\{a_i, b_i\}$ sowie $\{b_i, v_i\}$ für alle $i < |V|$. Des Weiteren enthält E' genau dann die Kanten $\{v_i, u_{ij}\}$ und $\{u_{ij}, v_j\}$, wenn $\{v_i, v_j\} \in E$. Wir bemerken, dass alle Knoten in G' geraden Grad besitzen und G' genau dann zusammenhängend ist, wenn G zusammenhängend ist. Somit besitzt G' gemäß Beobachtung 2.1.9 genau dann einen Eulerkreis, wenn G zusammenhängend ist. Diese Konstruktion stammt von Chandra u. a. (1984).

Es bleibt zu zeigen, dass die Reduktion auf OBDD-Ebene gemäß der Definition impliziter Reduktionen realisierbar ist. Sei $\pi^{\mathrm{id}}_{2k,n}$ gemäß Definition 7.2.10 die Variablenordnung des Eingabe-OBDDs für χ_G. Wir definieren $\chi_{G'}$ auf $2(2k+1)n$ Variablen mit der Ordnung $\pi^{\mathrm{id}}_{2(2k+1),n}$. Eine Knotennummer $x \in \mathbb{B}^{(2k+1)n}$ besteht aus zwei Komponenten aus jeweils k Argumenten der Länge n, um Knotennummern i und j aus dem Intervall $\{0, \ldots, |V| - 1\}$ darstellen zu können, sowie einem Argument der Länge n, um den Knotentyp (also v, u, a oder b) codieren zu können. Dies ist offenbar für $n \geq 2$ möglich. Wir referenzieren diese drei Komponenten einer Knotennummer x durch $i(x)$, $j(x)$ und $T(x)$ und definieren

$$\begin{aligned} H(x,y) := \big[&(T(x) = T(y) = v) \wedge \chi_G(i(x), i(y))\big] \\ &\vee \big[(T(x) = v) \wedge (T(y) = a) \wedge (i(x) = i(y))\big] \\ &\vee \big[(T(x) = a) \wedge (T(y) = b) \wedge (i(x) = i(y))\big] \\ &\vee \big[(T(x) = b) \wedge (T(y) = v) \wedge (i(x) = i(y))\big] \\ \vee \big[&(T(x) = v) \wedge (T(y) = u) \wedge (i(x) = i(y)) \wedge \chi_G(i(y), j(y))\big] \\ \vee \big[&(T(x) = u) \wedge (T(y) = v) \wedge (j(x) = j(y)) \wedge \chi_G(i(x), j(x))\big]. \end{aligned} \quad (7.2.1)$$

Nach dem symmetrischen Abschluss $\chi_{G'}(x, y) := H(x, y) \vee H(y, x)$ stellt $\chi_{G'}$ offenbar korrekt die Kanten des Graphen G' dar; zusätzliche isolierte Knoten, die durch die Codierung entstehen, beeinflussen die Existenz eines Eulerkreises nicht. Gleichheitstests zweier Argumente wie etwa $i(x) = i(y)$ haben bei gemischten Variablenordnungen, wie schon oft in dieser Dissertation diskutiert, konstante Voll-OBDD-Breite. Dies gilt daher offenbar auch für Typprüfungen wie $T(x) = v$, wobei wir für jede der vier Typen eine beliebige entsprechende Belegung der $T(x)$-Variablen festlegen können. Allerdings muss im Falle $T(x) \neq u$ garantiert werden, dass die überflüssigen Variablen $j(x)$ zur Codierung des j-Indexes einen einheitlichen Wert haben. So sei jede Typprüfung zusätzlich konjunktiv mit $T(x) \neq u \Rightarrow |j(x)| = 0$ verknüpft.

Jede der konstant vielen in Gleichung (7.2.1) auftauchenden Bausteinfunktionen hat also konstante Voll-OBDD-Breite bez. $\pi^{\mathrm{id}}_{2(2k+1),n}$. Funktion χ_G habe Voll-OBDD-Breite w bez. $\pi^{\mathrm{id}}_{2k,n}$. Gemäß Theorem 3.7.4 kann $\chi_{G'}$ in Zeit $\alpha(w) \cdot \mathrm{poly}(n)$ konstruiert

werden und hat höchstens Voll-OBDD-Breite $\beta(w)$ bez. $\pi^{\mathrm{id}}_{2(2k+1),n}$ für geeignete Funktionen $\alpha, \beta\colon \mathbb{N} \to \mathbb{N}$. □

Theorem 7.2.15. Ungerichtete *q*-*s*-Pfad-Existenz* $\prec$ Bipartitheit.

Beweis: Wir konstruieren aus dem in der Reduktionseingabe $\chi_G \in B_{2kn}$ implizit dargestellten Graphen $G = (V, E)$ die Funktion $\chi_{G'} \in B_{2(2k+1)n}$ eines Bildgraphen $G' = (V', E')$ mit $V' := (V \cup E) \times \{1, 2\} \cup \{w\}$. E' enthält die Kanten $\big\{(v, r), (e, \ell)\big\}$ für $e \in E$, $v \in V \cap e$, und $r = \ell$. Außerdem enthält E' die Kanten $\big\{(q, 1), (q, 2)\big\}$, $\big\{(s, 1), w\big\}$ und $\big\{(s, 2), w\big\}$ für $q, s \in V$. G' enthält nun genau dann einen Kreis ungerader Länge (ist also nicht bipartit), wenn q und s in G durch einen Pfad verbunden sind. Diese Konstruktion stammt von Jones u. a. (1976).

Es bleibt zu zeigen, dass die Reduktion auf OBDD-Ebene gemäß der Definition impliziter Reduktionen realisierbar ist. Sei $\pi^{\mathrm{id}}_{2k,n}$ gemäß Definition 7.2.10 die Variablenordnung des Eingabe-OBDDs für χ_G. Wir definieren $\chi_{G'}$ auf $2(2k+1)n$ Variablen mit Ordnung $\pi^{\mathrm{id}}_{2(2k+1),n}$. Eine Knotennummer $x \in \mathbb{B}^{(2k+1)n}$ besteht aus zwei Komponenten aus jeweils k Argumenten der Länge n, um Knotennummern i und j aus dem Intervall $\{0, \ldots, |V| - 1\}$ darstellen zu können, sowie einem Argument der Länge n, um zum einen den Knotentyp (also v, u, a oder b) wie auch den Index der Graphkopie (also 1 oder 2) codieren zu können. Dies ist offenbar für $n \geq 4$ möglich. Wir referenzieren diese vier Komponenten einer Knotennummer x durch $i(x)$, $j(x)$, $T(x)$ und $c(x)$ und definieren

$$\begin{aligned}
\chi_{G'}(x, y) := \\
&\big[(T(x) = v) \wedge (T(y) = e) \wedge (i(x) = i(y)) \wedge (c(x) = c(y)) \wedge \chi_G(i(y), j(y))\big] \\
\vee\, &\big[(T(x) = e) \wedge (T(y) = v) \wedge (j(x) = j(y)) \wedge (c(x) = c(y)) \wedge \chi_G(i(x), j(x))\big] \\
\vee\, &\big[(T(x) = T(y) = v) \wedge (v_{|i(x)|} = v_{|i(y)|} = q) \wedge (c(x) \neq c(y))\big] \\
\vee\, &\big[(T(x) = v) \wedge (T(y) = w) \wedge (v_{|i(x)|} = s)\big]. \qquad (7.2.2)
\end{aligned}$$

Nach dem symmetrischen Abschluss $\chi_{G'}(x, y) := H(x, y) \vee H(y, x)$ stellt $\chi_{G'}$ offenbar korrekt die Kanten des Graphen G' dar; zusätzliche isolierte Knoten, die durch die Codierung entstehen, beeinflussen die Bipartitheit nicht. Gleichheitstests zweier Argumente wie etwa $i(x) = i(y)$ haben bei gemischten Variablenordnungen konstante Voll-OBDD-Breite. Dies gilt daher offenbar auch für Typprüfungen wie $T(x) = v$ und Kopienindexvergleiche, wobei wir jeder der acht verschiedenen zu codierenden Kombinationen eine beliebige entsprechende Variablenbelegung festlegen können. Allerdings muss im Falle $T(x) = v$ die Bedingung $|j(x)| = 0$ ebenso garantiert werden wie $|i(x)| = |j(x)| = 0$ für $T(x) = w$. So sei jede Typprüfung zusätzlich konjunktiv mit $T(x) = v \Rightarrow |j(x)| = 0$ und $T(x) = w \Rightarrow |i(x)| = |j(x)| = 0$ verknüpft.

Jede der konstant vielen in Gleichung (7.2.2) auftauchenden Bausteinfunktionen hat also konstante Voll-OBDD-Breite bez. $\pi^{\mathrm{id}}_{2(2k+1),n}$. Funktion χ_G habe Voll-OBDD-Breite w bez. $\pi^{\mathrm{id}}_{2k,n}$. Gemäß Theorem 3.7.4 kann $\chi_{G'}$ in Zeit $\alpha(w) \cdot \mathrm{poly}(n)$ konstruiert

werden und hat höchstens Voll-OBDD-Breite $\beta(w)$ bez. $\pi^{\mathrm{id}}_{2(2k+1),n}$ für geeignete Funktionen $\alpha, \beta\colon \mathbb{N} \to \mathbb{N}$. □

Theorem 7.2.16. Ungerichtete *q*-*s*-Pfad-Existenz* $\prec$ Planarität.

Beweis: Wir konstruieren aus dem in der Reduktionseingabe $\chi_G \in B_{2kn}$ implizit dargestellten Graphen $G = (V, E)$ die Funktion $\chi_{G'} \in B_{2(k+1)n}$ eines Bildgraphen $G' = (V', E')$ mit $V' := V \cup \{w_1, w_2, w_3\}$ und definieren des Weiteren $w_4 := q$ und $w_5 := s$. Wir erhalten E', indem wir die Kanten des vollständigen Graphen auf den Knoten $w_1, \ldots, w_5$ zu E hinzufügen, ausgenommen die Kanten $\{w_4, w_5\}$. Da G nach Voraussetzung planar ist, ist auch G' genau dann planar, wenn es keinen Pfad zwischen $q = w_4$ und $s = w_5$ gibt. Diese Konstruktion stammt von Balcázar und Lozano (1989).

Es bleibt zu zeigen, dass die Reduktion auf OBDD-Ebene gemäß der Definition impliziter Reduktionen realisierbar ist. Sei $\pi^{\mathrm{id}}_{2k,n}$ gemäß Definition 7.2.10 die Variablenordnung des Eingabe-OBDDs für χ_G. Wir definieren $\chi_{G'}$ auf $2(k+1)n$ Variablen mit Ordnung $\pi^{\mathrm{id}}_{2(k+1),n}$. Eine Knotennummer $x \in \mathbb{B}^{(k+1)n}$ besteht aus einer Komponente $i(x)$ aus k Argumenten der Länge n, um eine Knotennummer $i \in \{0, \ldots, |V| - 1\}$ darstellen zu können, sowie einem Argument $T(x)$ der Länge n, um die drei weiteren Knoten w_1, w_2, w_3 darzustellen. Für $|i(x)| = 0$ und $|T(x)| = j \in \{1, 2, 3\}$ codiere x also w_j, für $|T(x)| = 0$ codiere x den Knoten $v_{|i(x)|} \in V$. Dann lässt sich $\chi_{G'}$ folgendermaßen darstellen:

$$\begin{aligned}\chi_{G'}(x,y) := {} & \big[(|T(x)| = |T(y)| = 0) \wedge \chi_G(i(x), i(y))\big] \\ & \vee \big[(|i(x)| = 0) \wedge (|T(x)| \in \{1,2,3\}) \wedge (|T(y)| = 0) \wedge (v_{|i(x)|} \in \{q, s\})\big] \\ & \vee \big[(|i(x)| = |i(y)| = 0) \wedge (|T(x)|, |T(y)| \in \{1,2,3\})\big]. \quad (7.2.3)\end{aligned}$$

Nach dem symmetrischen Abschluss $\chi_{G'}(x, y) := H(x, y) \vee H(y, x)$ stellt $\chi_{G'}$ offenbar korrekt die Kanten des Graphen G' dar; zusätzliche isolierte Knoten, die durch die Codierung entstehen, beeinflussen die Planarität nicht. Gleichheitstests wie etwa $i(x) = i(y)$ oder $|T(x)| = 0$ haben bei gemischten Variablenordnungen konstante Voll-OBDD-Breite. Mengenausdrücke wie $|T(x)| \in \{1, 2, 3\}$ können offenbar durch die Disjunktion konstant vieler Gleichheitstest ersetzt werden.

Jede der konstant vielen in Gleichung (7.2.3) auftauchenden Bausteinfunktionen hat also konstante Voll-OBDD-Breite bez. $\pi^{\mathrm{id}}_{2(k+1),n}$. Funktion χ_G habe Voll-OBDD-Breite w bez. $\pi^{\mathrm{id}}_{2k,n}$. Gemäß Theorem 3.7.4 kann $\chi_{G'}$ in Zeit $\alpha(w) \cdot \mathrm{poly}(n)$ konstruiert werden und hat höchstens Voll-OBDD-Breite $\beta(w)$ bez. $\pi^{\mathrm{id}}_{2(k+1),n}$ für geeignete Funktionen $\alpha, \beta\colon \mathbb{N} \to \mathbb{N}$. □

Theorem 7.2.17. Ungerichtete *q*-*s*-Pfad-Existenz* $\prec$ Zweizusammenhang.

Beweis: Der Eingabegraph G besteht nach Voraussetzung aus zwei Zusammenhangskomponenten. Wir fügen die Kante $\{q, s\}$ sowie für einen beliebig gewählten Knoten

$u \notin \{q, s\}$ alle Kanten $\{u, w\}$ für alle $w \in V$ ein. Lagen q und s in G in der gleichen Zusammenhangskomponente, so ist u ein Schnittpunkt in G' – seine Entfernung lässt G' wieder in zwei Komponenten zerfallen. Anderenfalls besitzt G' keinen Schnittpunkt und G' ist zweizusammenhängend.

Die Reduktion berechnet also analog zum Beweis von Theorem 7.2.12 aus der impliziten Eingabe $\chi_G(x, y)$ die Funktion $\chi_{G'}(x, y) := \chi_G(x, y) \vee [E(x, y) \wedge E(y, x)]$ für $E(x, y) := \big[(v_{|x|} = q) \wedge (v_{|y|} = s)\big] \vee (v_{|x|} = u)$. Gemäß Theorem 3.7.4 kann $\chi_{G'}$ in Zeit $\alpha(w) \cdot \text{poly}(n)$ berechnet werden und hat höchstens Voll-OBDD-Breite $\beta(w)$ bez. $\pi^{\text{id}}_{2k,n}$ für geeignete Funktionen $\alpha, \beta \colon \mathbb{N} \to \mathbb{N}$. □

Theorem 7.2.18. Ungerichtete *q*-*s*-Pfad-Existenz* $\prec$ Bipartite Perfekte-Matching-Existenz.

Beweis: Wir transformieren den Eingabegraphen $G = (V, E)$ nun in einen bipartiten Graphen $G' = (V', E')$, sodass q und s genau dann durch einen Pfad verbunden sind, wenn G' ein perfektes Matching besitzt. Dazu legen wir von jedem Knoten in G eine Kopie an, also $V' := V \times \{1, 2\}$ und verbinden für $u, w \in V$ zwei solche neuen Knoten (u, c) und (w, d) mit einer Kante, wenn $\{u, w\} \in E \setminus \big\{\{q, s\}\big\}$ und $c \neq d$. Außerdem seien $(u, 1)$ und $(u, 2)$ für alle $u \in V \setminus \{q, s\}$ adjazent. Schließlich verbinden wir $(q, 1)$ mit allen Knoten $(u, 2)$, wenn $\{q, u\} \in E$; ebenso verbinden wir $(s, 2)$ mit allen Knoten $(w, 1)$, wenn $\{s, w\} \in E$. Damit verbleiben $(q, 2)$ und $(s, 1)$ isoliert. Offenbar ist G' bipartit.

Da $(q, 1)$ und $(q, 2)$ ebenso wenig adjazent zueinander sind wie $(s, 1)$ und $(s, 2)$, muss ein perfektes Matching M eine Kante $\big\{(q, 1), (u, 2)\big\}$ enthalten. Dann dürfen wir aber nicht mehr $\big\{(u, 1), (u, 2)\big\}$ ins Matching aufnehmen. So haben wir das Problem an $(u, 1)$ weitergereicht. Dieses Problem kann nur gelöst werden, indem wir iterativ schließlich die 1-Kopie eines Knotens u erreichen, dessen 2-Kopie isoliert ist. Dies gilt aber nur für $u = s$. Somit können wir $(q, 1)$ und $(s, 2)$ nur dann in ein perfektes Matching aufnehmen, wenn die Knoten in G durch einen Pfad verbunden sind. Für die übrigen Knoten $w \in V$ nehmen wir einfach die Kante $\big\{(w, 1), (w, 2)\big\}$ ins Matching M auf.

Es bleibt zu zeigen, dass die Reduktion auf OBDD-Ebene gemäß der Definition impliziter Reduktionen realisierbar ist. Sei $\pi^{\text{id}}_{2k,n}$ gemäß Definition 7.2.10 die Variablenordnung des Eingabe-OBDDs für χ_G. Wir definieren $\chi_{G'}$ auf $2(k+1)n$ Variablen mit Ordnung $\pi^{\text{id}}_{2(k+1),n}$. Eine Knotennummer $x \in \mathbb{B}^{(k+1)n}$ besteht aus einer Komponente $i(x)$ aus k Argumenten der Länge n, um eine Knotennummer $i \in \{0, \ldots, |V|-1\}$ darstellen zu können, sowie einem Argument der Länge n, um den Kopienindex $c(x) \in \{1, 2\}$ zu codieren. Nun lässt sich $\chi_{G'}$ folgendermaßen darstellen:

$$\begin{aligned}\chi_{G'}(x, y) := \big[&(v_{|i(x)|} = q) \wedge (c(x) = 1 \neq c(y)) \wedge \chi_G(i(x), i(y))\big] \\ \vee \big[&(v_{|i(x)|} = s) \wedge (c(x) = 2 \neq c(y)) \wedge \chi_G(i(x), i(y))\big] \\ \vee \big[&(q \neq v_{|i(x)|}, v_{|i(y)|} \neq s) \wedge (c(x) \neq c(y)) \wedge \chi_G(i(x), i(y))\big]\end{aligned}$$

$$\vee \left[(q \neq v_{|i(x)|}, v_{|i(y)|} \neq s) \wedge (c(x) \neq c(y)) \wedge (|i(x)| = i(y)|)\right]. \quad (7.2.4)$$

Nach dem symmetrischen Abschluss $\chi_{G'}(x,y) := H(x,y) \vee H(y,x)$ stellt $\chi_{G'}$ offenbar korrekt die Kanten des Graphen G' dar; zusätzliche isolierte Knoten, die durch die Codierung entstehen, beeinflussen die Planarität nicht. Gleichheitstests wie etwa $i(x) = i(y)$ haben bei gemischten Variablenordnungen konstante Voll-OBDD-Breite. Tests des Kopienindex $c(x)$ werden durch einen Vergleich der codierenden Variablen mit zwei festgelegten Belegungen realisiert, die den Kopien zugeordnet werden. Zudem sei jeweils $c(x), c(y) \in \{1,2\}$ garantiert.

Jede der konstant vielen in Gleichung (7.2.3) auftauchenden Bausteinfunktionen hat also konstante Voll-OBDD-Breite bez. $\pi^{\mathrm{id}}_{2(k+1),n}$. Funktion χ_G habe Voll-OBDD-Breite w bez. $\pi^{\mathrm{id}}_{2k,n}$. Gemäß Theorem 3.7.4 kann $\chi_{G'}$ in Zeit $\alpha(w) \cdot \mathrm{poly}(n)$ konstruiert werden und hat höchstens Voll-OBDD-Breite $\beta(w)$ bez. $\pi^{\mathrm{id}}_{2(k+1),n}$ für geeignete Funktionen $\alpha, \beta \colon \mathbb{N} \to \mathbb{N}$. □

Wir verlassen nun die Entscheidungsprobleme und gehen über zu Suchproblemen. Wir haben bisher noch keine Übertragung des Problems **Transitiver Abschluss** ins implizite Szenario vorgenommen. Da bei Eingabe $G = (V,E)$ die Ausgabe jedoch wieder ein Graph $A = (V,E')$ ist, soll auch die implizite Ausgabe $\chi_A(x,y)$ bei impliziter Eingabe $\chi_G \in B_{2n}$ kanonisch durch $\chi_A(x,y) = 1 :\Leftrightarrow (v_{|x|}, v_{|y|}) \in E'$ definiert sein.

Die Ausgabe der impliziten Version des Problems **Flussmaximierung** auf einer Eingabe $\chi_G \in B_{3n}$ eines gewichteten Graphen $G = (V,E,\phi)$ definieren wir außerdem kanonisch als charakteristische Funktion $\chi_F \in B_{3n}$ eines maximalen Flusses $F \colon E \to \mathbb{N}$ mit $\chi_F(x,y,a) = 1 :\Leftrightarrow (v_{|x|}, v_{|y|}) \in E \wedge F(v_{|x|}, v_{|y|}) = |a|$.

Anmerkung 7.2.19. *Bevor wir zu Suchproblemen übergehen, müssen wir noch ein Detail besprechen: die Ausgabelänge der Orakelbefragungen. Da die Rechenzeit der impliziten Reduktion durch $\alpha(w) \cdot \mathrm{poly}(\Lambda(I))$ beschränkt ist, können also auch nur Orakelausgaben dieser Länge verarbeitet werden. Orakel in klassischen Turingreduktionen produzieren typischerweise nur Ausgaben polynomieller Länge. In unserem Fall können wir die OBDD-Größen der Orakelausgabe nicht von vornherein beschränken. Um die Laufzeitschranke der Reduktionen in jedem Fall zu garantieren, müssten die im Folgenden vorgestellten impliziten Reduktionen, die als Orakelausgabe OBDDs zur Weiterverarbeitung erhalten, deren Größe mit einer entsprechend gewählten Schranke vergleichen und bei Überschreitung abbrechen. Diesen formalen Kunstgriff werden wir aber wieder vernachlässigen, da diese Bedingung im interessanten Fall, dass Orakel durch entsprechende* I-FPT-T*-Algorithmen realisiert werden, offensichtlich gewährleistet ist.*

Theorem 7.2.20.

$$\textbf{Ungerichtete } \boldsymbol{q}\textbf{-}\boldsymbol{s}\textbf{-Pfad-Existenz}^* \prec \begin{array}{l} \boldsymbol{q}\textbf{-Erreichbarkeit}, \\ \textbf{Transitiver Abschluss}, \\ \textbf{SSSP}, \\ \textbf{APSP}. \end{array}$$

Beweis: Wir konstruieren aus dem in der Reduktionseingabe $\chi_G \in B_{2kn}$ mit Voll-OBDD-Breite w bez. $\pi^{\mathrm{id}}_{2k,n}$ implizit dargestellten Graphen $G = (V, E)$ die Funktion $\chi_{G'}$ eines Bildgraphen G'. Für die ersten beiden Probleme führen wir die gleiche Reduktion wie im Beweis von Theorem 7.2.12 durch und wenden das Orakel für ***q*-Erreichbarkeit** bzw. **Transitiver Abschluss** auf ihr Ergebnis $\chi_{G'} \in B_{2n}$ an. Genau dann, wenn q und s in verschiedenen Zusammenhangskomponenten von G liegen, ist G' zusammenhängend. Genau dann repräsentiert die Ausgabe von ***q*-Erreichbarkeit** alle Knoten durch $\chi_R(x) = 1$ und die Ausgabe von **Transitiver Abschluss** alle Knotenpaare durch $\chi_A(x, y) = 1$.

Im Falle der Kürzeste-Wege-Probleme müssen wir außerdem eine Kantengewichtsfunktion $\phi(e) := 0$ zu G' hinzufügen, also $\chi_{G'}(x, y, a) := \chi_{G'}(x, y) \wedge (|a| = 0)$. Hier ist $\chi_{G'}$ nun auf $3kn$ Variablen mit Ordnung $\pi^{\mathrm{id}}_{3k,n}$ definiert. Genau dann, wenn G' zusammenhängend ist, gilt $\chi_{\Delta,\mathrm{SSSP}}(x, a) = (|a| = 0)$ für die **SSSP**-Ausgabe, alle Knoten sind auf Pfaden der Länge 0 von q aus erreichbar. Ebenso ist die Ausgabe für **APSP** dann $\chi_{\Delta,\mathrm{APSP}}(x, y, a) = (|a| = 0)$.

Die Berechnung von $\chi_{G'}(x, y)$ ist bekanntermaßen eine implizite Reduktion. Die Vergleichsfunktion $|a| = 0$ hat konstante Voll-OBDD-Breite, und gemäß Theorem 3.7.4 lassen sich die jeweiligen Orakeleingaben $\chi_{G'}$ in Zeit $\alpha(w) \cdot \mathrm{poly}(n)$ konstruieren und haben höchstens Voll-OBDD-Breite $\beta(w)$ bez. $\pi^{\mathrm{id}}_{2k,n}$ bzw. $\pi^{\mathrm{id}}_{3k,n}$ für geeignete Funktionen $\alpha, \beta\colon \mathbb{N} \to \mathbb{N}$. Die jeweilige Orakelausgabe vergleichen wir mit der Einsfunktion bzw. mit $|a| = 0$ und verursachen dadurch nur Laufzeit $\mathcal{O}(n)$. □

Dieser Beweis konstruiert Kürzeste-Wege-Instanzen mit Kantengewichten 0. Dies wird für eine Erweiterung des Resultats durch Theorem 7.2.30 entscheidend sein. Offenbar funktioniert obige Reduktion jedoch auch mit Kantengewichten 1, und es folgt direkt:

Korollar 7.2.21.

$$\textbf{Ungerichtete } \textbf{\textit{q}-\textit{s}-Pfad-Existenz}^* \prec \begin{array}{l}\textbf{SSSP}^+,\\ \textbf{APSP}^+.\end{array}$$

Theorem 7.2.22.

$$\textbf{\textit{q}-\textit{s}-Pfad-Existenz}^* \prec \begin{array}{l}\textbf{0-1-Flussmaximierung},\\ \textbf{Flussmaximierung}.\end{array}$$

Beweis: Wir konstruieren aus dem in der Reduktionseingabe $\chi_G \in B_{2kn}$ mit Voll-OBDD-Breite w bez. $\pi^{\mathrm{id}}_{2k,n}$ implizit dargestellten Graphen $G = (V, E)$ die Funktion $\chi_{G'}$ eines Bildgraphen G'. Für die **0-1-Flussmaximierung** übernehmen wir G unverändert: Das Ergebnis $\chi_F(x, y)$ ist genau dann die Nullfunktion, wenn kein q-s-Pfad in G existiert.

Analog zum Vorgehen bei den Kürzeste-Wege-Problemen fügen wir für das allgemeinere Problem **Flussmaximierung** die Kantengewichtsfunktion $\phi(e) := 1$

durch $\chi_{G'}(x, y, a) := \chi_G(x, y) \wedge (|a| = 1)$ hinzu und können wie beim Problem **0-1-Flussmaximierung** argumentieren: Das Ergebnis $\chi_F(x, y, a)$ ist genau dann $\chi_G(x, y) \wedge (|a| = 0)$, wenn kein q-s-Pfad in G existiert.

Vergleichsfunktionen wie $|a| = 0$ haben konstante Voll-OBDD-Breite, und gemäß Theorem 3.7.4 lassen sich die jeweiligen Orakeleingaben $\chi_{G'}$ in Zeit $\alpha(w) \cdot \text{poly}(n)$ konstruieren und haben höchstens Voll-OBDD-Breite $\beta(w)$ bez. $\pi^{\text{id}}_{2k,n}$ bzw. $\pi^{\text{id}}_{3k,n}$ für geeignete Funktionen $\alpha, \beta \colon \mathbb{N} \to \mathbb{N}$. Die jeweilige Orakelausgabe vergleichen wir mit der Nullfunktion bzw. mit $\chi_G(x, y) \wedge (|a| = 0)$ und verursachen dadurch nur Laufzeit $\mathcal{O}(wn)$. □

Die Probleme **Ψ-Existenz** für $\Psi \in \{$***q*-*s*-Pfad**, **Kreis**, **Perfekte-Matching**$\}$ sind leicht ersichtlich implizit auf ihre Suchvarianten reduzierbar. Letztere haben als Ausgabe eine Kantenmenge M, und dementsprechend sollen ihre impliziten Versionen die charakteristische Funktion χ_M so einer Kantenmenge ausgeben.

Theorem 7.2.23.

$$\textbf{Ψ-Existenz} \prec \textbf{Ψ-Suche}$$

für $\Psi \in \{$***q*-*s*-Pfad**, **Kreis**, **Perfekte-Matching**$\}$.

Beweis: Wir rufen das Orakel für **Ψ-Suche** auf der unveränderten Eingabe für **Ψ-Existenz** auf. Ist die entsprechende Ausgabe die implizite leere Menge $\chi_M(x, y) = 0$, so lautet die Antwort auf das Entscheidungsproblem „Nein“, ansonsten „Ja“. □

Für das Problem **Eulerkreis-Existenz** haben wir keine Suchvariante definiert, da ihre implizite Version ein neues implizites Codierungskonzept neben denen der in dieser Dissertation betrachteten ungewichteten und gewichteten Graphen erfordern würde. Für sinnvolle implizite Repräsentationen eines Eulerkreises lässt sich jedoch ebenso einfach eine Reduktion angeben wie im Beweis von Theorem 7.2.23.

Ähnlich einfach lassen sich die Probleme **Zusammenhang**, **Zweizusammenhang** und **Starker Zusammenhang** implizit auf ihre Zerlegungsvarianten reduzieren. In allen vier Fällen ist die Ausgabe der Zerlegungsvariante eine Menge von Knotenmengen $W_1, \ldots, W_r$, und dementsprechend sei die Ausgabe ihrer impliziten Versionen kanonisch als Menge der charakteristischen Funktionen $\chi_{W_1}, \ldots, \chi_{W_r}$ definiert.

Theorem 7.2.24.

$$\Psi \prec \textbf{Ψ-Zerlegung}$$

für $\Psi \in \{$**Zusammenhang**, **Zweizusammenhang**, **Starker Zusammenhang**$\}$.

Beweis: Wir rufen das Orakel für **Ψ-Zerlegung** auf der unveränderten Eingabe für **Ψ** auf. Besteht die Ausgabe aus nur einer Funktion, so besteht G offenbar aus nur einer Komponente bez. des jeweils betrachteten Zusammenhangsbegriffs und die Ausgabe des Entscheidungsproblems lautet „Ja“, ansonsten lautet sie „Nein“. □

Auch die Ausgabe des Problems **Maximales Matching** ist eine Kantenmenge M und so definieren wir die Ausgabe der impliziten Problemversion kanonisch als χ_M.

Theorem 7.2.25. Perfekte-Matching-Existenz $\prec$ Maximales Matching.

Beweis: Wir rufen das Orakel für **Maximales Matching** auf der unveränderten Eingabe χ_G für **Perfekte-Matching-Existenz** auf und zählen die erfüllenden Belegungen der Ausgabe χ_M in linearer Zeit bez. ihrer OBDD-Größe. Diese Anzahl entspricht der Größe eines maximalen Matchings in G. Ist sie $|V|/2$, so existiert ein perfektes Matching, ansonsten nicht. □

Die Ausgabe der impliziten Version des Problems **Topologisches Sortieren** sei kanonisch als die charakteristische Funktion χ_Δ der ursprünglichen Ausgabeabbildung $\Delta\colon V \to \mathbb{N}$ mit $\chi_\Delta(x,a) = 1 :\Leftrightarrow \Delta(v_{|x|}) = |a|$ definiert.

Theorem 7.2.26. *q*-*s*-Pfad-Existenz* $\prec$ Topologisches Sortieren.

Beweis: Wir rufen das Orakel für **Topologisches Sortieren** auf der zunächst unveränderten Reduktionseingabe $\chi_G \in B_{2kn}$ eines implizit dargestellten Graphen $G = (V,E)$ auf. In der so erhaltenen Ausgabe $\chi_\Delta(x,a)$ ersetzen wir x durch $\langle s\rangle_2$ und extrahieren in linearer Zeit bez. ihrer OBDD-Größe die einzige erfüllende Belegung a^* der a-Variablen. Wenn ein q-s-Pfad in G existiert, so entspricht $|a^*|$ nach Problemdefinition der maximalen Länge eines Pfades in G. Ansonsten muss dies nicht notwendigerweise der Fall sein.

Wir konstruieren nun die Funktion $\chi_{G'} \in B_{2(k+1)n}$ eines Graphen $G' = (V', E')$ mit $V' := V \cup \{v^*\}$ und $E' := E \cup \{(v^*, q)\}$ für einen neuen zusätzlichen Knoten v^*. Wenn $\Delta'(s) = |a^*| + 1$ in G' gilt, so war $|a^*|$ tatsächlich die maximale Pfadlänge in G, die wir durch die zusätzliche Kante (v^*, q) um den Wert 1 erhöht haben – es existiert ein q-s-Pfad. Ansonsten existiert kein solcher Pfad.

Es bleibt zu zeigen, dass die Erzeugung des OBDDs für $\chi_{G'}$ tatsächlich als implizite Reduktion realisierbar ist. Sei $\pi^{\mathrm{id}}_{2k,n}$ gemäß Definition 7.2.10 die Variablenordnung des Eingabe-OBDDs für χ_G. Wir definieren $\chi_{G'}$ auf $2(k+1)n$ Variablen mit Ordnung $\pi^{\mathrm{id}}_{2(k+1),n}$. Eine Knotennummer $x \in \mathbb{B}^{(k+1)n}$ besteht aus einer Komponente $i(x)$ aus k Argumenten der Länge n, um eine Knotennummer $i \in \{0,\ldots,|V|-1\}$ darstellen zu können, sowie einem Argument $j(x)$ der Länge n, um den zusätzlichen Knoten v^* codieren zu können. Ursprüngliche Knoten $v \in V$ codieren wir durch x mit $i(x) = \langle v\rangle_2$ und $j(x) = (0)_2$, für Knoten v^* sei $i(x) = (0)_2$ und $j(x) = (1)_2$. Nun lässt sich $\chi_{G'}$ folgendermaßen darstellen:

$$\chi_{G'}(x,y) := \left[(|j(x)| = |j(y)| = 0) \wedge \chi_G(i(x), i(y))\right] \vee \left[(v_{|x|} = v^*) \wedge (v_{|y|} = q)\right].$$

Offenbar stellt $\chi_{G'}$ korrekt die Kanten des Graphen G' dar; zusätzliche isolierte Knoten, die durch die Codierung entstehen, beeinflussen die Orakelausgabe $\chi_{\Delta'} \in B_{2kn}$

nicht. Den Wert $\Delta'(s)$ erhalten wir schließlich wieder durch Ersetzung der x-Variablen von $\chi_{\Delta'}(x,a)$ durch $\langle s\rangle_2$ und anschließender Extrahierung der erfüllenden a-Belegung. Gleichheitstests wie $v_{|x|} = v^*$ haben bei gemischten Variablenordnungen konstante Voll-OBDD-Breite. Funktion χ_G habe Voll-OBDD-Breite w bez. $\pi^{\mathrm{id}}_{2k,n}$. Gemäß Theorem 3.7.4 lassen sich die jeweiligen Orakeleingaben $\chi_{G'}$ in Zeit $\alpha(w)\cdot\mathrm{poly}(n)$ konstruieren und haben höchstens Voll-OBDD-Breite $\beta(w)$ bez. $\pi^{\mathrm{id}}_{2(k+1),n}$ für geeignete Funktionen $\alpha,\beta\colon \mathbb{N}\to\mathbb{N}$. Die Variablenersetzung sowie die Extrahierung der erfüllenden Belegung geschehen in linearer Zeit bez. der Orakelausgabelänge. □

Die Ausgabe des letzten hier betrachteten Graphproblems **Minimaler Spannbaum** ist ein Teilgraph $T = (V, E', \phi)$ des gewichteten Eingabegraphen $G = (V, E, \phi)$, und so definieren wir die Ausgabe der impliziten Problemversion kanonisch als die charakteristische Funktion χ_T der Ausgabe T.

Theorem 7.2.27.

Ungerichtete *q*-*s*-Pfad-Existenz* $\prec$ Minimaler Spannbaum.

Beweis: Wir konstruieren aus dem in der Reduktionseingabe $\chi_G \in B_{2kn}$ implizit dargestellten Graphen $G = (V, E)$ die Funktion $\chi_{G'} \in B_{3kn}$ eines gewichteten Bildgraphen $G' = (V, E', \phi')$ mit $E' := E \cup \big\{\{q, s\}\big\}$ und $\phi'(e) := 0$ für $e \in E$ und $\phi'(\{q, s\}) := 1$. Offenbar enthält ein minimaler Spannbaum T von G' genau dann die Kante $\{q, s\}$, wenn q und s in G nicht durch einen Pfad verbunden sind.

Es bleibt zu zeigen, dass die Erzeugung des OBDDs für $\chi_{G'}$ tatsächlich als implizite Reduktion realisierbar ist. Sei $\pi^{\mathrm{id}}_{2k,n}$ gemäß Definition 7.2.10 die Variablenordnung des Eingabe-OBDDs für χ_G. Wir definieren $\chi_{G'}$ auf $3k$ Variablen mit Ordnung $\pi^{\mathrm{id}}_{3k,n}$, wobei das neue dritte Argument der Codierung der Kantengewichte dient. Nun lässt sich $\chi_{G'}$ folgendermaßen darstellen:

$$\chi_{G'}(x, y, a) := \big[\chi_G(x, y, a) \wedge (|a| = 0)\big] \vee \big[(v_{|x|} = q) \wedge (v_{|y|} = s) \wedge (|a| = 1)\big]\,.$$

Offenbar stellt $\chi_{G'}$ korrekt die Kanten des Graphen G' dar. Gleichheitstests wie $v_{|x|} = q$ haben konstante Voll-OBDD-Breite. Funktion χ_G habe Voll-OBDD-Breite w bez. $\pi^{\mathrm{id}}_{2k,n}$. Gemäß Theorem 3.7.4 lassen sich die jeweiligen Orakeleingaben $\chi_{G'}$ in Zeit $\alpha(w)\cdot\mathrm{poly}(n)$ konstruieren und haben höchstens Voll-OBDD-Breite $\beta(w)$ bez. $\pi^{\mathrm{id}}_{2(k+1),n}$ für geeignete Funktionen $\alpha,\beta\colon \mathbb{N}\to\mathbb{N}$. Wir wenden das Orakel auf $\chi_{G'}$ an und erhalten den impliziten minimalen Spannbaum χ_T. Um zu testen, ob er $\{q, s\}$ enthält, führen wir die Ersetzung $(x, s, a) := (\langle q\rangle_2, \langle s\rangle_2, (1)_2)$ in linearer Zeit bez. der Orakelausgabelänge durch. □

Korollar 7.2.28. *Keines der in Abschnitt 2.1.1 eingeführten Graphprobleme ist in* I-FPT-T*, es sei denn* P = PSPACE*.*

Aus den Ergebnissen für I-FPT-T lassen sich für die meisten der betrachteten Suchprobleme nun auch entsprechende Negativaussagen für die Zugehörigkeit zu IO-FPT-T erhalten. Für Entscheidungsprobleme macht die Einbeziehung der Ausgabe keinen Sinn, weil sie stets konstante Größe hat.

Theorem 7.2.29. *Sei Π' ein OBDD-Entscheidungsproblem und sei Π'' ein OBDD-Suchproblem, sodass*

$$\textbf{(Ungerichtete) } \boldsymbol{q}\textbf{-}\boldsymbol{s}\textbf{-Pfad-Existenz}^* \prec \Pi' \prec \Pi''$$

gilt. Wenn für Π'-Eingaben $I' \in \Pi'$ oder aber $I' \notin \Pi'$ die Π''-Orakelausgaben stets höchstens maximale Voll-OBDD-Breite $\delta(w'')$ für die maximale Voll-OBDD-Breite w'' der Π''-Orakeleingaben und ein geeignetes monotones $\delta\colon \mathbb{N} \to \mathbb{N}$ haben, so gilt $\Pi'' \notin$ IO-FPT-T*, es sei denn* P = PSPACE*.*

Beweis: Wir verwenden die gleichen Bezeichner wie im Beweis von Theorem 7.2.11 und argumentieren auch analog. Angenommen Π'' ist in IO-FPT-T, so verwenden wir einen entsprechenden FPT-Algorithmus A zur Realisierung des Π''-Orakels. Zudem impliziere o. B. d. A. $I' \in \Pi'$ eine Π''-Orakelausgabe-Voll-OBDD-Breite von höchstens $\delta(w'')$. Dann hat A im Falle $I' \in \Pi'$ Laufzeit $\gamma(\delta(w'')) \cdot \text{poly}(\Lambda(I''))$ für Π''-Orakeleingaben I'' mit maximaler Voll-OBDD-Breite w'' und ein geeignetes $\gamma\colon \mathbb{N} \to \mathbb{N}$, da wir von monotonem δ mit $\max\{w'', \delta(w'')\} = \delta(w'')$ ausgehen. Überschreitet A jedoch diese Laufzeitschranke, brechen wir die Berechnung ab und es folgt $I' \notin \Pi'$. Wir bemerken $w'' = \beta'(\beta(w))$ und $\Lambda(I'') = w'' \cdot n''$. Aufgrund der Transitivität von $\prec$ ist die Komposition der beiden impliziten Reduktionen zusammen mit A ein I-FPT-T-Algorithmus für **(Ungerichtete) $\boldsymbol{q}$-$\boldsymbol{s}$-Pfad-Existenz***. □

Können wir also für eine der in diesem Abschnitt vorgestellten impliziten Reduktionen eines Entscheidungsproblems Π' auf ein Suchproblem Π'' nachweisen, dass für eine der beiden möglichen Entscheidungen die Π''-Orakelausgabe stets Voll-OBDD-Breite $\mathcal{O}(w'')$ für die Voll-OBDD-Breite w'' der entsprechenden Π''-Orakeleingabe hat, so haben wir $\Pi'' \notin$ IO-FPT-T unter der Annahme P $\neq$ PSPACE bewiesen. Wir betrachten daher die Reduktionen erneut:

Theorem 7.2.30. *Mit Ausnahme von Korollar 7.2.21, Theorem 7.2.25 und Theorem 7.2.26 erfüllen alle vorgestellten impliziten Reduktionen auf Suchprobleme die Bedingungen von Theorem 7.2.29.*

Beweis: Wir verwenden die gleiche Notation wie in Theorem 7.2.29. In Theorem 7.2.20 haben die Orakelausgaben Größe $\mathcal{O}(n'')$ bei konstanter Voll-OBDD-Breite, wenn q und s in G nicht verbunden sind.

In Theorem 7.2.22 haben die Orakelausgaben Größe $\mathcal{O}(w''n'')$ bei Voll-OBDD-Breite $\mathcal{O}(w'') = \mathcal{O}(w')$, wenn q und s in G nicht verbunden sind – die Konjunktion von $\chi_G(x, y)$ mit $(|a| = 0)$ erhöht die Breite nur um einen konstanten Faktor.

In Theorem 7.2.23 haben die Orakelausgaben konstante Größe, wenn die Antwort auf das Entscheidungsproblem „Nein“ lautet.

In Theorem 7.2.24 haben die Orakelausgaben konstante Größe, wenn die Antwort auf das Entscheidungsproblem „Ja“ lautet – die Zerlegung gibt eine Funktion $\chi_{W_1}(x) = 1$ aus.

In Theorem 7.2.27 haben die Orakelausgaben Größe $\mathcal{O}(w''n'')$ bei Voll-OBDD-Breite $w'' = \mathcal{O}(w')$, wenn q und s in G nicht verbunden sind. Dann wird nämlich die zusätzlich eingefügte Kante $\{q, s\}$ mit Gewicht 1 benötigt, und es ist $\chi_T(x, y, a) = \chi_{G'}(x, y, a)$. □

Korollar 7.2.31. *Für jedes der in Abschnitt 2.1.1 eingeführten Suchprobleme mit Ausnahme von* **SSSP$^+$**, **APSP$^+$**, **Maximales Matching** *und* **Topologisches Sortieren** *folgt aus Theorem 7.2.30, dass es nicht in* IO-FPT-T *ist, es sei denn* P = PSPACE.

Anmerkung 7.2.32. *Damit ist auch für* **APSP$^+$** *die Zugehörigkeit zu* IO-FPT-T *noch offen, da wir diese Klasse unabhängig von einer Variablenordnung definiert haben. Im Beweis der FPT-Eigenschaften des in Abschnitt 5.1 vorgestellten Algorithmus 5.1.2 wird jedoch formal von der Verwendung der Variablenordnung $\pi^{\mathrm{id}}_{3,n}$ ausgegangen. Mithilfe von Anmerkung 5.1.4 und durch Einführung von redundanten Kantengewichtsargumenten (siehe Anmerkung 5.3.3) können wir diese Bedingung jedoch so lockern, dass auch Variablenordnungen $\pi^{\mathrm{id}}_{3k,n}$ für konstantes k zulässig wären. Genau für diese Klasse von Variablenordnungen wurden in diesem Abschnitt aber die PSPACE-Härteresultate bewiesen.*

Bezüglich einer weniger eleganten Definition der Klasse IO-FPT-T *wäre* **APSP$^+$** *also in ihr enthalten und daher kann unsere implizite Reduktionstechnik für dieses Problem nicht erfolgreich sein, es sei denn* P = PSPACE.

Für keines der betrachteten Probleme existiert unter der allgemein anerkannten Annahme P $\neq$ NP ein Festparameteralgorithmus bez. der Eingabe-OBDD-Breite. Unter weiterer Einbeziehung des Ausgabe-OBDDs bleibt die Frage nur für manche Probleme offen, während das Negativresultat in vielen Fällen übertragbar ist. Für die meisten der betrachteten grundlegenden Graphprobleme kann es also wahrscheinlich noch nicht einmal dann polynomielle OBDD-basierte Algorithmen geben, wenn Eingabe- und Ausgabe-OBDDs konstante vollständige Breite besitzen. Dabei reicht es, die natürliche gemischte Variablenordnung als Gegenbeispiel zu nennen. Zugleich haben wir die Resultate somit aber für eine wichtige Variablenordnung gezeigt, da sie zu einer kompakten OBDD-Darstellung des Konfigurationsübergangsgraphen von Turingmaschinen führt. Zudem ermöglicht sie den Einsatz von Theorem 3.7.4, um komplexere Funktionen aus einfacheren zu komponieren, ohne die Voll-OBDD-Breite in Abhängigkeit von der Variablenanzahl n wesentlich zu vergrößern. Es mag ebenso Variablenordnungen π geben, für die die besprochenen impliziten Graphprobleme FPT-Algorithmen besitzen – wenn dies jedoch daran liegt, dass die Voll-OBDD-Breite

bez. π bereits für einfache Funktionen exponentiell in n ist, dann hat ein derartiges „Positivresultat“ keinen großen Wert.

In welchem Verhältnis stehen diese Resultate zu denen aus Abschnitt 5.3? Dort wurde mit größerem Aufwand eine exponentielle untere Schranke lediglich für eine eingeschränkte Klasse impliziter **APSP**-Algorithmen bewiesen. Diese hängt jedoch nicht von der Annahme P $\neq$ PSPACE ab und basiert tatsächlich sogar auf einer $\mathbf{APSP}^+$-Instanz. Somit existiert keine direkte Implikation zwischen beiden Resultaten.

Schließlich führen wir noch die Klassen I-P-T und IO-P-T für Probleme mit polynomiellen OBDD-basierten Algorithmen bez. Eingabe- bzw. Eingabe- und Ausgabe-OBDD-Größe ein. Existieren nicht einmal entsprechende FPT-Algorithmen, so können insbesondere keine allgemeinen polynomiellen Algorithmen existieren, und es folgen einfache Teilmengenbeziehungen.

Definition 7.2.33 (I-P-T). I-P-T *ist die Klasse der OBDD-Suchprobleme* Π*, für die ein Algorithmus A mit folgenden Eigenschaften existiert: A berechnet ein* $O \in \Pi(I)$ *für jede* Π*-Eingabe I in Zeit und auf Platz* $\mathrm{poly}(\Lambda(I))$.

Definition 7.2.34 (IO-P-T). IO-P-T *ist die Klasse der OBDD-Suchprobleme* Π*, für die ein Algorithmus A mit folgenden Eigenschaften existiert: A berechnet ein* $O \in \Pi(I)$ *für jede* Π*-Eingabe I in Zeit und auf Platz* $\mathrm{poly}(\max_{O \in \Pi(I)}\{\Lambda(I), \Lambda(O)\})$.

Fakt 7.2.35. *Offenbar gelten die Beziehungen*

$$\text{I-P-T} \subseteq \text{I-FPT-T}, \quad \text{IO-P-T} \subseteq \text{IO-FPT-T}, \quad \text{I-P-T} \subseteq \text{IO-P-T}.$$

Schließlich sei erwähnt, dass ähnlich zu den Problemen ***q*-*s*-Pfad-Suche**, ***q*-Erreichbarkeit** und **Transitiver Abschluss** auch entsprechende Varianten für den Zweizusammenhang von Graphen definiert werden können. Ihre Betrachtung ist aber unüblich, und es ergeben sich leicht ersichtlich die gleichen Zuordnungen zu den in diesem Kapitel behandelten Komplexitätsklassen wie für den Fall einfachen Zusammenhangs. Ebenso können die PSPACE-Härte-Resultate für die betrachteten Optimierungsprobleme leicht auf die jeweiligen Entscheidungsvarianten mit der Fragestellung „Existiert eine Lösung mit minimalem/maximalem Wert x?“ übertragen werden.

7.3 Platzkomplexität

Schließlich betrachten wir in diesem Abschnitt die Platzkomplexität OBDD-basierter Graphprobleme. Zunächst wird ein einfaches Schema vorgestellt, mit dem viele grundlegende Graphprobleme auf polynomiellem Platz bez. des Maximums von Eingabe- und Ausgabe-OBDD-Größe gelöst werden können. Dieses Schema kann auch als Festparameteralgorithmus bez. des Maximums der entsprechenden OBDD-Breiten aufgefasst werden. Es ist im Falle der betrachteten Entscheidungsprobleme sogar nur von

Größe bzw. Breite des Eingabe-OBDDs abhängig. Interessanter wird es jedoch bei den betrachteten grundlegenden Such- und Optimierungsproblemen. Dort gelingt es in keinem einzigen Fall, einen Festparameteralgorithmus in alleiniger Abhängigkeit von der vollständigen Eingabe-Breite anzugeben. Für die drei betrachteten Zusammenhangs-Zerlegungs-Probleme sowie für **SSSP$^+$**, **APSP$^+$**, **Flussmaximierung**, **Topologisches Sortieren** und eingeschränkte Varianten von ***q*-Erreichbarkeit** und **Transitiver Abschluss** gelingt es sogar, die Nichtexistenz derartiger Algorithmen zu beweisen. In diesem Kontext wird auch die erste exponentielle untere Schranke für die OBDD-Größe des höchsten Bits der ganzzahligen Multiplikation für eine spezielle Variablenordnung bewiesen. Abschließend geben wir einen Überblick über die erzielten Komplexitätsresultate.

Zunächst jedoch definieren wir die Problemklassen I-FPT-S, IO-FPT-S, I-P-S und IO-P-S analog zu ihren Laufzeitäquivalenten mit dem Suffix „T“. Diesmal ist also der Platz die Ressource, die durch einen Ausdruck $\alpha(\bullet) \cdot \text{poly}(\bullet)$ bzw. $\text{poly}(\bullet)$ beschränkt sein soll.

Definition 7.3.1 (I-FPT-S). I-FPT-S *ist die Klasse der OBDD-Suchprobleme* Π*, für die ein Algorithmus A mit folgenden Eigenschaften existiert: A berechnet ein* $O \in \Pi(I)$ *für jede* Π*-Eingabe I auf Platz* $\alpha(w) \cdot \text{poly}(\Lambda(I))$ *für ein geeignetes* $\alpha\colon \mathbb{N} \to \mathbb{N}$*, wenn w eine obere Schranke für die maximale vollständige Breite der OBDDs in* $S(I)$ *ist.*

Definition 7.3.2 (IO-FPT-S). IO-FPT-S *ist die Klasse der OBDD-Suchprobleme* Π*, für die ein Algorithmus A mit folgenden Eigenschaften existiert: A berechnet ein* $O \in \Pi(I)$ *für jede* Π*-Eingabe I auf Platz* $\alpha(w) \cdot \text{poly}(\Lambda(I))$ *für ein geeignetes* $\alpha\colon \mathbb{N} \to \mathbb{N}$*, wenn w eine obere Schranke für die maximale vollständige Breite der OBDDs in* $S(I) \cup \bigcup_{O \in \Pi(I)} S(O)$ *ist.*

Definition 7.3.3 (I-P-S). I-P-S *ist die Klasse der OBDD-Suchprobleme* Π*, für die ein Algorithmus A mit folgenden Eigenschaften existiert: A berechnet ein* $O \in \Pi(I)$ *für jede* Π*-Eingabe I auf Platz* $\text{poly}(\Lambda(I))$*.*

Definition 7.3.4 (IO-P-S). IO-P-S *ist die Klasse der OBDD-Suchprobleme* Π*, für die ein Algorithmus A mit folgenden Eigenschaften existiert: A berechnet ein* $O \in \Pi(I)$ *für jede* Π*-Eingabe I auf Platz* $\text{poly}(\max_{O \in \Pi(I)}\{\Lambda(I), \Lambda(O)\})$*.*

Wir betrachten also die Größe der Ausgabe nicht (wie in der Komplexitätstheorie sonst üblich) getrennt vom Platzbedarf des Algorithmus. Wir gehen somit nicht von der Existenz eines zusätzlichen Ausgabebands aus, sondern lassen die Ausgabe in den regulären Speicher des Algorithmus schreiben. Die so erhaltenen unteren Schranken sind demnach aus klassischer Sicht schwächer als bei getrennter Betrachtung von Arbeitsspeicher und Ausgabe, in praktischer Hinsicht sind sie jedoch gleichwertig – die Ausgabe-OBDDs sollen schließlich nicht nur auf ein (möglicherweise großes, externes) Speichermedium geschrieben werden, sondern zur weiteren Verarbeitung im Arbeitsspeicher verfügbar sein.

Fakt 7.3.5. *Offenbar gelten die Beziehungen*

$$\text{I-P-S} \subseteq \text{I-FPT-S}, \quad \text{IO-P-S} \subseteq \text{IO-FPT-S}, \quad \text{I-FPT-S} \subseteq \text{IO-FPT-S}, \quad \text{I-P-S} \subseteq \text{IO-P-S}.$$

7.3.1 Polynomieller Platz in Eingabe- und Ausgabe-Größe

Wir beginnen mit der Betrachtung der Klasse IO-P-S und stellen ein Schema vor, mit dem für viele Probleme auf einfache Art und Weise ein entsprechender Algorithmus erhalten werden kann.

Definition 7.3.6 (Polynomialplatz-Schema). *Seien Π ein OBDD-Suchproblem (zu denen wir hier auch explizit die Entscheidungsprobleme als Suchprobleme mit Lösungsgrundmenge $\mathbb{B}$ zählen möchten), I eine Π-Eingabe sowie $O^{\max} \in \Pi(I)$ eine gültige Lösung maximaler Größe $\Lambda(O^{\max})$.*

Um eine beliebige (optimale) Lösung auf Platz $B(I) := p(\max\{\Lambda(I), \Lambda(O^{\max})\})$ für ein Polynom p zu erhalten, zählen wir alle potenziellen Lösungen O bis zur maximalen Länge $\Lambda(O^{\max})$ kanonisch auf und prüfen jeweils auf Platz $B(I)$, ob $O \in \Pi(I)$ gilt.

Offenbar folgt aus der Existenz eines Polynomialplatz-Schemas für Π die Zugehörigkeit $\Pi \in$ IO-P-S. Ist zudem ein Entscheidungsproblem in IO-P-S, so ist es natürlich auch in I-P-S. Doch wie prüfen wir auf Platz $B(I)$, ob eine potenzielle Lösung O tatsächlich in $\Pi(I)$ enthalten ist?

Eine komfortable Entwurfsmethode ist hier, einen nichtdeterministischen Algorithmus A anzugeben, der nach dem Satz von Savitch bekanntlich auf quadratischem Platz deterministisch simuliert werden kann (siehe z. B. Wegener (2003)). Besteht die Eingabe nun aus OBDDs für charakteristische Funktionen auf $n = \log|V| + \mathcal{O}(1)$ Variablen, so darf A auf jeden Fall Platz $\text{poly}(n) = \text{polylog}(|V|)$ verwenden. Entsprechende Algorithmen können für alle in dieser Dissertation betrachteten Graphprobleme und die entsprechenden Tests $O \in \Pi(I)$ angegeben werden: Die Entscheidungsprobleme befinden sich bekanntermaßen alle in der Klasse NL der Entscheidungsprobleme mit nichtdeterministischen, logarithmisch platzbeschränkten Turingmaschinen (siehe z. B. für **Planarität** Allender und Mahajan (2000)). Durch die Auswertung der durch das Eingabe-OBDD dargestellten charakteristischen Funktion χ_G kann auf Platz $\mathcal{O}(\max\{\Lambda(I), n\})$ die Existenz einer Kante geprüft werden. Ebenso kann durch weitere Extrahierung der erfüllenden Belegung nach einer konstanten Ersetzung einer Variablenteilmenge auf gleichem Platz die Menge der zu einem Knoten adjazenten Knoten traversiert werden oder das Gewicht einer Kante erfragt werden.

Für die betrachteten Suchprobleme können ebenfalls einfache nichtdeterministische Algorithmen mit Platz $B(I)$ angegeben werden. Wir möchten dies hier nur für das Problem **Transitiver Abschluss** beispielhaft durchführen:

Bei Eingabe $I = \pi\text{-OBDD}[\chi_G]$ mit $\chi_G \in B_{2n}$ und $n = \log|V| + \mathcal{O}(1)$ soll für $\pi\text{-OBDD}[f]$ mit $f \in B_{2n}$ geprüft werden, ob f die korrekte Ausgabe χ_A des transitiven Abschlusses A von G dargestellt. Für jedes Knotenpaar $(v_{|x|}, v_{|y|})$ mit $x, y \in \mathbb{B}^n$ soll

also die geforderte Äquivalenz $f(x,y) = 1 \Leftrightarrow \chi_A(x,y) = 1$ geprüft werden. Dazu zählen wir alle Belegungen x^*, y^* von x und y auf Platz $\mathcal{O}(n)$ auf und werten jeweils $f(x^*,y^*) =: c$ auf Platz $\mathcal{O}(\Lambda(I))$ aus. Im Falle $c = 1$ suchen wir nichtdeterministisch einen $v_{|x^*|}$-$v_{|y^*|}$-Pfad. Dabei merken wir uns stets eine Knotennummer z der Länge n, raten eine Nachfolgernummer z', prüfen, ob $v_{|z|}$ und $v_{|z'|}$ adjazent sind, und fahren fort, bis $v_{|y^*|}$ gefunden wurde. Wir haben Platz $\mathcal{O}(n)$ verwandt, und auf gleichem Platz können wir nach dem Satz von Immerman und Szelepcsényi nichtdeterministisch die Nichtexistenz eines $v_{|x^*|}$-$v_{|y^*|}$-Pfad im Falle $c = 0$ prüfen (siehe z. B. Wegener (2003)).

Auf ähnliche Weise kann auch für die übrigen betrachteten Suchprobleme die Zugehörigkeit zu IO-P-T gezeigt werden.

Theorem 7.3.7. *Alle in Abschnitt 2.1.1 eingeführten Suchprobleme sind in* IO-P-S $\subseteq$ IO-FPT-S. *Alle in Abschnitt 2.1.1 eingeführten Entscheidungsprobleme sind in* I-P-S $\subseteq$ I-FPT-S.

7.3.2 Zusammenhangs-Zerlegungs-Probleme

Einerseits haben wir mit den PSPACE-Härteresultaten dieses Kapitels unter der Annahme P $\neq$ PSPACE untere Schranken für die Laufzeit OBDD-basierter impliziter Algorithmen bez. des Maximums von Eingabe- und Ausgabe-Größe erhalten. Andererseits haben wir polynomielle obere Platzschranken bez. des Maximums von Eingabe- und Ausgabe-Größe gezeigt. So stellt sich die Frage, ob sich die betrachteten Graphprobleme durch OBDD-basierte Algorithmen auf polynomiellem Platz bez. ihrer Eingabe-OBDD-Größe lösen lassen – oder ob die Platzkomplexität wenigstens durch einen Ausdruck $\alpha(w)\cdot\text{poly}(\Lambda(I))$ für die maximale Vollständige-Eingabe-OBDD-Breite w und Eingabegröße $\Lambda(I)$ beschränkt ist.

Diese Frage ist offenbar nur für Suchprobleme sinnvoll, da für Entscheidungsprobleme stets ein Polynomialplatz-Schema angewandt werden kann. So werden die letzten Ergebnisse dieser Dissertation konkrete exponentielle untere Platzschranken für einige der betrachteten Suchprobleme auf Graphen sein, die daher nicht in I-FPT-S enthalten sind. Für die übrigen Probleme bleibt diese Frage noch offen. Wir beginnen mit sehr einfachen Resultaten für die drei Zusammenhangs-Zerlegungs-Probleme **Zusammenhangs-Zerlegung**, **Zweizusammenhangs-Zerlegung** und **Starke-Zusammenhangs-Zerlegung**.

Theorem 7.3.8.

$$\left.\begin{array}{r}\textbf{Zusammenhangs-Zerlegung},\\ \textbf{Zweizusammenhangs-Zerlegung},\\ \textbf{Starke-Zusammenhangs-Zerlegung}\end{array}\right\} \notin \text{I-FPT-S}.$$

Beweis: Der Graph G_m mit Knotenmenge $V_m := \{v_0,\ldots,v_{2^m-1}\}$ und leerer Kantenmenge $E_m := \emptyset$ hat offenbar die charakteristische Funktion $\chi_{G_m} = 0$ in B_{2m},

das entsprechende OBDD hat konstante Größe. G_m kann sowohl als ungerichteter als auch gerichteter Graph aufgefasst werden und besteht aus 2^m nicht vergrößerbaren zusammenhängenden, zweizusammenhängenden bzw. stark zusammenhängenden Teilgraphen, die jeweils nur aus einem isolierten Knoten bestehen. Ausgabe aller drei Probleme sind also 2^m OBDDs. □

Eine für die Praxis sinnvollere Definition der drei Probleme könnte derartig entartete Ausgaben ausschließen. Interessanter wäre daher, analog zu den folgenden Ergebnissen für Kürzeste-Wege-Probleme, Erreichbarkeitsprobleme und das **Topologische Sortieren** auch exponentielle untere Schranken für die Größe einzelner Ausgabe-OBDDs zu zeigen, die spezielle nicht vergrößerbare Teilgraphen implizit darstellen. Dies ist bisher jedoch noch nicht gelungen.

7.3.3 SSSP und APSP

Kommen wir nun zum Kürzeste-Wege-Problem **SSSP**. Eingabe sei also ein durch π-OBDD$[\chi_G]$ OBDD-repräsentierter Graph $G = (V, E, \phi)$, während die Ausgabe π-OBDD$[\chi_\Delta]$ die Distanzfunktion $\Delta\colon V \to \mathbb{N}$ darstellt.

Wir definieren die pathologischen Graphen $(G_m)_m$ mit entsprechenden Lösungen $(\Delta_m)_m$. $G_m := (V_m, E_m, \phi_m)$ besteht aus 2^m Komponenten H_i mit $0 \leq i < 2^m$. Jedes H_i ist ein Pfad aus 2^m Knoten $w_{i,0}, \ldots, w_{i,2^m-1}$ und Kanten $(w_{i,j}, w_{i,j+1})$ mit Länge i für $0 \leq i < 2^m$. Außerdem gibt es einen gemeinsamen Quellenknoten q, der mit allen H_is durch Kanten $(q, w_{i,0})$ der Länge 0 verbunden ist. Somit hat der Pfad $(q, \ldots, w_{i,j})$ die Länge ij und es ist folglicherweise $\Delta(w_{i,j}) = ij$. Wir haben also ähnlich wie in Abschnitt 5.3 den Graphen der Multiplikation in χ_Δ codiert. Zudem fügen wir im Sinne einer komfortablen impliziten Darstellung $2^{2(m+1)} - (2^{2m}+1)$ zusätzliche isolierte Knoten zu G hinzu und nummerieren seine Knoten $V_m := \{v_0, \ldots, v_{2^{2(m+1)}-1}\}$ durch $w_{i,j} := v_{i2^{m+1}+j}$ und $q := v_{2^{2m+1}}$.

Wir betrachten die Voll-OBDD-Breite von G_m.

Lemma 7.3.9. *Die Funktion χ_{G_m} hat konstante Voll-OBDD-Breite bez. $\pi^{\mathrm{id}}_{6,m+1}$.*

Beweis: Sowohl Knotennummern als auch Kantenlängen seien also durch jeweils $2(m+1)$ Bits codiert. Wir referenzieren die Teilargumente einer Knotennummer $x \in \mathbb{B}^{2(m+1)}$ mit $i(x) := (x_{m+1}, \ldots, x_{2m+1})$ und $j(x) := (x_0, \ldots, x_m)$. Wir beginnen mit der Definition einer Hilfsfunktion $\chi^*_{G_m}(x, y, a)$ für $x, y, a \in \mathbb{B}^{2(m+1)}$:

$$\begin{aligned}\chi^*_{G_m}(x,y,a) := {} & \left[(|x| = 2^{2m+1} \neq |y|) \wedge (|j(y)| = |a| = 0)\right] \\ & \vee \left[(|x| \neq 2^{2m+1} \neq |y|) \wedge (|i(x)| = |i(y)| = |a|) \wedge (|j(y)| = |j(x)| + 1)\right].\end{aligned}$$

Obwohl diese Definition den Graphen G_m korrekt widerzuspiegeln scheint, haben wir noch nicht die zusätzlich entstandenen isolierten Knoten beachtet. Daher sei die

Funktion $K \in B_{2(m+1)}$ durch

$$K(x) := (|x| = 2^{2m+1}) \vee (|i(x)|, |j(x)| < 2^m)$$

definiert, die genau für alle ursprünglichen Knoten von G_m auf 1 abbildet. So erhalten wir

$$\chi_{G_m}(x, y, a) := K(x) \wedge K(y) \wedge \chi^*_{G_m}(x, y, a).$$

Wir haben die charakteristische Funktion χ_{G_m} durch einen Ausdruck konstanter Länge gemäß Theorem 3.7.4 dargestellt. Seine grundlegenden Bestandteile sind multivariate Vergleichsfunktionen mit konstantem maximalem absolutem Gewicht, die nach Korollar 3.7.5 konstante Voll-OBDD-Breite bez. $\pi^{\mathrm{id}}_{6,m+1}$ haben. Insgesamt folgt also auch für χ_{G_m} konstante Voll-OBDD-Breite bez. $\pi^{\mathrm{id}}_{6,m+1}$. □

Lemma 7.3.10. *OBDDs für χ_{Δ_m} haben exponentielle Größe in m.*

Beweis: Wir zeigen eine untere Schranke für eine Subfunktion $f_m \in B_{4m}$ von $\chi_{\Delta_m}(x, a)$, die wir durch die Ersetzung der Variablen x_{2m+1}, x_m, a_{2m+1} sowie a_{2m} durch 0 erhalten. Somit repräsentiert das Argument x einen $w_{i,j}$-Knoten mit $i, j < 2^m$ und es gilt

$$f_m(x, a) = 1 \Leftrightarrow |i(x)| \cdot |j(x)| = |a|$$

mit $|a| < 2^{2m}$. Es ist daher $f_m = MUL_m\text{-}GRAPH$ und hat laut Theorem 4.3.1 exponentielle OBDD-Größe für jede Variablenordnung. □

Somit haben wir gezeigt, dass es eine Eingabeinstanz für **SSSP** mit konstanter Voll-OBDD-Breite bzgl. einer geeigneten Variablenordnung gibt, für die jedoch jeder OBDD-basierte Algorithmus ein exponentiell größeres Ausgabe-OBDD produzieren muss.

Theorem 7.3.11. SSSP $\notin$ I-FPT-S.

Da die Ausgabe $\chi_{\Delta,\mathrm{SSSP}}(x, a)$ des Problems **SSSP** eine Subfunktion der Ausgabe $\chi_{\Delta,\mathrm{APSP}}(x, y, a)$ des Problems **APSP** ist, gilt ebenso:

Korollar 7.3.12. APSP $\notin$ I-FPT-S.

G_m enthält im Sinne einer einfacheren Notation Nullkanten $(q, w_{i,0})$, die wir jedoch leicht ersichtlich mit ein wenig technischem Aufwand durch Kanten der Länge 1 ersetzen können, und so gilt ebenfalls:

Theorem 7.3.13. SSSP$^+$, APSP$^+$ $\notin$ I-FPT-S.

7.3.4 Flussmaximierung

Wir fahren mit einer exponentiellen unteren Schranke für die Platzkomplexität der OBDD-basierten **Flussmaximierung** fort. Eingabe ist also ein OBDD π-OBDD$[\chi_G]$ eines gewichteten Graphen $G = (V, E, \phi)$ sowie Quelle $q \in V$ und Senke $s \in V$, für den wir einen maximalen q-s-Fluss $F\colon E \to \mathbb{N}$ berechnen wollen. Diese Ausgabe soll ebenfalls kanonisch implizit durch ein OBDD π-OBDD$[\chi_F]$ mit $\chi_F(x, y, a) = 1 :\Leftrightarrow (v_{|x|}, v_{|y|}) \in E \wedge F(v_{|x|}, v_{|y|}) = |a|$ präsentiert werden.

Wir definieren die pathologischen Graphen $(G_m)_m$ mit eindeutigem maximalem Fluss F_m. $G_m := (V_m, E_m, \phi_m)$ besteht aus 2^{2m} Komponenten $H_{i,j}$ für $0 \leq i, j < 2^m$. Jedes $H_{i,j}$ besteht aus $j + 2$ Knoten $q_{i,j}$, $s_{i,j}$ und $w_{i,j,\ell}$ für $0 \leq \ell < j$. $H_{i,j}$ enthält $2j$ Kanten $(q_{i,j}, w_{i,j,\ell})$ sowie $(w_{i,j,\ell}, s_{i,j})$ mit jeweiliger Kapazität i. Globale Quelle q und und Senke s sind mit allen Komponenten $H_{i,j}$ durch Kanten $(q, q_{i,j})$ und $(s_{i,j}, s)$ der Kapazität 2^{2m} verbunden. Offenbar schickt F_m genau ij Flusseinheiten durch jedes $H_{i,j}$ und es gilt $F_m(q, q_{i,j}) = ij$.

Die Knoten in $V_m := \{v_0, \ldots, v_{2^{3(m+1)}-1}\}$ nummerieren wir nach dem folgenden Schema:

$$\begin{aligned} w_{i,j,\ell} &:= v_{i2^{2(m+1)}+j2^{m+1}+\ell}, \\ q_{i,j} &:= v_{i2^{2(m+1)}+j2^{m+1}+2^m}, \\ s_{i,j} &:= v_{i2^{2(m+1)}+j2^{m+1}+2^m+1}, \\ q &:= v_{2^{2m+1}}, \\ s &:= v_{2^{2m+1}+1}. \end{aligned}$$

Lemma 7.3.14. *Die Funktion χ_{G_m} hat konstante Voll-OBDD-Breite bez. $\pi^{\mathrm{id}}_{9,m+1}$.*

Beweis: Sowohl Knotennummern als auch Kantenkapazitäten seien also durch jeweils $3(m + 1)$ Bits codiert. Wir referenzieren die Teilargumente einer Knotennummer $x \in \mathbb{B}^{3(m+1)}$ mit $i(x) := (x_{2m+2}, \ldots, x_{3m+2})$, $j(x) := (x_{m+1}, \ldots, x_{2m+1})$ und $\ell(x) := (x_0, \ldots, x_m)$. Wir beginnen mit der Definition einer Hilfsfunktion $\chi^*_{G_m}(x, y, a)$ für $x, y, a \in \mathbb{B}^{3(m+1)}$:

$$\begin{aligned} \chi^*_{G_m}(x, y, a) := {} & \big[(|x| = 2^{2m+1}) \wedge (v_{|y|} \in \{q_{i,j} \mid i, j\}) \wedge (|a| = 2^{2m})\big] \\ & \vee \big[(v_{|x|} \in \{s_{i,j} \mid i, j\}) \wedge (|y| = 2^{2m+1} + 1) \wedge (|ab| = 2^{2m})\big] \\ & \vee \big[(v_{|x|} \in \{q_{i,j}\}_{i,j}) \wedge (v_{|y|} \in \{w_{i,j,\ell} \mid i, j, \ell\}) \\ & \wedge (|i(x)| = |i(y)| = |a|) \wedge (|\ell(y)| < |j(x)| = |j(y)|)] \\ & \vee \big[(v_{|x|} \in \{w_{i,j,\ell} \mid i, j, \ell\}) \wedge (v_{|y|} \in \{s_{i,j} \mid i, j\}) \\ & \qquad \wedge (|i(x)| = |i(y)| = |a|) \wedge (|\ell(x)| < |j(x)| = |j(y)|)\big]. \end{aligned}$$

Obwohl bereits $\chi^*_{G_m}$ den Graphen G_m korrekt widerzuspiegeln scheint, haben wir noch nicht die zusätzlich entstandenen isolierten Knoten beachtet. Daher sei die Funktion $K \in B_{3(m+1)}$ durch

$$K(x) := (v_{|x|} \in \{q, q_{i,j}, w_{i,j,\ell}, s_{i,j}, s \mid i, j, \ell\})$$

definiert, die genau für alle ursprünglichen Knoten von G_m auf 1 abbildet. So erhalten wir wieder

$$\chi_{G_m}(x, y, a) := K(x) \wedge K(y) \wedge \chi^*_{G_m}(x, y, a).$$

Wir haben die charakteristische Funktion χ_{G_m} durch einen Ausdruck konstanter Länge gemäß Theorem 3.7.4 dargestellt. Seine grundlegenden Bestandteile sind einerseits multivariate Vergleichsfunktionen mit konstantem maximalem absolutem Gewicht, die nach Korollar 3.7.5 konstante Voll-OBDD-Breite bez. $\pi^{\mathrm{id}}_{9,m+1}$ haben. Andererseits lassen sich Prüfungen des Knotentyps wie etwa

$$v_{|x|} \in \{q_{i,j} \mid i, j\} \Leftrightarrow (|i(x)|, |j(x)| < 2^m) \wedge (|\ell(m)| = 2^m)$$

ebenfalls durch Komposition derartiger multivariater Vergleichsfunktionen repräsentieren. Insgesamt folgt also auch für χ_{G_n} konstante Voll-OBDD-Breite bez. $\pi^{\mathrm{id}}_{9,m+1}$. □

Lemma 7.3.15. *OBDDs für χ_{F_m} haben exponentielle Größe in m.*

Beweis: Wir zeigen eine untere Schranke für eine Subfunktion $f_m \in B_{4m}$ von $\chi_{F_m}(x, y, a)$. Wir erhalten f_m, indem wir die Ersetzungen $x := (2^{2m+1})_2$, $y_{3m+2}, y_{2m+1} := 0$, $\ell(y) := (2^m)_2$ sowie $(a_{2m}, \ldots, a_{3m+2}) := (0)_2$ durchführen. Somit stellen die verbleibenden freien y-Variablen einen $q_{i,j}$-Knoten mit $i, j < 2^m$ dar. Der maximale Fluss F_m schickt $ij < 2^{2m}$ Flusseinheiten durch $(q, q_{i,j})$ und so gelten

$$f_m(y, a) = 1 \Leftrightarrow |i(y)| \cdot |j(y)| = |a|$$

und $f_m = MUL_m\text{-}GRAPH$. Damit hat χ_{F_m} laut Theorem 4.3.1 exponentielle OBDD-Größe für jede Variablenordnung. □

Somit haben wir gezeigt, dass es eine Eingabeinstanz für **Flussmaximierung** mit konstanter Voll-OBDD-Breite bzgl. einer geeigneten Variablenordnung gibt, für die jedoch jeder OBDD-basierte Algorithmus ein exponentiell größeres Ausgabe-OBDD produzieren muss.

Theorem 7.3.16. Flussmaximierung $\notin$ I-FPT-S.

7.3.5 *q*-Erreichbarkeit und Transitiver Abschluss

Das Problem ***q*-Erreichbarkeit** hat wichtige Anwendungen in den Gebieten CAD und Model Checking (siehe z. B. Abschnitte 13.2 und 13.3 in Wegener (2000)). Auch hier werden wir eine exponentielle untere Schranke für die Platzkomplexität beweisen. Allerdings wird hier die Codierung einer für OBDDs schwierigen Funktion in die Eingabeinstanz nicht so nahe liegend sein wie in den beiden vorigen Abschnitten, und so gelingt nur ein eingeschränktes Resultat.

Wir erinnern uns, dass wir die durch $\chi_R \in B_n$ implizit repräsentierte Menge $R \in V$ der von einem Startknoten q aus in $G = (V, E)$ erreichbaren Knoten berechnen möchten. Eingabe ist also wieder ein OBDD für $\chi_G \in B_{2n}$ mit $n = \log|V| + \mathcal{O}(1)$. Die beiden letzten Theoreme nutzen eine untere Schranke für das Ausgabe-OBDD des jeweiligen Problems und sind somit unabhängig vom verwandten Algorithmus. Hier sind wir jedoch leider auf eine untere Schranke für die OBDD-Größe einer Zwischenfunktion angewiesen, die von einer wichtigen Klasse von Algorithmen berechnet wird. Alle dem Autor bekannten impliziten Algorithmen für das Problem ***q*-Erreichbarkeit** lösen nämlich sukzessiv Teilprobleme, in denen die Länge erlaubter Pfade q zu einem erreichbaren Knoten iterativ erhöht wird. Das Ergebnis dieses Abschnitts wird also nicht allgemeingültig sein, jedoch für eine Problemvariante gelten, die in praktischer Hinsicht als gleichberechtigt zum allgemeinen Problem betrachtet werden kann.

Definition 7.3.17 (*q*-Erreichbarkeit*). *Gegeben seien ein Graph $G = (V, E)$ und ein Knoten $q \in V$. Gib die Mengen $R_1, \ldots, R_p$ aus, sodass R_i alle Knoten $v \in V$ mit einem q-v-Pfad in G mit nicht mehr als 2^i Kanten für $i \in \{1, \ldots, p\}$ und $p := \lceil \log|V| \rceil$ enthält.*

Da wir weiterhin eine untere Schranke für die OBDD-Größe der Funktion MUL_{2n-1} verwenden werden, die momentan nur für eine Teilmenge der möglichen Variablenordnungen gelingt, gilt die untere Platzschranke nur für Algorithmen, die die Anfangsvariablenordnung der Eingabe nicht wesentlich verändern.

Wir beginnen mit der Konstruktion einer pathologischen Eingabeinstanz $(G_m)_m$, die aus 2^{2m} Komponenten $H_{i,j}$ für $0 \leq i, j < 2^m$ besteht. Jedes $H_{i,j}$ ist die Konkatenation $P_{i-1} \ldots P_0$ der Pfade $P_\ell := (w_{i,j,\ell,j-1}, \ldots, w_{i,j,\ell,0})$ für $0 \leq \ell < i$. Benachbarte Teilpfade P_ℓ und $P_{\ell-1}$ sind durch die Kanten $(w_{i,j,\ell,0}, w_{i,j,\ell-1,j-1})$ verbunden. Außerdem ist ein gemeinsamer Quellenknoten q an alle $H_{i,j}$s durch Kanten $(q, w_{i,j,i-1,j-1})$ angebunden. Die Knoten in $V_m := \{v_0, \ldots, v_{2^{4(m+1)}-1}\}$ nummerieren wir durch $w_{i,j,\ell,r} := v_{i2^{3(m+1)}+j2^{2(m+1)}+\ell 2^{m+1}+r}$ und $q := v_{2^{4m+3}}$.

Lemma 7.3.18. *Die Funktion χ_{G_m} hat konstante Voll-OBDD-Breite bez. $\pi^{\mathrm{id}}_{8,m+1}$.*

Beweis: Knotennummern seien also durch jeweils $4(m+1)$ Bits codiert. Wir referenzieren die Teilargumente einer Knotennummer $x \in \mathbb{B}^{4(m+1)}$ mit $i(x) := (x_{3m+3}, \ldots, x_{4m+3})$, $j(x) := (x_{2m+2}, \ldots, x_{3m+2})$, $\ell(x) := (x_{m+1}, \ldots, x_{2m+1})$ und $r(x) := (x_0, \ldots, x_m)$. Wir beginnen mit der Definition einer Hilfsfunktion $\chi^*_{G_m}(x, y)$ für $x, y \in \mathbb{B}^{4(m+1)}$:

$$\begin{aligned}\chi^*_{G_m}(x,y) := \big[(|x| = 2^{4m+3} \neq |y|) \wedge (|\ell(y)| = |i(y)| - 1) \wedge (|r(y)| = |j(y)| - 1)\big] \\ \vee \big[(|x| \neq 2^{2m+1} \neq |y|) \wedge (|i(x)| = |i(y)|) \wedge (|j(x)| = |j(y)|) \\ \wedge (|\ell(x)| = |\ell(y)| < |i(x)|) \wedge (|j(x)| > |r(x)| = |r(y)| + 1)\big]\end{aligned}$$

$$\vee \big[(|x| \neq 2^{2m+1} \neq |y|) \wedge (|i(x)| = |i(y)|) \wedge (|j(x)| = |j(y)| \\ \wedge (|i(x)| > |\ell(x)| = |\ell(y)| + 1) \wedge (|r(x)| = 0) \wedge (|r(y)| = |j(y)| - 1)\big].$$

Obwohl bereits $\chi^*_{G_m}$ den Graphen G_m korrekt widerzuspiegeln scheint, haben wir noch nicht die zusätzlich entstandenen isolierten Knoten beachtet. Daher sei die Funktion $K \in B_{4(m+1)}$ durch

$$K(x) := (v_{|x|} = 2^{4m+3}) \vee (|i(x)|, |j(x)|, |\ell(x)|, |r(x)| < 2^m)$$

definiert, die genau für alle ursprünglichen Knoten von G_m auf 1 abbildet. So erhalten wir wieder

$$\chi_{G_m}(x, y, a) := K(x) \wedge K(y) \wedge \chi^*_{G_m}(x, y, a).$$

Wir haben die charakteristische Funktion χ_{G_m} durch einen Ausdruck konstanter Länge gemäß Theorem 3.7.4 dargestellt. Seine grundlegenden Bestandteile sind einerseits multivariate Vergleichsfunktionen mit konstantem maximalem absolutem Gewicht, die nach Korollar 3.7.5 konstante Voll-OBDD-Breite bez. $\pi^{\mathrm{id}}_{8,m+1}$ haben. Insgesamt folgt also auch für χ_{G_n} konstante Voll-OBDD-Breite bez. $\pi^{\mathrm{id}}_{8,m+1}$. □

Bevor wir zu einer konkreten unteren Platzschranke kommen, müssen wir zunächst ein Hilfsmittel zur Verfügung stellen. Während die OBDD-Größe der Funktion $MUL_{n,n-1}$ bereits umfassend untersucht wurde, so existierte für $MUL_{n,2n-1}$ bisher nicht einmal eine nicht triviale untere Schranke für eine konkrete Variablenordnung $\pi \in \Sigma_{2n}$ (siehe Problem 4.12 in Wegener (2000)). Wir stellen eine entsprechende Schranke vor und wenden sie im Rahmen der unteren Platzschranke für ***q*-Erreichbarkeit*** an.

Theorem 7.3.19. *Die Größe von* $\pi^{\mathrm{id}}_{2,n}$-OBDD$[MUL_{n,2n-1}]$ *ist mindestens* $2^{(n-5)/6} - o(1)$.

Wir wenden dieses Resultat zunächst an und beweisen es anschließend. Ein OBDD-basierter Algorithmus für ***q*-Erreichbarkeit*** berechnet nach Definition 7.3.17 unter anderem ein OBDD für die Funktion $\chi_{R_{m,2m-1}} \in B_m$, die alle Knoten $v \in V_m$ mit einem q-v-Pfad mit höchstens 2^{2m-1} Kanten darstellt (beachte $2m - 1 < \lfloor \log |V_n| \rfloor$).

Lemma 7.3.20. *Die Größe von* $\pi^{\mathrm{id}}_{4,m+1}$-OBDD$[\chi_{R_{m,2m-1}}]$ *ist exponentiell in* m.

Beweis: Wir zeigen eine untere Schranke für eine Subfunktion $f_m \in B_m$ von $\chi_{R_{m,2m-1}}$, die wir durch die Ersetzung von x_{4m+3}, x_{3m+2} und $x_0, \ldots, x_{2m+1}$ durch 0 erhalten. Argument x stellt also einen $w_{i,j,0,0}$-Knoten dar, der genau dann von q aus über höchstens 2^{2m-1} Kanten erreichbar ist, wenn die Länge ij des q-$w_{i,j,0,0}$-Pfades in G_m nicht größer als 2^{2m-1} ist. Es gilt also

$$f_m(x) = 1 \Leftrightarrow |i(x)| \cdot |j(x)| \leq 2^{2m-1}.$$

Sei $g_m(x) = 1 :\Leftrightarrow |i(x)| \cdot |j(x)| = 2^{2m-1}$. Leicht ersichtlich ist die $\pi^{\mathrm{id}}_{2,m}$-OBDD-Größe von g_m $\mathcal{O}(m^2)$. Das $\pi^{\mathrm{id}}_{2,m}$-OBDD der Funktion

$$h_m(x) := f_m(x) \wedge \overline{g(x)} = (|i(x)| \cdot |j(x)| < 2^{2m-1})$$

ist daher höchstens polynomiell größer als das der Funktion f_m. Aufgrund von

$$MUL_{m,2m-1}(x, y) = 1 :\Leftrightarrow |x| \cdot |y| \geq 2^{2m-1}$$

für $x, y \in \mathbb{B}^m$ gilt weiterhin $\overline{h_m} = MUL_{m,2m-1}$.

Insgesamt haben wir damit gezeigt, dass die $\pi^{\mathrm{id}}_{2,m}$-OBDD-Größe von $MUL_{m,2m-1}$ höchstens polynomiell größer ist als die von $\chi_{R_{m,2m-1}}$, woraus die Aussage des Lemmas folgt. □

Somit haben wir gezeigt, dass es eine Eingabeinstanz für ***q*-Erreichbarkeit*** mit konstanter Voll-OBDD-Breite bzgl. der Variablenordnung $\pi^{\mathrm{id}}_{8,m+1}$ gibt, für die jedoch jeder OBDD-basierte Algorithmus ein exponentiell größeres Ausgabe-OBDD produzieren muss, der ausschließlich diese Variablenordnung verwendet. Die entsprechende Teilmenge von I-FPT-S bezeichnen wir mit I-FPT-S*.

Theorem 7.3.21. ***q*-Erreichbarkeit*** $\notin$ I-FPT-S* $\subseteq$ I-FPT-S.

Analog können wir auch **Transitiver Abschluss** einschränken und direkt ein entsprechendes Ergebnis erhalten.

Definition 7.3.22 (Transitiver Abschluss*). *Gegeben sei ein Graph $G = (V, E)$. Gib die Menge der Graphen $A_1 := (V, E_1), \ldots, A_p := (V, E_p)$ aus, sodass $(u, w) \in E_i$ genau dann, wenn G einen u-w-Pfad mit höchstens 2^i Kanten für $i \in \{1, \ldots, p\}$ und $p := \lceil \log |V| \rceil$ enthält.*

Korollar 7.3.23. *Da die Ausgabe $\chi_{R_{m,2m-1}}(x)$ des Problems* q-Erreichbarkeit* *eine Subfunktion der Ausgabe $\chi_{A_{m,2m-1}}(x, y)$ des Problems* Transitiver Abschluss* *auf der Instanz χ_{G_m} ist, gilt ebenso* **Transitiver Abschluss*** $\notin$ I-FPT-S*.

Da $\chi_{R_{m,2m-1}}(x)$ bez. der Problemvarianten ***q*-Erreichbarkeit*** und **Transitiver Abschluss*** Teil der Ausgabe ist, haben wir lediglich Resultate für die eingabebezogene Klasse I-FPT-S* erhalten. Dies Erweiterung der Ausgabe war jedoch ein formaler Kunstgriff – tatsächlich sind wir nur an den eigentlichen Ausgaben $\chi_R(x)$ und $\chi_A(x, y)$ interessiert. Diese wiederum haben für G_m offenbar konstante Voll-OBDD-Breite: G_m ist zusammenhängend, und die Ausgabefunktionen müssen lediglich testen, ob ihre beiden Knotenargumente tatsächliche Graphknoten repräsentieren oder isolierte Knoten, die bei der binären Codierung entstanden sind. So ist dieses Resultat letztlich wieder vergleichbar mit den Resultaten für IO-FPT-S: Trotz konstanter Eingabe- und Ausgabe-Breite können OBDD-basierte Algorithmen bei der Berechnung von Teillösungen in Schwierigkeiten geraten.

Schließlich verbleibt der Beweis von Theorem 7.3.19. Anschließend werden wir in Anmerkung 7.3.28 die Annahme einer unveränderlichen Variablenordnung etwas lockern können und diese neue Annahme werten. Zuvor führen wir benötigte Grundlagen aus der analytischen Zahlentheorie ein (siehe z. B. Kuipers und Niederreiter (1974)). In diesem Kontext sei i die imaginäre Zahl, π bezeichne die mathematische Konstante und e sei die eulersche Zahl. Zudem notieren wir die Ableitungsfunktion einer differenzierbaren reellen Funktion f mit f'.

Definition 7.3.24. *Sei $\sigma := \sigma_1, \ldots, \sigma_M$ eine endliche Folge reeller Zahlen.*

(a) Für $\alpha, \beta \in [0,1]$ bezeichne $A([\alpha,\beta);\sigma)$ die Anzahl der Elemente σ_ℓ in σ, deren gebrochener Teil $(\sigma_\ell \bmod 1)$ in $[\alpha,\beta)$ liegt.

(b) Die Zahl

$$D(\sigma) := \sup_{\alpha,\beta} \left| |\beta - \alpha| - \frac{A([\alpha,\beta);\sigma)}{M} \right|$$

heißt Diskrepanz *von σ.*

Das folgende Theorem geht auf Erdös und Turán zurück (siehe Kapitel 1, Theorem 2.7 in Kuipers und Niederreiter (1974)).

Theorem 7.3.25. *Für jede endliche Folge $\sigma := \sigma_1, \ldots, \sigma_M$ reeller Zahlen und jede positive Konstante $k \in \mathbb{N}^+$ gilt*

$$D(\sigma) \leq \frac{6}{k+1} + \frac{4}{\pi} \sum_{h=1}^{k} \left(\frac{1}{h} - \frac{1}{k+1} \right) \left| \frac{1}{M} \sum_{\ell=1}^{M} \mathrm{e}^{2\pi \mathrm{i} h \sigma_\ell} \right|. \tag{7.3.1}$$

Um später die exponentielle Summe in Gleichung (7.3.1) abschätzen zu können, führen wir ein Ergebnis von van der Corput ein (siehe Kapitel 2, Theorem 2.5 in Kuipers und Niederreiter (1974)).

Theorem 7.3.26. *Seien u und v ganze Zahlen mit $u < v$ und sei $\psi\colon \mathbb{R} \to \mathbb{R}$ zweifach differenzierbar auf $[u,v]$ mit $\psi''(z) \geq \rho > 0$ oder $\psi''(z) \leq -\rho < 0$ für $z \in [u,v]$. Dann gilt*

$$\left| \sum_{\ell=u}^{v} \mathrm{e}^{2\pi \mathrm{i} \psi(\ell)} \right| \leq (|\psi'(v) - \psi'(u)| + 2)(4/\sqrt{\rho} + 3).$$

Wir nehmen vorübergehend $n = 6m$ für ein $m \in \mathbb{N}$ an. Sei $a \in \{0, \ldots, 2^n - 1\}$ mit $a = a_{\mathrm{H}} \cdot 2^m + a_{\mathrm{L}}$ für $a_{\mathrm{L}} \in \{0, \ldots, 2^m - 1\}$ und $a_{\mathrm{H}} \in \{0, \ldots, 2^{5m} - 1\}$. Seien zudem b, b_{L} und b_{H} in der gleichen Weise definiert. Wir legen $b_{\mathrm{L}} := 0$ fest und betrachten $ab = a_{\mathrm{L}} b_{\mathrm{H}} 2^m + a_{\mathrm{H}} b_{\mathrm{H}} 2^{2m}$.

Lemma 7.3.27. *Für hinreichend großes m existieren für jedes $a_{\mathrm{L}} > 0$ entsprechende a_{H} und b_{H}, sodass $(a-1)b < 2^{2n-1} \leq ab$ gilt.*

Beweis: Wir legen $a_\mathrm{H}, b_\mathrm{H} \geq 2^{5m-1}+1$ fest und schreiben $(a-1)b < 2^{2n-1} \leq ab$ als

$$\begin{aligned} & (a_\mathrm{L}-1)\cdot b_\mathrm{H}\cdot 2^m + a_\mathrm{H}\cdot b_\mathrm{H}\cdot 2^{2m} < 2^{12m-1} \leq a_\mathrm{L}\cdot b_\mathrm{H}\cdot 2^m + a_\mathrm{H}\cdot b_\mathrm{H}\cdot 2^{2m} \\ \Leftrightarrow\ & (a_\mathrm{L}-1)\cdot b_\mathrm{H}\cdot 2^m < 2^{12m-1} - a_\mathrm{H}\cdot b_\mathrm{H}\cdot 2^{2m} \leq a_\mathrm{L}\cdot b_\mathrm{H}\cdot 2^m \\ \Leftrightarrow\ & \frac{a_\mathrm{L}-1}{2^m} < \frac{2^{10m-1}}{b_\mathrm{H}} - a_\mathrm{H} \leq \frac{a_\mathrm{L}}{2^m} \end{aligned}$$

Nun wählen wir $a_\mathrm{H} := \lfloor 2^{10m-1}/b_\mathrm{H} \rfloor$. Die Aussage des Lemmas ist nun äquivalent zu

$$\frac{a_\mathrm{L}-1}{2^m} < \frac{2^{10m-1}}{b_\mathrm{H}} \bmod 1 \leq \frac{a_\mathrm{L}}{2^m}, \tag{7.3.2}$$

wobei $z \bmod 1$ wieder den gebrochenen Teil einer Zahl $z \in \mathbb{R}$ bezeichnet. Wir zeigen, dass für hinreichend großes m jedes $a_\mathrm{L} \in \{1,\ldots,2^m-1\}$ ein entsprechendes $b_\mathrm{H} \in \{2^{5m-1}+1,\ldots,2^{5m}-1\}$ existiert, das Gleichung (7.3.2) erfüllt.

Diese Behauptung ist äquivalent zu

$$A\left(\left(\frac{a_\mathrm{L}-1}{2^m}, \frac{a_\mathrm{L}}{2^m}\right]; \sigma\right) \geq 1$$

für $\sigma := (2^{10m-1}/b_\mathrm{H})_{b_\mathrm{H}}$. Für ein $\varepsilon > 0$ und hinreichend großes m gilt

$$\frac{1}{2^{m+1}} \leq \frac{1-\varepsilon}{2^m} - \frac{1}{2^{5m-1}-1}$$

und wir haben

$$\begin{aligned} D(\sigma) &\overset{!}{\leq} \frac{1}{2^{m+1}} \\ &\overset{\text{Def. 7.3.24}}{\Rightarrow} \frac{1-\varepsilon}{2^m} - A\left(\left[\frac{a_\mathrm{L}-1+\varepsilon}{2^m}, \frac{a_\mathrm{L}}{2^m}\right); \sigma\right) \Big/ \left(2^{5m-1}-1\right) \leq \frac{1-\varepsilon}{2^m} - \frac{1}{2^{5m-1}-1} \\ &\Rightarrow A\left(\left[\frac{a_\mathrm{L}-1+\varepsilon}{2^m}, \frac{a_\mathrm{L}}{2^m}\right); \sigma\right) \geq 1 \Rightarrow A\left(\left(\frac{a_\mathrm{L}-1}{2^m}, \frac{a_\mathrm{L}}{2^m}\right]; \sigma\right) \geq 1. \end{aligned}$$

Es verbleibt zu zeigen, dass $D(\sigma) \leq 1/2^{m+1}$ gilt. Um die Notation zu vereinfachen, definieren wir $\sigma' := (2^{10m-1}/(\ell + 2^{5m-1}))_\ell$ für $\ell \in \{1,\ldots,2^{5m-1}-1\}$. Außerdem seien $\psi_h(z) := h \cdot 2^{10m-1}/(z+2^{5m-1})$ für $h \in \mathbb{N}^+$ und $z \in \mathbb{R}$. Wir bemerken, dass $\psi_h(\ell) = h \cdot \sigma'_\ell = h \cdot \sigma_{\ell+2^{5m-1}}$. Für $z \in [1, 2^{5m-1}-1]$ gilt nun

$$\psi''_h(z) = \frac{h \cdot 2^{10m}}{(z+2^{5m-1})^3} > \frac{h}{2^{5m}} \geq \frac{1}{2^{5m}} =: \rho.$$

Wir wenden Theorem 7.3.26 an.

$$\left|\sum_{\ell=1}^{2^{5m-1}-1} \mathrm{e}^{2\pi \mathrm{i}\psi_h(\ell)}\right| \leq (|\psi_h'(2^{5m-1}-1) - \psi_h'(1)| + 2)(4/\sqrt{\rho} + 3)$$
$$= \left(\left|-\frac{h \cdot 2^{10m-1}}{(2^{5m}-1)^2} + \frac{h \cdot 2^{10m-1}}{(2^{5m-1}+1)^2}\right| + 2\right)\left(4 \cdot \sqrt{2^{5m}} + 3\right)$$
$$< \left(\left|\frac{h \cdot 2^{10m-1}}{2^{10m-2}} - \frac{h \cdot 2^{10m-1}}{2^{10m}}\right| + 2\right) \cdot 2^{5m/2+3} < (h+1) \cdot 2^{5m/2+4} \quad (7.3.3)$$

Diese Abschätzung wird nun benutzt, um die exponentielle Summe in Gleichung (7.3.1) zu ersetzen.

$$D(\sigma') \leq \frac{6}{k+1} + \frac{4}{\pi}\sum_{h=1}^{k}\left(\frac{1}{h} - \frac{1}{k+1}\right)\left|\frac{1}{2^{5m-1}-1}\sum_{\ell=1}^{2^{5m-1}-1} \mathrm{e}^{2\pi \mathrm{i} h\sigma'_\ell}\right|$$
$$\overset{(7.3.3)}{<} \frac{6}{k+1} + \frac{4}{\pi}\sum_{h=1}^{k}\left(\frac{1}{h} - \frac{1}{k+1}\right)\frac{(h+1)\cdot 2^{5m/2+4}}{2^{5m-1}-1}$$
$$< \frac{6}{k+1} + \frac{4}{\pi} \cdot \frac{2^{5m/2+4}}{2^{5m-2}}\sum_{h=1}^{k}\left(\frac{1}{h} - \frac{1}{k+1}\right)(h+1)$$
$$= \frac{6}{k+1} + \frac{1}{\pi 2^{5m/2-8}}\left(\sum_{h=1}^{k}\left(\frac{h+1}{h} - \frac{h}{k+1}\right) - \frac{k}{k+1}\right)$$
$$< \frac{8}{k+1} + \frac{1}{2^{5m/2-8}}\left(2k - \frac{k}{2} - \frac{k}{k+1}\right) < \frac{8}{k} + \frac{k}{2^{5m/2-9}}$$

Die Wahl von $k := 2^{m+5}$ ergibt schließlich

$$D(\sigma) = D(\sigma') < \frac{8}{2^{m+5}} + \frac{2^{m+5}}{2^{5m/2-9}} = \frac{1}{2^{m+2}} + \frac{1}{2^{3m/2-14}} < \frac{1}{2^{m+1}}$$

für hinreichend großes m. □

Nun können wir endlich Theorem 7.3.19 beweisen.

Beweis von Theorem 7.3.19: Wir nehmen wieder $n = 6m$ an. Seien $x, y \in \mathbb{B}^n$ die Variablenvektoren, die den Faktoren der Multiplikation entsprechen, auf denen $MUL_{n,2n-1}$ definiert ist. Wir betrachten die Subfunktion f_n von $MUL_{n,2n-1}$, die wir durch Ersetzung der Variablen $y_0, \ldots, y_{m-1}$ durch 0 erhalten.

Gemäß $\pi^{\mathrm{id}}_{2,n}$ lesen wir die Variablen $x_0, \ldots, x_{m-1}$ sowohl vor $x_m, \ldots, x_{n-1}$ als auch vor $y_m, \ldots, y_{n-1}$. Wir betrachten die Subfunktionen g_n und h_n von f_n, die wir durch Ersetzung von $x_0, \ldots, x_{m-1}$ durch Konstanten $c_0, \ldots, c_{m-1}$ bzw. $d_0, \ldots, d_{m-1}$ mit $|c| < |d|$ erhalten. Für hinreichend großes m gibt es nach Lemma 7.3.27 $x^*, y^* \in \mathbb{B}^{n-m}$, sodass

$g_n(x^*, y^*) = 0 \neq 1 = h(x^*, y^*)$ gilt. Damit ist also $g_n \neq h_n$ für jedes Paar c, d verschiedener konstanter Variablenersetzungen. Es folgt, dass eine Ersetzung der ersten m Variablen von f_n bez. $\pi^{\mathrm{id}}_{2,n}$ eine Anzahl von $2^m = 2^{n/6}$ verschiedenen Subfunktionen ergibt. Die entsprechende untere Schranke für die $\pi^{\mathrm{id}}_{2,n}$-OBDD-Größe von f_n und damit auch von $MUL_{n,2n-1}$ folgt aus Theorem 2.3.6.

Ist n kein Vielfaches von 6, so betrachten wir $MUL_{n',2n'-1}$ für die größte natürliche Zahl $n' < n$ mit $n' = 6m$ für ein $m \in \mathbb{N}$. Die $\pi^{\mathrm{id}}_{2,n'}$-OBDD-Größe von $MUL_{n',2n'-1}$ ist mindestens $2^{n'/6} \geq 2^{(n-5)/6}$. Andererseits gilt offenbar

$$MUL_{n',2n'-1}(x, y) = 1 \Leftrightarrow |x| \cdot |y| \cdot 2^{2(n-n')} \geq 2^{2n-1}$$

und wir erhalten $MUL_{n',2n'-1}$ aus $MUL_{n,2n-1}$, indem wir $x_0, \ldots, x_{n-n'-1}$ und $y_0, \ldots, y_{n-n'-1}$ durch 0 ersetzen und x_ℓ als $x_{\ell-(n-n')}$ sowie y_ℓ als $y_{\ell-(n-n')}$ interpretieren. Somit ist $MUL_{n',2n'-1}$ eine Subfunktion von $MUL_{n,2n-1}$ und die untere Schranke gilt auch für $MUL_{n,2n-1}$. □

Anmerkung 7.3.28. *Im Beweis von Theorem 7.3.19 verwenden wir lediglich, dass die Variablenordnung die Variablen $x_0, \ldots, x_{m-1}$ vor den Variablen $x_m, \ldots, x_{n-1}$ und $y_m, \ldots, y_{n-1}$ liest. In diesem Rahmen dürfen Algorithmen also schon die in Theorem 7.3.19 vorausgesetzte Variablenordnung $\pi^{\mathrm{id}}_{2,n}$ verändern, ohne die untere Schranke zu tangieren. Andererseits ist keine Variablenordnung bekannt, die zu einer polynomiellen OBDD-Größe für $MUL_{n,2n-1}$ führt, und so ist fraglich, ob eine derartige Ordnung (so sie denn überhaupt existiert) von einer der gängigen effizienten Variablenordnungs-Optimierungsheuristiken gefunden werden kann (siehe Abschnitt 5.6 in Wegener (2000)).*

7.3.6 Topologisches Sortieren

Schließlich können wir die Methoden des vorigen Abschnitts direkt für das Problem **Topologisches Sortieren** einsetzen. Wir erinnern uns, dass die Eingabe hier ebenfalls das OBDD einer Funktion $\chi_G \in B_{2n}$ eines gerichteten Graphen $G = (V, E)$ mit $n = \log|V| + \mathcal{O}(1)$ ist und dass das Ausgabe-OBDD für $\chi_\Delta(x, a) \in B_{3n}$ implizit die Funktion $\Delta\colon V \to \mathbb{N}$ darstellt, die Knoten $v \in V$ auf die maximale Länge eines u-v-Pfades abbildet.

Theorem 7.3.29. Topologisches Sortieren $\notin$ I-FPT-S.

Beweis: Wir verwenden den Graphen G_m aus Abschnitt 7.3.5. Seine charakteristische Funktion χ_{G_m} hat nach Lemma 7.3.18 konstante Voll-OBDD-Breite bez. $\pi^{\mathrm{id}}_{8,m+1}$. Andererseits hat jedes OBDD für χ_{Δ_m} exponentielle Größe in m: Wir definieren eine Subfunktion f_m, indem wir zunächst wieder die gleichen Variablenersetzungen von x_{4m+3}, x_{3m+2} und $x_0, \ldots, x_{2m+1}$ durch 0 wie in Lemma 7.3.20 durchführen. Argument x stellt also wieder einen $w_{i,j,0,0}$-Knoten dar. Es gilt offenbar $\Delta_m(w_{i,j,0,0}) = ij$ und somit auch $f_m = MUL_m\text{-}GRAPH$. Die Aussage folgt nun aus Theorem 4.3.1. □

7.4 Übersicht der Komplexitätsresultate

Wir verschaffen uns in Tabelle 7.4.1 einen Überblick über die zeit- und platzbezogenen Komplexitätsergebnisse dieses Kapitels. Dabei nehmen wir P $\neq$ PSPACE an. Wir haben Aussagen mit einem „$*$“ markiert, die nur unter der Annahme unveränderlicher Variablenordnungen bewiesen wurden. Für Entscheidungsprobleme wurden die redundanten Spalten für ausgabebezogene Klassen ausgelassen. Ein wesentlicher Anteil der Enthaltenseinsbeziehungen konnte von uns beantwortet werden, und wir können ablesen, dass es sich bei

IO-FPT-T $\subset$ IO-FPT-S, IO-P-T $\subset$ IO-P-S, I-FPT-S $\subset$ IO-FPT-S, I-P-S $\subset$ IO-P-S

um echte Teilmengenbeziehungen handelt.

Problem / Klasse	I-FPT-T	IO-FPT-T	I-FPT-S	IO-FPT-S	I-P-T	IO-P-T	I-P-S	IO-P-S
q-s-Pfad-Suche	$\notin$	$\notin$	?	$\in$	$\notin$	$\notin$	?	$\in$
q-Erreichbarkeit	$\notin$	$\notin$	?	$\in$	$\notin$	$\notin$	?	$\in$
q-Erreichbarkeit*	$\notin$	$\notin$	$\notin^*$	$\in$	$\notin$	$\notin$	$\notin^*$	$\in$
Transitiver Abschl.	$\notin$	$\notin$	?	$\in$	$\notin$	$\notin$	?	$\in$
Transitiver Abschl.*	$\notin$	$\notin$	$\notin^*$	$\in$	$\notin$	$\notin$	$\notin^*$	$\in$
Kreis-Suche	$\notin$	$\notin$	?	$\in$	$\notin$	$\notin$	?	$\in$
ZH-Zerl.	$\notin$	$\notin$	$\notin$	$\in$	$\notin$	$\notin$	$\notin$	$\in$
Zwei-ZH-Zerl.	$\notin$	$\notin$	$\notin$	$\in$	$\notin$	$\notin$	$\notin$	$\in$
Starke-ZH-Zerl.	$\notin$	$\notin$	$\notin$	$\in$	$\notin$	$\notin$	$\notin$	$\in$
Maximales Match.	$\notin$	?	?	$\in$	$\notin$	?	?	$\in$
Perf.-Match.-Suche	$\notin$	$\notin$	?	$\in$	$\notin$	$\notin$	?	$\in$
Flussmax.	$\notin$	$\notin$	$\notin$	$\in$	$\notin$	$\notin$	$\notin$	$\in$
0-1-Flussmax.	$\notin$	$\notin$	?	$\in$	$\notin$	$\notin$	?	$\in$
SSSP	$\notin$	$\notin$	$\notin$	$\in$	$\notin$	$\notin$	$\notin$	$\in$
SSSP$^+$	$\notin$	?	$\notin$	$\in$	$\notin$	?	$\notin$	$\in$
APSP	$\notin$	$\notin$	$\notin$	$\in$	$\notin$	$\notin$	$\notin$	$\in$
APSP$^+$	$\notin$	?	$\notin$	$\in$	$\notin$	?	$\notin$	$\in$
Topologisches Sort.	$\notin$	?	$\notin$	$\in$	$\notin$	?	$\notin$	$\in$
Minimaler Spannb.	$\notin$	$\notin$	?	$\in$	$\notin$	$\notin$	?	$\in$
Bipartitheit	$\notin$	-	$\in$	-	$\notin$	-	$\in$	-
Planarität	$\notin$	-	$\in$	-	$\notin$	-	$\in$	-
q-s-Pfad-Exist.	$\notin$	-	$\in$	-	$\notin$	-	$\in$	-
Kreis-Existenz	$\notin$	-	$\in$	-	$\notin$	-	$\in$	-
Eulerkreis-Exist.	$\notin$	-	$\in$	-	$\notin$	-	$\in$	-
ZH	$\notin$	-	$\in$	-	$\notin$	-	$\in$	-
Zwei-ZH	$\notin$	-	$\in$	-	$\notin$	-	$\in$	-
Starker ZH	$\notin$	-	$\in$	-	$\notin$	-	$\in$	-
Perf.-Match.-Exist.	$\notin$	-	$\in$	-	$\notin$	-	$\in$	-

Tabelle 7.4.1: Die betrachteten Graphprobleme und ihre Einordnung in zeit- und platzbezogene Komplexitätsklassen.

8 Fazit

Zu guter Letzt sprechen wir die verschiedenen Ergebnisse der Dissertation zusammenfassend an, ziehen ein Fazit und zeigen Möglichkeiten für weitergehende Forschungen auf.

Im Fokus dieser Dissertation standen Heuristiken für polynomiell lösbare Graphprobleme, die auf implizit OBDD-repräsentierten Instanzen geeigneter Struktur wesentlich effizienter und/oder platzsparender arbeiten als klassische Algorithmen auf üblichen Graphdarstellungen. Nach der Einführung von zum Verständnis wichtigen Grundlagen vereinbarten wir in Kapitel 2 den Begriff impliziter und OBDD-basierter Graphalgorithmen. Ein Hauptziel war es, Algorithmen anzugeben, die auf großen Graphinstanzen $G = (V, E)$ bzw. $G = (V, E, \phi)$ jedoch eingeschränkter struktureller Komplexität polylogarithmische Laufzeit in $|V|$ bzw. $|V| \cdot \max_e\{\phi(e)\}$ haben.

Es ist noch offen, ob das Problem **0-1-Flussmaximierung** in NC enthalten ist, und so ist entsprechend der Ergebnisse in Abschnitt 7.1 noch kein entsprechender impliziter P-Ops-Algorithmus bekannt. Dennoch gelang uns in Kapitel 3, die bisherigen impliziten Algorithmen (Hachtel und Somenzi, 1997; Sawitzki, 2002) zu verbessern, sodass stets nur $\log^2 |V|$ OBDD-Operationen pro Niveaunetzwerkkonstruktion und pro Berechnung mindestens eines verbessernden Pfades ausgeführt werden. Eine von der Breite vollständiger OBDDs abhängige Technik erlaubte in der Folge auch die exemplarische Analyse der Gesamtlaufzeit des neuen Algorithmus auf einfachen Gittergraphen. Sie stellen Instanzen dar, auf denen eine exponentiell bessere Laufzeit im Vergleich zu klassischen Flussalgorithmen und auch zum Algorithmus von Hachtel und Somenzi (1997) erreicht wird. Eine Fortsetzung in diesem Bereich könnte in der Entwicklung impliziter Flussmaximierungsalgorithmen für allgemeine natürliche Kantenkapazitäten bestehen.

In Kapitel 4 wandten wir uns der Übertragung bekannter exponentieller unterer Schranken für die OBDD-Größe der drei grundlegenden Funktionen MUL, ISA und HWB auf den Fall entsprechender Graphen zu, die als Konjunktion der einzelnen Ausgabefunktionen eines Funktionenvektors definiert sind. Motiviert war dies durch die Beobachtung, dass sich Graphen von Funktionen oft natürlicher in implizit dargestellte Instanzen von Graphproblemen codieren lassen, um so exponentielle untere Schranken für die Laufzeit entsprechender OBDD-basierter Algorithmen zu erhalten. Das Ergebnis für die Multiplikation MUL wurde in späteren Kapiteln genau zu diesem Zweck verwandt.

In Kapitel 5 betrachteten wir zunächst das Problem **APSP** und stellten zwei implizite Algorithmen gegenüber: Unter der Annahme, dass der Eingabegraph keine

Kanten der Länge 0 besitzt, kann das Problem durch einen FPT-Algorithmus gelöst werden. Parameter ist das Maximum der Voll-OBDD-Breiten von Eingabe- und Ausgabe-OBDD bez. einer natürlichen gemischten Variablenordnung. Dieser Parameter ist für viele Basisgraphen konstant und bleibt es nach Anwendung grundlegender Graphoperationen auch. Andererseits wurde gezeigt, dass nahe liegende Iterative-Quadrieren-Ansätze, die prinzipiell auch mit Nullkanten umgehen können, auf speziellen Instanzen (sogar ohne Nullkanten) exponentiell größere Laufzeit als der vorgestellte FPT-Algorithmus haben. Schließlich wurde unabhängig vom Ziel polylogarithmischer Laufzeit experimentell untersucht, wie sich OBDD-basierte Versionen der klassischen Algorithmen von Dijkstra und Bellman-Ford auf durchschnittlichen Graphen verhalten. Auch wenn hier die erwartete exponentielle Eingabe-OBDD-Breite dem Verhältnis von Laufzeit/Platz zur Eingabe-Größe zugutekommt, so konnte stellenweise ein nochmals wesentlich besseres Verhalten beobachtet werden, als es obere Schranken nahelegen. Insbesondere zeigte der Dijkstra-Ansatz in allen Versuchsanordnungen einen linearen Platzbedarf bez. der Eingabe-OBDD-Größe. Insgesamt zeigte der Bellman-Ford-Ansatz überlegene Laufzeit und der Dijkstra-Ansatz überlegenen Platzbedarf.

Ähnlich wurde in Kapitel 6 vorgegangen: Hier wurden OBDDs verwandt, um die Dynamische-Programmieren-Tabelle eines Algorithmus für ein eingeschränktes Stundenplanproblem kompakt zu repräsentieren. Diese Tabelle kann zusammen mit Zeigern zur Lösungskonstruktion auch als Graph aufgefasst werden. Jedoch liegt die Eingabe in ursprünglicher und OBDD-basierter Problemvariante zunächst in gleicher Form vor. So sollte der implizite Ansatz der exponentiell in der Anzahl k verschiedener Bearbeitungszeiten wachsenden Tabellengröße entgegenwirken. Wieder interessierten uns experimentell beobachtete Laufzeit und Platzbedarf auf durchschnittlichen Instanzen, die nach üblichen Verteilungen erzeugt wurden. Für jeweils feste zu verteilende Gesamtlast und festes k wurde das Verhältnis von durchschnittlicher Bearbeitungszeit und Anzahl der zu planenden Aufgaben variiert. Dabei nahm die Überlegenheit des OBDD-basierten Algorithmus in dieser Aufgabenanzahl, die auch als die eigentliche Problemgröße betrachtet werden kann, zu. Insgesamt wiesen die Experimente darauf hin, dass mit dem Einsatz von OBDDs zur impliziten Darstellung der Dynamische-Programmieren-Tabelle für hinreichend große Gesamtlast und ab einer gewissen Problemgröße eine praktisch wesentlich bessere Laufzeit erreicht werden kann (unabhängig von konstanten Laufzeitfaktoren konkreter Implementationen).

Bis zu diesem Punkt war die Dissertation vor allem auf die Entwicklung effizienter OBDD-basierter Heuristiken ausgerichtet. In Kapitel 7 wandten wir uns schließlich den Grenzen solcher Ansätze zu. Zunächst wurde dabei die Verwandtschaft impliziter Algorithmen zu parallelen Algorithmen durch ein Simulationsresultat herausgearbeitet. So können wir für P-vollständige Probleme auf ungewichteten Graphen nicht hoffen, entsprechende implizite Algorithmen mit einer in $|V|$ polylogarithmischen Laufzeit zu erhalten.

Eine Ausweitung eines PSPACE-Härteresultats für ***q*-*s*-Pfad-Existenz** auf OBDD-dargestellten Graphen führte im Anschluss zu Nichtexistenzergebnissen von Festparameteralgorithmen für einige grundlegende Graphprobleme, deren Parameter die Vollständige-Eingabe-OBDD-Breite ist (unter der Annahme P $\neq$ PSPACE). Teilweise konnten diese Negativresultate auch auf das Maximum der Vollständige-OBDD-Breiten von Eingabe- und Ausgabe-OBDDs erweitert werden. So einschränkend die Bedingungen für die in dieser Dissertation erzielten oberen Schranke auch zu sein scheinen – sie reichen dennoch für die meisten der betrachteten Probleme nicht aus, um effizientes Verhalten zu garantieren.

Schließlich wurde die Platzkomplexität betrachtet: Wenn Ausgabe-OBDDs polynomielle Größe bez. der Eingabe-OBDD-Größen haben, so lässt sich für alle betrachteten Probleme leicht ein Polynomialplatzschema angeben. Im Allgemeinen kann die Ausgabegröße der drei betrachteten Zusammenhangs-Zerlegungs-Probleme sowie der Probleme **Flussmaximierung**, **SSSP$^+$**, **APSP$^+$** und **Topologisches Sortieren** jedoch exponentiell größer sein als die der Eingabe. In eingeschränkter Weise gelang uns ein derartiges Ergebnis auch für die Probleme ***q*-Erreichbarkeit** und **Transitiver Abschluss**. In diesem Zusammenhang wurde auch die erste exponentielle untere Schranke für die OBDD-Größe der Funktion $MUL_{n,2n-1}$ für eine konkrete Variablenordnung vorgestellt.

So hat es sich also gezeigt, dass selbst unter starken Einschränkungen und bei üblichen komplexitätstheoretischen Annahmen keine effizienten Algorithmen für die meisten der betrachteten Probleme existieren. Dies widerspricht dem großen praktischen Erfolg OBDD-basierter Heuristiken aber nur scheinbar: Die Negativresultate beziehen sich auf ein Modell, in dem polylogarithmische Laufzeit in der Graphgröße bei konstanten (oder in der Praxis kleinen) Eingabe-/Ausgabe-OBDD-Breiten erreicht werden soll. Die Betrachtung der OBDD-Breiten als Festparameter, durch dessen Einschränkung wir immerhin erste obere Laufzeitschranken nachweisen konnten, war dabei lediglich ein erstes kanonisches Strukturmerkmal auf der Suche nach geeigneten Eingabemerkmalen, die den Erfolg in praktischen Anwendungen herbeiführen. Außerdem ist es möglich, dass in der Praxis erfolgreiche implizite Algorithmen asymptotisch betrachtet dennoch schwach exponentiell (bzw. polynomiell in $|V|$) sind.

In dieser Dissertation stand also die Auslotung der prinzipiellen Möglichkeiten OBDD-basierter impliziter Algorithmen aus theoretischer Sicht im Vordergrund. Im stärkeren Hinblick auf eine Erklärung des praktischen Erfolgs dieser Heuristiken führt jedoch kein Weg an einer strukturellen Betrachtung der in solchen Anwendungen auftretenden Probleminstanzen vorbei. Zudem sollten neben Laufzeiten der Größenordnung polylog($|V|$) auch weniger strikte Modelle betrachtet werden, um wenigstens für grundlegende Graphprobleme obere Schranken mit theoretischen Methoden unter moderaten Annahmen beweisen zu können. Jedoch ist noch unklar, wie solche Modelle und Methoden aussehen könnten – die Sichtweisen dieser Dissertation wurden nicht zuletzt durch problematische Binäre-Synthese-Sequenzen im Rahmen von Quantifizierungsblöcken herbeigeführt. Bei ihrer Analyse soll nicht jede schwer zu interpretierende

Zwischenfunktion einzeln betrachtet werden, und so erzwang die Exponenzierung der OBDD-Breite eine konzeptuelle Trennung dieser Breite von der Gesamt-OBDD-Größe.

Ein alternativer Ansatz zur Erklärung des praktischen OBDD-Erfolgs auf dem Hintergrund der Worst-case-Betrachtungen dieser Dissertation wäre zudem eine (zunächst experimentelle) Untersuchung impliziter Iterative-Quadrieren-Ansätze auf zufälligen Instanzen mit jedoch beschränkter Voll-OBDD-Breite. So ist in diesem Szenario die erwartete Laufzeit möglicherweise wesentlich kleiner und die Reduktionen und konkreten unteren Platzschranken stellen pathologische Fälle dar.

Schließlich soll auf Gründe für die ausschließliche Betrachtung von OBDDs als Datenstruktur für charakteristische boolesche Funktionen eingegangen werden. OBDDs gehören einerseits zu den Datenstrukturen, die in praktischen impliziten Algorithmen am verbreitetsten und etabliertesten sind. Andererseits sind ihre Eigenschaften auch in der Theorie umfangreich untersucht worden. Letzteres trifft auf viele oft nur minimal variierte BDD-Varianten nicht zu, deren Nützlichkeit nur für sehr spezielle Probleme und Instanzen experimentell belegt werden konnte. Die einfache Definition und Struktur von OBDDs ermöglichte in dieser Dissertation die Einführung und Anwendung überschaubarer Techniken. Damit stellen OBDDs also einen sinnvollen Startpunkt bei der theoretischen Betrachtung impliziter Heuristiken dar.

Lediglich eine Teilmenge der im Laufe der Zeit eingeführten Branchingprogramm-Varianten sind aufgrund ihrer algorithmischen Eigenschaften im Rahmen impliziter Algorithmen interessant. Dazu zählen k-OBDDs, k-IBDDs, FBDDs, ZBDDs, OFDDs, OKFDDs, PBDDs und EXOR-OBDDs (siehe Kapitel 12 in Wegener (2000)), die hier nur genannt werden sollen. Jedoch haben von diesen Varianten nur die ZBDDs ebenso umfassende algorithmische Eigenschaften wie OBDDs, deren Größe allerdings nicht wesentlich mehr als um einen Faktor n für die Variablenanzahl n von der entsprechender OBDDs abweicht.

Die PSPACE-Härteresultate basierten zudem auf der kompakten OBDD-Darstellung des Konfigurationsübergangsgraphen von Turingmaschinen, die durch die Lokalitätseigenschaften seiner charakteristischen Funktion ermöglicht wurden. Eine Funktion mit derartig einfacher Struktur soll einerseits von jeder BDD-Variante kompakt dargestellt werden können, die wir im algorithmischen Kontext sinnvoll betrachten möchten. Andererseits tangieren kompaktere Darstellungen dieser Funktion die erzielten Härteresultate nicht.

Abschließend stellen wir also fest, dass sich wichtige Negativresultate dieser Dissertation direkt auf andere BDD-Varianten übertragen lassen, während dies für die Positivresultate gerade nicht der Fall zu sein scheint. In anderen Situationen ist eine Übertragung aufwendiger und nur von begrenztem Interesse. So erscheinen die weiter oben angesprochenen Anknüpfungspunkte an die bereits erzielten Ergebnisse vielversprechender als der Übergang zu weiteren Datenstrukturen für boolesche Funktionen.

Symbolverzeichnis

$\mathbb{N}$	Menge der natürlichen Zahlen: $0, 1, 2, \ldots$
$\mathbb{N}^+$	Menge der positiven natürlichen Zahlen: $1, 2, \ldots$
$\mathbb{Z}$	Menge der ganzen Zahlen
$\mathbb{R}$	Menge der reellen Zahlen
e	eulersche Zahl
i	imaginäre Zahl
$\lvert a \rvert$	Betrag der Zahl $a \in \mathbb{R}$
$\lceil a \rceil$	kleinste ganze Zahl $\geq a \in \mathbb{R}$
$\lfloor a \rfloor$	größte ganze Zahl $\leq a \in \mathbb{R}$
$\log(a)$	Logarithmus zur Basis 2
$[a, b]$	abgeschlossenes, reellwertiges Interval mit Grenzen $a, b \in \mathbb{R}$
$]a, b[$	offenes, reellwertiges Interval mit Grenzen $a, b \in \mathbb{R}$
$n!$	Fakultätsfunktion $\Pi_{i=1}^{n} i$
$\binom{n}{k}$	Binomialkoeffizient $n!/(k!(n-k)!)$
$\text{poly}(n)$	$n^{\mathcal{O}(1)}$ (polynomiell in n beschränkt)
$\text{polylog}(n)$	$(\log n)^{\mathcal{O}(1)}$ (polylogarithmisch in n beschränkt)
$\text{Prob}(A)$	Wahrscheinlichkeit des Ereignisses A
$\text{E}[X]$	Erwartungswert der Zufallsvariablen X
$\lvert M \rvert$	Kardinalität der Menge M
$\mathcal{P}(M)$	Potenzmenge der Menge M

M^* ..... kleenscher Abschluss der Menge M

$M_1 \subseteq M_2$ ..... $(e \in M_1) \Rightarrow (e \in M_2)$

$M_1 \subset M_2$ ..... $(M_1 \subseteq M_2) \wedge (M_1 \neq M_2)$

$\Lambda(I)$ ..... Länge N eines Vektors $I \in M^N \subset M^*$

Σ_M ..... Menge aller bijektiven Abbildungen $\{1, \ldots, |M|\} \to M$

Σ_n ..... Menge aller bijektiven Abbildungen $\{1, \ldots, n\} \to \{1, \ldots, n\}$

$\mathbb{B}$ ..... $\{0, 1\}$

$|x|$ ..... Wert $\sum_i x_i \cdot 2^i$ des binären Vektors $x \in \mathbb{B}^n$

$||x||$ ..... Anzahl $\sum_i x_i$ der Einsen im binären Vektor $x \in \mathbb{B}^n$

$(a)_2$ ..... Binärdarstellung x von $a \in \mathbb{N}$ mit $|x| = a$ und kontextabhängiger Länge

$\langle v \rangle$ ..... Nummer i des Graphknotens $v = v_i$

$\langle v \rangle_2$ ..... $(\langle v \rangle)_2$

$B_{n,m}$ ..... Menge aller booleschen Funktionen $\mathbb{B}^n \to \mathbb{B}^m$

B_n ..... Kurzform für $B_{n,1}$

$f \mid x := c$ ..... Subfunktion einer booleschen Funktion f für die Konstantsetzung $x := c$

χ_M ..... charakteristische boolesche Funktion der Menge M

$\mathbb{T}^W_{k,n}$ ..... Menge der k-variaten Thresholdfunktionen auf kn Variablen mit maximalem absolutem Gewicht W

$\mathbb{V}^W_{k,n}$ ..... Menge der k-variaten Vergleichsfunktionen auf kn Variablen mit maximalem absolutem Gewicht W

$\mathbb{M}^M_{k,n}$ ..... Menge der k-variaten Modulo-M-Funktionen auf kn Variablen

Abbildungsverzeichnis

Algorithmenverzeichnis

Literaturverzeichnis

Abello, J., Buchsbaum, A. und Westbrook, J. (2002): A functional approach to external graph algorithms, *Algorithmica*, **32**, S. 437–458

Ahuja, R. K., Magnanti, T. L. und Orlin, J. B. (1993): *Network Flows*, Prentice Hall

Albers, S. und Schröder, B. (2002): An experimental study of online scheduling algorithms, *Journal of Experimental Algorithms*, **7**, S. 3

Allender, E. und Mahajan, M. (2000): The complexity of planarity testing, *Information and Computation*, **189**(1), S. 117–134

Alon, N. und Maass, W. (1988): Meanders and their applications in lower bound arguments, *Journal of Computer and System Sciences*, **37**, S. 118–129

Bahar, R. I., Frohm, E. A., Gaona, C. M., Hachtel, G. D., Macii, E., Pardo, A. und Somenzi, F. (1993): Algebraic decision diagrams and their applications, in: *International Conference on Computer Aided Design 1993*, S. 188–191, IEEE Press

Bahar, R. I., Hachtel, G. D., Pardo, A., Poncino, M. und Somenzi, F. (1994): An ADD-based algorithm for shortest path back-tracing of large graphs, in: *Great Lakes Symposium on VLSI 1994*, S. 248–251, IEEE Press

Balcázar, J. L. und Lozano, A. (1989): The complexity of graph problems for succinctly represented graphs, in: *Graph-Theoretic Concepts in Computer Science 1989*, Band 411 von *Lecture Notes in Computer Science*, S. 277–285, Springer

Beame, P., Jayram, T. S. und Saks, M. E. (2001): Time-space tradeoffs for branching programs, *Journal of Computer and System Sciences*, **63**(4), S. 542–572

Becker, B., Behle, M., Eisenbrand, F. und Wimmer, R. (2005): BDDs in a branch and cut framework, in: *International Workshop on Experimental and Efficient Algorithms 2005*, Band 3503 von *Lecture Notes in Computer Science*, S. 452–463, Springer

Bellman, R. (1958): On a routing problem, *Quarterly of Applied Mathematics*, **16**, S. 87–90

Berghammer, R. und Neumann, F. (2005): RELVIEW - an OBDD-based computer algebra system for relations, in: *Workshop on Computer Algebra in Scientific Computing 2005*, Band 3718 von *Lecture Notes in Computer Science*, S. 40–51, Springer

Bloem, R., Gabow, H. N. und Somenzi, F. (2000): An algorithm for strongly connected component analysis in $n \log n$ symbolic steps, in: *Formal Methods in Computer-Aided Design 2000*, Band 1954 von *Lecture Notes in Computer Science*, S. 37–54, Springer

Bollig, B. und Wegener, I. (1996): Improving the variable ordering of OBDDs is NP-complete, *IEEE Transactions on Computers*, **45**, S. 993–1002

Borodin, A. und Cook, S. (1982): A time–space tradeoff for sorting on a general sequential model of computation, *SIAM Journal on Computing*, **11**, S. 287–297

Brandstädt, A., Le, V. B. und Spinrad, J. P. (1999): *Graph Classes: A Survey*, Monographs on Discrete Mathematics and Applications, SIAM Press

Breitbart, Y., Hunt III, H. B. und Rosenkrantz, D. (1995): On the size of binary decision diagrams representing Boolean functions, *Theoretical Computer Science*, **145**, S. 45–69

Bryant, R. E. (1985): Symbolic manipulation of Boolean functions using a graphical representation, in: *Design Automation Conference 1985*, S. 688–694, ACM Press

Bryant, R. E. (1986): Graph-based algorithms for Boolean function manipulation, *IEEE Transactions on Computers*, **35**, S. 677–691

Bryant, R. E. und Chen, Y.-A. (1997): Verification of arithmetic functions with binary moment diagrams, in: *Design Automation Conference 1997*, S. 535–541, IEEE Press

Buchsbaum, A. L., Goldwasser, M., Venkatasubramian, S. und Westbrook, J. R. (2000): On external memory graph traversal, in: *Symposium on Discrete Algorithms 2000*, S. 859–860, ACM Press

Burch, J. R. (1991): Using BDDs to verify multipliers, in: *Design Automation Conference 1991*, S. 403–407, ACM Press

Burch, J. R., Clarke, E. M., McMillan, K. L., Dill, D. J. und Hwang, L. J. (1992): Symbolic model checking: 10^{30} states and beyond, *Information and Computation*, **98**, S. 142–170

Chandra, A. K., Stockmeyer, L. und Vishkin, U. (1984): Constant depth reducibility, *SIAM Journal on Computing*, **13**(2), S. 423–439

Chiang, Y.-J., Goodrich, M. T., Grove, E. F., Tamassia, R., Vengroff, D. E. und Vitter, J. S. (1995): External-memory graph algorithms, in: *Symposium on Discrete Algorithms 1995*, S. 139–149, ACM Press

Cormen, T. H., Leiserson, C. E., Rivest, R. L. und Stein, C. (2001): *Introduction to Algorithms*, MIT Press

Coudert, O. (1995): Doing two-level logic minimization 100 times faster, in: *Symposium on Discrete Algorithms 1995*, S. 112–121, ACM Press

Dehne, F., Hutchinson, D. und Maheshwari, A. (1999): Reducing I/O-complexity by simulating coarse grained parallel algorithms, in: *International Parallel Processing Symposium 1999*, S. 14–20, IEEE Press

Dijkstra, E. W. (1959): A note on two problems in connexion with graphs, *Numerische Mathematik*, **1**, S. 269–271

Downey, R. G. und Fellows, M. R. (1999): *Parameterized Complexity*, Springer

Drechsler, R., Becker, B. und Ruppertz, S. (1996): K*BMDs: A new data structure for verification, in: *European Design and Test Conference 1996*, S. 2–8, IEEE Press

Droste, S., Jansen, T. und Wegener, I. (2002): On the analysis of the (1+1) evolutionary algorithm, *Theoretical Computer Science*, **276**, S. 51–81

Eppstein, D., Galil, Z., Italiano, G. F. und Nissenzweig, A. (1997): Sparsification—a technique for speeding up dynamic graph algorithms, *Journal of the ACM*, **44**, S. 669–696

Even, S. (1979): *Graph Algorithms*, Computer Science Press

Feigenbaum, J., Kannan, S., Vardi, M. Y. und Viswanathan, M. (1998): Complexity of problems on graphs represented as OBDDs, in: *Symposium on Theoretical Aspects of Computer Science 1998*, Band 1373 von *Lecture Notes in Computer Science*, S. 216–226, Springer

Fleischer, L. und Skutella, M. (2002): The quickest multicommodity flow problem, in: *Integer Programming and Combinatorial Optimization 2002*, Band 2337 von *Lecture Notes in Computer Science*, S. 36–53, Springer

Ford, L. R. und Fulkerson, D. R. (1958): Constructing maximal dynamic flows from static flows, *Operations Research*, **6**, S. 419–433

Fujita, M., McGeer, P. C. und Yang, J. (1997): Multi-terminal binary decision diagrams: An efficient data structure for matrix representation, *Formal Methods in System Design*, **10**, S. 149–169

Galperin, H. und Wigderson, A. (1983): Succinct representations of graphs, *Information and Control*, **56**, S. 183–198

Garey, M. R. und Johnson, D. S. (1979): *Computers and Intractability: A Guide to the Theory of NP-Completeness*, Freeman

Gavalda, R. und Guijarro, D. (1995): Learning ordered binary decision diagrams, in: *Algorithmic Learning Theory 1995*, Band 997 von *Lecture Notes in Computer Science*, S. 228–238, Springer

Gentilini, R., Piazza, C. und Policriti, A. (2003): Computing strongly connected components in a linear number of symbolic steps, in: *Symposium on Discrete Algorithms 2003*, S. 573–582, ACM Press

Gentilini, R. und Policriti, A. (2003): Biconnectivity on symbolically represented graphs: A linear solution, in: *International Symposium on Algorithms and Computation 2003*, Band 2906 von *Lecture Notes in Computer Science*, S. 554–564, Springer

Gergov, J. (1994): Time-space tradeoffs for integer multiplication on various types of input oblivious sequential machines, *Information Processing Letters*, **51**, S. 265–269

Greenlaw, R., Hoover, H. J. und Ruzzo, W. L. (1995): *Limits to Parallel Computation*, Oxford University Press

Hachtel, G. D., Hermida, M., Pardo, A., Poncino, M. und Somenzi, F. (1994): Re-encoding sequential circuits to reduce power dissipation, in: *International Conference on Computer Aided Design 1994*, S. 70–73, IEEE Press

Hachtel, G. D. und Somenzi, F. (1996): *Logic Synthesis and Verification Algorithms*, Kluwer Academic Publishers

Hachtel, G. D. und Somenzi, F. (1997): A symbolic algorithm for maximum flow in 0–1 networks, *Formal Methods in System Design*, **10**, S. 207–219

Hojati, R., Touati, H., Kurshan, R. P. und Brayton, R. K. (1993): Efficient ω-regular language containment, in: *Computer-Aided Verification 1993*, Band 663 von *Lecture Notes in Computer Science*, S. 396–409, Springer

Holzer, M., Schulz, F. und Willhalm, T. (2004): Combining speed-up techniques for shortest-path computations, in: *International Workshop on Experimental and Efficient Algorithms 2004*, Band 3059 von *Lecture Notes in Computer Science*, S. 269–284, Springer

Hosaka, K., Takenaga, Y., Kaneda, T. und Yajima, S. (1997): Size of ordered binary decision diagrams representing threshold functions, *Theoretical Computer Science*, **180**(1–2), S. 47–60

Hromkovič, J. (1997): *Communication Complexity and Parallel Computing*, Springer

Ishiura, N., Sawada, H. und Yajima, S. (1991): Minimization of binary decision diagrams based on exchanges of variables, in: *International Conference on Computer Aided Design 1991*, S. 472–475, IEEE Press

Jones, N. D., Lien, Y. E. und Laaser, W. T. (1976): New problems complete for nondeterministic log space, *Mathematical Systems Theory*, **10**, S. 1–17

Jukna, S. (1995): The graph of integer multiplication is hard for read-k-times networks, Technischer Bericht 95–10, Universität Trier

Keim, M., Drechsler, R., Becker, B., Martin, M. und Molitor, P. (2003): Polynomial formal verification of multipliers, *Formal Methods in System Design*, **22**, S. 39–58

Kuipers, L. und Niederreiter, H. (1974): *Uniform Distribution of Sequences*, Wiley

Lai, Y.-T., Pedram, M. und Vrudhula, S. B. K. (1994): EVBDD-based algorithms for integer linear programming, spectral transformation, and function decomposition, *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, **13**, S. 959–975

Leung, J. Y.-T. (1982): On scheduling independent tasks with restricted execution times, *Operations Research*, **30**(1), S. 163–171

Meer, K. und Rautenbach, D. (2006): On the OBDD size for graphs of bounded tree- and clique-width, Technischer Bericht 06956, Forschungsinstitut für Diskrete Mathematik, Universität Bonn

Munagala, K. und Ranade, A. (1999): I/O-complexity of graph algorithms, in: *Symposium on Discrete Algorithms 1999*, S. 687–694, ACM Press

Nunkesser, R. und Woelfel, P. (2005): Representation of graphs by OBDDs, in: *International Symposium on Algorithms and Computation 2005*, Band 3827 von *Lecture Notes in Computer Science*, S. 1132–1142, Springer

Papadimitriou, C. H. und Yannakakis, M. (1986): A note on succinct representations of graphs, *Information and Control*, **71**, S. 181–185

Sawitzki, D. (2002): *Implizite Algorithmen für Graphprobleme*, Diplomarbeit, Lehrstuhl Informatik 2, Universität Dortmund

Sawitzki, D. (2004a): A symbolic approach to the all-pairs shortest-paths problem, in: *Graph-Theoretic Concepts in Computer Science 2004*, Band 3353 von *Lecture Notes in Computer Science*, S. 154–167, Springer

Sawitzki, D. (2004b): Experimental studies of symbolic shortest-path algorithms, in: *International Workshop on Experimental and Efficient Algorithms 2004*, Band 3059 von *Lecture Notes in Computer Science*, S. 482–497, Springer

Sawitzki, D. (2004c): Implicit flow maximization by iterative squaring, in: *SOFSEM 2004: Theory and Practice of Computer Science*, Band 2932 von *Lecture Notes in Computer Science*, S. 301–313, Springer

Sawitzki, D. (2004d): Implicit maximization of flows over time, Technischer Bericht, Universität Dortmund

Sawitzki, D. (2004e): On graphs with characteristic bounded-width functions, Technischer Bericht, Universität Dortmund

Sawitzki, D. (2005a): Lower bounds on the OBDD size of graphs of some popular functions, in: *SOFSEM 2005: Theory and Practice of Computer Science*, Band 3381 von *Lecture Notes in Computer Science*, S. 298–309, Springer

Sawitzki, D. (2005b): On symbolic scheduling independent tasks with restricted execution times, in: *International Workshop on Experimental and Efficient Algorithms 2005*, Band 3503 von *Lecture Notes in Computer Science*, S. 277–289, Springer

Sawitzki, D. (2006a): Exponential lower bounds on the space complexity of OBDD-based graph algorithms, in: *Latin American Theoretical Informatics 2006*, Band 3887 von *Lecture Notes in Computer Science*, S. 781–792, Springer

Sawitzki, D. (2006b): The complexity of problems on implicitly represented inputs, in: *SOFSEM 2006: Theory and Practice of Computer Science*, Band 3831 von *Lecture Notes in Computer Science*, S. 471–482, Springer

Sawitzki, D. (2007): Lower bounds on the OBDD size of two fundamental functions' graphs, *Information Processing Letters*, **101**(2), S. 66–71

Sieling, D. und Wegener, I. (1993a): NC-algorithms for operations on binary decision diagrams, *Parallel Processing Letters*, **3**, S. 3–12

Sieling, D. und Wegener, I. (1993b): Reduction of OBDDs in linear time, *Information Processing Letters*, **48**, S. 139–144

Smith, T. F. und Waterman, M. S. (1981): Identification of common molecular subsequences, *Journal of Molecular Biology*, **147**, S. 195–197

Touati, H. J., Brayton, R. K. und Kurshan, R. P. (1995): Testing language containment for ω-automata using BDD's, *Information and Computation*, **118**, S. 101–109

Wegener, I. (2000): *Branching Programs and Binary Decision Diagrams*, Monographs on Discrete Mathematics and Applications, SIAM Press

Wegener, I. (2003): *Komplexitätstheorie*, Springer

Wegener, I. (2004): BDDs—design, analysis, complexity, and applications, *Discrete Applied Mathematics*, **138**, S. 229–251

Woelfel, P. (2001): New bounds on the OBDD-size of integer multiplication via universal hashing, in: *Symposium on Theoretical Aspects of Computer Science 2001*, Band 2010 von *Lecture Notes in Computer Science*, S. 563–574, Springer

Woelfel, P. (2003): Symbolic topological sorting with OBDDs, in: *Mathematical Foundations of Computer Science 2003*, Band 2747 von *Lecture Notes in Computer Science*, S. 671–680, Springer

Xie, A. und Beerel, P. A. (1999): Implicit enumeration of strongly connected components, in: *International Conference on Computer Aided Design 1999*, S. 37–40, ACM Press

Zeng, L. (2005): *Implizite Algorithmen für einfache Graphprobleme*, Diplomarbeit, Lehrstuhl Informatik 2, Universität Dortmund